Bios 6301: Assignment 4

Erin Fey

Grade: 47/50

Comment: Also, please note to not set working directory to anything other than current directory. I can help show how to make your code work in this manner during office hours if you'd like.

Due Tuesday, 01 November, 1:00 PM

$5^{n=day}$ points taken off for each day late.

50 points total.

Submit a single knitr file (named homework4.rmd), along with a valid PDF output file. Inside the file, clearly indicate which parts of your responses go with which problems (you may use the original homework document as a template). Add your name as author to the file's metadata section. Raw R code/output or word processor files are not acceptable.

Failure to name file homework4.rmd or include author name may result in 5 points taken off.

Question 1

15 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 $f'(x) = -\sin(x) - 1$.

```r
# secant method
test  <- function(x) cos(x) - x   #any function
test_deriv <- function(x) -sin(x) - 1   #the derivative of the function
secant <- function(fun, x_0, x_1){
  threshold <- 1e-5
  while (abs(x_0 - x_1) > threshold){
    older = x_0
    old = x_1
    x_0 = old
    x_1 = old - fun(old)*(old - older)/(fun(old) - fun(older))
  }
  return (x_1)
}

# newton-raphson method
newton <- function(F1, F2, guess){
  threshold <- 1e-5
  x_0 = guess
  x_1 = guess - F1(guess)/F2(guess)
  while(abs(x_0 - x_1) > threshold){
    x_0 = x_1
    x_1 = x_1 - F1(x_1)/F2(x_1)
  }
  return (x_1)
}
n = 1e5
system.time(replicate(n, newton(test, test_deriv, 0)))
```

```
##    user  system elapsed
##   1.993   0.034   2.076
```

```
system.time(replicate(n, secant(test, 0, 3.14/2)))
```

```
##    user  system elapsed
##   2.912   0.039   3.042
```

```
Q1 <- secant(test, 0, 3.14/2) # make a reasonable guess of ranges
```

With both methods we get 0.739085. Using the system.time function, the NR method is faster

Question 2

18 points

The game of craps is played as follows. First, you roll two six-sided dice; let x be the sum of the dice on the first roll. If x = 7 or 11 you win, otherwise you keep rolling until either you get x again, in which case you also win, or until you get a 7 or 11, in which case you lose.

Write a program to simulate a game of craps. You can use the following snippet of code to simulate the roll of two (fair) dice:

```
  x <- sum(ceiling(6*runif(2)))
```

The instructor should be able to easily import and run your program (function), and obtain output that clearly shows how the game progressed. Set the RNG seed with set.seed(100) and show the output of three games. (lucky 13 points)

```
craps <- function(roll1=NA){
    temp  <-  sum(ceiling(6*runif(2)))
    if(is.na(roll1)){
        roll1 = temp
        if(roll1 == 7 || roll1 == 11){
            return ("You win")
        }
        else{
            return (craps(roll1))
        }
    }
    else{
        if(temp == 7 | temp == 11){
            return ("You lose")
        }
        else if(temp == roll1){
            return ("You win")
        }
        else{
            return (craps(roll1))
        }
    }
}

set.seed(100)
craps()
```

```
## [1] "You lose"
```

```
craps()
```

```
## [1] "You lose"
```

```
craps()
```

```
## [1] "You lose"
```

JC Grading - 1

Comment: Missing the output as the game progresses.

Find a seed that will win ten straight games. Consider adding an argument to your function that disables output. Show the output of the ten games. (5 points)

```
res <- rep(NA, 10)
for(i in seq(1:1000)){
    set.seed(i)
    for(j in 1:10){
        res[j] <- craps()
    }
    if('You lose' %in% res == FALSE){
        print (i)
    }
}
```

```
## [1] 880
```

880

Question 3

12 points

Obtain a copy of the football-values lecture. Save the five 2016 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='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)) {
#read in the CSV files
#setwd("~/Desktop")
  k <- read.csv('proj_k16.csv', header=TRUE, stringsAsFactors=FALSE)
  qb <- read.csv('proj_qb16.csv', header=TRUE, stringsAsFactors=FALSE)
  rb <- read.csv('proj_rb16.csv', header=TRUE, stringsAsFactors=FALSE)
  te <- read.csv('proj_te16.csv', header=TRUE, stringsAsFactors=FALSE)
  wr <- read.csv('proj_wr16.csv', header=TRUE, stringsAsFactors=FALSE)

#calculate dollar values
  cols <- unique(c(names(k), names(qb), names(rb), names(te), names(wr)))
  k[,'pos'] <- 'k'
  qb[,'pos'] <- 'qb'
  rb[,'pos'] <- 'rb'
  te[,'pos'] <- 'te'
  wr[,'pos'] <- 'wr'
  cols <- c(cols, 'pos')

  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

  x <- rbind(k[,cols], qb[,cols], rb[,cols], te[,cols], wr[,cols])
  x[,'p_fg'] <- x[,'fg']*points['fg']
  x[,'p_xpt'] <- x[,'xpt']*points['xpt']
  x[,'p_pass_yds'] <- x[,'pass_yds']*points['pass_yds']
  x[,'p_pass_tds'] <- x[,'pass_tds']*points['pass_tds']
  x[,'p_pass_ints'] <- x[,'pass_ints']*points['pass_ints']
  x[,'p_rush_yds'] <- x[,'rush_yds']*points['rush_yds']
  x[,'p_rush_tds'] <- x[,'rush_tds']*points['rush_tds']
  x[,'p_fumbles'] <- x[,'fumbles']*points['fumbles']
  x[,'p_rec_yds'] <- x[,'rec_yds']*points['rec_yds']
  x[,'p_rec_tds'] <- x[,'rec_tds']*points['rec_tds']
  x[,'points'] <- rowSums(x[,grep("^p_", names(x))])

  x2 <- x[order(x[,'points'], decreasing=TRUE),]
  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')

  x2[k.ix, 'marg'] <- x2[k.ix,'points'] - x2[k.ix[nTeams*posReq['k']],'points']
  x2[qb.ix, 'marg'] <- x2[qb.ix,'points'] - x2[qb.ix[nTeams*posReq['qb']],'points']
  x2[rb.ix, 'marg'] <- x2[rb.ix,'points'] - x2[rb.ix[nTeams*posReq['rb']],'points']
  x2[te.ix, 'marg'] <- x2[te.ix,'points'] - x2[te.ix[nTeams*posReq['te']],'points']
  x2[wr.ix, 'marg'] <- x2[wr.ix,'points'] - x2[wr.ix[nTeams*posReq['wr']],'points']
  x3 <- x2[x2[,'marg'] >= 0,]
  x3 <- x3[order(x3[,'marg'], decreasing=TRUE),]
  rownames(x3) <- NULL
  x3[,'value'] <- x3[,'marg']*(nTeams*cap-nrow(x3))/sum(x3[,'marg']) + 1

  x4 <- x3[,c('PlayerName','pos','points','value')]

# save dollar values as CSV file
write.csv(file=file,x4)
# return data.frame with dollar values
return(x4)
  }
```

1. Call x1 <- ffvalues('.')

```
x1 <- ffvalues('.')
```

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

```
sum(x1$value>20)
```

```
## [1] 46
```

46 players

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

```
x1[which(x1$pos=='rb'),][15,1]
```

```
## [1] "Carlos Hyde"
```

Carlos Hyde

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

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

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

```
sum(x2$value>20)
```

```
## [1] 49
```

49 players

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

```
top40<-x2[1:40,]
length(top40[which(top40$pos=='wr'),1])
```

```
## [1] 18
```

18

3. Call:

JC Grading -2 The calls below didn't work because of an error:

Error in `[<-.data.frame(*tmp*`, k.ix, "marg", value = numeric(0)) : replacement has length zero

to solve this so that it can read cleanly, add an if statement such that when 0 players are required in a given position (such as kicker position), it returns -1 marginal points. Then remove those rows with negative marginal points.

x3 <- ffvalues('.', 'qbheavy.csv', 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))

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

sum(x3[,'value'] > 20)

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

sum(x3[1:30, "pos"] == "qb")

Question 4

5 points

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

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

Using this list, write code to answer these questions.

Which function has the most arguments? (3 points)

```
which.max(lapply(funs, function(x) length(formals(x))))
```

```
## scan
##  938
```

scan How many functions have no arguments? (2 points)

```
length(funs[lapply(funs, function(x) length(formals(x)))==0])
```

```
## [1] 225
```

225 Hint: find a function that returns the arguments for a given function.