BIOS 6301: Assignment 5

Charles Rhea

39/40 Good!
Please put answers on their own lines rather than in the questions

Due Thursday, 12 October, 1:00 PM

$5^{n=day}$ points taken off for each day late.

40 points total.

Submit a single knitr file (named `homework5.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 `homework5.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:

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

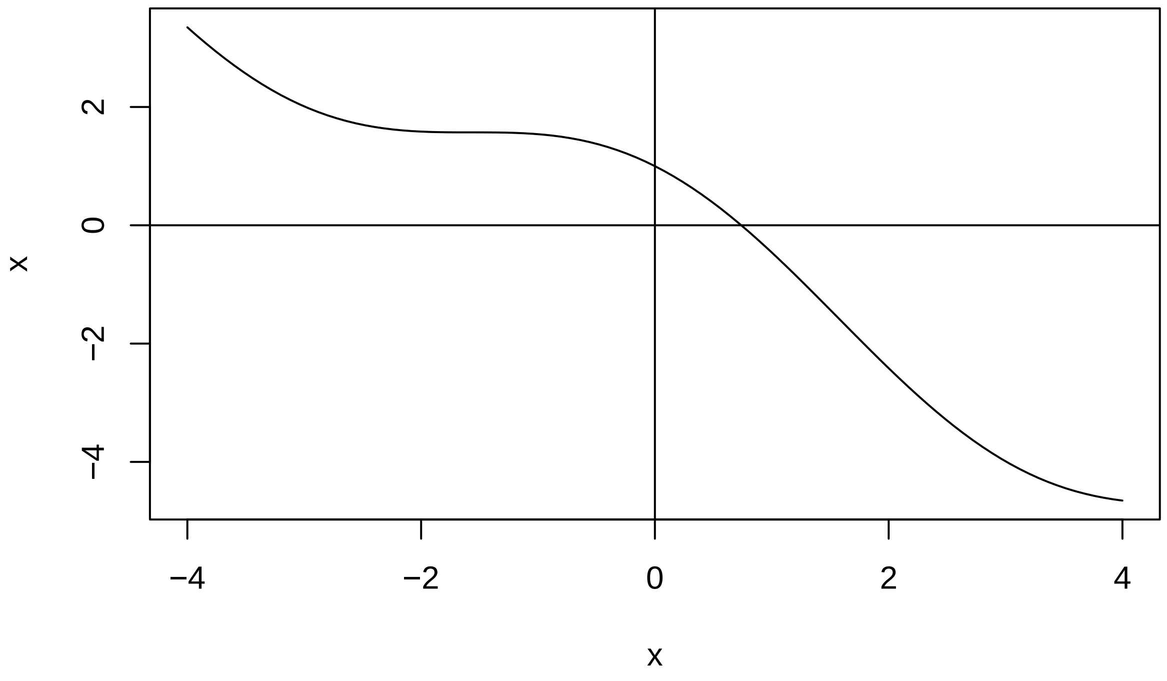
Notice that we no longer need to know f' but in return we have to provide *two* initial points, x_0 and x_1.

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 $f'(x) = -\sin(x) - 1$. #Using the secant method the root is 0.725. Using the Newton-Raphson method the root is 0.739. Both methods ran very quickly (maybe 1-2 second) and were not drastically different from on another.#

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

#Plotting the function to see where the root
plot(x, xlim = c(-4,4))
```

```
abline(h=0)
abline(v=0)
```



```
#The root is between 0.5 and 1

#Using Secent to find root of f(x)
#Define 2 guesses for 'x' based on our guesses for the root
x_1 = 1
x_0 = 0.5

#Iterate Secant method
secant.start <- Sys.time()
while(abs(x(x_1) - x(x_0))> 1e-8){
 x_2 = x_1 - (x(x_1)*((x_1 - x_0)/(x(x_1) -x(x_0))))
 x_1 = x_2
 x_0 = x_1
}
secant.end <- Sys.time()
x_2
```

need to change order of these two. you just set x2 = x1 = x0 instead of x1 and x0 being different

```
## [1] 0.7254816
```

```
#Using Newton Raphson method TO FIND THE ROOT OF f(x)
xprime = function(x) -sin(x) -1
```

```r
#Define 1 guess for 'x'
x_old = 0.5

#Iterate Newton Raphson method until the difference between f(x_old) is less than 0.00000001
newton.starttime <- Sys.time()
while(abs(x(x_old))> 1e-8){
  x_new = x_old - x(x_old)/xprime(x_old)
  x_old = x_new
}
newton.endtime <- Sys.time()
x_new
```

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

Question 2

20 points

The game of craps is played as follows (this is simplified). 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:

```r
craps_game <- function() {
  first_roll <- sum(ceiling(6*runif(2)))
  if (first_roll == 7 | first_roll == 11) print(paste("First Role Win"))
  else {
  second_roll <- sum(ceiling(6*runif(2)))
  while (second_roll != first_roll & second_roll != 7 & second_roll != 11)
  {
  second_roll <- sum(ceiling(6*runif(2)))
  }
  if (second_roll == first_roll) print("Equal Roll Win")
  else {print("Lose")
  }}
}

craps_game()
```

```
## [1] "Equal Roll Win"
```

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)

```r
set.seed(100)
craps_game <- function(ngames = 1) {
  for (i in 1:ngames){
    roll_num = 1
    first_roll <- sum(ceiling(6*runif(2)))
    if (first_roll == 7 | first_roll == 11) {print(paste("Game #", i, "First Role Win =", first_roll))
    i = i+1 }
    else {
      roll_num = roll_num + 1
      roll_sum <- sum(ceiling(6*runif(2)))
      while (roll_sum != first_roll & roll_sum != 7 & roll_sum != 11) {
        roll_num = roll_num + 1
        roll_sum <- sum(ceiling(6*runif(2)))
        }
        if (first_roll == roll_sum) {print(paste("Game #", i, "You Win After", roll_sum, "Rolls =", rol
        i= i+1 }
      else {
        print(paste("Game #", i, "You Lose After", roll_sum, "Rolls =", roll_sum))
        i= i+1 }
  }}
}

craps_game(ngames = 3)
```

```
## [1] "Game # 1 You Lose After 11 Rolls = 11"
## [1] "Game # 2 You Lose After 11 Rolls = 11"
## [1] "Game # 3 You Lose After 7 Rolls = 7"
```

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. (7 points)

```
craps_wins = function(seed = 1, tries = 1, show = 0) {
  wins = 0
  set.seed(seed)
  for (i in 1:tries) {
    number_roll = 1
    roll1_sum = sum(ceiling(6 * runif(2)))
    if (roll1_sum %in% c(7, 11)) {
      if (show == 1) {
        print(paste("Game #", i, ": You Won After", number_roll, "Roll =", "\n"))
      }
      i = i + 1
      wins = wins + 1
    }
    else {
      number_roll = number_roll + 1
      roll2_sum = sum(ceiling(6 * runif(2)))
      while (roll2_sum != roll1_sum & roll2_sum != 7 & roll2_sum != 11) {
        number_roll = number_roll + 1
        roll2_sum = sum(ceiling(6 * runif(2)))
      }
      if (roll1_sum == roll2_sum) {
        if (show == 1) {
          print(paste("Game #", i, ": You Won After", number_roll, "Rolls =", "\n"))
        }
        i = i + 1
        wins = wins + 1
      }
```

```r
      else {
        if (show == 1) {
          print(paste("Game #", i,": You Lost After", number_roll,"Rolls =", "\n"))
        }
        i = i + 1
        wins = wins
      }
    }
  }
  if (show == 1) {
    print(paste("Using Seed =",seed,", You Won", wins, "Out of", tries, "Games.","\n"))
  }
  else {
    wins = return(wins)
  }
}

startseed = 100
results = craps_wins(startseed, 10)
while (results != 10) {
  startseed = startseed + 1
  results = craps_wins(startseed, 10)
}
cat("Seed =",startseed,"Returns 10 Consecutive Wins.","\n")
```

```
## Seed = 880 Returns 10 Consecutive Wins.
```

```r
craps_wins(startseed, 10, 1)
```

```
## [1] "Game # 1 : You Won After 1 Roll = \n"
## [1] "Game # 2 : You Won After 6 Rolls = \n"
## [1] "Game # 3 : You Won After 2 Rolls = \n"
## [1] "Game # 4 : You Won After 2 Rolls = \n"
## [1] "Game # 5 : You Won After 1 Roll = \n"
## [1] "Game # 6 : You Won After 2 Rolls = \n"
## [1] "Game # 7 : You Won After 2 Rolls = \n"
## [1] "Game # 8 : You Won After 1 Roll = \n"
## [1] "Game # 9 : You Won After 2 Rolls = \n"
## [1] "Game # 10 : You Won After 1 Roll = \n"
## [1] "Using Seed = 880 , You Won 10 Out of 10 Games. \n"
```

Question 3

5 points

This code makes a list of all functions in the base package:

```r
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) #The ‘scan’ function has the most arguement = 983#

```r
argCount = function(x){
  length(formalArgs(x))
}
#which.max(sapply(names(funs),argCount))
```

2. How many functions have no arguments? (2 points) #There are 229 functions with no arguements#

```r
#argList = sapply(names(funs),argCount)
#sum(argList == 0)
```

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