Bios 6301: Assignment 4

Josh DeClercq

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 (guess1, guess2, f, tol = 10e-7, maxiter =10000)
{
  i <- 1
  while (abs(f(guess1)) > tol && i < maxiter){
   guess3 <- guess2 - f(guess2)*((guess2 - guess1)/(f(guess2)-f(guess1)))
   guess1 <- guess2
   guess2 <- guess3
   i <- i + 1
  }
  if(i == maxiter) {
```

```r
    print('failed to converge')
    return (NULL)
  }else{
    print(sprintf('converges at %s', i))
  }
  guess1
}
```

```r
newton <- function(guess,f, fp, tol = 10e-7, maxiter = 10000){
  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)
  }else{
    print(sprintf('converges at %s', i))
  }
  guess
}
```

```r
f <- function(x) cos(x) - x
fp <- function(x) -sin(x) - 1
secant(-10,10, f)
```

```
## [1] "converges at 9"

## [1] 0.7390851
```

```r
newton(10,f,fp)
```

```
## [1] "converges at 49"

## [1] 0.7390852
```

```r
# time1 <- system.time(replicate(10000,newton(10,f,fp)))
# time2 <- system.time(replicate(10000,secant(-10,10, f)))
```

```r
library(tictoc)
tic()
invisible(capture.output(replicate(10000,newton(10,f,fp))))
newton_time <-toc()
```

```
## 2.057 sec elapsed
```

```r
newton_time <- newton_time$toc - newton_time$tic

tic()
invisible(capture.output(replicate(10000,secant(-10,10, f))))
secant_time <-toc()
```

```
## 1.058 sec elapsed
```

```
secant_time <- secant_time$toc - secant_time$tic

result <- newton_time - secant_time
ratio <- round(newton_time/secant_time,2)
```

The secant method is 0.999 seconds faster than the newton method, when both are simulated 10,000 times. The secant method is 1.94 times faster than the newton method.

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)
craps <- function(){
  win <- 2
  i <- 1
    x <- sum(ceiling(6*runif(2)))
    rolls <- x
     if (x == 7 | x == 11){
       i <- 0
        win <- 1
     }else{point <- x}

  while(i != 0){
  x <- sum(ceiling(6*runif(2)))
  rolls[i+1] <- x
    if(x == 7 | x == 11){
      i <- 0
      win <- 0
    }else if(x == point){
      i <- 0
        win <- 1
    }else {i <- i+1}
  }

   if (win == 1) {call <- "You Win!"
   } else {call <- "You lose!"}
```

```
    result <- list("rolls" = rolls, "outcome" = call)
    return(result)

}
craps()
```

```
## $rolls
##  [1]  4  5  6  8  6 10  5 10  5  8  9  9  5 11
##
## $outcome
## [1] "You lose!"
```

```
craps()
```

```
## $rolls
## [1]  6  9  9 11
##
## $outcome
## [1] "You lose!"
```

```
craps()
```

```
## $rolls
## [1] 6 7
##
## $outcome
## [1] "You lose!"
```

1. 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)

```
TEN <- FALSE
i <- 1
while (TEN == FALSE){
set.seed(i)
if (sum(invisible(replicate(10, craps()$outcome =="You Win!"))) == 10)
{
  TEN <- TRUE
  print(i)
  }else{i <- i+1}
}
```

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

```
set.seed(i) #The first seed that works is 880.
x <- list("game" = replicate(10, craps()$rolls))
x
```

```
## $game
## $game[[1]]
## [1] 7
##
## $game[[2]]
## [1]  8  9  3 10  6  8
##
## $game[[3]]
## [1] 10 10
##
## $game[[4]]
## [1] 9 9
##
## $game[[5]]
## [1] 11
##
## $game[[6]]
## [1] 8 8
##
## $game[[7]]
## [1] 5 5
##
## $game[[8]]
## [1] 7
##
## $game[[9]]
## [1] 9 9
##
## $game[[10]]
## [1] 7
```

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):

```r
# 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)) {
```

```r
#read in csv files
positions <- c('k','qb','rb','te','wr')
year <- 2016
csvfile <- paste('proj_', positions, substr(year, 3, 4), '.csv', sep='')
files <- file.path(year, csvfile)
names(files) <- positions
k <- read.csv(files['k'], header=TRUE, stringsAsFactors=FALSE)
qb <- read.csv(files['qb'], header=TRUE, stringsAsFactors=FALSE)
rb <- read.csv(files['rb'], header=TRUE, stringsAsFactors=FALSE)
te <- read.csv(files['te'], header=TRUE, stringsAsFactors=FALSE)
wr <- read.csv(files['wr'], header=TRUE, stringsAsFactors=FALSE)

# 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
# values are recylcled
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])

  ## calculate dollar values
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']

# sum selected column values for every row
x[,'points'] <- rowSums(x[,grep("^p_", names(x))])

# create new data.frame ordered by points descendingly
```

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

# determine the row indeces 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')

# calculate marginal points by subtracting "baseline" player's points
x2[k.ix, 'marg'] <- x2[k.ix,'points'] - x2[k.ix[nTeams],'points']
x2[qb.ix, 'marg'] <- x2[qb.ix,'points'] - x2[qb.ix[nTeams],'points']
x2[rb.ix, 'marg'] <- x2[rb.ix,'points'] - x2[rb.ix[2*nTeams],'points']
x2[te.ix, 'marg'] <- x2[te.ix,'points'] - x2[te.ix[nTeams],'points']
x2[wr.ix, 'marg'] <- x2[wr.ix,'points'] - x2[wr.ix[3*nTeams],'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(x3))/sum(x3[,'marg']) + 1

# create a data.frame with more interesting columns
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('.')`

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

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

```r
nrow(x1[x1[,'value'] > 20,])
```

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

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

```r
x1$PlayerName[x1$pos == 'rb'][15]
```

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

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

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

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

```
nrow(x2[x2[,'value'] > 20,])
```

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

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

```
nrow(na.exclude(x2[1:40,][x2$pos == 'wr',]))
```

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

1. 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))
```

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

```
nrow(x3[x3[,'value'] > 20,])
```

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

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

```
nrow(na.exclude(x3[1:30,][x3$pos == 'qb',]))
```

```
## [1] 5
```

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)

```r
arg_length <- sapply(funs, function(x) length(formals(x)))
max_arg <- which(arg_length == max(arg_length))
names(funs[max_arg])
```

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

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

```r
length(which(arg_length == 0))
```

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

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