Bios 6301: Assignment 4

Zi Ye

Grade: 49/50

Comment: There's a place in the final problem where you set c <- c(). This turns out okay because you're initializing c to a NULL vector. But, be careful to not save objects with the same name as a function (ex. c = c()). I didn't take off for this, just a general comment.

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

Anwser: For secant method:

```
secant = function(x0, x1, max_iter=1000, tol=10e-7, fun) {
  iter = 1
  y = abs(fun(x1))
  while (iter <= max_iter & y > tol) {
```

```
  x2 = x1-fun(x1)*(x1-x0)/(fun(x1)-fun(x0))
  x0 = x1
  x1 = x2
  iter = 1 + iter
  y = abs(fun(x2))
  }
  if (iter == max_iter) {
    print('failed to coverage')
    return(NULL)
  } else {
    print(sprintf('converge at %s', iter))
    return(x2)
  }
}
secant(9,10,fun=function(x) {cos(x)-x})
```

```
## [1] "converge at 8"
```

```
## [1] 0.7390849
```

For newton method:

```
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)
  } else {
    print(sprintf('converge at %s', i))
  }
  guess
}
f <- function(x) cos(x)-x
fp <- function(x) -sin(x)-1
newton(10,f,fp)
```

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

```
## [1] 0.7390852
```

I ran `system.time(replicate(10000,secant(9,10,fun=function(x) {cos(x)-x})))` and `system.time(replicate(1000` `newton(10,f,fp)))` seperately since it created a long output and I don't know how to ignore the information from `print()` and just show the time. For secant method, the user time is 0.960, and the system time is 0.128, and the elasped time is 1.061. For Newton method, the user time is 3.520, and the system time is 0.156, and the elasped time is 3.611. The secant method is three-times 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)))
```

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)

```
craps_game <- function(count = FALSE, rep = 1) {
timer = 0
for (i in seq(rep)) {
x <- sum(ceiling(6*runif(2)))
if (x == 7 | x == 11) {
  counter = 1
  if (count == FALSE) {print(sprintf('you rolled %s, you win!', x))}
} else {
  x_true <- x
  if (count == FALSE) {print(sprintf('you rolled %s, you need to roll %s again to win', x, x))}
  for (i in seq(10000)) {
    x <- sum(ceiling(6*runif(2)))
    if (x == x_true) {
      counter = 1
      if (count == FALSE) {print(sprintf('you rolled %s again, you win!', x_true))}
      break
    } else if (x == 7 | x ==11) {
      if (count == FALSE) {print(sprintf('you rolled %s, you lose!', x))}
      counter = 0
      break
    }
  }
}
timer = counter + timer
}
while (count == TRUE) return(timer)
}

set.seed(1000)
craps_game(rep = 3)
```

```
## [1] "you rolled 7, you win!"
## [1] "you rolled 6, you need to roll 6 again to win"
## [1] "you rolled 6 again, you win!"
## [1] "you rolled 7, you win!"
```

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)

```
sa <- sample(1:10000,10000)
for (i in 1:10000) {
  set.seed(sa[i])
  x <- craps_game(count = TRUE, rep = 10)
  if (x == 10) {
    t <- sa[i]
    break
  }
}
t
```

```
## [1] 4411
```

```
set.seed(t)
craps_game(rep = 10)
```

```
## [1] "you rolled 7, you win!"
## [1] "you rolled 10, you need to roll 10 again to win"
## [1] "you rolled 10 again, you win!"
## [1] "you rolled 7, you win!"
## [1] "you rolled 7, you win!"
## [1] "you rolled 6, you need to roll 6 again to win"
## [1] "you rolled 6 again, you win!"
## [1] "you rolled 7, you win!"
## [1] "you rolled 11, you win!"
## [1] "you rolled 10, you need to roll 10 again to win"
## [1] "you rolled 10 again, you win!"
## [1] "you rolled 7, you win!"
## [1] "you rolled 11, you win!"
```

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
#somehow I could not read the values inside the c(). So I used a stupid way to take them out.
  fg <- data.frame(points)['fg',]
  xpt <- data.frame(points)['xpt',]
  pass_yds <- data.frame(points)['pass_yds',]
  pass_tds <- data.frame(points)['pass_tds',]
  pass_ints <- data.frame(points)['pass_ints',]
  rush_yds <- data.frame(points)['rush_yds',]
  rush_tds <- data.frame(points)['rush_tds',]
  fumbles <- data.frame(points)['fumbles',]
  rec_yds <- data.frame(points)['rec_yds',]
  rec_tds <- data.frame(points)['rec_tds',]
  qb <- data.frame(posReq)['qb',]
  rb <- data.frame(posReq)['rb',]
  wr <- data.frame(posReq)['wr',]
  te <- data.frame(posReq)['te',]
```

```
  k <- data.frame(posReq)['k',]

  ## read in CSV files

  setwd(path)
  qbdat = read.csv('proj_qb16.csv', header = T)
  rbdat = read.csv('proj_rb16.csv', header = T)
  wrdat = read.csv('proj_wr16.csv', header = T)
  tedat = read.csv('proj_te16.csv', header = T)
  kdat = read.csv('proj_k16.csv', header = T)

  ## calculate dollar values

  qb_pts = data.frame(PlayerName=qbdat$PlayerName, pos='qb', points = (qbdat$pass_yds*pass_yds) + (qbda
  rb_pts = data.frame(PlayerName=rbdat$PlayerName, pos='rb', points = (rbdat$rush_yds*rush_yds) + (rbda
  wr_pts = data.frame(PlayerName=wrdat$PlayerName, pos='wr', points = (wrdat$rush_yds*rush_yds) + (wrda
  te_pts = data.frame(PlayerName=tedat$PlayerName, pos='te', points = (tedat$fumbles*fumbles) + (tedat$
  k_pts = data.frame(PlayerName=kdat$PlayerName, pos='k', points = (kdat$fg*fg) + (kdat$xpt*xpt))

  qb_pts <- qb_pts[order(qb_pts[,'points'], decreasing=TRUE),]
  rb_pts <- rb_pts[order(rb_pts[,'points'], decreasing=TRUE),]
  wr_pts <- wr_pts[order(wr_pts[,'points'], decreasing=TRUE),]
  te_pts <- te_pts[order(te_pts[,'points'], decreasing=TRUE),]
  k_pts <- k_pts[order(k_pts[,'points'], decreasing=TRUE),]

  qb_pts[,'marg'] <- qb_pts[,'points'] - qb_pts[qb*nTeams, 'points']
  rb_pts[,'marg'] <- rb_pts[,'points'] - rb_pts[rb*nTeams, 'points']
  wr_pts[,'marg'] <- wr_pts[,'points'] - wr_pts[wr*nTeams, 'points']
  te_pts[,'marg'] <- te_pts[,'points'] - te_pts[te*nTeams, 'points']
  k_pts[,'marg'] <- k_pts[,'points'] - k_pts[k*nTeams, 'points']

  x1 <- rbind(qb_pts, rb_pts, wr_pts, te_pts, k_pts)
  x2 <- x1[order(x1[,'marg'], decreasing = T),]
  x2[,'value'] <- x2[,'marg']*(nTeams*cap-nTeams*sum(posReq))/sum(x2[1:(sum(posReq)*nTeams),'marg']) + 
  x2[(sum(posReq)*nTeams+1):nrow(x2),'value'] = 0

  ## save dollar values as CSV file

  write.csv(data.frame(x2, row.names = NULL), file=file)

  ## return data.frame with dollar values

  data.frame(x2, row.names = NULL)
}
```

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

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

```
  #setwd('~/Downloads/football-values-master/2016/')
  x1 <- ffvalues('.')
  nrow(x1[x1[,'value']>20,])
```

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

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

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

```
##     PlayerName pos points   marg    value
## 47 Carlos Hyde  rb 145.47 18.145 19.76574
```

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

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

```
    #setwd('~/Downloads/football-values-master/2016/')
    x2 <- ffvalues(getwd(), '16team.csv', nTeams=16, cap=150)
    nrow(x2[x2[,'value']>20,])
```

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

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

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

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

3. Call:

```
    #setwd('~/Downloads/football-values-master/2016/')
    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] 51
```

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

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

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

I don't understand why I always got errors if I don't `setwd()` for every question. I wrote `setwd(path)` in the function.

Comment I suspect it's because

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)

```
c <- c()
for (i in 1:length(funs)) {
  x <- length(as.list(args(funs[[i]])))
  c[i] <- x
}
ma <- which.max(c)
funs[[ma]]
```

```
## function (file = "", what = double(), nmax = -1L, n = -1L, sep = "",
##     quote = if (identical(sep, "\n")) "" else "'\"", dec = ".",
##     skip = 0L, nlines = 0L, na.strings = "NA", flush = FALSE,
##     fill = FALSE, strip.white = FALSE, quiet = FALSE, blank.lines.skip = TRUE,
##     multi.line = TRUE, comment.char = "", allowEscapes = FALSE,
##     fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
## {
##     na.strings <- as.character(na.strings)
##     if (!missing(n)) {
##         if (missing(nmax))
##             nmax <- n/pmax(length(what), 1L)
##         else stop("either specify 'nmax' or 'n', but not both.")
##     }
##     if (missing(file) && !missing(text)) {
##         file <- textConnection(text, encoding = "UTF-8")
##         encoding <- "UTF-8"
##         on.exit(close(file))
##     }
##     if (is.character(file))
##         if (file == "")
##             file <- stdin()
##         else {
##             file <- if (nzchar(fileEncoding))
##                 file(file, "r", encoding = fileEncoding)
##             else file(file, "r")
##             on.exit(close(file))
##         }
##     if (!inherits(file, "connection"))
##         stop("'file' must be a character string or connection")
##     .Internal(scan(file, what, nmax, sep, dec, quote, skip, nlines,
##         na.strings, flush, fill, strip.white, quiet, blank.lines.skip,
##         multi.line, comment.char, allowEscapes, encoding, skipNul))
## }
## <bytecode: 0x7fc17cca3498>
## <environment: namespace:base>
```

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

```
tru <- c()
for (i in 1:length(funs)) {
  tru[i] <- (c[i] == 0)
}
length(tru[tru==TRUE])
```

```
## [1] 25
```

JC Grading -1

Comment Instead of x <- length(as.list(args(funs[[i]]))), call x <- length(formas(funs[[i]])). You'll find the answer of 225.

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