Bios 6301: Assignment 5

Haley Yaremych

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

```
set.seed(321)
# function for validation
f = function(x) cos(x)-x


# write the secant algorithm
secant = function(func, x0, x1){
```

```
  orig=x0
  old=x1
  current = old - func(old)*((old-orig)/(func(old)-func(orig)))

  while(abs(func(current)) > 1e-8){

    new = current - func(current)*((current-old)/(func(current)-func(old)))
    old=current
    current=new

  }

  return(new)

}


# compare solutions
secant(f, -1, 1)
```

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

```
uniroot(f, c(-10,10))$root
```

```
## [1] 0.7390799
```

```
# compare run time
s.start = Sys.time()
secant(f, -1, 1)
```

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

```
s.end = Sys.time()

u.start = Sys.time()
uniroot(f, c(-10,10))$root
```

```
## [1] 0.7390799
```

```
u.end = Sys.time()

s.end-s.start ; u.end-u.start
```

```
## Time difference of 0.001828909 secs
```

```
## Time difference of 0.005404949 secs
```

The secant function yields the same answer as the NR method down to 4 decimal places. The secant function runs faster than the NR method, which is reflected in a smaller difference between the start time and end time of the function. The difference is only a few thousands of a second.

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(){

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

  if (x==7 | x==11){

    cat("you win! you rolled a 7 or 11 on your first roll.")

  } else {

    while(TRUE){
      new_x <- sum(ceiling(6*runif(2)))
      print(new_x)

      if (new_x == x){
        cat("you win! a subsequent roll matched your first roll. ")
        break
      }

      if (new_x == 7 | new_x == 11){
        cat("you lose!")
        break
      }
    }
  }


}

set.seed(100)
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
## you lose!
```

```
craps()
```

```
## [1] 6
## [1] 9
## [1] 9
## [1] 11
## you lose!
```

```
craps()
```

```
## [1] 6
## [1] 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)

```
craps2 = function(print = TRUE){

  x <- sum(ceiling(6*runif(2)))
  if (print==TRUE) {print(x)}

  if (x==7 | x==11){

    if (print==TRUE) {cat("you win! you rolled a 7 or 11 on your first roll.")}
    return(1)

  } else {

    while(TRUE){
      new_x <- sum(ceiling(6*runif(2)))
      if (print==TRUE) {print(new_x)}

      if (new_x == x){
        if (print==TRUE) {cat("you win! a subsequent roll matched your first roll. ")}
        return(1)
        break
      }

      if (new_x == 7 | new_x == 11){
        if (print==TRUE) {cat("you lose!")}
        return(0)
        break
      }
    }
  }


}


seed = 1

while(TRUE){
```

```
  results = numeric(10)
  for (i in 1:10){
    results[i] = craps2(print=FALSE)
  }

  if (results[i]==1 & length(unique(results==1))==1){
    print(seed)
    print(results)
    break
  } else {seed = seed+1}
}
```

```
## [1] 1631
##  [1] 1 1 1 1 1 1 1 1 1 1
```

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

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

```
args = numeric(length(funs))

for (i in 1:length(funs)){
  args[i] = length(formals(funs[[i]]))
}

which(args==max(args))
```

```
## [1] 961
```

```
length(which(args==0))
```

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