Homework 2 on Newton's methods

Mengyu Zhang / mz2777

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

Problem 1

Design an optimization 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

By using Golden Section search in $[0, 1.5]$, $x_1 = upper.bound - golden.ratio \times (upper.bound - lower.bound)$, $x_2 = upper.bound - golden.ratio^2 \times (upper.bound - lower.bound)$. The minimum of $f(x)$ is -0.3223969 at point 0.7854006.

```
df = function(x){
  return(-exp(-x)*sin(x))
}

golden.section.search = function(f, lower.bound, upper.bound, tolerance)
{
   golden.ratio = 2/(sqrt(5) + 1)

   ### Use the golden ratio to set the initial test points
   x1 = upper.bound - golden.ratio*(upper.bound - lower.bound)
   x2 = upper.bound - golden.ratio^2*(upper.bound - lower.bound)

   ### Evaluate the function at the test points
   f1 = f(x1)
   f2 = f(x2)

   iteration = 0

   res = NULL

   while (abs(upper.bound - lower.bound) > tolerance)
   {
      iteration = iteration + 1

      if (f2 > f1)
      {
         ### Set the new upper bound
         upper.bound = x2
         x2 = x1
         f2 = f1

         x1 = upper.bound - golden.ratio*(upper.bound - lower.bound)
         f1 = f(x1)
```

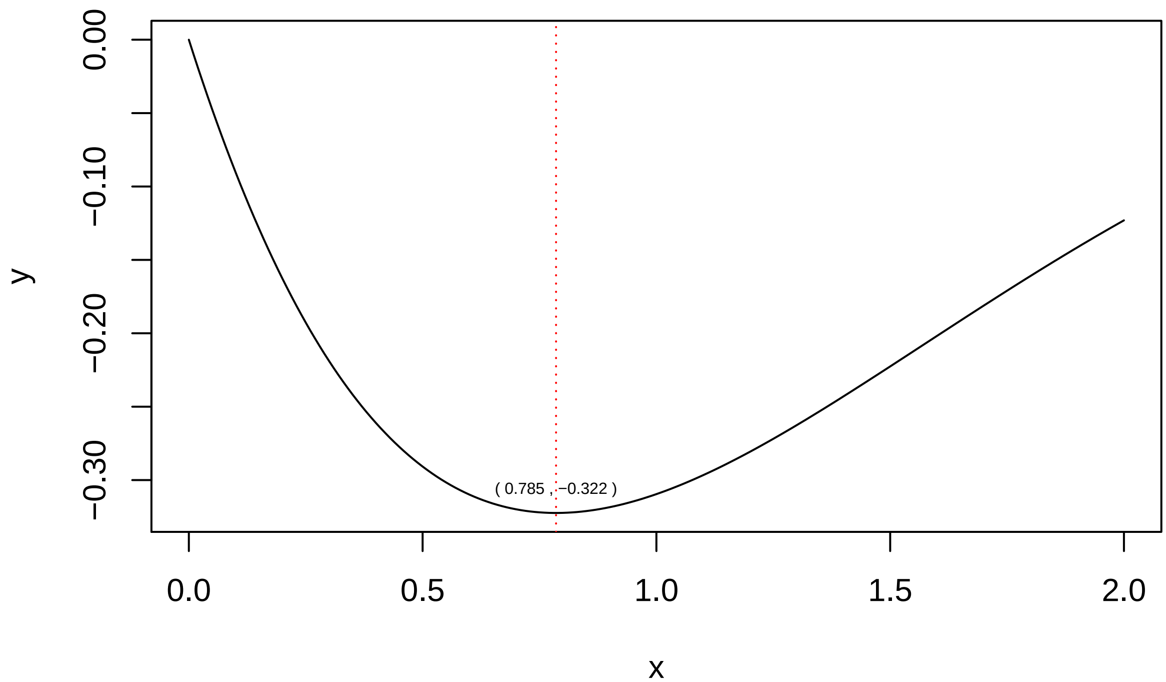
```r
      }
      else
      {
         lower.bound = x1
         x1 = x2
         f1 = f2
         x2 = upper.bound - golden.ratio^2*(upper.bound - lower.bound)
         f2 = f(x2)
      }
      res = rbind(res, c(iteration, f1, f2, lower.bound, upper.bound, x1, x2))
   }
   cat('', '\n')
   cat('Final Lower Bound =', round(lower.bound,3), '\n')
   cat('Final Upper Bound =', round(upper.bound,3), '\n')
   estimated.minimizer = round((lower.bound + upper.bound)/2,3)
   cat('Estimated Minimizer =', round(estimated.minimizer,3), '\n')
   estimated.minimum = f(round(estimated.minimizer,3))
   cat('Estimated Minimum =', round(estimated.minimum,3), '\n')
   colnames(res) = c("Iteration","f1","f2","New Lower Bound","New Upper Bound","New Lower Test Point","
   return(res)
}

golden.section.search(df, 0, 1.5, 1e-5)
```

```
##
## Final Lower Bound = 0.785
## Final Upper Bound = 0.785
## Estimated Minimizer = 0.785
## Estimated Minimum = -0.322
```

```
##       Iteration         f1         f2 New Lower Bound New Upper Bound
##  [1,]         1 -0.3165172 -0.2896674       0.5729490       1.5000000
##  [2,]         2 -0.3223838 -0.3165172       0.5729490       1.1458980
##  [3,]         3 -0.3203750 -0.3223838       0.5729490       0.9270510
##  [4,]         4 -0.3223838 -0.3213516       0.7082039       0.9270510
##  [5,]         5 -0.3221832 -0.3223838       0.7082039       0.8434588
##  [6,]         6 -0.3223838 -0.3221806       0.7598667       0.8434588
##  [7,]         7 -0.3223861 -0.3223838       0.7598667       0.8115295
##  [8,]         8 -0.3223391 -0.3223861       0.7598667       0.7917961
##  [9,]         9 -0.3223861 -0.3223965       0.7720626       0.7917961
## [10,]        10 -0.3223965 -0.3223960       0.7796001       0.7917961
## [11,]        11 -0.3223942 -0.3223965       0.7796001       0.7871376
## [12,]        12 -0.3223965 -0.3223969       0.7824792       0.7871376
## [13,]        13 -0.3223969 -0.3223968       0.7842586       0.7871376
## [14,]        14 -0.3223969 -0.3223969       0.7842586       0.7860379
## [15,]        15 -0.3223969 -0.3223969       0.7849382       0.7860379
## [16,]        16 -0.3223969 -0.3223969       0.7849382       0.7856179
## [17,]        17 -0.3223969 -0.3223969       0.7851978       0.7856179
## [18,]        18 -0.3223969 -0.3223969       0.7851978       0.7854574
## [19,]        19 -0.3223969 -0.3223969       0.7852970       0.7854574
## [20,]        20 -0.3223969 -0.3223969       0.7853583       0.7854574
## [21,]        21 -0.3223969 -0.3223969       0.7853583       0.7854196
## [22,]        22 -0.3223969 -0.3223969       0.7853817       0.7854196
```

```
## [23,]        23 -0.3223969 -0.3223969       0.7853817       0.7854051
## [24,]        24 -0.3223969 -0.3223969       0.7853906       0.7854051
## [25,]        25 -0.3223969 -0.3223969       0.7853962       0.7854051
##        New Lower Test Point New Upper Test Point
##  [1,]             0.9270510            1.1458980
##  [2,]             0.7917961            0.9270510
##  [3,]             0.7082039            0.7917961
##  [4,]             0.7917961            0.8434588
##  [5,]             0.7598667            0.7917961
##  [6,]             0.7917961            0.8115295
##  [7,]             0.7796001            0.7917961
##  [8,]             0.7720626            0.7796001
##  [9,]             0.7796001            0.7842586
## [10,]             0.7842586            0.7871376
## [11,]             0.7824792            0.7842586
## [12,]             0.7842586            0.7853583
## [13,]             0.7853583            0.7860379
## [14,]             0.7849382            0.7853583
## [15,]             0.7853583            0.7856179
## [16,]             0.7851978            0.7853583
## [17,]             0.7853583            0.7854574
## [18,]             0.7852970            0.7853583
## [19,]             0.7853583            0.7853962
## [20,]             0.7853962            0.7854196
## [21,]             0.7853817            0.7853962
## [22,]             0.7853962            0.7854051
## [23,]             0.7853906            0.7853962
## [24,]             0.7853962            0.7853996
## [25,]             0.7853996            0.7854017
```

```r
x = seq(0,2,0.01)
y = -exp(-x)*sin(x)
plot(x,y, 'l')
abline(v = 0.7854006, lty = 3, col="red")
text(0.7854006,-0.3223969,paste("(",0.785,",",-0.322,")"),pos=3,cex=0.5)
```



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:

I use a simulation to test the reliability of my codes. 70 observations that following poisson distribution has been randomly generated with true $\alpha = 1$ and true $\beta = -2$. Newton Raphson algorithm starts at point (0,0) and converges at the 5th irritation with estimated $\hat{\alpha} = 0.9819444$ and $\hat{\beta} = -2.007312$

```
# Function to compute the loglikelihood, the gradient, and hessian matrix
poissonstuff <- function(dat, betavec) {
  u <- betavec[1] + betavec[2] * dat$x
  # lambda
  lambda <- exp(u)
  loglik <- sum(dat$y * u - lambda - log(factorial(dat$y)))
  grad <- c(sum(dat$y - lambda), sum(dat$x * (dat$y - lambda)))
# gradient at betavec
  Hess <- - matrix(c(sum(lambda),
                    rep(sum(dat$x * lambda),2),
                    sum(dat$x^2 * lambda)), ncol=2)
# Hessian at betavec
  return(list(loglik = loglik, grad = grad, Hess = Hess))
}
```

```
# Newton Raphson function
NewtonRaphson <- function(dat, func, start, tol=1e-10, maxiter = 20000) {
  i <- 0
  cur <- start
  stuff <- func(dat, cur)
  res <- c(0, stuff$loglik, cur)
  prevloglik <- -Inf      # To make sure it iterates

  while(i < maxiter && abs(stuff$loglik - prevloglik) > tol)
    {
    i <- i + 1
    prevloglik <- stuff$loglik
    prev <- cur
    cur <- prev - solve(stuff$Hess) %*% stuff$grad
    prevstuff<- stuff
    stuff <- func(dat, cur)        # log-lik, gradient, Hessian
    gamma <- 0.01
    # ensure that the direction of the step is an ascent direction
    while(max(eigen(stuff$Hess)$value) > 0){

      stuff$Hess <- stuff$Hess - diag(2) * gamma
      gamma <- gamma + 0.01
    }

    # step-halving
    if (stuff$loglik > prevloglik)
    {
        res <- rbind(res, c(i, stuff$loglik, cur))# Add current values to results matrix
    }
    else
    {
      lambda <- 1
```

```
      while (stuff$loglik < prevloglik) {
        lambda <- lambda / 2 # step-halving
        cur <- prev - lambda * solve(prevstuff$Hess) %*% prevstuff$grad
        stuff <- func(dat, cur)        # log-lik, gradient, Hessian
      }
        res <- rbind(res, c(i, stuff$loglik, cur))# Add current values to results matrix
    }
    }
  return(res)
}
```

```
# data generation
set.seed(123)
# generate some data
n <- 70
truebeta <- c(1, -2) # true beta
x <- rnorm(n)
lambda <- exp(truebeta[1] + truebeta[2] * x)
y <- rpois(n, lambda)

NewtonRaphson(list(x=x,y=y), poissonstuff,c(0, 0)) # start point (0,0)
```

```
##     [,1]       [,2]      [,3]      [,4]
## res    0 -1751.3284 0.0000000  0.000000
##        1  -120.3668 1.1854974 -1.835343
##        2  -116.1500 0.9971172 -1.999663
##        3  -116.1298 0.9820472 -2.007252
##        4  -116.1298 0.9819444 -2.007312
##        5  -116.1298 0.9819444 -2.007312
```

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:

With start point $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, this algorithm converges to (0.1772807, 0.1832544, 0.6394649), representing allele frequencies, P_A, P_B and P_O, at 15th irritation.

E-step

$$
\begin{aligned}
N_{A/A}^{(k)} &= E\left(N_{AA} | N_{\text{obs}}, p^{(k)}\right) = N_A \times \frac{p_A^{(k)^2}}{p_A^{(k)^2} + 2p_A^{(k)} p_O^{(k)}} \\
N_{AO}^{(k)} &= E\left(N_{A/0} | N_{\text{obs}}, p^{(k)}\right) = N_A \times \frac{2p_A^{(k)} p_O^{(k)}}{p_A^{(k)} + 2p_A^{(k)} p_O^{(k)}} \\
N_{B/B}^{(k)} &= E\left(N_{BB} | N_{\text{obs}}, p^{(k)}\right) = N_B \times \frac{p_B^{(k)^2}}{p_B^{(k)^2} + 2p_B^{(k)} p_O^{(k)}} \\
N_{BlO}^{(k)} &= E\left(N_{BIO} | N_{\text{obs}}, p^{(k)}\right) = N_B \times \frac{2p_B^{(k)} p_O^{(k)}}{p_B^{(k)^2} + 2p_B^{(k)} p_O^{(k)}}
\end{aligned}
$$

$$
\begin{aligned}
E\left(N_{A/B} | N_{\text{obs}}, p^{(k)}\right) &= N_{A/B} \\
E\left(N_{O/O} | N_{\text{obs}}, p^{(k)}\right) &= N_{O/O}
\end{aligned}
$$

M-step

$$
\begin{aligned}
p_A^{(k+1)} &= \frac{2N_{A/A}^{(k)} + N_{AO}^{(k)} + N_{AB}^{(k)}}{2n} \\
p_B^{(k+1)} &= \frac{2N_{B/B}^{(k)} + N_{B/O}^{(k)} + N_{A/B}^{(k)}}{2n} \\
p_O^{(k+1)} &= \frac{2N_{O/O}^{(k)} + N_{A/O}^{(k)} + N_{B/O}^{(k)}}{2n}
\end{aligned}
$$

```
# N=(Na,Nb,Nab,No)
# p=(pa,pb,po)
emstep <- function(N,p) {
  #E-step
  Naa <- N[1] * p[1]^2 / (p[1]^2 + 2 * p[1] * p[3])
  Nao <- N[1] * 2 * p[1] * p[3] / (p[1]^2 + 2 * p[1] * p[3])
  Nbb <- N[2] * p[2]^2 / (p[2]^2 + 2 * p[2] * p[3])
  Nbo <- N[2] * 2 * p[2] * p[3] / (p[2]^2 + 2 * p[2] * p[3])
  #M-step
  n <- sum(N)
  p[1] = (2 * Naa + Nao + N[3]) / (2 * n)
  p[2] = (2 * Nbb + Nbo + N[3]) / (2 * n)
  p[3] = (Nao + Nbo + 2 * N[4]) / (2 * n)
  p
  return(p)
}

EMmix <- function(N, p, nreps = 15) {
i <- 0
res <- c(0, p)
while(i < nreps) {
i <- i + 1
p <- emstep(N,p)
res <- rbind(res, c(i,p))
}
return(rbind(res))
}

#Data
N <- c(26,27,7,42)
```

```r
#Starting value
p <- c(1,1,1) / 3


EMmix(N,p)
```

```
##     [,1]      [,2]      [,3]      [,4]
## res    0 0.3333333 0.3333333 0.3333333
##        1 0.2042484 0.2107843 0.5849673
##        2 0.1807081 0.1868720 0.6324199
##        3 0.1776974 0.1837038 0.6385988
##        4 0.1773313 0.1833096 0.6393591
##        5 0.1772869 0.1832611 0.6394520
##        6 0.1772815 0.1832552 0.6394633
##        7 0.1772808 0.1832545 0.6394647
##        8 0.1772807 0.1832544 0.6394649
##        9 0.1772807 0.1832544 0.6394649
##       10 0.1772807 0.1832544 0.6394649
##       11 0.1772807 0.1832544 0.6394649
##       12 0.1772807 0.1832544 0.6394649
##       13 0.1772807 0.1832544 0.6394649
##       14 0.1772807 0.1832544 0.6394649
##       15 0.1772807 0.1832544 0.6394649
```