Homework 2 on Newton's methods

Leave your name and uni here

Due: 03/18/2020, Wednesday, by 1pm

Problem 1

Design an optmization algorithm to find the minimum of the continuously differentiable function $f(x) = -e^{-x}\sin(x)$ on the closed interval $[0, 1.5]$. Write out your algorithm and implement it into **R**.

Answer:

$$f(x) = -e^{-x} * \sin(x)$$

```
f = function(x){
  return(-exp(-x)*sin(x))
} # original function

find_min = function(){
  w = 0.618
  a = 0
  b = 1.5
  x1 = (1 - w) * (b - a) + a
  x2 = x1 + w * (b - a) * (1 - w)
  while (abs(x1 - x2) > 1e-20) {
    if( f(x1) < f(x2) ) {
      a = a
      b = x2
      #print(a,b)
    }
    else {
      a = x1
      b = b
      #print(a,b)
    }
     x1 = (1 - w) * (b - a) + a
     x2 = x1 + w * (b - a) * (1 - w)
    result = f(a)
  }
  return(result)
  }
```

```r
# result
find_min()
```

```
## [1] -0.3223969
```

```r
f(pi/4)
```

```
## [1] -0.3223969
```

Problem 2

The Poisson distribution is often used to model "count" data — e.g., the number of events in a given time period.
The Poisson regression model states that

$$Y_i \sim \text{Poisson}(\lambda_i),$$

where

$$\log \lambda_i = \alpha + \beta x_i$$

for some explanatory variable x_i. The question is how to estimate α and β given a set of independent data $(x_1, Y_1), (x_2, Y_2), \ldots, (x_n, Y_n)$.

1. Modify the Newton-Raphson function from the class notes to include a step-halving step.
2. Further modify this function to ensure that the direction of the step is an ascent direction. (If it is not, the program should take appropriate action.)
3. Write code to apply the resulting modified Newton-Raphson function to compute maximum likelihood estimates for α and β in the Poisson regression setting.

The Poisson distribution is given by

$$P(Y = y) = \frac{\lambda^y e^{-\lambda}}{y!}$$

for $\lambda > 0$.

Answer: your answer starts here...

log-likelihood function of the posisson distribution:

$$L(\lambda; y) = \sum_{i=1}^{n} [y_i \log \lambda - \lambda - \log y_i!]$$

```r
# define a function to get loglikelihood function, gradience and Hessian matrix
poissonstuff = function(dat, betavec) {
  log_lambda = betavec[1] + betavec[2] * dat$x
  lambda = exp(log_lambda)
  loglik <- sum(dat$y * log_lambda - lambda - log(factorial(dat$y))) # Log-likelihood at betavec
```

```
  grad <- c(sum(dat$y - lambda), sum(dat$x * dat$y - dat$x * lambda))
  # gradient at betavec
  Hess <- matrix(c(sum((-1)*lambda),rep(sum((-1)*dat$x * lambda),2),sum(((-1)*dat$x^2 * lambda))), ncol
  # Hessian at betavec
  return(list(loglik = loglik, grad = grad, Hess = Hess))
  }
```

```
set.seed(886)

generate_data = function(alpha,beta){
  x = rnorm(5000,sd = 0.1)
  true_beta = c(alpha,beta) # alpha ,beta
  lambda = exp(true_beta[1] + true_beta[2] * x)
  y = NULL
  for (i in 1:5000) {
    y[i] = rpois(1,lambda = lambda[i])
  }
  df = tibble::tibble(x = x,
                      y = y)
return(df)

}
```

```
NewtonRaphson = function(dat, start, tol=1e-10, maxiter = 2000, func = poissonstuff) {
  i = 0
  cur = start
  stuff = func(dat,cur)
  res = c(0, stuff$loglik, cur)
  loglik = stuff$loglik
  prevloglik = -Inf  # To make sure it iterates
  while (i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
    l = 1 #lambda / step
    i = i + 1
    prevloglik = stuff$loglik
    hessian  = stuff$Hess
    grad = stuff$grad
    prev = cur
    if( t(grad) %*% hessian %*% grad > 0 ){
      #print("hessian is not negative definite and has been transformed")
      hessian = hessian - 10*max(diag(hessian)) * diag(nrow(hessian))
      }
    else{hessian = hessian}
    cur = prev - l * solve(hessian) %*% grad
    stuff = func(dat, cur) # log-lik, gradient, Hessian
    while (stuff$loglik < prevloglik) {
      l = 0.5 * l
      cur = prev - l * solve(hessian) %*% grad
      stuff = func(dat, cur)
    }
    res = rbind(res, c(i, stuff$loglik, cur)) # Add current values to results matrix
  }
  colnames(res) = c("iter","likelihood","alpha","beta")
  return(res)
```

```
  }
```

```
set.seed(886)
# true alpha = 2, true beta = 7
dat = generate_data(2,7)
#start = c(10,10)
res_1 = NewtonRaphson(dat,start = c(3,7))
#start = c(-1,1)
res_2 = NewtonRaphson(dat,start = c(3,1))
#start = c(-5,-5)
res_3 = NewtonRaphson(dat,start = c(40,5))
#start = c(-2,8)
res_4 = NewtonRaphson(dat,start = c(-2,8))

set.seed(886)
# true alpha = 1, true beta = 9
dat_2 = generate_data(1,9)
#start = c(10,10)
res_1_2 = NewtonRaphson(dat_2,start = c(2,2))
#start = c(3,1)
res_2_2 = NewtonRaphson(dat_2,start = c(0,0))
#start = c(400,5)
res_3_2 = NewtonRaphson(dat_2,start = c(1,1))
#start = c(-2,8)
res_4_2 = NewtonRaphson(dat_2,start = c(-0.5,7))
```

problem 3

Consider the ABO blood type data, where you have $N_{\text{obs}} = (N_A, N_B, N_O, N_{AB}) = (26, 27, 42, 7)$.

- design an EM algorithm to estimate the allele frequencies, P_A, P_B and P_O; and
- Implement your algorithms in R, and present your results..

Answer: your answer starts here. . .

```
#R codes:
```