Bios 6301: Assignment 5

Lan Shi

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

```
# secant method
secant = function(f,x0=0,x1=1,epsilon=1e-8){
  # check the difference
  while(abs(x1-x0)>epsilon){
    fx0 = f(x0)
    fx1 = f(x1)
    x2 = x1-fx1*(x1-x0)/(fx1-fx0)
```

```
    x0 = x1
    x1 = x2
  }
  return(x1)
}
start_time <- Sys.time()
secant(\(x) cos(x)-x)
```

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

```
end_time <- Sys.time()
# running time for secant method:
time_secant = difftime(end_time, start_time, units = "secs")
time_secant
```

```
## Time difference of 0.02593994 secs
```

```
# Newton-Raphson method
NR = function(f,df,x0=0,epsilon=1e-8){
  x1 = x0 + 1 # initialize x1
  while(abs(x1-x0)>epsilon){
    x0 = x1
    fx0 = f(x0)
    dfx0 = df(x0)
    x1 = x0 - fx0/dfx0
  }
  return(x1)
}
# running time for NR method:
start_time <- Sys.time()
NR(\(x) cos(x)-x, \(x) -sin(x)-x)
```

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

```
end_time <- Sys.time()
# running time for Newton-Raphson method:
time_NR = difftime(end_time, start_time, units = "secs")
time_NR
```

```
## Time difference of 0.007876158 secs
```

```
if (time_NR < time_secant){
  sprintf("Newton-Raphson method is %s seconds faster than Secant method.",round(time_secant-time_NR,8))
}else{
 sprintf("Secant method is %s seconds faster than Newton-Raphson method.", round(time_NR-time_secant,8))
}
```

```
## [1] "Newton-Raphson method is 0.01806378 seconds faster than 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:

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(win_num0=c(7,11),gen.output=T){
  if (gen.output) cat("==========\nNew Game Starts:\n")
  # first game
  idx = 1
  x = sum(ceiling(6*runif(2)))
  if (gen.output) cat(sprintf("No.%s roll sum is %s.\n",idx,x))
  # if first roll not in win_num0, roll again.
  if (!(x %in% win_num0)){
    win_num1 = x
    x = 0  # restore x
    while ( !(x %in% c(win_num0,win_num1)) ){
      x = sum(ceiling(6*runif(2)))
      idx = idx + 1
      if (gen.output) cat(sprintf("No.%s roll sum is %s.\n",idx,x))
    }
    if (x %in% win_num0){
      if (gen.output) cat("Haha, You lose!\nGame Ends.\n==========\n")
      win = 0
    }else{  # x==win_num1
      if (gen.output) cat("Congrat, You win!\nGame Ends.\n==========\n")
      win = 1
    }
  # if first roll win:
  }else{
    if (gen.output) cat("Super Lucky, You win in the first roll!\nGame Ends.\n==========\n")
    win = 1
  }
}

set.seed(100)
craps()
```

```
## ==========
## New Game Starts:
## No.1 roll sum is 4.
## No.2 roll sum is 5.
## No.3 roll sum is 6.
## No.4 roll sum is 8.
## No.5 roll sum is 6.
## No.6 roll sum is 10.
## No.7 roll sum is 5.
## No.8 roll sum is 10.
## No.9 roll sum is 5.
## No.10 roll sum is 8.
## No.11 roll sum is 9.
## No.12 roll sum is 9.
## No.13 roll sum is 5.
## No.14 roll sum is 11.
## Haha, You lose!
```

```
## Game Ends.
## ==========
```

```
craps()
```

```
## ==========
## New Game Starts:
## No.1 roll sum is 6.
## No.2 roll sum is 9.
## No.3 roll sum is 9.
## No.4 roll sum is 11.
## Haha, You lose!
## Game Ends.
## ==========
```

```
craps()
```

```
## ==========
## New Game Starts:
## No.1 roll sum is 6.
## No.2 roll sum is 7.
## Haha, You lose!
## Game Ends.
## ==========
```

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)

```
# for each seed run 10 times game, stop till find the one.
for (seed in 1:1e5){
  set.seed(seed)
  win_num = replicate(n=10,craps(gen.output = F))
  if (sum(win_num)==10){
    cat(sprintf("Seed that win 10 straight games is %s.\n",seed))
    set.seed(seed)
    replicate(n=10,craps())
    break
  }
}
```

```
## Seed that win 10 straight games is 880.
## ==========
## New Game Starts:
## No.1 roll sum is 7.
## Super Lucky, You win in the first roll!
## Game Ends.
## ==========
## ==========
## New Game Starts:
## No.1 roll sum is 8.
## No.2 roll sum is 9.
## No.3 roll sum is 3.
## No.4 roll sum is 10.
## No.5 roll sum is 6.
## No.6 roll sum is 8.
## Congrat, You win!
## Game Ends.
```

```
## ==========
## ==========
## New Game Starts:
## No.1 roll sum is 10.
## No.2 roll sum is 10.
## Congrat, You win!
## Game Ends.
## ==========
## ==========
## New Game Starts:
## No.1 roll sum is 9.
## No.2 roll sum is 9.
## Congrat, You win!
## Game Ends.
## ==========
## ==========
## New Game Starts:
## No.1 roll sum is 11.
## Super Lucky, You win in the first roll!
## Game Ends.
## ==========
## ==========
## New Game Starts:
## No.1 roll sum is 8.
## No.2 roll sum is 8.
## Congrat, You win!
## Game Ends.
## ==========
## ==========
## New Game Starts:
## No.1 roll sum is 5.
## No.2 roll sum is 5.
## Congrat, You win!
## Game Ends.
## ==========
## ==========
## New Game Starts:
## No.1 roll sum is 7.
## Super Lucky, You win in the first roll!
## Game Ends.
## ==========
## ==========
## New Game Starts:
## No.1 roll sum is 9.
## No.2 roll sum is 9.
## Congrat, You win!
## Game Ends.
## ==========
## ==========
## New Game Starts:
## No.1 roll sum is 7.
## Super Lucky, You win in the first roll!
## Game Ends.
## ==========
```

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)

```
len_argus = sapply(funs,\(x) length(formals(x)))
len_argus[which.max(len_argus)]
```

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

```
#funs[[which.max(len_argus)]]
```

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

```
sum(len_argus==0)
```

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

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