Bios 6301: Assignment 5

Yiqing Pan

39

Due Thursday, 10 October, 1:00 PM

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

40 points total.

Submit a single quarto file (named `homework5.qmd`), 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.qmd` 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 <- function(x0, x1, tol, fx) {

  sims = 0

  while (abs(x1-x0) >= tol) {
    f_x0 = fx(x0)
```

```r
    f_x1 = fx(x1)
    new_x1 = x1 - f_x1*((x1-x0)/(f_x1-f_x0))
    x0 = x1
    x1 = new_x1
    sims = sims + 1

  }

  return(c(x1, sims))
}


NR <- function(x0, tol, fx, dx) {
  sims = 0

  f_x0 = fx(x0)
  d_x0 = dx(x0)
  x1 = x0 - f_x0/d_x0
  sims = sims + 1

  while (abs(x1-x0) >= tol) {
    x0 = x1
    f_x0 = fx(x0)
    d_x0 = dx(x0)
    x1 = x0 - f_x0/d_x0
    sims = sims + 1

  }

  return(c(x1, sims))

}
```

```r
fx <- function(X){
fx <- cos(X) - X
return(fx)
}

dx <- function(X){
dx <- -sin(X) - 1
return(dx)
}


cat("Secant Algorithm result with x0 = 0, x1 = 1:", secant(0,1, 1e-5, fx)[1], ", Simulations:", secant(
```

```
## Secant Algorithm result with x0 = 0, x1 = 1: 0.7390851 , Simulations: 5
```

```r
cat("\nSecant Algorithm result with x0 = 1, x1 = 2:", secant(1,2, 1e-5, fx)[1], ", Simulations:", secan
```

```
##
## Secant Algorithm result with x0 = 1, x1 = 2: 0.7390851 , Simulations: 5
```

```
cat("\nSecant Algorithm result with x0 = 1, x1 = 100:", secant(1,100, 1e-5, fx)[1], ", Simulations:", se
```

```
##
## Secant Algorithm result with x0 = 1, x1 = 100: 0.7390851 , Simulations: 6
```

```
cat("\nNewton-Raphson Algorithm result with x0 = 0 :", NR(0, 1e-5, fx, dx)[1], ", Simulations:", NR(0, 1
```

```
##
## Newton-Raphson Algorithm result with x0 = 0 : 0.7390851 , Simulations: 5
```

```
cat("\nNewton-Raphson Algorithm result with x0 = 1 :", NR(1, 1e-5, fx, dx)[1], ", Simulations:", NR(1, 1
```

```
##
## Newton-Raphson Algorithm result with x0 = 1 : 0.7390851 , Simulations: 4
```

```
cat("\nNewton-Raphson Algorithm result with x0 = 2 :", NR(2, 1e-5, fx, dx)[1], ", Simulations:", NR(2, 1
```

```
##
## Newton-Raphson Algorithm result with x0 = 2 : 0.7390851 , Simulations: 3
```

```
cat("\nNewton-Raphson Algorithm result with x0 = 100 :", NR(100, 1e-5, fx, dx)[1], ", Simulations:", NR
```

```
##
## Newton-Raphson Algorithm result with x0 = 100 : 0.7390851 , Simulations: 9
```

Comparing the performance of the Secant Algorithm with the Newton-Raphson Algorithm, it is hard to say which performances better arbitrarily. In this example, the simulation they took do not differ very much and rely heavily on the initially selected (x0, x1) / x0 value. If the starting value is close to the true root, Newton-Raphson Algorithm seems to be a little faster by 2 less simulation required. If the starting value is very far from the true root, Secant Algorithm is faster with 3 less simulation needed.

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)))
```

```
crap <- function() {
  roll = 0
  sum_x <- sum(ceiling(6*runif(2)))
  roll = roll + 1
  cat(c("you get", sum_x, "on the first roll;"))
```

```r
  if (sum_x == 7 | sum_x == 11) {
    cat(c("\nyou win on the first roll with", sum_x, "!"))
    return("GAME FINISHED 1")

  } else {
    roll_x <- sum(ceiling(6*runif(2)))
    roll = roll + 1
    cat(c("\nyou get", roll_x, "on roll 2 ;"))

    while (!(roll_x %in% c(sum_x, 7, 11))) {
      roll_x <- sum(ceiling(6*runif(2)))
      roll = roll + 1
      cat(c("\nyou get", roll_x, "on roll", roll,";"))
    }
    if (roll_x == sum_x) {
      cat("\nyou win on roll", roll ,"with", roll_x,"!")
      return("GAME FINISHED 2") } else {
        cat(c("\nyou lose on roll", roll ,"with", roll_x), "!")
        return("GAME FINISHED 3")
      }
  }
}

crap_multi<- function(repeats) {
  for (i in (1:repeats)) {
    cat("GAME", i, "STARTS: \n")
    crap()
    cat("\n")
  }
}
```

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)

crap_multi(3)
```

```
## GAME 1 STARTS:
## you get 4 on the first roll;
## you get 5 on roll 2 ;
## you get 6 on roll 3 ;
## you get 8 on roll 4 ;
## you get 6 on roll 5 ;
## you get 10 on roll 6 ;
## you get 5 on roll 7 ;
## you get 10 on roll 8 ;
## you get 5 on roll 9 ;
## you get 8 on roll 10 ;
## you get 9 on roll 11 ;
## you get 9 on roll 12 ;
```

```
## you get 5 on roll 13 ;
## you get 11 on roll 14 ;
## you lose on roll 14 with 11 !
## GAME 2 STARTS:
## you get 6 on the first roll;
## you get 9 on roll 2 ;
## you get 9 on roll 3 ;
## you get 11 on roll 4 ;
## you lose on roll 4 with 11 !
## GAME 3 STARTS:
## you get 6 on the first roll;
## you get 7 on roll 2 ;
## you lose on roll 2 with 7 !
```

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)

```
crap_new <- function() {
  roll = 0
  sum_x <- sum(ceiling(6*runif(2)))
  roll = roll + 1
  #cat(c("you get", sum_x, "on the first roll;"))

  if (sum_x == 7 | sum_x == 11) {
    #cat(c("\nyou win on the first roll with", sum_x, "!"))
    return(TRUE)

  } else {
    roll_x <- sum(ceiling(6*runif(2)))
    roll = roll + 1
    #cat(c("\nyou get", roll_x, "on roll 2 ;"))

    while (!(roll_x %in% c(sum_x, 7, 11))) {
      roll_x <- sum(ceiling(6*runif(2)))
      roll = roll + 1
      #cat(c("\nyou get", roll_x, "on roll", roll,";"))
    }
    if (roll_x == sum_x) {
      #cat("\nyou win on roll", roll ,"with", roll_x,"!")
      return(TRUE) } else {
        #cat(c("\nyou lose on roll", roll ,"with", roll_x), "!")
        return(FALSE)
      }
  }
}

crap_multi_new <- function(repeats) {
  result <- NULL
  for (i in (1:repeats)) {
    #cat("GAME", i, "STARTS: \n")
    result[i]= crap_new()
  }
  return(result)
}
```

```r
i = 0

while (i < 1e10) {
  set.seed(i)
  if (all(crap_multi_new(10))) {
    print(i)
    return()
  } else {
    i = i+1
  }
}
```

```
## [1] 880
```

```r
#Verify seed 880

set.seed(880)

crap_multi(10)
```

```
## GAME 1 STARTS:
## you get 7 on the first roll;
## you win on the first roll with 7 !
## GAME 2 STARTS:
## you get 8 on the first roll;
## you get 9 on roll 2 ;
## you get 3 on roll 3 ;
## you get 10 on roll 4 ;
## you get 6 on roll 5 ;
## you get 8 on roll 6 ;
## you win on roll 6 with 8 !
## GAME 3 STARTS:
## you get 10 on the first roll;
## you get 10 on roll 2 ;
## you win on roll 2 with 10 !
## GAME 4 STARTS:
## you get 9 on the first roll;
## you get 9 on roll 2 ;
## you win on roll 2 with 9 !
## GAME 5 STARTS:
## you get 11 on the first roll;
## you win on the first roll with 11 !
## GAME 6 STARTS:
## you get 8 on the first roll;
## you get 8 on roll 2 ;
## you win on roll 2 with 8 !
## GAME 7 STARTS:
## you get 5 on the first roll;
## you get 5 on roll 2 ;
## you win on roll 2 with 5 !
## GAME 8 STARTS:
## you get 7 on the first roll;
## you win on the first roll with 7 !
```

```
## GAME 9 STARTS:
## you get 9 on the first roll;
## you get 9 on roll 2 ;
## you win on roll 2 with 9 !
## GAME 10 STARTS:
## you get 7 on the first roll;
## you win on the first roll with 7 !
```

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)

```r
arg_count <- sapply(funs, function(f) length(formals(f)))
max_arg <- names(which.max(arg_count))
max_arg
```

```
## [1] "scan"
```

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

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

```r
no_arg <- sapply(funs, function(f) length(formals(f)) == 0)
no_arg_count <- sum(no_arg)
no_arg_count
```

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