

# Bios 6301: Assignment 4

Valerie Welty

Due Tuesday, 01 November, 1:00 PM

$5^{n=\text{day}}$  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  $x_2$  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  $f'(x) = -\sin(x) - 1$ .

```
secant <- function(guess1, guess2, f, tol=10e-7, maxiter=1000) {
  x <- vector(mode="numeric", length=maxiter)
  i <- 2
  x[1]=guess1; x[2]=guess2
  while(abs(f(x[i])) > tol && i < maxiter) {
    x[i+1] = x[i] - f(x[i]) * (x[i]-x[i-1])/(f(x[i])-f(x[i-1]))
    i <- i + 1
  }
  if(i == maxiter) {
    print('failed to converge')
```

```

    return(NULL)
  } else {
    ## only want the print if failed
  }
}

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 {
    ## only want the print if failed
  }
}

```

```

f <- function(x) cos(x) - x
fp <- function(x) -sin(x) - 1

secant.time <- system.time(replicate(1e2, secant(10,11,f)))
newton.time <- system.time(replicate(1e2, newton(10,f,fp)))
secant.time; newton.time

```

```

##      user  system elapsed
##  0.006   0.001   0.007

```

```

##      user  system elapsed
##  0.015   0.001   0.016

```

```

set.seed(23456)
reps <- c(1e1,1e2,1e3,1e4,1e5)
times <- matrix(data=NA, nrow=length(reps), ncol=3)
f <- function(x) cos(x) - x
fp <- function(x) -sin(x) - 1

for(i in seq_along(reps)) {
  times[i,1] <- system.time(replicate(reps[i], secant(10,11,f)))[3]
  times[i,2] <- system.time(replicate(reps[i], newton(10,f,fp)))[3]
  # secant.time; newton.time
  times[i,3] <- times[i,2]/times[i,1]
}
times

```

```

##      [,1]  [,2]  [,3]
## [1,] 0.001  0.002 2.000000
## [2,] 0.004  0.012 3.000000

```

```
## [3,] 0.051 0.134 2.627451
## [4,] 0.501 1.337 2.668663
## [5,] 4.832 13.604 2.815397
```

The secant method is markedly faster; on average, over multiple increments of replication times (10, 100, 1000,  $10^4$ ,  $10^5$ ), the Newton-Raphson method takes about 2.6223022 times as long as the Secant method to run.

## 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.

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 <- function(numgames) {
  # store result of win or loss in vector 'result'
  result <- vector(mode = "numeric", length = numgames)
  for (i in 1:numgames) {
    # x will be the target value (value of first roll if not 7 or 11)
    x <- 0
    # first roll
    roll.1 <- sum(ceiling(6 * runif(2)))
    # beginning of a vector which holds all the rolls of a game
    rolls.rec <- roll.1
    # set win to -1 if neither win nor lose yet, to use in the 'while' argument
    # a win is win=1 and a loss is win=0
    if ((roll.1 == 7) | (roll.1 == 11)) {
      win <- 1
    } else {
      win <- -1
      x = roll.1
    }
    # keep rolling until we either win or lose
    while (win == -1) {
      roll <- sum(ceiling(6 * runif(2)))
      # add this roll to the roll record
      rolls.rec = c(rolls.rec, roll)
      # if the roll is our target value 'x' we win, if the roll is 7 or 11 we
      # lose, else keep rolling
      if (roll == x) {
        win = 1
      } else if ((roll == 7) | (roll == 11)) {
        win = 0
      } else {
        win == -1
      }
    }
  }
}
```

```

    }
    print(sprintf("game %s rolls:", i))
    print(rolls.rec)
    print(sprintf("win game? %s", as.logical(win)))
    # store result of each game in 'result'
    result[i] = win
  }
  print(sprintf("total number of wins: %s", sum(result)))
  # important output value is the # of games won out of 'numgames'
  return(sum(result))
}

```

```

set.seed(101)
craps(3)

```

```

## [1] "game 1 rolls:"
## [1] 4 9 4
## [1] "win game? TRUE"
## [1] "game 2 rolls:"
## [1] 7
## [1] "win game? TRUE"
## [1] "game 3 rolls:"
## [1] 8 11
## [1] "win game? FALSE"
## [1] "total number of wins: 2"

## [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)

```

for (i in 1000:5000) {
  set.seed(i)
  capture.output(x <- craps(10))
  # print(x[1])
  if (x[1]==10) {break}
}
i

```

```

## [1] 1639

```

```

set.seed(i)
craps(10)

```

```

## [1] "game 1 rolls:"
## [1] 5 10 6 6 5
## [1] "win game? TRUE"
## [1] "game 2 rolls:"
## [1] 6 6
## [1] "win game? TRUE"
## [1] "game 3 rolls:"

```

```
## [1] 7
## [1] "win game? TRUE"
## [1] "game 4 rolls:"
## [1] 7
## [1] "win game? TRUE"
## [1] "game 5 rolls:"
## [1] 5 2 4 8 5
## [1] "win game? TRUE"
## [1] "game 6 rolls:"
## [1] 7
## [1] "win game? TRUE"
## [1] "game 7 rolls:"
## [1] 7
## [1] "win game? TRUE"
## [1] "game 8 rolls:"
## [1] 9 5 6 6 6 9
## [1] "win game? TRUE"
## [1] "game 9 rolls:"
## [1] 6 8 8 9 6
## [1] "win game? TRUE"
## [1] "game 10 rolls:"
## [1] 11
## [1] "win game? TRUE"
## [1] "total number of wins: 10"

## [1] 10
```

(solution for disabling output found on <http://stackoverflow.com/questions/8797314/suppress-messages-displayed-by-print-instead-of-message-or-warning-in-r>)

### 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):

```
# 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

path <- "~/examples/homework"

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
k <- read.csv(paste(path, "/proj_k16.csv", sep = ""))
qb <- read.csv(paste(path, "/proj_qb16.csv", sep = ""))
rb <- read.csv(paste(path, "/proj_rb16.csv", sep = ""))
te <- read.csv(paste(path, "/proj_te16.csv", sep = ""))
wr <- read.csv(paste(path, "/proj_wr16.csv", sep = ""))
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])

## calculate dollar values
met <- c("fg", "xpt", "pass_yds", "pass_tds", "pass_ints", "rush_yds", "rush_tds",
        "fumbles", "rec_yds", "rec_tds")
for (i in seq_along(met)) {
  x[, paste("p_", met[i])] <- x[, met[i]] * points[i]
}
x[, "points"] <- rowSums(x[, grep("^p_", names(x))])

x2 <- x[order(x[, "points"], decreasing = TRUE), ]
k.ix <- which(x2[, "pos"] == "k" & x2[, "points"] != 0)
k.ix
qb.ix <- which(x2[, "pos"] == "qb" & x2[, "points"] != 0)
qb.ix
rb.ix <- which(x2[, "pos"] == "rb" & x2[, "points"] != 0)
te.ix <- which(x2[, "pos"] == "te" & x2[, "points"] != 0)
wr.ix <- which(x2[, "pos"] == "wr" & x2[, "points"] != 0)

x2[k.ix, "marg"] <- x2[k.ix, "points"] - x2[k.ix[nTeams * posReq["k"]],
        "points"]

x2[qb.ix, "marg"] <- x2[qb.ix, "points"] - x2[qb.ix[nTeams * posReq["qb"]],
        "points"]
x2[rb.ix, "marg"] <- x2[rb.ix, "points"] - x2[rb.ix[nTeams * posReq["rb"]],
        "points"]
x2[te.ix, "marg"] <- x2[te.ix, "points"] - x2[te.ix[nTeams * posReq["te"]],
        "points"]
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), ]
rownames(x3) <- NULL

x4 <- na.omit(x3)

x4[, "value"] <- (nTeams * cap - nTeams * sum(posReq)) * x4[, "marg"]/sum(x4[,
  "marg"]) + 1

## save dollar values as CSV file
val <- x4[, c("PlayerName", "pos", "points", "value")]
write.csv(val, file, row.names = FALSE)
## return data.frame with dollar values
x5 <- read.csv(file)
return(x5)
}

```

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

```

path <- "~/examples/homework"
x_1 <- ffvalues(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))

```

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

```
length(which(x_1[, 'value'] > 20))
```

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

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

```
x_1[which(x_1[, 'pos'] == 'rb')[15], ][1]
```

```
##      PlayerName
## 47 Carlos Hyde
```

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

```

x_2 <- ffvalues(getwd(), file = "16team.csv", nTeams = 16, cap = 150, 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))

```

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

```
length(which(x_2[, 'value'] > 20))
```

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

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

```
length(which(x_2[1:40, ], 'pos' == 'wr'))
```

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

1. Call:

```
x_3 <- fvalues(getwd(), "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)

```
length(which(x_3[, 'value'] > 20))
```

```
## [1] 51
```

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

```
length(which(x_3[1:30, ], 'pos' == 'qb'))
```

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

## 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)

```
str(funs, max.level = 1, list.len = 6)
```

```
## List of 1193
## $ - :function (e1, e2)
## $ -.Date :function (e1, e2)
## $ -.POSIXt :function (e1, e2)
## $ : :.Primitive(":")
## $ :: :function (pkg, name)
## $ ::: :function (pkg, name)
## [list output truncated]
```



```
num_arg <- sapply(funs, function(x) length(formals(x)))
max <- which.max(num_arg)
(which <- names(funs)[max])
```

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

```
(num <- length(formals(funs[[max]])))
```

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

Thus the function scan has the most arguments at 22.

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

```
table(num_arg, useNA = "always")
```

```
## num_arg
##      0      1      2      3      4      5      6      7      8      9     10     12     14     16     22
## 225  196  349  188   91   82   31   12    8    3    2    3    1    1    1
## <NA>
##      0
```

```
length(no_args <- which(num_arg == 0))[1]
```

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

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