Assignment 04

Ya-Chen Lin

2015 M10 17

Question 1

15 points Write a function that implements the secant algorithm. Function for secant algorithm

```
formula <- function(x) cos(x) - x
secant <- function(a,tol=1e-9, max_iter=1000){
  x <- c()
  x[1] <- a
  x[2] <- a + 1e6
  iter <- 3
  while(abs(formula(x[iter-1])-formula(x[iter-2])) > tol && (iter < max_iter)){
    x[iter] <-  x[iter-1] - formula(x[iter-1]) * ((x[iter-1] - x[iter-2])/(formula(x[iter-1]) - formula
    iter <- iter + 1
  }
  x[iter-1]
}
secant(10)
```

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

We can see that it converges to 0.7390851

```
#Newton-Raphson method megthod:
formula <- function(x) cos(x) - x
derivative <- function(x) -sin(x) - 1
newton <- function(c, tol=1e-9, max_iter=1000){
  x <- c()
  iter <- 3
  x[1] <- c
  x[2] <- c + 1e6
  while(abs(formula(x[iter-1])-formula(x[iter-2])) > tol && (iter < max_iter)){
    x[iter] <- x[iter-1] - (formula(x[iter-1])) / (derivative(x[iter-1]))
    iter <- iter + 1
  }
  x[iter-1]
}
newton(10)
```

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

We can see it converges to the same thing. Now we are interested in the executing time.

```
system.time(replicate(1000,newton(10)))
```

```
##    user  system elapsed
##    5.25    0.00    5.26
```

```
system.time(replicate(1000, secant(10)))
```

```
##    user  system elapsed
##     0.1     0.0     0.1
```

Here I replicate the function 1000 times and measure the time with random starting x0. We can see that my secant method is way faster than my Newton-Raphson method method by at least 5 seconds with 1000 replicates! (slightly different for each run)

Question 2

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:

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)

```
playCraps <- function(firstroll, rollAgain, rolls){
  firstroll <- sum(ceiling(6*runif(2)))
  if (firstroll == 7 | firstroll == 11){
    rollAgain <- firstroll
    rolls <- 1
    return(list('You win!', rollAgain, rolls))
  }else{
    rollAgain <- c()
    rollAgain <- c(firstroll, sum(ceiling(6*runif(2))))
    rolls <- 2
    while(rollAgain[rolls] != firstroll && rollAgain[rolls] != 7 && rollAgain[rolls] !=11){
      rollAgain <- c(rollAgain, sum(ceiling(6*runif(2))))
      rolls <- rolls + 1
    }
    if (rollAgain[rolls] == firstroll){
      return(list('You win!', rollAgain, rolls))
    }else{
      return(list('You lost!', rollAgain, rolls))
    }
  }

}
```

```
require('foreach')
```

```
## Loading required package: foreach
```

```
set.seed(100)
foreach(i=1:3) %do% playCraps(i)
```

```
## [[1]]
## [[1]][[1]]
## [1] "You lost!"
##
## [[1]][[2]]
##  [1]  4  5  6  8  6 10  5 10  5  8  9  9  5 11
##
## [[1]][[3]]
## [1] 14
##
##
## [[2]]
## [[2]][[1]]
## [1] "You lost!"
##
## [[2]][[2]]
## [1]  6  9  9 11
##
## [[2]][[3]]
## [1] 4
##
##
## [[3]]
## [[3]][[1]]
## [1] "You lost!"
##
## [[3]][[2]]
## [1] 6 7
##
## [[3]][[3]]
## [1] 2
```

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)

```r
#find the seed where we win ten straight games
k <- 1
b<- c()
while(sum(b[] == "You win!") != 10){
  set.seed(k)
  x <- foreach(i=1:10) %do% playCraps()
  k <- k+ 1
  b <- c(x[[1]][[1]],x[[2]][[1]],x[[3]][[1]],x[[4]][[1]],x[[5]][[1]],
         x[[6]][[1]],x[[7]][[1]],x[[8]][[1]],x[[9]][[1]],x[[10]][[1]])
}
#x here will display results of 10 games (win/lose, rolls summary, #of rolls)
set.seed(k)
x
```

```
## [[1]]
## [[1]][[1]]
## [1] "You win!"
##
## [[1]][[2]]
## [1] 7
```

```
##
## [[1]][[3]]
## [1] 1
##
##
## [[2]]
## [[2]][[1]]
## [1] "You win!"
##
## [[2]][[2]]
## [1]  8  9  3 10  6  8
##
## [[2]][[3]]
## [1] 6
##
##
## [[3]]
## [[3]][[1]]
## [1] "You win!"
##
## [[3]][[2]]
## [1] 10 10
##
## [[3]][[3]]
## [1] 2
##
##
## [[4]]
## [[4]][[1]]
## [1] "You win!"
##
## [[4]][[2]]
## [1] 9 9
##
## [[4]][[3]]
## [1] 2
##
##
## [[5]]
## [[5]][[1]]
## [1] "You win!"
##
## [[5]][[2]]
## [1] 11
##
## [[5]][[3]]
## [1] 1
##
##
## [[6]]
## [[6]][[1]]
## [1] "You win!"
##
## [[6]][[2]]
## [1] 8 8
##
## [[6]][[3]]
## [1] 2
##
##
## [[7]]
## [[7]][[1]]
## [1] "You win!"
##
## [[7]][[2]]
## [1] 5 5
##
## [[7]][[3]]
## [1] 2
##
##
## [[8]]
## [[8]][[1]]
## [1] "You win!"
##
## [[8]][[2]]
## [1] 7
##
## [[8]][[3]]
## [1] 1
##
##
## [[9]]
## [[9]][[1]]
## [1] "You win!"
##
## [[9]][[2]]
## [1] 9 9
##
## [[9]][[3]]
## [1] 2
##
##
## [[10]]
## [[10]][[1]]
## [1] "You win!"
##
## [[10]][[2]]
## [1] 7
##
## [[10]][[3]]
## [1] 1
```

Question 3

12 points

Obtain a copy of the football-values lecture. Save the five 2015 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
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('proj_k15.csv')
  qb <- read.csv('proj_qb15.csv')
  rb <- read.csv('proj_rb15.csv')
  te <- read.csv('proj_te15.csv')
  wr <- read.csv('proj_wr15.csv')
  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')
  if(posReq['k'] == 0) {
    x2[k.ix, 'marg'] <- 0
  }else{
    x2[k.ix, 'marg'] <- x2[k.ix,'points'] - x2[k.ix[nTeams*posReq['k']],'points']
  }
  if(posReq['qb'] == 0) {
    x2[qb.ix, 'marg'] <-  0
  }else{
    x2[qb.ix, 'marg'] <- x2[qb.ix,'points'] - x2[qb.ix[nTeams*posReq['qb']],'points']
  }
  if(posReq['rb'] == 0){
    x2[rb.ix, 'marg'] <- 0
  }else{
    x2[rb.ix, 'marg'] <- x2[rb.ix,'points'] - x2[rb.ix[nTeams*posReq['rb']],'points']
  }
  if(posReq['te'] == 0){
    x2[te.ix, 'marg'] <- 0
  }else{
    x2[te.ix, 'marg'] <- x2[te.ix,'points'] - x2[te.ix[nTeams*posReq['te']],'points']
  }
  if(posReq['wr'] == 0){
    x2[wr.ix, 'marg'] <- 0
  }else{
    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),]
  x3[,'value'] <- x3[,'marg']*(nTeams*cap-nrow(x3))/sum(x3[,'marg']) + 1
  for ( i in 1:length(posReq)){
    if (posReq[i] == 0){
      x3 <- x3[!x3[,'pos'] == names(posReq[i]),]
    }
  }
  x4 <- x3[,c('PlayerName','pos','points','value')]
  write.csv(x4, file=file)
  data.frame(x4)
}
```

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

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

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

```
b <- x1[,'value'] > 20
sum(b)
```

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

We can see that 40 players are worth more than $20.

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

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

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

We can see that Melvin Gordon is 15th most valuable running back.

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

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

i.How many players are worth more than $20?

```r
c <- x2[,'value'] > 20
sum(c)
```

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

We can see that in this case, 41 players are worth more than $20.

ii. How many wide receivers (wr) are in the top 40?

```r
top40 <- x2[1:40,]
wrtop40 <- top40[,'pos'] == 'wr'
sum(wrtop40)
```

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

We can see that 13 wide receivers are in the top 40.

1. Call:

   ```r
   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))
   ```

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

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

```
## [1] 44
```

We can see that 44 players are worth more than $20

ii. How many quarterbacks (qb) are in the top 30?

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

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

In this case, there are 13 quarterbacks in the top 30.

Question 4

5 points

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

```r
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
objs <- mget(ls("package:base"), inherits = TRUE)
funs <- Filter(is.function, objs)
arg_length <- lapply(funs, function(x) length(formals(x)))
arg_length_number <- as.numeric(arg_length) #make it into numeric from 'list' type.
max_length_arg <- which(arg_length == max(arg_length_number))
names(funs[max_length_arg])
```

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

We can see the it's "scan".

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

```r
sum(arg_length_number == 0)
```

```
## [1] 224
```

We can see that 224 functions have 0 arguments.