Homework4

Molly Olson

October 26, 2015

Question 1

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.

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

```
#Newton method
#function that takes a guess of the root, x, and f, an expression of a function
#root finding algorithm
newton <- function(x, f, tol=1e-7, iter=100){
  i <- 1
  while(abs(eval(f)) > tol & i < iter){
    x <- x - eval(f)/eval(D(f,'x'))
    i <- i + 1
  }
  if(i == iter){
    stop('method did not converge')
  }
  x
}

newton(1, expression(cos(x)-x) )
```

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

```
#Secant method:
#function that takes a function f, two guesses for the root: older and newer.
#root finding algorithm
secant <- function(f, older, newer, tol=1e-7, iter=100){
  i <- 1
  while(abs(f(newer)) > tol & i < iter){
```

```
    newest <- newer - (f(newer)*((newer-older)/(f(newer)-f(older))))
    older <- newer
    newer <- newest

    i <- i + 1
  }
  if(i == iter){
    stop('method did not converge')
  }
  newest
}
```

```
f <- function(x) cos(x)-x
```

```
secant(f,0,1)
```

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

```
system.time({
  for(i in seq(1000)){
  secant(f,0,1)
  }
  })
```

```
##    user  system elapsed
##   0.023   0.004   0.028
```

```
newton(1, expression(cos(x)-x))
```

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

```
system.time({
  for(i in seq(1000)){
  newton(1,expression(cos(x)-x))
  }
  })
```

```
##    user  system elapsed
##   0.029   0.002   0.030
```

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:

```
x <- sum(ceiling(6*runif(2)))
```

```
roll <- function(){
  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)

```
#helper function for Craps, will take the intial roll that is calculated in #craps and compare it to an
#initial roll, it returns Win, if it isn't, it checks if it equal to
#7 or 11. if it is, it returns lose.
#if neither of these conditions are met, it will repeat the function,
#reroll and check conditions again.
#print2 turns on and off printing of results
helperCraps <- function(initialRoll, print2 ){
  repeat{
  subsequentRoll <- roll()
  if(print2 == "on"){
  print("Subsequent roll value:")
  print(subsequentRoll)
  }
  if(subsequentRoll == initialRoll){
    return("WIN!")

  }
  if(subsequentRoll == 7 | subsequentRoll== 11){
    return("LOSE!")
  }
  }

}
```

```
#craps, will roll an initial roll and check if it is 7 or 11. if it isn't,
# it will go to the helper function. if it is, it will return Win.
#print1 and print2 turns on and off printing
craps <- function(seed = 100, print1 = "off", print2 = "off"){
  set.seed(seed)
    initialRoll <- roll()

    if(print1 == "on"){
      print("initial roll value:")
      print(initialRoll)
    }

    if(initialRoll == 7 | initialRoll == 11){
        return("WIN!")
    } else {
    helperCraps(initialRoll,print2)
        }
}
```

```
craps(print1="on",print2="on")
```

```
## [1] "initial roll value:"
## [1] 4
## [1] "Subsequent roll value:"
## [1] 5
## [1] "Subsequent roll value:"
## [1] 6
## [1] "Subsequent roll value:"
## [1] 8
## [1] "Subsequent roll value:"
## [1] 6
## [1] "Subsequent roll value:"
## [1] 10
## [1] "Subsequent roll value:"
## [1] 5
## [1] "Subsequent roll value:"
## [1] 10
## [1] "Subsequent roll value:"
## [1] 5
## [1] "Subsequent roll value:"
## [1] 8
## [1] "Subsequent roll value:"
## [1] 9
## [1] "Subsequent roll value:"
## [1] 9
## [1] "Subsequent roll value:"
## [1] 5
## [1] "Subsequent roll value:"
## [1] 11

## [1] "LOSE!"
```

```
craps(print1="on",print2="on")
```

```
## [1] "initial roll value:"
## [1] 4
## [1] "Subsequent roll value:"
## [1] 5
## [1] "Subsequent roll value:"
## [1] 6
## [1] "Subsequent roll value:"
## [1] 8
## [1] "Subsequent roll value:"
## [1] 6
## [1] "Subsequent roll value:"
## [1] 10
## [1] "Subsequent roll value:"
## [1] 5
## [1] "Subsequent roll value:"
## [1] 10
## [1] "Subsequent roll value:"
```

```
## [1] 5
## [1] "Subsequent roll value:"
## [1] 8
## [1] "Subsequent roll value:"
## [1] 9
## [1] "Subsequent roll value:"
## [1] 9
## [1] "Subsequent roll value:"
## [1] 5
## [1] "Subsequent roll value:"
## [1] 11

## [1] "LOSE!"
```

```
craps(print1="on",print2="on")
```

```
## [1] "initial roll value:"
## [1] 4
## [1] "Subsequent roll value:"
## [1] 5
## [1] "Subsequent roll value:"
## [1] 6
## [1] "Subsequent roll value:"
## [1] 8
## [1] "Subsequent roll value:"
## [1] 6
## [1] "Subsequent roll value:"
## [1] 10
## [1] "Subsequent roll value:"
## [1] 5
## [1] "Subsequent roll value:"
## [1] 10
## [1] "Subsequent roll value:"
## [1] 5
## [1] "Subsequent roll value:"
## [1] 8
## [1] "Subsequent roll value:"
## [1] 9
## [1] "Subsequent roll value:"
## [1] 9
## [1] "Subsequent roll value:"
## [1] 5
## [1] "Subsequent roll value:"
## [1] 11

## [1] "LOSE!"
```

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)

```
#find a seed that will win 10 games
#play 10 games
```

```
vector <- rep(0,10)
for(j in seq(1000)){
  for(i in seq(10)){
    if(craps(seed = j)=="WIN!"){
      vector[i] <- 1
    }

  }
   if(sum(vector)==10){
     print(c("seed is:",j))
     break
   }
}
```

```
## [1] "seed is:" "1"
```

So set seed=1

```
for(i in seq(10)){
    if(craps(seed = 1)=="WIN!"){
      print("WIN!")
    }
}
```

```
## [1] "WIN!"
## [1] "WIN!"
## [1] "WIN!"
## [1] "WIN!"
## [1] "WIN!"
## [1] "WIN!"
## [1] "WIN!"
## [1] "WIN!"
## [1] "WIN!"
## [1] "WIN!"
```

Question 3

Obtain a copy of the football-values lecture. Save the file 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 ordered by value descendingly. Do not round dollar values.
Note that the returned data.frame should have `sum(posReq)*nTeams` rows.

```
setwd("/Users/mollyolson/documents/Vanderbilt_2015_Fall/Statistical_Computing/football-values-master")
```

```
ffvalues <- function(path, file='outfile.csv', nTeams=12, cap=200, posReq=c(qb=1, rb=2, wr=3,te=1, k=1)

  x <- data.frame(id=1:3, letter=letters[1:3], value=rnorm(3), even=(1:3)%%2==0)
  x
```

```
nrow(x)
ncol(x)
names(x)
rownames(x)
str(x)
year <- 2015

#read in data
k <- read.csv(file.path(year,'proj_k15.csv'), header=TRUE, stringsAsFactors=FALSE)
qb <- read.csv(file.path(year,'proj_qb15.csv'), header=TRUE, stringsAsFactors=FALSE)
rb <- read.csv(file.path(year,'proj_rb15.csv'), header=TRUE, stringsAsFactors=FALSE)
te <- read.csv(file.path(year,'proj_te15.csv'), header=TRUE, stringsAsFactors=FALSE)
wr <- read.csv(file.path(year,'proj_wr15.csv'), 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
  # concept: ?Extract
  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])

  # guess how to combine by column

   # calculate new columns
  # convert NFL stat to fantasy points
  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
# this is total fantasy points for each player
x[,'points'] <- rowSums(x[,grep("^p_", names(x))])

# create new data.frame ordered by points descendingly
x2 <- x[order(x[,'points'], decreasing=TRUE),]

# which rows contain quarterbacks?
#which(x2[,'pos']=='qb')

# which row contains the 12th best quarterback?
#which(x2[,'pos']=='qb')[12]

# which row contains the 90th best quarterback?
#which(x2[,'pos']=='qb')[90]

# 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')


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']


# 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

head(x4)
tail(x4)
```

```
    write.csv(x4,file)

    return(x4)

}
```

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

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

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

```
## 'data.frame':    3 obs. of  4 variables:
##  $ id    : int  1 2 3
##  $ letter: Factor w/ 3 levels "a","b","c": 1 2 3
##  $ value : num  -0.82 0.487 0.738
##  $ even  : logi  FALSE TRUE FALSE
```

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

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

ii. Who is the 15th most valuable running back?

```
#we know they are ordered
x1[x1[,'pos'] == "rb",][15,]
```

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

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

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

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

```
## 'data.frame':    3 obs. of  4 variables:
##  $ id    : int  1 2 3
##  $ letter: Factor w/ 3 levels "a","b","c": 1 2 3
##  $ value : num  0.576 -0.305 1.512
##  $ even  : logi  FALSE TRUE FALSE
```

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

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

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

```
counter <- 0
for(i in 1:40){
  if(x2[i,"pos"] == "wr"){
    counter <- counter + 1
  }
}
counter
```

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

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

```
x3 <- ffvalues('.', 'qbheavy.csv', posReq=c(qb=2, rb=2, wr=3, te=1, k=0), points=c(fg=0, xpt=0, pass_yds
```

```
## 'data.frame':    3 obs. of  4 variables:
##  $ id    : int  1 2 3
##  $ letter: Factor w/ 3 levels "a","b","c": 1 2 3
##  $ value : num  0.39 -0.621 -2.215
##  $ even  : logi  FALSE TRUE FALSE
```

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

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

```
counter <- 0
for(i in 1:30){
  if(x2[i,"pos"] == "qb"){
    counter <- counter + 1
  }
}
counter
```

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

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:

1. Which function has the most arguments?

```
max <- 0
functionName <- ""
for(i in 1:length(funs)){
  if(length((formals(names(funs)[i]))) > max){
   max <- length((formals(names(funs)[i])))
   functionName <- names(funs[i])
  }
}
print(functionName)
```

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

```
print(max)
```

```
## [1] 22
```

So the function scan has the most arguments with 22 arguments.

2. How many functions have no arguments?

```
counter <- 0
for(i in 1:length(funs)){
  if(length(formals(names(funs)[i])) == 0){
    counter <- counter + 1
  }
}
counter
```

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

There are 225 functions without arguments.