

# Computing Assignment 4

Angela Huynh

10/18/2016

Grade: 42/50

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

```
newton <- function(guess, f, fp, tol = 1e-06, 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 {
    print(sprintf("converges at %s", i))
  }
  guess
}
f <- function(x) cos(x) - x
nr <- function(x) -sin(x) - 1
newton(10, f, nr)
```

```
## [1] "converges at 49"
```

```
## [1] 0.7390852
```

It took 2 seconds.

## JC Grading - 6

Hey Angela, nice job on the newton-rhapson. Maybe I'm missing it, but I'm not seeing the implementation of the secant function.

## 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. You can use the following snippet of code to simulate the roll of two (fair) dice:

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

- a) 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.

```
set.seed(100)
# Create craps function
craps <- function() {
  roll <- TRUE
  firstRoll <- TRUE
  point <- 0
  while (roll == TRUE) {

    if (firstRoll == TRUE) {
      result <- sum(ceiling(6 * runif(2)))
      if (result == 7 | result == 11) {
        print(result)
        print(paste("You win"))
        roll <- FALSE
      } else {
        print(result)
        point <- result
        firstRoll <- FALSE
        roll <- TRUE
      }
    } else {
      result <- sum(ceiling(6 * runif(2)))
      if (result == 7 | result == 11) {
        print(result)
        print(paste("You lose"))
        roll <- FALSE
      } else {
        if (result == point) {
          print(result)
          print(paste("You win!"))
        }
      }
    }
  }
}
```

```

        roll <- FALSE
      } else {
        print(result)
      }
    }
  }
}

for (i in seq(1:3)) {
  craps()
}

```

```

## [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] "You lose"
## [1] 6
## [1] 9
## [1] 9
## [1] 11
## [1] "You lose"
## [1] 6
## [1] 7
## [1] "You lose"

```

- b) 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.

```

craps.10 <- function() {
  roll <- TRUE
  firstRoll <- TRUE
  point <- 0
  win <- 0
  while (roll == TRUE) {
    if (firstRoll == TRUE) {
      result <- sum(ceiling(6 * runif(2)))
      if (result == 7 | result == 11) {
        roll <- FALSE
        win <- 1
      } else {
        point <- result
        firstRoll <- FALSE
        roll <- TRUE
      }
    }
  }
}

```

```

        win <- 0
      }
    } else {
      result <- sum(ceiling(6 * runif(2)))
      if (result == 7 | result == 11) {
        roll <- FALSE
        win <- 0
      } else {
        if (result == point) {
          roll <- FALSE
          win <- 1
        } else {
          win <- 0
          roll <- TRUE
        }
      }
    }
  }
}
return(win)
}

```

*# run craps 10 times*

```

run.craps <- function() {
  runs <- vector()
  for (i in seq(1:10)) {
    runs[i] <- craps.10()
  }
  sum.runs <- sum(runs)
  return(sum.runs)
}

```

*# finding seed that gives you 10 straight wins*

```

sample.runs <- vector()
for (i in seq(1:1000)) {
  set.seed(i)
  sample.runs[i] <- run.craps()
  if (sample.runs[i] == 10) {
    print(i)
    (break)()
  }
}

```

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

Seed 880 is the first one that gives me 10 straight wins!

## 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  $\text{sum}(\text{posReq}) * n\text{Teams}$  rows.

```
# 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 <- setwd('~Documents/Vanderbilt 1/Semester 1/BIOS 6301/Assignments')
path <- "."
# Create function fvalues
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 = ""))

  # generate unique list of column names
  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"

  # 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])

  # calculate new columns convert NFL stat to fantasy points
  x[, "p_fg"] <- x[, "fg"] * 4
  x[, "p_xpt"] <- x[, "xpt"] * 1
  x[, "p_pass_yds"] <- x[, "pass_yds"]/25
  x[, "p_pass_tds"] <- x[, "pass_tds"] * 4
  x[, "p_pass_ints"] <- x[, "pass_ints"] * -2
  x[, "p_rush_yds"] <- x[, "rush_yds"]/10
  x[, "p_rush_tds"] <- x[, "rush_tds"] * 6
```

```

x[, "p_fumbles"] <- x[, "fumbles"] * -2
x[, "p_rec_yds"] <- x[, "rec_yds"]/20
x[, "p_rec_tds"] <- x[, "rec_tds"] * 6

# sum selected column values for every row this is total fantasy points for
# each player
x[, "points"] <- rowSums(x[, grep("^p_", names(x))])

# calculate dollar values

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

# determine the row indeces for each position
k.ix <- which(x2[, "pos"] == "k" & x2[, "points"] != 0)
qb.ix <- which(x2[, "pos"] == "qb" & x2[, "points"] != 0)
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)

# calculate marginal points by subtracting 'baseline' player's points

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[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"]

# 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

x4 <- na.omit(x3)

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

x5 <- x4[, c("PlayerName", "pos", "points", "marg", "value")]
write.csv(x5, file, row.names = FALSE)
x6 <- read.csv(file)
return(x6)
}

```

1. Call

```
# path <- setwd('~/Documents/Vanderbilt 1/Semester 1/BIOS 6301/Assignments')
path <- "."
x1 <- ffvalues(path)
```

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

```
(length(which(x1[, "value"] > 20)))
```

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

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

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

```
##      PlayerName pos points   marg   value
## 47 Carlos Hyde   rb 145.47 18.145 19.76574
```

2. Call

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

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

```
(length(which(x2[, "value"] > 20)))
```

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

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

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

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

3. Call:

```
# path <- setwd('~/Documents/Vanderbilt 1/Semester 1/BIOS 6301/Assignments')
path <- "."

x3 <- ffvalues(path, "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?

```
(length(which(x3[, "value"] > 20)))
```

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

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

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

```
## [1] 0
```

**JC Grading -2** The answer to the two questions above are 51 and 10. Check out where your code starts like: # convert NFL stat to fantasy points. Rather than hard code the conversion, use the “points” input in the function.

## 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?

```
require(plyr)
```

```
## Loading required package: plyr
# create a vector of lengths
arg_length <- lapply(funs, function(x) (length(formals(x))))

# find the max
max_args <- which(arg_length == max(arg_length))

# get the name of the function
names(funs[max_args])
```

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

2. How many functions have no arguments? Hint: find a function that returns the arguments for a given function.

```
(length(which(arg_length == 0)))
```

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