Bios 6301: Assignment 4

Yudong Cao

Grade: 48/50

Comment: In the football problem you've added a requisite that there is at least 1 player in each position via the following code: x2[qb.ix, 'marg'] <- x2[qb.ix,'points'] - x2[qb.ix[max(1,posReq['qb'])*nTeams],'points'] . I did not take off for this because it was a logical assumption. But FYI, Cole's intent is to allow for zero players in a position for the fantasy team.

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

```
secant<-function(f,x,tol=10e-7) {
    while (abs(f(x[2]))>tol) {
      y<-x[2]
      x[2]<-x[2]-f(x[2])*(x[2]-x[1])/(f(x[2])-f(x[1]))
      x[1]<-y
```

```
    }
  x[2]
}
f<-function(x) cos(x)-x
x<-c(0,1)
secant(f,x)
```

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

```
system.time(replicate(1000, secant(f,x)))
```

```
##    user  system elapsed
##   0.037   0.003   0.040
```

The root of the function $f(x) = \cos(x) - x$ under the secant method is 0.739.

```
newton<-function(guess,f,fp,tol=10e-7,maxiter=1000) {
  i<-1
  while(abs(f(guess))>tol && i<maxiter) {
    guess<-guess-f(guess)/fp(guess)
    i<-i+1
  }
  if(i==maxiter) {
    print('failed to converge')
    return(NULL)
  }
  guess
}
f<-function(x) cos(x)-x
fp<-function(x) -sin(x)-1
newton(0,f,fp)
```

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

```
system.time(replicate(1000,newton(0,f,fp)))
```

```
##    user  system elapsed
##   0.028   0.001   0.031
```

The root of the function $f(x) = \cos(x) - x$ under the Newton-Raphson method is 0.739.

Both methods return the same root. The Newton-Raphson method is faster by about 50% with 1000 replications.The secant method takes about twice the time as the Newton-Raphson does.

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)))
```

1. 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)

```
set.seed(100)
game<-function() {
  x<-sum(ceiling(6*runif(2)))
  if (x==7||x==11) {
    return('You win!')
  } else {
     z <- sum(ceiling(6*runif(2)))
     while (!(z==x)) {
       if (z==7||z==11) {
         return('You lose!')
       } else {
         z <- sum(ceiling(6*runif(2)))
       }
     }
     return('You win!')
  }
}
game()
```

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

```
game()
```

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

```
game()
```

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

JC Grading -2 Question also asks output to show progress of game.

2. 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)

```
n<-0
set.seed(n)
while(sum(replicate(10,tryCatch(game())=='You win!'))!=10) {
  n<-n+1
  set.seed(n)
}
print(n)
```

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

```
set.seed(n)
replicate(10,game())
```

```
##  [1] "You win!" "You win!" "You win!" "You win!" "You win!" "You win!"
##  [7] "You win!" "You win!" "You win!" "You win!"
```

When seed is set at 880, we have ten wins in a row.

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):

```
# 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)) {
  ## read in CSV files
  ## calculate dollar values
  ## save dollar values as CSV file
  ## return data.frame with dollar values
  path<-getwd()
  setwd(path) #assuming all the files are saved in the wd
  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)
  cols <- unique(c(names(k), names(qb), names(rb), names(te), names(wr)))
  k[,'pos'] <- 'k' #add a "position" column in dataset
  qb[,'pos'] <- 'qb'
  rb[,'pos'] <- 'rb'
  te[,'pos'] <- 'te'
  wr[,'pos'] <- 'wr'
  cols <- c(cols, 'pos')
  k[,setdiff(cols, names(k))] <- 0 #add columns not found in k but in all cols and set all values = 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[1]
  x[,'p_xpt'] <- x[,'xpt']*points[2]
  x[,'p_pass_yds'] <- x[,'pass_yds']*points[3]
  x[,'p_pass_tds'] <- x[,'pass_tds']*points[4]
  x[,'p_pass_ints'] <- x[,'pass_ints']*points[5]
  x[,'p_rush_yds'] <- x[,'rush_yds']*points[6]
  x[,'p_rush_tds'] <- x[,'rush_tds']*points[7]
  x[,'p_fumbles'] <- x[,'fumbles']*points[8]
  x[,'p_rec_yds'] <- x[,'rec_yds']*points[9]
  x[,'p_rec_tds'] <- x[,'rec_tds']*points[10]

  ## calculate dollar values
```

```
  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[qb.ix, 'marg'] <- x2[qb.ix,'points'] - x2[qb.ix[max(1,posReq['qb'])*nTeams],'points']
  x2[rb.ix, 'marg'] <- x2[rb.ix,'points'] - x2[rb.ix[max(1,posReq['rb'])*nTeams],'points']
  x2[wr.ix, 'marg'] <- x2[wr.ix,'points'] - x2[wr.ix[max(1,posReq['wr'])*nTeams],'points']
  x2[te.ix, 'marg'] <- x2[te.ix,'points'] - x2[te.ix[max(1,posReq['te'])*nTeams],'points']
  x2[k.ix, 'marg'] <- x2[k.ix, 'points'] - x2[k.ix[max(1,posReq['k'])*nTeams],'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(x4, file)

  ## return data.frame with dollar values
  return(x4)
}
```

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

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

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

```
sum(value>20)
```

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

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

```
x1[x1[,'pos'] == "rb",][15,1]
```

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

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

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

```
## The following objects are masked from x1:
##
##     PlayerName, points, pos, value
```

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

```
sum(value > 20)
```

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

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

```
nrow(x2[1:40,][x2[1:40,]['pos']=='wr',])
```

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

3. Call:

```
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))
attach(x3)
```

```
## The following objects are masked from x2:
##
##     PlayerName, points, pos, value
```

```
## The following objects are masked from x1:
##
##     PlayerName, points, pos, value
```

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

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

```
## [1] 50
```

```
x3[x3$value>20,]
```

```
##            PlayerName pos  points    value
## 1       Aaron Rodgers  qb 395.586 94.19072
## 2          Cam Newton  qb 384.832 85.04614
## 3      Russell Wilson  qb 374.956 76.64816
## 4         Andrew Luck  qb 366.980 69.86583
## 5     Adrian Peterson  rb 201.420 64.00612
## 6       Antonio Brown  wr 146.740 62.06309
## 7         Todd Gurley  rb 198.005 61.10220
## 8          Drew Brees  qb 356.022 60.54778
## 9   Odell Beckham Jr.  wr 138.855 55.35814
## 10     Jamaal Charles  rb 190.290 54.54181
## 11        Julio Jones  wr 132.535 49.98397
## 12      David Johnson  rb 183.780 49.00608
## 13       Lamar Miller  rb 181.915 47.42019
## 14     Rob Gronkowski  te 114.950 47.17784
## 15 Ben Roethlisberger  qb 340.088 46.99842
## 16    DeAndre Hopkins  wr 128.560 46.60386
## 17      Carson Palmer  qb 334.408 42.16848
## 18    Ezekiel Elliott  rb 172.825 39.69058
## 19         Dez Bryant  wr 120.155 39.45673
## 20       Le'Veon Bell  rb 170.180 37.44142
## 21      Blake Bortles  qb 327.998 36.71778
## 22         A.J. Green  wr 116.775 36.58258
## 23       Jordy Nelson  wr 114.615 34.74584
## 24        Eli Manning  qb 324.790 33.98988
## 25         Eddie Lacy  rb 164.770 32.84107
## 26      Philip Rivers  qb 323.424 32.82831
## 27       LeSean McCoy  rb 164.065 32.24158
## 28    Devonta Freeman  rb 162.295 30.73647
## 29        Doug Martin  rb 161.830 30.34106
## 30     Allen Robinson  wr 109.280 30.20926
```

```
## 31     Alshon Jeffery  wr 108.915 29.89889
## 32        Mark Ingram  rb 160.260 29.00603
## 33       Thomas Rawls  rb 158.400 27.42439
## 34   Brandon Marshall  wr 105.195 26.73561
## 35      Sammy Watkins  wr 104.380 26.04258
## 36         Mike Evans  wr 103.930 25.65993
## 37        T.Y. Hilton  wr 103.875 25.61316
## 38      Brandin Cooks  wr 103.055 24.91588
## 39       Randall Cobb  wr 100.625 22.84955
## 40         Greg Olsen  te  86.065 22.61570
## 41       Kirk Cousins  qb 311.308 22.52557
## 42          Tony Romo  qb 310.968 22.23645
## 43        Andy Dalton  qb 310.570 21.89801
## 44        Jordan Reed  te  84.965 21.68033
## 45       Tyrod Taylor  qb 309.618 21.08849
## 46   Demaryius Thomas  wr  98.515 21.05532
## 47       Keenan Allen  wr  98.330 20.89801
## 48      C.J. Anderson  rb 150.680 20.85975
## 49   Matthew Stafford  qb 309.338 20.85039
## 50         Derek Carr  qb 308.990 20.55447
```

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

```
nrow(x3[1:30,][x3[1:30,]['pos']=='qb',])
```

```
## [1] 10
```

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.

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

```
which.max(sapply(funs,function(f) length(formals(f))))
```

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

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

```
length(funs[(sapply(funs,function(f) length(formals(f)))==0)])
```

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

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