Bios 6301: Assignment 5

Jeongwon Choi

Due Thursday, 13 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 $f'(x) = -\sin(x) - 1$.

```
secant.method <- function(f, x0, x1, tol = 1e-100, n = 10000) {
  for (i in 1:n) {
    x2 <- x1 - f(x1) / ((f(x1) - f(x0)) / (x1 - x0)) # Calculate the new x value
    if (abs(x2 - x1) < tol) { # If the difference between the
      #new value and the previous value is small enough, end iteration and output root.
      x_root <- x2
```

```
      break
      }
    # If the root was not determined in the previous iteration,
    # update the values and proceed to the next iteration.
    x0 <- x1
    x1 <- x2
    return(x2)
  }

  ##### If there are no case that makes abs(x2-x1)<tol true in the loop.
  if (i==n){
    x_root <- x2
    print('Convergence was failed.')
  }
  return(x_root)
}

func <- function(x) {
  cos(x)-x
}
secant.method(func, 1, 3)
```

```
## [1] 0.7395698
```

```
newton <- function(f, delta = 0.0000001, x_0 = 2, n=1000){
  h = 0.0000001
  i = 1; x1 = x_0
  p = numeric(n)
  while (i <= n) {
    df.dx = (f(x_0 + h) - f(x_0)) / h
    x1 = (x_0 - (f(x_0) / df.dx))
    p[i] = x1
    i = i+1
    if (abs(x1 - x_0) < delta) break
    x_0 = x1
  }
  return(x1)
}
newton(func)
```

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

```
root.secant = secant.method(func,1,3)
root.newton = newton(func)
func(root.secant)
```

```
## [1] -0.0008112277
```

```
func(root.newton)
```

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

```
startTime1 <- Sys.time()
secant.method(func, 1, 3)
```

```
## [1] 0.7395698
```

```
endTime1 <- Sys.time()
s_time <- print(endTime1 - startTime1)
```

```
## Time difference of 0.0007469654 secs
```

```
startTime2 <- Sys.time()
newton(func)
```

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

```
endTime2 <- Sys.time()
n_time <- print(endTime2 - startTime2)
```

```
## Time difference of 0.000579834 secs
```

```
s_time - n_time
```

```
## Time difference of 0.0001671314 secs
```

Newton method is faster in this case. However, in many cases, because the secant method needs one function while newton's method needs two functions per iteration, newton method takes more computation time than the secant method.

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:

```
    x <- 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)

```r
set.seed(100)

craps_testing <- function(verbose=TRUE,return_result=TRUE) {
  # verbose: if verbose is TRUE, print all the outputs of the games
  # return_result: if return_result is TRUE, this function
  # returns a result of the game (1: win, 0: lose)

  if (verbose){ print('<<Game Start!>>') }
  roll_1 <- sum(ceiling(6*runif(2)))

  counter = 1 # the current trial of the game

  if (verbose) { print(paste0('Trial ',as.character(counter),'----------')) }

  if (roll_1 == 7 | roll_1 == 11) {
    if (verbose) { print( paste0("You rolled a ", roll_1, ". You win!"))  }
    result = 1
    } else {

      if (verbose) { print( paste0("You rolled a ", roll_1, ". Keep going...")) }

      counter = counter+1
      if (verbose) { print(paste0('Trial ',as.character(counter),'----------')) }
      dice_roll <- sum(ceiling(6*runif(2)))

      while (TRUE) {
        # if you get x again, you win the game
        if (dice_roll == roll_1){
          if (verbose) { print( paste0('You rolled a ', dice_roll,
            ". Your roll matched your first roll. You win!") ) } # win the game
          result = 1 # win!
          break
        }
        # if you get 7 or 11, you lose the game
        if ( (dice_roll == 7 )|(dice_roll ==11) ) {
     if (verbose) { print(paste0("You rolled a ", dice_roll, ". You lose.")) }
          result = 0 # lose!
          break
        }
 if (verbose) { print( paste0('You rolled a ', dice_roll, ". Keep going...")) }
        dice_roll <- sum(ceiling(6*runif(2)))
    counter = counter+1
    if (verbose) { print(paste0('Trial ',as.character(counter),'----------')) }
      }
    }

  if (return_result){ return(result) } # return results
  #(either 0 or 1, depending on the game result) only if return_result is true.

  } # end of the function
```

```r
set.seed(100)
for (i in 1:3){
```

```
 craps_testing(verbose=TRUE,return_result=TRUE)
}
```

```
## [1] "<<Game Start!>>"
## [1] "Trial 1----------"
## [1] "You rolled a 4. Keep going..."
## [1] "Trial 2----------"
## [1] "You rolled a 5. Keep going..."
## [1] "Trial 3----------"
## [1] "You rolled a 6. Keep going..."
## [1] "Trial 4----------"
## [1] "You rolled a 8. Keep going..."
## [1] "Trial 5----------"
## [1] "You rolled a 6. Keep going..."
## [1] "Trial 6----------"
## [1] "You rolled a 10. Keep going..."
## [1] "Trial 7----------"
## [1] "You rolled a 5. Keep going..."
## [1] "Trial 8----------"
## [1] "You rolled a 10. Keep going..."
## [1] "Trial 9----------"
## [1] "You rolled a 5. Keep going..."
## [1] "Trial 10----------"
## [1] "You rolled a 8. Keep going..."
## [1] "Trial 11----------"
## [1] "You rolled a 9. Keep going..."
## [1] "Trial 12----------"
## [1] "You rolled a 9. Keep going..."
## [1] "Trial 13----------"
## [1] "You rolled a 5. Keep going..."
## [1] "Trial 14----------"
## [1] "You rolled a 11. You lose."
## [1] "<<Game Start!>>"
## [1] "Trial 1----------"
## [1] "You rolled a 6. Keep going..."
## [1] "Trial 2----------"
## [1] "You rolled a 9. Keep going..."
## [1] "Trial 3----------"
## [1] "You rolled a 9. Keep going..."
## [1] "Trial 4----------"
## [1] "You rolled a 11. You lose."
## [1] "<<Game Start!>>"
## [1] "Trial 1----------"
## [1] "You rolled a 6. Keep going..."
## [1] "Trial 2----------"
## [1] "You rolled a 7. You lose."
```

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. (7 points)

```
n_max = 10000 # from 1 to n_max
for (i in 1:n_max){
```

```
  # set random seed
  set.seed(i)

  # run 10 times of games (without printing outputs)
  res <- replicate(craps_testing(verbose=FALSE, return_result=TRUE),n=10)

  if (sum(res)==10) {
    print('The random seed that makes 10 consecutive won game was found!')
    print(i)
  }
}
```

```
## [1] "The random seed that makes 10 consecutive won game was found!"
## [1] 880
## [1] "The random seed that makes 10 consecutive won game was found!"
## [1] 1639
## [1] "The random seed that makes 10 consecutive won game was found!"
## [1] 4352
## [1] "The random seed that makes 10 consecutive won game was found!"
## [1] 4411
## [1] "The random seed that makes 10 consecutive won game was found!"
## [1] 8839
## [1] "The random seed that makes 10 consecutive won game was found!"
## [1] 9085
```

```
### Based on the result from above, confirm that it works well.
set.seed(880)

for (i in 1:10){
  print(paste0('----- Game repetition: ',as.character(i)))
  a<-craps_testing(verbose=TRUE, return_result=TRUE)
}
```

```
## [1] "----- Game repetition: 1"
## [1] "<<Game Start!>>"
## [1] "Trial 1----------"
## [1] "You rolled a 7. You win!"
## [1] "----- Game repetition: 2"
## [1] "<<Game Start!>>"
## [1] "Trial 1----------"
## [1] "You rolled a 8. Keep going..."
## [1] "Trial 2----------"
## [1] "You rolled a 9. Keep going..."
## [1] "Trial 3----------"
## [1] "You rolled a 3. Keep going..."
## [1] "Trial 4----------"
## [1] "You rolled a 10. Keep going..."
## [1] "Trial 5----------"
## [1] "You rolled a 6. Keep going..."
## [1] "Trial 6----------"
## [1] "You rolled a 8. Your roll matched your first roll. You win!"
## [1] "----- Game repetition: 3"
## [1] "<<Game Start!>>"
```

```
## [1] "Trial 1----------"
## [1] "You rolled a 10. Keep going..."
## [1] "Trial 2----------"
## [1] "You rolled a 10. Your roll matched your first roll. You win!"
## [1] "----- Game repetition: 4"
## [1] "<<Game Start!>>"
## [1] "Trial 1----------"
## [1] "You rolled a 9. Keep going..."
## [1] "Trial 2----------"
## [1] "You rolled a 9. Your roll matched your first roll. You win!"
## [1] "----- Game repetition: 5"
## [1] "<<Game Start!>>"
## [1] "Trial 1----------"
## [1] "You rolled a 11. You win!"
## [1] "----- Game repetition: 6"
## [1] "<<Game Start!>>"
## [1] "Trial 1----------"
## [1] "You rolled a 8. Keep going..."
## [1] "Trial 2----------"
## [1] "You rolled a 8. Your roll matched your first roll. You win!"
## [1] "----- Game repetition: 7"
## [1] "<<Game Start!>>"
## [1] "Trial 1----------"
## [1] "You rolled a 5. Keep going..."
## [1] "Trial 2----------"
## [1] "You rolled a 5. Your roll matched your first roll. You win!"
## [1] "----- Game repetition: 8"
## [1] "<<Game Start!>>"
## [1] "Trial 1----------"
## [1] "You rolled a 7. You win!"
## [1] "----- Game repetition: 9"
## [1] "<<Game Start!>>"
## [1] "Trial 1----------"
## [1] "You rolled a 9. Keep going..."
## [1] "Trial 2----------"
## [1] "You rolled a 9. Your roll matched your first roll. You win!"
## [1] "----- Game repetition: 10"
## [1] "<<Game Start!>>"
## [1] "Trial 1----------"
## [1] "You rolled a 7. You win!"
```

```
#Can also be done with
# replicate(craps_testing(verbose=TRUE,return_result=FALSE),n=10)
```

Question 3

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)

```
arglength <- sapply(funs, function(x) length(formals(x)))
arglength_sorted <- sort(arglength, decreasing = TRUE)
head(arglength_sorted)
```

```
##             scan   format.default           source          formatC
##               22               16               16               15
##          library merge.data.frame
##               13               13
```

The function `scan()` has the most arguments.

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

```
length(funs[arglength == 0])
```

```
## [1] 227
```

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

227 functions have no arguments.