Bios 6301: Assignment 4

Elizabeth Sigworth

due November 1, 2016

$5^{n=day}$ 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 $f'(x) = -\sin(x) - 1$.

```
f1 <- function(x) {
  return(cos(x)-x)
}

f2 <- function(x) {
  return(-sin(x)-1)
}

secant.func <- function(x1,tol = 1e-8,max_iter=100) {
  stopifnot(tol > 0, max_iter > 0)
```

```
  iter <- 0
  x0 <- x1 + 1e6
  while(abs(x1-x0) > tol && iter < max_iter) {
    outcome <- x1-(f1(x1)*((x1-x0)/(f1(x1)-f1(x0))))
    x0 <- x1
    x1 <- outcome
    iter <- iter + 1
  }
  if(abs(x1-x0) > tol) {
    x1 <- NULL
    print("Algorithm failed to converge!")
  }
  else {
    print(sprintf('coverges at %s', iter))
  }
  return(x1)
}

newtraph.func <- function(x0,tol = 1e-8,max_iter=100) {
  stopifnot(tol > 0, max_iter > 0)
  x1 <- x0 - f1(x0)/f2(x0)
  iter <- 1
  while(abs(x1-x0) > tol && iter < max_iter) {
    x0 <- x1
    x1 <- x0 - f1(x0)/f2(x0)
    iter <- iter + 1
  }
  if(abs(x1-x0) > tol) {
    x1 <- NULL
    print("Algorithm failed to converge!")
  }
  else {
    print(sprintf('coverges at %s', iter))
  }
  return(x1)
}
```

Now we find the root of the function $f(x) = \cos(x) - x$:

```
secant.func(10)
```

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

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

```
newtraph.func(10)
```

```
## [1] "coverges at 50"
```

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

Based on the above, the root of the function $f(x) = \cos(x) - x$ is 0.7390851.

```
log <- capture.output({
   sec.time <- system.time(replicate(10000, secant.func(10)));
   newtraph.time <- system.time(replicate(10000,newtraph.func(10)))
 })
 print(sec.time)
```

```
##    user  system elapsed
##   1.277   0.021   1.308
```

```
 print(newtraph.time);
```

```
##    user  system elapsed
##   3.421   0.139   3.633
```

The secant method was faster by a factor of 2.7775229.

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.

```
craps <- function() {
  values <- vector()
  initial <- sum(ceiling(6*runif(2)))
  values <- c(values,initial)
  rolls <- 1
  if (initial==7 | initial==1){
    print("Rolls")
    print(values)
    outcomes <- c(1,length(values))
    names(outcomes) <- c("Outcome","Length of Game")
    return(outcomes)
  }
  else {
    roll1 <- sum(ceiling(6*runif(2)))
    values <- c(values,roll1)
    rolls <- 2
    while (roll1 != 7 && roll1 != 11 && roll1 != initial) {
      roll1 <- sum(ceiling(6*runif(2)))
      rolls <- rolls + 1
      values <- c(values,roll1)
    }
    if (roll1 == 7 | roll1 == 11) {
      print("Rolls")
      print(values)
```

```
      outcomes <- c(0,length(values))
      names(outcomes) <- c("Outcome","Length of Game")
      return(outcomes)
    }
    else {
      print("Rolls")
      print(values)
      outcomes <- c(1,length(values))
      names(outcomes) <- c("Outcome","Length of Game")
      return(outcomes)
    }
  }
}
```

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)

*An "Outcome" of 1 indicates a win, and an "Outcome" of 0 indicates a loss.

```
set.seed(100)
craps()
```

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

##        Outcome Length of Game
##              0             14
```

```
craps()
```

```
## [1] "Rolls"
## [1]  6  9  9 11

##        Outcome Length of Game
##              0              4
```

```
craps()
```

```
## [1] "Rolls"
## [1] 6 7

##        Outcome Length of Game
##              0              2
```

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)

```r
#adding argument to disable output
craps <- function(d=1) {
  values <- vector()
  initial <- sum(ceiling(6*runif(2)))
  values <- c(values,initial)
  rolls <- 1
  if (initial==7 | initial==1){
    if(d==1) {
        print("Rolls")
        print(values)
    }
    outcomes <- c(1,length(values))
    names(outcomes) <- c("Outcome","Length of Game")
    return(outcomes)
  }
  else {
    roll1 <- sum(ceiling(6*runif(2)))
    values <- c(values,roll1)
    rolls <- 2
    while (roll1 != 7 && roll1 != 11 && roll1 != initial) {
      roll1 <- sum(ceiling(6*runif(2)))
      rolls <- rolls + 1
      values <- c(values,roll1)
    }
    if (roll1 == 7 | roll1 == 11) {
      if(d==1) {
        print("Rolls")
        print(values)
      }
      outcomes <- c(0,length(values))
      names(outcomes) <- c("Outcome","Length of Game")
      return(outcomes)
    }
    else {
      if(d==1) {
        print("Rolls")
        print(values)
      }
      outcomes <- c(1,length(values))
      names(outcomes) <- c("Outcome","Length of Game")
      return(outcomes)
    }
  }
}
```

```r
seeds <- rep(0,5000)
for (i in 1:5000) {
  set.seed(i)
  a <- craps(0)[1]
  b <- craps(0)[1]
  c <- craps(0)[1]
  d <- craps(0)[1]
  e <- craps(0)[1]
```

```
  f <- craps(0)[1]
  g <- craps(0)[1]
  h <- craps(0)[1]
  j <- craps(0)[1]
  k <- craps(0)[1]
  if (a+b+c+d+e+f+g+h+j+k == 10) {
    seeds[i] <- 1
  }
  else {
    seeds[i] <- 0
  }
}

which(seeds==1)
```

```
## [1] 4352
```

From this loop, I have found that the seed 4352 will give me ten wins in a row. I can verify this by setting the seed to 4352 and running the craps() function 10 times:

```
set.seed(4352)
craps()
```

```
## [1] "Rolls"
## [1]  6 12  3  5  6

##        Outcome Length of Game
##              1              5
```

```
craps()
```

```
## [1] "Rolls"
## [1] 5 5

##        Outcome Length of Game
##              1              2
```

```
craps()
```

```
## [1] "Rolls"
## [1] 5 9 5

##        Outcome Length of Game
##              1              3
```

```
craps()
```

```
## [1] "Rolls"
## [1] 7
```

```
##        Outcome Length of Game
##              1              1
```

```
craps()
```

```
## [1] "Rolls"
## [1] 3 5 5 3

##        Outcome Length of Game
##              1              4
```

```
craps()
```

```
## [1] "Rolls"
## [1] 7

##        Outcome Length of Game
##              1              1
```

```
craps()
```

```
## [1] "Rolls"
## [1] 7

##        Outcome Length of Game
##              1              1
```

```
craps()
```

```
## [1] "Rolls"
## [1] 6 6

##        Outcome Length of Game
##              1              2
```

```
craps()
```

```
## [1] "Rolls"
## [1] 6 6

##        Outcome Length of Game
##              1              2
```

```
craps()
```

```
## [1] "Rolls"
## [1] 6 5 9 6

##        Outcome Length of Game
##              1              4
```

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 ordered 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):

```
install.packages('dplyr', repos="http://cran.rstudio.com/")
```

```
##
## The downloaded binary packages are in
##  /var/folders/kp/zlsf12h14y92__lp668ljv2m0000gn/T//Rtmpa6gEDt/downloaded_packages
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
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)) {
  positions <- c('k','qb','rb','te','wr')
  csvfile <- paste('proj_', positions, '16', '.csv', sep='')
  files <- file.path(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)
  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')
  x2[k.ix, 'marg'] <- x2[k.ix,'points'] -
    ifelse(length(x2[k.ix[posReq['k']*nTeams], 'points'])==0,0, x2[k.ix[posReq['k']*nTeams],'points'])
  x2[qb.ix, 'marg'] <- x2[qb.ix,'points'] - x2[qb.ix[posReq['qb']*nTeams],'points']
  x2[rb.ix, 'marg'] <- x2[rb.ix,'points'] - x2[rb.ix[posReq['rb']*nTeams],'points']
  x2[te.ix, 'marg'] <- x2[te.ix,'points'] - x2[te.ix[posReq['te']*nTeams],'points']
  x2[wr.ix, 'marg'] <- x2[wr.ix,'points'] - x2[wr.ix[posReq['wr']*nTeams],'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
  x4 <- x3[,c('PlayerName','pos','points','value')]
  write.csv(x4,file)
  return(x4)
}
```

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

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

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

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

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

There are 46 players worth more than $20.

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

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

```
##      PlayerName pos points    value
## 157 Carlos Hyde  rb 145.47 19.76574
```

```
The 15th most valuable running back is Carlos Hyde.
```

2. 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
```

```
There are 49 players worth more than $20.
```

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

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

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

```
There are 18 wide receivers in the top 40.
```

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))
```

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

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

```
## [1] 50
```

```
There are 50 players worth more than $20.
```

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

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

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

```
There are 10 quarterbacks in the top 30.
```

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)

```
#Using the fact that the arguments are defined as the "formals" of a function
which.max(sapply(funs,function(x) length(formals(x))))
```

```
## scan
##  936
```

The scan function has the most arguments.

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

```
length(which(sapply(funs,function(x) length(formals(x))==0)))
```

```
## [1] 223
```

There are 223 functions in the base package that have no arguments.

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