Homework 2 on Newton's methods

Alyssa Vanderbeek (amv2187)

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

Problem 1

Design an optimization algorithm to find the minimum of the continuously differentiable function $f(x) = -e^{-1}\sin(x)$ on the closed interval $[0, 1.5]$. Write out your algorithm and implement it into **R**.

```
# Golden section search

f = function(x){
  y = -exp(-x)*sin(x)
  return(y)
}

a = 0
b = 1.5
w = 0.618

#optimize(f, lower = 0, upper = 1.5, maximum = FALSE) # truth
tol = 1e-10 # .Machine$double.eps^0.25 # tolerance with which to accept estimate of minimum

# beginning values of intervals
x1 = a
x2 = (a + b)*w

while (x2 - x1 > tol) { # while the size of the interval is greater than our tolerance level
  f1 = f(x1)
  f2 = f(x2)

  # evaluate values for interval and reassign new interval values
  if (f1 > f2) { # if f1 > f2, move to the right of x2
    x1 = x2
    x2 = x1 + (1 - w)*(b - x1)
  } else {
    x1 = x1
    x2 = x2 - w*(x2 - x1)
  }
}

min(f1, f2) # estimate of minimum on given interval
```

```
## [1] -0.3165212
```

The minimum is -0.3165212.

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 $\lambda > 0$.

```
poisson_obj <- function(dat, betavec){
  u <- betavec[1] + betavec[2] * dat$x
  mu <- exp(u)
  n <- length(mu)
  loglik <- sum(dat$y*u - mu - log(factorial(dat$y)))

  #scalar
  grad <- c(sum(dat$y - mu),
            sum(dat$x*(dat$y - mu)))

  #vector of 2
  Hess <- -(rbind(rep(1, n), dat$x) %*%
          diag(mu) %*%
          cbind(rep(1, n), dat$x))

  return(list(loglik = loglik, grad = grad, Hess = Hess))
}
```

```
# Step-halving
NR_half <- function(dat, stuff.func, start, tol = 1e-10, maxiter = 200) {
  i <- 0
  subit <- 1
  halves <- 0.5^(seq(1, 30, 1))
  cur <- start
  stuff <- stuff.func(dat, cur)
  res <- c(0, stuff$loglik, cur)
  prevloglik <- -Inf

  while (i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
```

```r
    i <- i + 1
    prevloglik <- stuff$loglik
    prev <- cur
    d <- -solve(stuff$Hess) %*% stuff$grad
    cur <- prev + d

    #No halving step -- lambda = 1
    if ( stuff.func(dat, cur)$loglik > stuff.func(dat, prev)$loglik ) {
      stuff <- stuff.func(dat, cur) # log-lik, gradient, Hessian
      res <- rbind(res, c(i, stuff$loglik, cur))
      # Add current values to results matrix
    }

    #Halving step -- lambda = 0.5, 0.25, ...
    else {
      half_cur <- prev + (halves[subit])*d
      while (stuff.func(dat, half_cur)$loglik <= stuff.func(dat, prev)$loglik) {
        subit <- subit + 1
        half_cur <- prev + (halves[subit])*d
      }
      cur <- half_cur
      stuff <- stuff.func(dat, cur) # log-lik, gradient, Hessian
      res <- rbind(res, c(i, stuff$loglik, cur))

    }
  }

  return(res)
}
```

```r
NR_ascent <- function(dat, stuff.func, start, tol=1e-10, maxiter = 200) {
  i <- 0
  subit <- 1
  halves <- 0.5^(seq(1, 30, 1))
  cur <- start
  stuff <- stuff.func(dat, cur)
  res <- c(0, stuff$loglik, cur)
  prevloglik <- -Inf

  while (i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
    i <- i + 1
    prevloglik <- stuff$loglik
    prev <- cur
    d <- -solve(stuff$Hess + diag(rep(max(stuff$Hess),2))) %*% stuff$grad # replaces Hessian with simil
    cur <- prev + d

    # For lambda = 1
    if (stuff.func(dat, cur)$loglik > stuff.func(dat, prev)$loglik ) {
      stuff <- stuff.func(dat, cur) # log-lik, gradient, Hessian
      res <- rbind(res, c(i, stuff$loglik, cur))

    } else {# For halving steps

      half_cur <- prev + (halves[subit])*d
```

```r
      while (stuff.func(dat, half_cur)$loglik <= stuff.func(dat, prev)$loglik) {
        subit <- subit + 1
        half_cur <- prev + (halves[subit])*d
      }
      cur <- half_cur
      stuff <- stuff.func(dat, cur) # log-lik, gradient, Hessian
      res <- rbind(res, c(i, stuff$loglik, cur))
    }
  }
  return(res)
}
```

```r
set.seed(2)
dat <- rpois(300, 0.8)
dat <- as.data.frame(table(dat))
names(dat) <- c("x", "y")
dat$x <- as.numeric(dat$x)
print(dat)
```

```
##   x   y
## 1 1 146
## 2 2  86
## 3 3  45
## 4 4  23
```

```r
summary(glm(y ~ x, data = dat, family = poisson()))
```

```
##
## Call:
## glm(formula = y ~ x, family = poisson(), data = dat)
##
## Deviance Residuals:
##        1         2         3         4
## -0.24187   0.47314   0.02976  -0.32034
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.60422    0.12263   45.70   <2e-16 ***
## x           -0.60067    0.05956  -10.09   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 117.70036  on 3  degrees of freedom
## Residual deviance:   0.38587  on 2  degrees of freedom
## AIC: 28.132
##
## Number of Fisher Scoring iterations: 3
```

```r
NR_half(list(x = dat$x, y = dat$y), poisson_obj, start = c(-10, 10))
```

```
##     [,1]          [,2]        [,3]         [,4]
## res    0 -1.068696e+13 -10.0000000 10.000000000
##        1 -3.931513e+12 -10.9999946  9.999998650
```

```
##        2 -1.446323e+12 -11.9999799  9.999994981
##        3 -5.320724e+11 -12.9999400  9.999985007
##        4 -1.957385e+11 -13.9998316  9.999957895
##        5 -7.200817e+10 -14.9995368  9.999884199
##        6 -2.649033e+10 -15.9987356  9.999683888
##        7 -9.745247e+09 -16.9965580  9.999139494
##        8 -3.585077e+09 -17.9906420  9.997660484
##        9 -1.318877e+09 -18.9745844  9.993646061
##       10 -4.851890e+08 -19.9311102  9.982777452
##       11 -1.784925e+08 -20.8142128  9.953552915
##       12 -6.566527e+07 -21.5056060  9.876400712
##       13 -2.415856e+07 -21.7290225  9.682253496
##       14 -8.889083e+06 -20.9906185  9.247648814
##       15 -3.271610e+06 -18.9311064  8.482760815
##       16 -1.204842e+06 -16.0620540  7.515470578
##       17 -4.442965e+05 -13.0687687  6.517075597
##       18 -1.642848e+05 -10.0813026  5.520009456
##       19 -6.105437e+04  -7.1152270  4.527952624
##       20 -2.286571e+04  -4.2063165  3.549296428
##       21 -8.623759e+03  -1.4452551  2.605385267
##       22 -3.233701e+03   0.9705974  1.743028772
##       23 -1.166412e+03   2.7811341  1.024995202
##       24 -3.821905e+02   3.9731426  0.457373374
##       25 -1.046486e+02   4.7669489 -0.002356481
##       26 -2.549247e+01   5.2809266 -0.351663394
##       27 -1.264421e+01   5.5378212 -0.546818204
##       28 -1.206743e+01   5.6007345 -0.597762322
##       29 -1.206576e+01   5.6042142 -0.600658538
##       30 -1.206576e+01   5.6042248 -0.600667451
##       31 -1.206576e+01   5.6042248 -0.600667451
```

```r
NR_ascent(list(x = dat$x, y = dat$y), poisson_obj, start = c(-10, 10))
```

```
##     [,1]          [,2]         [,3]        [,4]
## res    0 -1.068696e+13 -10.00000000 10.00000000
##        1 -4.156130e+12 -10.05555892  9.77777694
##        2 -1.616310e+12 -10.11111867  9.55555366
##        3 -6.285802e+11 -10.16667948  9.33333013
##        4 -2.444542e+11 -10.22224158  9.11110627
##        5 -9.506822e+10 -10.27780532  8.88888200
##        6 -3.697212e+10 -10.33337109  8.66665723
##        7 -1.437854e+10 -10.38893941  8.44443182
##        8 -5.591873e+09 -10.44451089  8.22220561
##        9 -2.174714e+09 -10.50008632  7.99997841
##       10 -8.457652e+08 -10.55566667  7.77774998
##       11 -3.289284e+08 -10.61125309  7.55551999
##       12 -1.279258e+08 -10.66684695  7.33328807
##       13 -4.975341e+07 -10.72244970  7.11105370
##       14 -1.935092e+07 -10.77806254  6.88881625
##       15 -7.526783e+06 -10.83368536  6.66657486
##       16 -2.928118e+06 -10.88931388  6.44432828
##       17 -1.139608e+06 -10.94493214  6.22207457
##       18 -4.440544e+05 -11.00049291  5.99981029
##       19 -1.735953e+05 -11.05586696  5.77752852
##       20 -6.847685e+04 -11.11071268  5.55521389
```

```
##        21 -2.766971e+04 -11.16414132  5.33283002
##        22 -1.187769e+04 -11.21386315  5.11028776
##        23 -5.814512e+03 -11.25403684  4.88736574
##        24 -3.530784e+03 -11.27002591  4.66351805
##        25 -2.704405e+03 -11.22667990  4.43744862
##        26 -2.415739e+03 -11.04819436  4.20639303
##        27 -2.282667e+03 -10.61149216  3.96593389
##        28 -2.148152e+03  -9.83845857  3.71343835
##        29 -1.986805e+03  -8.85096267  3.45360206
##        30 -1.819028e+03  -7.81830836  3.19272577
##        31 -1.652022e+03  -6.78753619  2.93268049
##        32 -1.486837e+03  -5.76449379  2.67392993
##        33 -1.324112e+03  -4.75201137  2.41692842
##        34 -1.164654e+03  -3.75360389  2.16229210
##        35 -1.009542e+03  -2.77399364  1.91087555
##        36 -8.602069e+02  -1.81951687  1.66386538
##        37 -7.185025e+02  -0.89853872  1.42289170
##        38 -5.867190e+02  -0.02170626  1.19012815
##        39 -4.674462e+02   0.79833091  0.96830756
##        40 -3.631840e+02   1.54822724  0.76053910
##        41 -2.757113e+02   2.21646975  0.56984591
##        42 -2.054664e+02   2.79637015  0.39852005
##        43 -1.513426e+02   3.28787135  0.24760667
##        44 -1.110603e+02   3.69703291  0.11680347
##        45 -8.184659e+01   4.03375296  0.00476159
##        46 -6.102806e+01   4.30923347 -0.09045739
##        47 -4.635095e+01   4.53420700 -0.17098740
##        48 -3.606433e+01   4.71807489 -0.23890495
##        49 -2.887398e+01   4.86867666 -0.29610130
##        50 -2.385111e+01   4.99238150 -0.34423623
##        51 -2.034041e+01   5.09430070 -0.38473615
##        52 -1.788372e+01   5.17851849 -0.41881273
##        53 -1.616202e+01   5.24829958 -0.44748875
##        54 -1.495350e+01   5.30626107 -0.47162480
##        55 -1.410390e+01   5.35450972 -0.49194388
##        56 -1.350574e+01   5.39474936 -0.50905311
##        57 -1.308406e+01   5.42836453 -0.52346229
##        58 -1.278643e+01   5.45648537 -0.53559957
##        59 -1.257614e+01   5.48003833 -0.54582471
##        60 -1.242743e+01   5.49978569 -0.55444007
##        61 -1.232217e+01   5.51635686 -0.56169992
##        62 -1.224763e+01   5.53027302 -0.56781811
##        63 -1.219481e+01   5.54196693 -0.57297462
##        64 -1.215735e+01   5.55179871 -0.57732091
##        65 -1.213079e+01   5.56006861 -0.58098451
##        66 -1.211194e+01   5.56702742 -0.58407281
##        67 -1.209856e+01   5.57288488 -0.58667626
##        68 -1.208906e+01   5.57781665 -0.58887107
##        69 -1.208231e+01   5.58196998 -0.59072144
##        70 -1.207752e+01   5.58546841 -0.59228146
##        71 -1.207412e+01   5.58841571 -0.59359671
##        72 -1.207170e+01   5.59089903 -0.59470563
##        73 -1.206998e+01   5.59299166 -0.59564060
##        74 -1.206876e+01   5.59475524 -0.59642892
```

```
##       75 -1.206789e+01   5.59624164 -0.59709359
##       76 -1.206727e+01   5.59749451 -0.59765402
##       77 -1.206684e+01   5.59855060 -0.59812655
##       78 -1.206652e+01   5.59944086 -0.59852498
##       79 -1.206630e+01   5.60019137 -0.59886093
##       80 -1.206614e+01   5.60082408 -0.59914420
##       81 -1.206603e+01   5.60135750 -0.59938305
##       82 -1.206595e+01   5.60180722 -0.59958444
##       83 -1.206590e+01   5.60218639 -0.59975426
##       84 -1.206586e+01   5.60250607 -0.59989744
##       85 -1.206583e+01   5.60277561 -0.60001818
##       86 -1.206581e+01   5.60300287 -0.60011998
##       87 -1.206579e+01   5.60319449 -0.60020582
##       88 -1.206578e+01   5.60335606 -0.60027820
##       89 -1.206578e+01   5.60349228 -0.60033924
##       90 -1.206577e+01   5.60360715 -0.60039070
##       91 -1.206577e+01   5.60370400 -0.60043409
##       92 -1.206576e+01   5.60378566 -0.60047068
##       93 -1.206576e+01   5.60385452 -0.60050153
##       94 -1.206576e+01   5.60391258 -0.60052755
##       95 -1.206576e+01   5.60396153 -0.60054948
##       96 -1.206576e+01   5.60400281 -0.60056798
##       97 -1.206576e+01   5.60403762 -0.60058358
##       98 -1.206576e+01   5.60406697 -0.60059673
##       99 -1.206576e+01   5.60409172 -0.60060782
##      100 -1.206576e+01   5.60411258 -0.60061717
##      101 -1.206576e+01   5.60413018 -0.60062505
##      102 -1.206576e+01   5.60414502 -0.60063170
##      103 -1.206576e+01   5.60415753 -0.60063730
##      104 -1.206576e+01   5.60416807 -0.60064203
##      105 -1.206576e+01   5.60417697 -0.60064602
##      106 -1.206576e+01   5.60418447 -0.60064938
##      107 -1.206576e+01   5.60419079 -0.60065221
##      108 -1.206576e+01   5.60419612 -0.60065460
##      109 -1.206576e+01   5.60420062 -0.60065662
##      110 -1.206576e+01   5.60420441 -0.60065831
##      111 -1.206576e+01   5.60420761 -0.60065975
##      112 -1.206576e+01   5.60421030 -0.60066095
##      113 -1.206576e+01   5.60421258 -0.60066197
##      114 -1.206576e+01   5.60421449 -0.60066283
##      115 -1.206576e+01   5.60421611 -0.60066356
##      116 -1.206576e+01   5.60421747 -0.60066417
##      117 -1.206576e+01   5.60421862 -0.60066468
##      118 -1.206576e+01   5.60421959 -0.60066512
##      119 -1.206576e+01   5.60422041 -0.60066548
##      120 -1.206576e+01   5.60422110 -0.60066579
##      121 -1.206576e+01   5.60422168 -0.60066605
##      122 -1.206576e+01   5.60422217 -0.60066627
```

The Poisson model is estimated to be $log(\lambda_i) = 5.604 - 0.6x_i$.

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..

```r
# E-step evaluating conditional means E(Z_i | X_i , pars)
# X = c(Na, Nb, Nab, No)
# pars = c(pa, pb, po)
delta <- function(X, pars){

  n_aa = X[1] * (pars[["pa"]]^2 / (pars[["pa"]]^2 + 2*pars[["pa"]]*pars[["po"]]))
  n_ao = X[1] * ((2*pars[["pa"]]*pars[["po"]]) / (pars[["pa"]]^2 + 2*pars[["pa"]]*pars[["po"]]))
  n_bb = X[2] * (pars[["pb"]]^2 / (pars[["pb"]]^2 + 2*pars[["pb"]]*pars[["po"]]))
  n_bo = X[2] * ((2*pars[["pb"]]*pars[["po"]]) / (pars[["pb"]]^2 + 2*pars[["pb"]]*pars[["po"]]))
  n_ab = X[3]
  n_oo = X[4]

  return(unname(c(n_aa, n_ao, n_bb, n_bo, n_ab, n_oo)))
}

# M-step - updating the parameters
mles <- function(Z, X) {
  n <- sum(X)

  pa = (2*Z[1] + Z[2] + Z[5])/(2*n)
  pb = (2*Z[3] + Z[4] + Z[5])/(2*n)
  po = (2*Z[6] + Z[2] + Z[4])/(2*n)

  return(list(pa = pa, pb = pb, po = po))
}


# X - the
EMmix <- function(X, start, nreps = 10) {
  i <- 0
  Z <- delta(X, start)
  newpars <- start
  res <- c(0, t(as.matrix(newpars)))
  while (i < nreps) {
  # This should actually check for convergence
    i <- i + 1
    newpars <- mles(Z, X)
    Z <- delta(X, newpars)
    res <- rbind(res, c(i, t(as.matrix(newpars))))
  }
  return(res)
}
```

```r
start = c(pa = 0.3, pb = 0.1, po = 0.6)
X = c(na = 26, nb = 27, nab = 7, no = 42)
```

```r
EMmix(X = X, start = start)
```

```
##     [,1] [,2]      [,3]      [,4]
## res 0    0.3       0.1       0.6
##     1    0.1872549 0.1768477 0.6358974
##     2    0.1781218 0.1828241 0.6390541
##     3    0.1773541 0.1832296 0.6394163
##     4    0.1772874 0.1832535 0.6394591
##     5    0.1772814 0.1832544 0.6394642
##     6    0.1772808 0.1832544 0.6394648
##     7    0.1772807 0.1832544 0.6394649
##     8    0.1772807 0.1832544 0.6394649
##     9    0.1772807 0.1832544 0.6394649
##     10   0.1772807 0.1832544 0.6394649
```

EM algorithm estimates $(p_a, p_b, p_o) = (0.17, 0.18, 0.64)$.