Bios 6301: Assignment 4

Nick Strayer

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

So we start by coding up the secant method. We will use recursion and make sure we only calculate values once to make it as fast as possible.

```
secantMethod <- function(f, x0, x1, tol = 0.0001){
  #Set up function for recurance equation
  nextGuess <- function(f, x0, x1) {
    fx1 <- f(x1) #We use this value twice, so let's only calculate it once.
    return(x1 - fx1 * (x1 - x0)/(fx1 - f(x0)))
  }

  #Get the first guess
  x2 <- nextGuess(f, x0, x1)

  #Check for tolerance level:
  if(abs(x1 - x2) > tol){
    secantMethod(f, x1, x2, tol) #recursively call the function again.
  } else {
    return(x2) #Return the final value.
  }
}
```

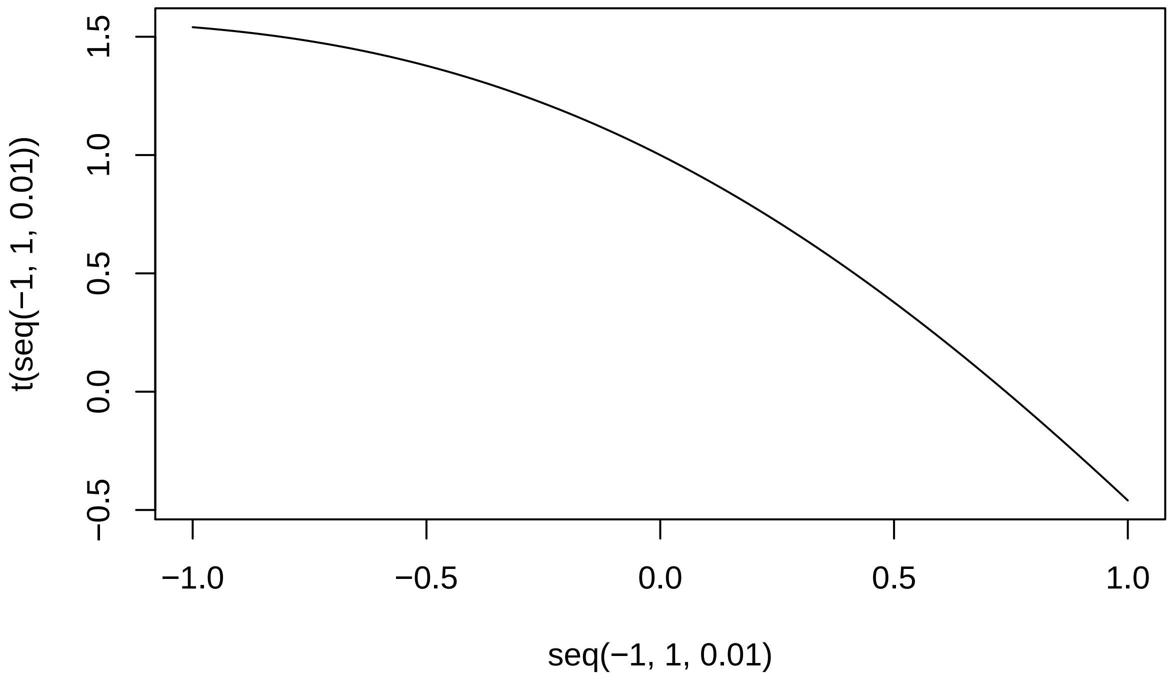
Now let's test it on the function we were given.

```
t <- function(x) cos(x) - x
secantMethod(t, 2.1, 2.5)
```

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

Just to make sure that this answer is right, let's plot our test function real quick.

```
plot(seq(-1,1,0.01), t(seq(-1,1,0.01)), type = 'l')
```



Great. It looks like it's working. Now to compare with NR.

First we bring in our NR algorithm from class. I have written it in as similar a format as possible to make a speed comparison fair.

```
dt = function(x) -sin(x) - 1 #Code in the derivative.
x0  = 2.1 #initial value

NR = function(f, df, x0, tol = 0.0001){
  #recurance equation
  nextGuess <- function(f, df, x0) x0 - (f(x0))/(df(0))

  #Get the first guess
  x1 <- nextGuess(f, df, x0)

  #Check for tolerance level:
  if(abs(x0 - x1) > tol){
    NR(f, df, x1, tol) #recursively call the function again.
  } else {
    return(x1) #Return the final value.
  }
}

#A quick test.
NR(t, dt, x0)
```

```
## [1] 0.7390502
```

So we're getting the same answers from both algorithms. Now for the fun stuff.

We will set our tolerance to a very small number and then repeat the function a ton of times. Fastest one wins!

I have chosen to use a normally drawn value as the intial guess as it helps us get a better sense of the performance of the algorithms true performance from different starting points rather than just one. This is important because we could choose a point that takes a particularly long or short period of time to converge for a particular algorithm and it would throw off our comparison.

Since only one draw is done from the same function for each test the compute time added will be identical and it won't throw off the conclusion.

We start with the **secant method**:

```
tollerance = 0.00001
numberOfReps = 100000
system.time(replicate(numberOfReps, secantMethod(t, rnorm(1, 0, 2), 2.5, tol = tollerance)))
```

```
##    user  system elapsed
##   3.944   0.041   4.041
```

...and on to the **newton-raphson method**:

```
system.time(replicate(numberOfReps, NR(t, dt, rnorm(1, 0, 2), tol = tollerance)))
```

```
##    user  system elapsed
##  13.469   0.144  13.781
```

It is rather conclusive that the secant method is the fastest. By almost 2 times (although the magnitude of difference could be thrown off by the `rnorm` call.)

Question 2

> 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:

```
#Dice rolling function
rollDice <- function(n = 2, d = c(1,2,3,4,5,6), printGame = F){
  res = sum(sample(d, n, replace = T))
  if(printGame){print(res)} #Do we want to see the way the game plays out?
  return(res)
}

#Let's play craps!
playCraps <- function(firstRoll = 0, printGame = F){
  if(firstRoll == 0){ #if it is our first time playing the game, roll for the first time.
    firstRoll <- rollDice(printGame = printGame) #First roll
    if (firstRoll %in% c(7,11)) return("win") #win if it's a 7 or 11
  }

  r <- rollDice(printGame = printGame)
  if(r == firstRoll){ #Check if our roll is the first roll.
```

```r
        return("win") #if it is, we win
      } else if(r %in% c(7,11)) { #If we didn't win, did we lose by rolling a 7 or 11?
        return("lose")
      } else { #If we didn't win or lose, play again.
        playCraps(firstRoll = firstRoll, printGame = printGame)
      }
    }
```

So now that it's in, we test it:

```r
playCraps(printGame = T)
```

```
## [1] 3
## [1] 3

## [1] "win"
```

Good, looks like it's working.

> 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)

```r
set.seed(100)
replicate(3,playCraps(printGame = T))
```

```
## [1] 4
## [1] 5
## [1] 6
## [1] 8
## [1] 6
## [1] 10
## [1] 5
## [1] 10
## [1] 5
## [1] 8
## [1] 9
## [1] 9
## [1] 5
## [1] 11
## [1] 6
## [1] 9
## [1] 9
## [1] 11
## [1] 6
## [1] 7

## [1] "lose" "lose" "lose"
```

> 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)

To do this we will scan through possible seeds until we hit one that allows us to win 10 games in a row.

```
s <- 10 #initialize seed counter.
while(sum(replicate(10, playCraps()) == "win") != 10){
  s <- s + 1  #increment our seed
  set.seed(s) #set the new seed.
}
print(s)
```

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

Cool. So 880 gives us what we need. Let's just make sure this is correct.

```
set.seed(880)
replicate(10, playCraps())
```

```
##  [1] "win" "win" "win" "win" "win" "win" "win" "win" "win" "win"
```

Schweet.

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):

```
# setwd("/Users/Nick/Dropbox/vandy/computing/homework/data")
# 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
  setwd(path) #Get to the right place.
  k  <- read.csv("proj_k15.csv")
```

```r
qb <- read.csv("proj_qb15.csv")
rb <- read.csv("proj_rb15.csv")
te <- read.csv("proj_te15.csv")
wr <- read.csv("proj_wr15.csv")

k[,'pos']  <- 'k'
qb[,'pos'] <- 'qb'
rb[,'pos'] <- 'rb'
te[,'pos'] <- 'te'
wr[,'pos'] <- 'wr'

# generate unique list of column names
cols <- unique(c(names(k), names(qb), names(rb), names(te), names(wr)))

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

#Create one big dataframe.
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"]

## calculate dollar values

x[,'points'] <- rowSums(x[,grep("^p_", names(x))])
x2 <- x[order(x[,'points'], decreasing=TRUE),]

# determine the row indices 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')


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']
```

```r
  # 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','marg','value')]
  x4[,'marg'] <- NULL
  ## save dollar values as CSV file
  write.csv(x4, file)
  ## return data.frame with dollar values
  return(x4)
}
```

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

```r
setwd("/Users/Nick/Dropbox/vandy/computing/homework/data")
x1 <- ffvalues('.')
```

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

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

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

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

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

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

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

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

```r
setwd("/Users/Nick/Dropbox/vandy/computing/homework/data")
x2 <- ffvalues(getwd(), '16team.csv', nTeams=16, cap=150)
sum(x2[,'value'] > 20)
```

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

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

```
sum(x2[1:40, "pos" ] == "wr")
```

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

I added the following lines to the `ffvalues` function to allow for the 0 values.

Now we continue.

```
setwd("/Users/Nick/Dropbox/vandy/computing/homework/data")
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))
```

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

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

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

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

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

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

Question 4

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.

Which function has the most arguments? (3 points)

```
argLengths = lapply(funs, function(f) length(formals(f)))
which.max(argLengths)
```

```
## scan
##  910
```

How many functions have no arguments? (2 points)

```
sum(argLengths == 0)
```

```
## [1] 221
```