Bios 301: Assignment 2

Utsav Kumar

Question 1

20 points

A problem with the Newton-Raphson algorithm is that it needs the derivative f'. If the derivative is hard to compute or does not exist, then we can use the *secant method*, which only requires that the function f is continuous.

Like the Newton-Raphson method, the **secant method** is based on a linear approximation to the function f. Suppose that f has a root at a. For this method we assume that we have *two* current guesses, x_0 and x_1, for the value of a. We will think of x_0 as an older guess and we want to replace the pair x_0, x_1 by the pair x_1, x_2, where x_2 is a new guess.

To find a good new guess x2 we first draw the straight line from $(x_0, f(x_0))$ to $(x_1, f(x_1))$, which is called a secant of the curve $y = f(x)$. Like the tangent, the secant is a linear approximation of the behavior of $y = f(x)$, in the region of the points x_0 and x_1. As the new guess we will use the x-coordinate x_2 of the point at which the secant crosses the x-axis.

The general form of the recurrence equation for the secant method is:

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

Notice that we no longer need to know f' but in return we have to provide *two* initial points, x_0 and x_1.

Write a function that implements the secant algorithm. Validate your program by finding the root of the function $f(x) = \cos(x) - x$. Compare its performance with the Newton-Raphson method – which is faster, and by how much? For this example $f'(x) = -\sin(x) - 1$.

```
# Given function and its derivative
f <- function(x)
{
  cos(x)-x
}

fder <- function(x)
{
  -sin(x)-1
}

# Secant method function

secant <- function (f, x0, x1, eps)
{
  i <- 0
  while (abs(f(x1)) > eps)
  {
    x2 = x1 - f(x1) * (x1-x0)/(f(x1)-f(x0))
    x0 = x1
    x1 = x2
    i = i + 1
  }
  return (x1)
}

# secant function call

ptm <- proc.time()
secant_root <- secant (f, 0, 1, 1e-20)
proc.time() - ptm
```

```
##    user  system elapsed
##   0.003   0.000   0.004
```

```
secant_root
```

```
## [1] 0.7391
```

```
# Newton Raphson function

newton <- function (f, fder, x0, eps)
{
  i=0
  while (abs(f(x0)) > eps)
  {
    x1 = x0 - f(x0)/fder(x0)
    x0 = x1
    i = i + 1
  }
  return (x0)
}

# newton raphson function call

ptm <- proc.time()
newton_root <- newton (f, fder, 0, 1e-20)
proc.time() - ptm
```

```
##    user  system elapsed
##   0.002   0.000   0.002
```

```
newton_root
```

```
## [1] 0.7391
```

In general secant method has slower convergence as compared to newton-raphson but in this case the problem is not as complex to actually judge speed and both methods takes negligible time of 0.02s to solve.

Question 2

18 points

Import the HAART dataset (`haart.csv`) from the GitHub repository into R, and perform the following manipulations:

1. Convert date columns into a usable (for analysis) format.

```r
# haart.csv file present in the working directory
haart <- read.csv("haart.csv")
haart$init.date <- as.Date(haart$init.date, format="%m/%d/%y")
haart$last.visit <- as.Date(haart$last.visit, format="%m/%d/%y")
haart$date.death <- as.Date(haart$date.death, format="%m/%d/%y")
```

2. Create an indicator variable (one which takes the values 0 or 1 only) to represent death within 1 year of the initial visit.

```r
# need to check for the indicator only for those who are dead
# i.e. equating indicator column to death column
haart$indicator <- haart$death

# finding number of rows
i_max <- length(haart$death)
for (i in 1:i_max)
{
  # only if default indicator value is 1 as it is by default
  # equated to the death column
  if (haart$indicator[i] == 1)
  {
    # for those indicator value where death time is within 1 year of initial visit
    # keeping that one and the ones more than one year are changed to 0
    if (difftime(haart$date.death[i], haart$init.date[i], units="days") < 365)
    {
      haart$indicator[i] = 1
    }
    else
    {
      haart$indicator[i] = 0
    }
  }
}
```

3. Use the `init.date`, `last visit` and `death.date` to calculate a followup time, which is the difference between the first and either the last visit or a death event (whichever comes first). If these times are longer than 1 year, censor them.

```r
haart$follow.up <- NA
time1 <- c()
time2 <- c()
i_max <- length(haart$death)
for (i in 1:i_max)
{
  # column matrix to store time difference between last & initial visit for each row
  time1[i] <- difftime (haart$last.visit[i], haart$init.date[i], units="days")
  # checking for column not having NA values
  if (!is.na(time1[i]))
  {
    # changing the cell value to NA where time difference was more than an year
    if (time1[i] > 365)
    {
      time1[i] = NA
    }
  }
  # col matrix to store time diff between date of death & initial visit for each row
  time2[i] <- difftime (haart$date.death[i], haart$init.date[i], units="days")
  # checking for column not having NA values
  if (!is.na(time2[i]))
  {
    # changing the cell value to NA where time difference was more than an year
    if (time2[i] > 365)
    {
      time2[i] = NA
    }
  }
}
  # creating a new column matrix storing the difference between time1 and time2
  # so only those row will have values for which time1 and time2 value exist
  # i.e. it is not NA and it means that both are less than year and we need to
  # find the one which is earliest or smaller difference
  time12 <- (time1 - time2)

for (i in 1:1000)
{
  # if row entry is a number and not NA
  if (!is.na(time1[i]))
  # if time difference is less than an year
  {if (time1[i] < 365){haart$follow.up[i] = time1[i]}}
  # if row entry is a number and not NA
  if (!is.na(time2[i]))
  # if time difference is less than an year
  {if (time2[i] < 365){haart$follow.up[i] = time2[i]}}
  # for those rows which has values for both time1 and time2 and is less than
  # an year either time1 and time2 whichever has lower value replaces the cell
  # with correct entry i.e. only for common cell it overwrites the value
  # written by the previous two if statements
  if (!is.na(time12[i]))
  {
    if (time12[i] < 0)
    {haart$follow.up[i] = time1[i]}
    else
    {haart$follow.up[i] = time2[i]}
  }
}
```

4. Create another indicator variable representing loss to followup; that is, if their status 1 year after the first visit was unknown.

```
# assuming that loss of follow up to all entries
haart$loss.follow.up <- 1
i_max <- length(haart$death)
for (i in 1:i_max)
{
  # for those entry for which follow up time is less than an year which
  # is already stored in follow.up column in the previous case the loss
  # of follow up is 0 rest it means their status was unknown after an year
  if (!is.na(haart$follow.up[i]))
  {haart$loss.follow.up[i] = 0}
}
```

5. Recall our work in class, which separated the `init.reg` field into a set of indicator variables, one for each unique drug. Create these fields and append them to the database as new columns.

```
library(stringr)
# Splitting the first three rows into separate columns
haart.split <- (str_split_fixed(haart$init.reg, ",", 3))
# Splitting the last column some of which might have to be separated
haart.split2 <- str_split_fixed(haart.split[,3], ",", 2)
# combining to haart
haart$init.reg1 <- haart.split[,1]
haart$init.reg2 <- haart.split[,2]
haart$init.reg3 <- haart.split2[,1]
haart$init.reg4 <- haart.split2[,2]
```

6. The dataset `haart2.csv` contains a few additional observations for the same study. Import these and append them to your master dataset (if you were smart about how you coded the previous steps, cleaning the additional observations should be easy!).

```
# reading haart2 file
haart2 <- read.csv("haart2.csv")
cols <- names(haart)
# adding extra olumns to hart2 with default as NA
haart2[,setdiff(cols, names(haart2))] <- NA
# combining the additional entries to haart
haart <- rbind(haart,haart2)
```

Question 3

12 points

Obtain a copy of the football-values lecture. Save the five CSV files in your working directory.

Modify the code to create a function. This function will create dollar values given information (as arguments) about a league setup. It will return a data.frame and write this data.frame to a CSV file. The final data.frame should contain the columns 'PlayerName', 'pos', 'points', 'value' and be orderd by value descendingly. Do not round dollar values.

Note that the returned data.frame should have `sum(posReq)*nTeams` rows.

Define the function as such (6 points):

```
ffvalues <- function(path, file, nTeams, cap, posReq=c(qb, rb, wr, te, k),
                     points=c(fg, xpt, pass_yds, pass_tds, pass_ints,
                              rush_yds, rush_tds, fumbles, rec_yds, rec_tds))
{
# finding current path
current_path = getwd()
# setting path where data files are present
setwd(file.path(path))
```

```r
# Reading data files
  k <- read.table('proj_k14.csv', header=TRUE, stringsAsFactors=FALSE)
  qb <- read.table('proj_qb14.csv', header=TRUE, stringsAsFactors=FALSE)
  rb <- read.table('proj_rb14.csv', header=TRUE, stringsAsFactors=FALSE)
  te <- read.table('proj_te14.csv', header=TRUE, stringsAsFactors=FALSE)
  wr <- read.table('proj_wr14.csv', header=TRUE, stringsAsFactors=FALSE)

# shifting back to working directory
setwd(current_path)
# generate unique list of column names
  cols <- unique(c(names(k), names(qb), names(rb), names(te), names(wr)))

# create a new column in each data.frame with playing position type
  k[,'pos'] <- 'k'
  qb[,'pos'] <- 'qb'
  rb[,'pos'] <- 'rb'
  te[,'pos'] <- 'te'
  wr[,'pos'] <- 'wr'

# append 'pos' to unique column list
  cols <- c(cols, 'pos')

# create common columns in each data.frame
# initialize values to zero
  k[,setdiff(cols, names(k))] <- 0
  qb[,setdiff(cols, names(qb))] <- 0
  rb[,setdiff(cols, names(rb))] <- 0
  te[,setdiff(cols, names(te))] <- 0
  wr[,setdiff(cols, names(wr))] <- 0

# combine data.frames by row, using consistent column order
  x <- rbind(k[,cols], qb[,cols], rb[,cols], te[,cols], wr[,cols])

# rename column names, by removing periods
  names(x) <- gsub('[.]', '', names(x))

# points supplied as a parameter to the function right now assumed as variable
  fg = as.numeric(points["fg"])
  xpt = as.numeric(points["xpt"])
  pass_yds = as.numeric(points["pass_yds"])
  pass_tds = as.numeric(points["pass_tds"])
  pass_ints = as.numeric(points["pass_ints"])
  rush_yds = as.numeric(points["rush_yds"])
  rush_tds = as.numeric(points["rush_tds"])
  fumbles = as.numeric(points["fumbles"])
  rec_yds = as.numeric(points["rec_yds"])
  rec_tds = as.numeric(points["rec_tds"])

# convert NFL stat to fantasy points
  x[,'p_fg'] <- x[,'fg']*fg
  x[,'p_xpt'] <- x[,'xpt']*xpt
  x[,'p_pass_yds'] <- x[,'pass_yds']*pass_yds
  x[,'p_pass_tds'] <- x[,'pass_tds']*pass_tds
  x[,'p_pass_ints'] <- x[,'pass_ints']*pass_ints
  x[,'p_rush_yds'] <- x[,'rush_yds']*rush_yds
  x[,'p_rush_tds'] <- x[,'rush_tds']*rush_tds
  x[,'p_fumbles'] <- x[,'fumbles']*fumbles
  x[,'p_rec_yds'] <- x[,'rec_yds']*rec_yds
  x[,'p_rec_tds'] <- x[,'rec_tds']*rec_tds

# summing the points along the row
  x[,'points'] <- rowSums(x[,grep("^p_", names(x))])
# setting up the points column in decreasing order
  x2 <- x[order(x[,'points'], decreasing=TRUE),]

# determine the row indices for each position
  k.ix <- which(x2[,'pos']=='k')
  qb.ix <- which(x2[,'pos']=='qb')
  rb.ix <- which(x2[,'pos']=='rb')
  te.ix <- which(x2[,'pos']=='te')
  wr.ix <- which(x2[,'pos']=='wr')

# no. of player in the team required given as parameter to function
  qb = as.numeric(posReq["qb"])
  rb = as.numeric(posReq["rb"])
  wr = as.numeric(posReq["wr"])
  te = as.numeric(posReq["te"])
  k = as.numeric(posReq["k"])

# calculate marginal points by subtracting "baseline" player's points
# in case number of k required in the team is 0
  if (k > 0)
  {x2[k.ix, 'marg'] <- x2[k.ix,'points'] - x2[k.ix[nTeams*k],'points']}
  else
  {x2[k.ix, 'marg'] <- -1}
  x2[qb.ix, 'marg'] <- x2[qb.ix,'points'] - x2[qb.ix[nTeams*qb],'points']
  x2[rb.ix, 'marg'] <- x2[rb.ix,'points'] - x2[rb.ix[nTeams*rb],'points']
  x2[te.ix, 'marg'] <- x2[te.ix,'points'] - x2[te.ix[nTeams*te],'points']
  x2[wr.ix, 'marg'] <- x2[wr.ix,'points'] - x2[wr.ix[nTeams*wr],'points']

# create a new data.frame subset by non-negative marginal points
  x3 <- x2[x2[,'marg'] >= 0,]

# re-order by marginal points
  x3 <- x3[order(x3[,'marg'], decreasing=TRUE),]

# reset the row names
  rownames(x3) <- NULL

# calculation for player value
  x3[,'value'] <- x3[,'marg']*(nTeams*cap-nrow(x))/sum(x3[,'marg']) + 1

# create a data.frame with required columns needed in data.frame
  player.each.team <- qb + rb + wr + te + k
  x4 <- x3[1:(nTeams*player.each.team), c('PlayerName','pos','points','value')]
```

```
# writing to csv file as an output
  write.csv(file, x=x4)
}
```

1. Call `x1 <- ffvalues(’.’)`

    ```
    # calling function with req. path, correct input values and output file name
    ffvalues("~/Documents/BIOS301/football-values/", file='outfile.csv',
             nTeams=12, cap=200, posReq=c(qb=1, rb=2, wr=3, te=1, k=1),
             points=c(fg=4, xpt=1, pass_yds=1/25, pass_tds=4, pass_ints=-2,
              rush_yds=1/10, rush_tds=6, fumbles=-2, rec_yds=1/20, rec_tds=6))
    ```

    1. How many players are worth more than $20? (1 point)

    ```
    fvalues <- data.frame(read.csv("outfile.csv"))
    players.above20 <- sum((fvalues[,'value']) > 20)
    players.above20
    ```

    ```
    ## [1] 37
    ```

    2. Who is 15th most valuable running back (rb)? (1 point)

    ```
    rb_indices <- which(fvalues[,'pos']=='rb')
    fvalues[rb_indices[15], 'PlayerName']
    ```

    ```
    ## [1] Toby Gerhart
    ## 96 Levels: A.J. Green Aaron Rodgers Adam Vinatieri ... Zach Ertz
    ```

2. Call `x2 <- ffvalues(getwd(), ’16team.csv’, nTeams=16, cap=150)`

    ```
    ffvalues("~/Documents/BIOS301/football-values/", file='16team.csv',
             nTeams=16, cap=150, posReq=c(qb=1, rb=2, wr=3, te=1, k=1),
             points=c(fg=4, xpt=1, pass_yds=1/25, pass_tds=4, pass_ints=-2,
              rush_yds=1/10, rush_tds=6, fumbles=-2, rec_yds=1/20, rec_tds=6))
    ```

    1. How many players are worth more than $20? (1 point)

    ```
    fvalues1 <- data.frame(read.csv("16team.csv"))
    players.above20 <- sum((fvalues1[,'value']) > 20)
    players.above20
    ```

    ```
    ## [1] 32
    ```

    2. How many wide receivers (wr) are in the top 40? (1 point)

    ```
    wr_indices <- which(fvalues1[,'pos']=='wr')
    wr.top40 <- sum(wr_indices < 40)
    wr.top40
    ```

    ```
    ## [1] 12
    ```

3. Call:

```r
ffvalues("~/Documents/BIOS301/football-values/", 'qbheavy.csv',
         nTeams = 12, cap = 200, posReq=c(qb=2, rb=2, wr=3, te=1, k=0),
         points=c(fg=0, xpt=0, pass_yds=1/25, pass_tds=6, pass_ints=-2,
         rush_yds=1/10, rush_tds=6, fumbles=-2, rec_yds=1/20, rec_tds=6))
```

1. How many players are worth more than $20? (1 point)

```r
fvalues2 <- data.frame(read.csv("qbheavy.csv"))
players.above20 <- sum((fvalues2[,'value']) > 20)
players.above20
```

```
## [1] 38
```

2. How many quarterbacks (qb) are in the top 30? (1 point)

```r
qb_indices <- which(fvalues2[,'pos']=='qb')
qb.top30 <- sum(qb_indices < 30)
qb.top30
```

```
## [1] 16
```

Question 4

5 bonus points

This code makes a list of all functions in the base package:

```r
objs <- mget(ls("package:base"), inherits = TRUE)
funs <- Filter(is.function, objs)
```

Using this list, write code to answer these questions.

1. Which function has the most arguments? (3 points)
2. How many functions have no arguments? (2 points)

Grade = 50 - 1 - 2 - 1 - 1- 1 = 44

Comments:

- Please see Cole's solution for the following questions:
- Question 2.5 - 1 point. The idea was to have a separate column for each medication.
- Question 2.6 -2 points. The additional rows were supposed to be merged after performing data manipulation steps from Q2.1-5.
- Question 3.1 -1 one point.
- Question 3.3 -1 one point.
- Question 3.5 -1 one point.