Bios 6301: Assignment 5

Lydia Yao

Due Thursday, 14 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 $f'(x) = -\sin(x) - 1$.

```
start_time <- Sys.time()
fofx <- function(x) {cos(x) - x}
secant <- function(x1,x2) {
  x2 -fofx(x2)*(x2 - x1)/(fofx(x2) - fofx(x1))
}

error = 1e-7
```

```
y = Inf
x1 = 0
x2 = 1
count = 0
while(y > error){
  xn = secant(x1,x2)
  x1 = x2
  x2 = xn
  y = fofx(xn)
  count = count + 1
}
end_time <- Sys.time()
print(end_time - start_time)
```

```
## Time difference of 0.03879189 secs
```

```
print(paste("Secant method took" ,count,"iterations starting at 0 and 1"))
```

```
## [1] "Secant method took 3 iterations starting at 0 and 1"
```

```
start_time <- Sys.time()
fofx_d <- function(x) {-sin(x) - 1}
NR <- function(x) {x - fofx(x)/fofx_d(x)}
x1 = 1
y = Inf
count = 0
while(abs(y) > error){
  xn = NR(x1)
  x1 = xn
  y = fofx(xn)
  count = count + 1
}
end_time <- Sys.time()
print(end_time - start_time)
```

```
## Time difference of 0.01639295 secs
```

```
print(paste("Newton-Raphson method took" ,count,"iterations starting at 1"))
```

```
## [1] "Newton-Raphson method took 3 iterations starting at 1"
```

From the simulation above, we see that the Newton-Raphson method is indeed faster by around 0.0112 seconds.

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:

```
roll <- function() 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)

```
set.seed(100)
craps <- function() {
  first <- 0
  count <- 1
  while(TRUE) {
    x <- roll()
    print(paste("Roll number", count, ":", x))
    if ((x == 7 | x == 11) && first == 0) {
      return("You Won!!")
    }
    if (first == 0) {first = x}
    if (x == 7 || x == 11) {
      return("You Lose.")
    }
    count = count + 1
  }

}
cat("Game 1\n")
```

```
## Game 1
```

```
craps()
```

```
## [1] "Roll number 1 : 4"
## [1] "Roll number 2 : 5"
## [1] "Roll number 3 : 6"
## [1] "Roll number 4 : 8"
## [1] "Roll number 5 : 6"
## [1] "Roll number 6 : 10"
## [1] "Roll number 7 : 5"
## [1] "Roll number 8 : 10"
## [1] "Roll number 9 : 5"
## [1] "Roll number 10 : 8"
## [1] "Roll number 11 : 9"
## [1] "Roll number 12 : 9"
## [1] "Roll number 13 : 5"
## [1] "Roll number 14 : 11"
## [1] "You Lose."
```

```
cat("Game 2\n")
```

```
## Game 2
```

```
craps()
```

```
## [1] "Roll number 1 : 6"
## [1] "Roll number 2 : 9"
## [1] "Roll number 3 : 9"
## [1] "Roll number 4 : 11"
## [1] "You Lose."
```

```
cat("Game 3\n")
```

```
## Game 3
```

```
craps()
```

```
## [1] "Roll number 1 : 6"
## [1] "Roll number 2 : 7"
```

```
## [1] "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)

```
library(sjmisc)
seed = 63398
  #1111480
while(TRUE) {
  print(seed)
  set.seed(seed)
  result = rep(NA, 30)
  for(i in 1:30){
      x <- capture.output(craps())
      if(str_contains(x, "You Won!!")) { result[i] = 1}
      else{result[i]= 0}
  }
  result
  consec <- rle(result)
  if (any(consec$lengths>=10 & consec$values==1)) {
    print(result)
    break
  }
  seed = seed + 1
}
```

```
library(sjmisc)
seed = 1
set.seed(seed)
result = rep(NA, 30)
  for(i in 1:17){
      x <- capture.output(craps())
      print(x)
      cat("\n")
      if(str_contains(x, "You Won!!")) { result[i] = 1}
      else{result[i]= 0}
  }
```

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)

```
#l <- function(x) length(formals(x))

l <- sapply(funs, function(x) length(formals(x)))
slength <- sort(l, decreasing = TRUE)
slength[1]
```

```
## scan
##   22
```

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

```
no_arguments <- l[l == 0]
length(no_arguments)
```

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

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