CHL5224 Assignment#1

Whitney (Huei-Chen) Chiu
Student Number 1004823596

The ABO blood group system in humans reflects a geneetic variation at a location which there are three alleles: A, B, O. In this system, the alleles A and B are each dominant to O, and A and B are codominant to each other. That is, there are 4 phenotypes, corresponding to genotypes {AA, AO}, {BB, BO}, {AB}, and {OO}. Let n be the number of people in the study, while nAA be the number of people with AA genotype, n_{AO} be the number of people with the AO genotype, n_{AB} be the number of people with the AB genotype, and n_B be the number of people with the BB genotype. The observed data is (n_A, n_{AB}, n_B, n_O). Let p, q and $1-p-q$ be the frequency of allele A, B and O respectively. We would like to obtain the maximum likelihood estimates (MLEs) of the underlying allele frequencies p, q, and $1-p-q$. For $\theta(p,q)$, the likelihood is

$$\begin{aligned}L(\theta;Y) = &\frac{n!}{n_A!\; n_B!\; n_{AB}!\; n_O!} + n_A[p^2 + 2p(1-p-q)] \\ &+ n_B[q^2 + 2q(1-p-q)] + n_{AB}(2pq) + n_O(1-p-q)^2.\end{aligned}$$

The log-likelihood can be approxiamted as

$$\begin{aligned}logL(\theta;Y) \approx &\; n_A[p^2 + 2p(1-p-q)] \\ &+ n_B[q^2 + 2q(1-p-q)] + n_{AB}(2pq) + n_O(1-p-q)^2.\end{aligned}$$

To estimate the allele frequency of the ABO blood type, three different types of methods were implemented in the R codes below, including two iterative algorithms and the grid search method.

The Estimation-Maximization algorithm, known as the EM algorithm, is an iterative way to approximate the maximum likelihood function. The algorithm iterates between the E-Step (Expectation) and the M-Step (Maximization). In the E-Step, we start with an first estimation of the log-likelihood by inputting an initial guess for the $\theta^0(p,q)$. Next, in the M-step, we calculate the expected n_{AA} and n_{BB}, given the observed data and θ^k to maximize the expected log-likelihood found on the E-step. The new $\theta^{k+1}(p,q)$ is used to create a better guess for the first set, and the iterative process continues until the algorithm converges on a fixed point.

The Newton-Raphson algorithm is another iterative approach used to obtain the local maximum or minimum of a function. The algorithm is based on the first and the second derivative of the function. By taking the first and second derivative of the function, a gradient vector f' and a Hessian matrix f'' is produced respectively. The full equation is

$$\theta^{k+1} = \theta^k - f''(\theta_k)^{-1} f'(\theta_k)$$

The algorithm also starts with a initial p and q value that we specify, and for every iteration, a new p and q updates according to the following function. In the end, the p and q converges to the value that maximizes the functions.

Grid searching is the other method that we implemented to find the probability of allele A, p, and allele B, q, that maximize the MLEs. First, an appropriate grid size is determined and the range of p and q is defined: $0 \le p \le 1$, $0 \le q \le 1$, $p + q \le 1$. In our case, a grid size of 0.01 was applied so there would be 100 possible values for both p and q with 10000 possible combinations when estimating the MLEs. After filtering all the invalid results such as $p + q \ge 1$, we produced a scatterplot accordingly to visualize all the probabilities.

By setting the same initial p and q ($p = 0.3$, $q = 0.4$) and the same stopping criteria ($\theta^{k+1} - \theta^k < 1e-12$) for the EM algorithm and the Newton-Raphson algorithm, we found that the speed of convergence was faster in

EM algorithm than in the Newton-Raphson algorithm with an average convergence time of 1.541773×10^{-5} compared to 0.002832413. It took 15 iterations for the EM algorithm to converge and 8 for the Newton-Raphson algorithm. In summary, EM is faster in average convergence speed for each iteration despite having done more iterations. Since there are only two parameters in this case, the two algorithms are both applicable. However, in cases which the number of parameters is large, it would be more feasible to implement the EM instead of the Newton-Raphson since the computational cost would be intractable to inverse the Hessian matrix f''.

Compared to the EM and the Newton-Raphson algorithm, grid searching is a brute force method which is inefficient and computationally heavy. It would never produce an estimation that is as accurate as the other two algorithms.

EM algorithm implementation

```
nA = 9123
nB = 2987
nAB = 1269
nO = 7725

# output: nAA, nAO, nBB, nBO, nAB, nOO.

p = 0.3
q = 0.4
n = nA + nB + nAB + nO

pOld = Inf
qOld = Inf

totalTime = 0
totalIter = 0
while((abs(p - pOld) > 1e-12) && (abs(q - qOld) > 1e-12))
{
  startTime <- Sys.time()

  # save old results
  pOld = p
  qOld = q

  # E step
  nAA = nA * (p*p)         / (p*p + 2*p*(1-p-q))
  nAO = nA * (2*p*(1-p-q)) / (p*p + 2*p*(1-p-q))
  nBB = nB * (q*q)         / (q*q + 2*q*(1-p-q))
  nBO = nB * (2*q*(1-p-q)) / (q*q + 2*q*(1-p-q))

  # M step
  p = (2*nAA + nAO + nAB) / (2*n)
  q = (2*nBB + nBO + nAB) / (2*n)

  endTime <- Sys.time()
  totalTime = (endTime - startTime) + totalTime
  totalIter = totalIter + 1

  cat("p=",p,", ")
  cat("q=",q,"\n")
```

```r
  Sys.sleep(0.01)
}
```

```
## p= 0.3182572 , q= 0.1291414
## p= 0.2945347 , q= 0.108238
## p= 0.2889643 , q= 0.106714
## p= 0.2879143 , q= 0.1065754
## p= 0.287726 , q= 0.1065582
## p= 0.2876927 , q= 0.1065555
## p= 0.2876868 , q= 0.1065551
## p= 0.2876858 , q= 0.106555
## p= 0.2876856 , q= 0.106555
## p= 0.2876856 , q= 0.106555
## p= 0.2876856 , q= 0.106555
## p= 0.2876856 , q= 0.106555
## p= 0.2876856 , q= 0.106555
## p= 0.2876856 , q= 0.106555
## p= 0.2876856 , q= 0.106555
```

```r
cat('avg run time:', totalTime/totalIter, '\n')
```

```
## avg run time: 1.993179e-05
```

```r
cat('total iterations:', totalIter, '\n')
```

```
## total iterations: 15
```

Newton-Raphson algorithm implementation

```r
nA = 9123
nB = 2987
nAB = 1269
nO = 7725

library(numDeriv)

f<-function(x)
{
  nA*log(x[1]^2+2*x[1]*(1-x[1]-x[2]))+
    nB*log(x[2]^2+2*x[2]*(1-x[1]-x[2]))+
    nAB*log(2*x[1]*x[2])+nO*log((1-x[1]-x[2])^2)
}

x0=c(0.3,0.4)

xk=x0
xkplusone=Inf
dif=abs(xkplusone-xk)
print(dif[1])
```

```
## [1] Inf
```

```r
dif<-c(1,1)

totalTime = 0
totalIter = 0
```

```r
while((dif[1]>1e-8 && dif[2]>1e-8))
{
  startTime <- Sys.time()

  fprime<-grad(f,xk)
  fpprime<-hessian(f,xk)
  invfpprime<-solve(fpprime)
  xkplusone=xk-invfpprime%*%fprime
  dif=abs(xkplusone-xk)
  xk=xkplusone
  print(xkplusone)

  endTime <- Sys.time()
  totalTime = (endTime - startTime) + totalTime
  totalIter = totalIter + 1

  cat("dif1 =", dif[1], ", dif2 =", dif[2], "\n")
  Sys.sleep(0.01)
}
```

```
##           [,1]
## [1,] 0.4295992
## [2,] 0.0343494
## dif1 = 0.1295992 , dif2 = 0.3656506
##            [,1]
## [1,] 0.28643911
## [2,] 0.05875079
## dif1 = 0.1431601 , dif2 = 0.02440139
##            [,1]
## [1,] 0.29422827
## [2,] 0.08588796
## dif1 = 0.007789157 , dif2 = 0.02713717
##           [,1]
## [1,] 0.2887908
## [2,] 0.1028507
## dif1 = 0.005437451 , dif2 = 0.01696271
##           [,1]
## [1,] 0.2877203
## [2,] 0.1064403
## dif1 = 0.001070538 , dif2 = 0.003589634
##           [,1]
## [1,] 0.2876856
## [2,] 0.1065549
## dif1 = 3.468849e-05 , dif2 = 0.0001145894
##           [,1]
## [1,] 0.2876856
## [2,] 0.1065550
## dif1 = 3.290026e-08 , dif2 = 1.090009e-07
##           [,1]
## [1,] 0.2876856
## [2,] 0.1065550
## dif1 = 1.245998e-11 , dif2 = 2.383371e-12
```

```r
end
```

```
## function (x, ...)
## UseMethod("end")
## <bytecode: 0x7f93abb3d718>
## <environment: namespace:stats>
```

```r
cat('avg run time:', totalTime/totalIter, '\n')
```

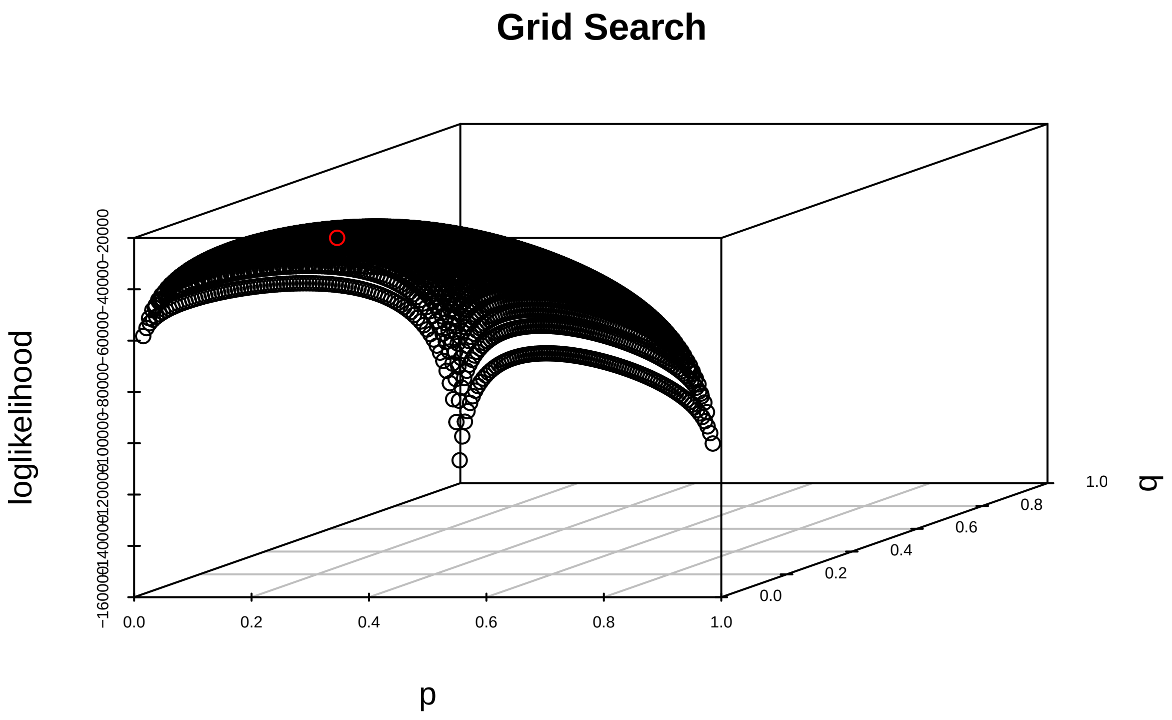
```
## avg run time: 0.003459275
```

```r
cat('total iterations:', totalIter, '\n')
```

```
## total iterations: 8
```

Grid search implementation

```r
library("scatterplot3d")
nA = 9123
nB = 2987
nAB = 1269
nO = 7725

# initialize loglikelihood function
objFunc<-function(p,q)
{
  nA*log(p^2+2*p*(1-p-q))+
    nB*log(q^2+2*q*(1-p-q))+
    nAB*log(2*p*q)+nO*log((1-p-q)^2)
}

# set grid size
gridSize = 0.01

# save p, q, val history
pList = c()
qList = c()
vList = c()

p = 0.0 # initial p
while( p <= 1.0 ){

  q = 0.0 # initial q
  while( q <= 1.0 && (p+q) <= 1.0){
    val = objFunc(p,q) # compute val

    # save history of p,q,val
    # cat("p=",p,", q=",q, ", v=",val,"\n")
    pList = c(pList, p)
    qList = c(qList, q)
    vList = c(vList, val)

    # update q
    q = q + gridSize
  }
  # update p
  p = p + gridSize
}

pList = pList[is.finite(vList)]
qList = qList[is.finite(vList)]
vList = vList[is.finite(vList)]

# print(pList)
# print(qList)
# print(vList)

scatterplot3d(pList,qList,vList, main="Grid Search",xlab="p",ylab="q",zlab="loglikelihood",cex.axis=0.5
```


Grid Search
loglikelihood
p
q