Bios 6301: Assignment 4

Brooklyn Stanley

Due Tuesday, 27 October, 1:00 PM

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

50 points total.

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

Question 1

18 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 x_2 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 $f'(x) = -\sin(x) - 1$.

```
f <- function(x) cos(x) - x

secant <- function(x1, x2, f, tol=1e-7, iter=100) {
  i <- 1
  while(abs(f(x2)) > tol & i < iter) {
    store <- x2
    x2 <- x2 - (f(x2) * (x2 - x1) / (f(x2) - f(x1)))
    x1 <- store
    i = i + 1
  }
  if (i == iter) {
    stop('Method did not converge')
  }
  x2
}
```

```
secant(1,2,f)
```

```
## [1] 0.7390851
```

```
ptm <- proc.time()
secant(1,2,f)
```

```
## [1] 0.7390851
```

```
proc.time() - ptm
```

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

```
ptm <- proc.time()
newtonRap(1,p)
```

```
## [1] 0.7390851
```

```
proc.time() - ptm
```

```
##    user  system elapsed
##   0.002   0.001   0.004
```

Question 2

20 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.

1. 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:

```
# Function to simulate the sum of a dice roll with 2 dice
diceRoll <- function() {
  roll <- sample(1:6,2,replace=T)
  return (sum(roll))
}

# Function to simulate a game of craps
craps <- function(output=T) {
  roll <- diceRoll()
  if (roll == 7 | roll == 11){
    W = 1
    if (output == T) print('Natural Win!')
    return (W)
  }
```

```
  point <- roll
  W = 0
  while (W == 0) {
    roll <- diceRoll()
    if (roll == 7) {
      W = -1
      if (output == T) print('7 or 11... You lose!')
      return (W)
    }
    if (roll == point) {
      W = 1
      if (output == T) print('Win by Point!')
      return (W)
    }
  }
}
```

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. (15 points)

```
set.seed(100)
games <- replicate(10, craps())
```

```
## [1] "7 or 11... You lose!"
## [1] "7 or 11... You lose!"
## [1] "Natural Win!"
## [1] "Win by Point!"
## [1] "7 or 11... You lose!"
## [1] "Natural Win!"
## [1] "Win by Point!"
## [1] "Natural Win!"
## [1] "7 or 11... You lose!"
## [1] "Natural Win!"
```

3. 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)

```
wins <- 0
seed <- 0
while (wins < 10) {
  seed = seed + 1
  set.seed(seed)
  wins <- sum(replicate(10, craps(output=F)))
}
seed
```

```
## [1] 825
```

```
set.seed(seed)
games <- replicate(10, craps())
```

```
## [1] "Natural Win!"
## [1] "Natural Win!"
## [1] "Win by Point!"
## [1] "Win by Point!"
## [1] "Win by Point!"
## [1] "Natural Win!"
## [1] "Win by Point!"
## [1] "Win by Point!"
## [1] "Win by Point!"
## [1] "Win by Point!"
```

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):

```
# path: directory path to input files
# file: name of the output file; it should be written to path
# nTeams: number of teams in league
# cap: money available to each team
# posReq: number of starters for each position
# points: point allocation for each category
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)) {
  k <- read.csv(file.path(path,'proj_k15.csv'), header=TRUE, stringsAsFactors=FALSE)
  qb <- read.csv(file.path(path,'proj_qb15.csv'), header=TRUE, stringsAsFactors=FALSE)
  rb <- read.csv(file.path(path,'proj_rb15.csv'), header=TRUE, stringsAsFactors=FALSE)
  te <- read.csv(file.path(path,'proj_te15.csv'), header=TRUE, stringsAsFactors=FALSE)
  wr <- read.csv(file.path(path,'proj_wr15.csv'), header=TRUE, stringsAsFactors=FALSE)

  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
```

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

  data[,'points'] <- rowSums(data[,grep("^p_", names(data))])

  data2 <- data[order(data[,'points'], decreasing=TRUE),]

  k.ix <- which(data2[,'pos']=='k')
  qb.ix <- which(data2[,'pos']=='qb')
  rb.ix <- which(data2[,'pos']=='rb')
  te.ix <- which(data2[,'pos']=='te')
  wr.ix <- which(data2[,'pos']=='wr')

  if (posReq['k'] == 0) {
    data2[k.ix, 'marg'] <- -1
  }
  else data2[k.ix, 'marg'] <- data2[k.ix,'points'] - data2[k.ix[posReq['k']*nTeams],'points']
  data2[qb.ix, 'marg'] <- data2[qb.ix,'points'] - data2[qb.ix[posReq['qb']*nTeams],'points']
  data2[rb.ix, 'marg'] <- data2[rb.ix,'points'] - data2[rb.ix[posReq['rb']*nTeams],'points']
  data2[te.ix, 'marg'] <- data2[te.ix,'points'] - data2[te.ix[posReq['te']*nTeams],'points']
  data2[wr.ix, 'marg'] <- data2[wr.ix,'points'] - data2[wr.ix[posReq['wr']*nTeams],'points']

  # create a new data.frame subset by non-negative marginal points
  data3 <- data2[data2[,'marg'] >= 0,]

  # re-order by marginal points
  data3 <- data3[order(data3[,'marg'], decreasing=TRUE),]

  # reset the row names
  rownames(data3) <- NULL

  data3[,'value'] <- data3[,'marg']*(nTeams*cap-nrow(data3))/sum(data3[,'marg']) + 1

  # create a data.frame with more interesting columns
  final <- data3[,c('PlayerName','pos','points','value')]
  write.csv(final, file)
  return(final)
}
```

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

```
x1 <- ffvalues(2015)
```

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

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

```
## [1] 40
```

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

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

```
##       PlayerName pos points    value
## 34 Melvin Gordon  rb 152.57 27.59549
```

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

```r
x2 <- ffvalues(2015, '16team.csv', nTeams=16, cap=150)
```

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

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

```
## [1] 41
```

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

```r
sum((which(x1[,'pos']=='wr') <= 40))
```

```
## [1] 14
```

3. Call:

```r
x3 <- ffvalues(2015, '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))
```

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

```r
sum(x3$value > 20)
```

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

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

```
sum((which(x3[,'pos']=='qb') <= 30))
```

```
## [1] 13
```

Question 4

5 bonus 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.

1. Which function has the most arguments? (3 points)

```
funs <- names(funs)
numArgs <- c()
for (i in 1:length(funs)) {
  numArgs[i] <- length(formals(funs[i]))
  if (is.na(numArgs[i])) numArgs[i] <- 0
}
funs[which.max(numArgs)]
```

```
## [1] "scan"
```

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

```
count = 0
for (i in 1:length(numArgs)){
  if (numArgs[i] == 0) count = count + 1
}
count
```

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