Bios 6301: Assignment 5

Yan Yan

Due Thursday, 18 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.

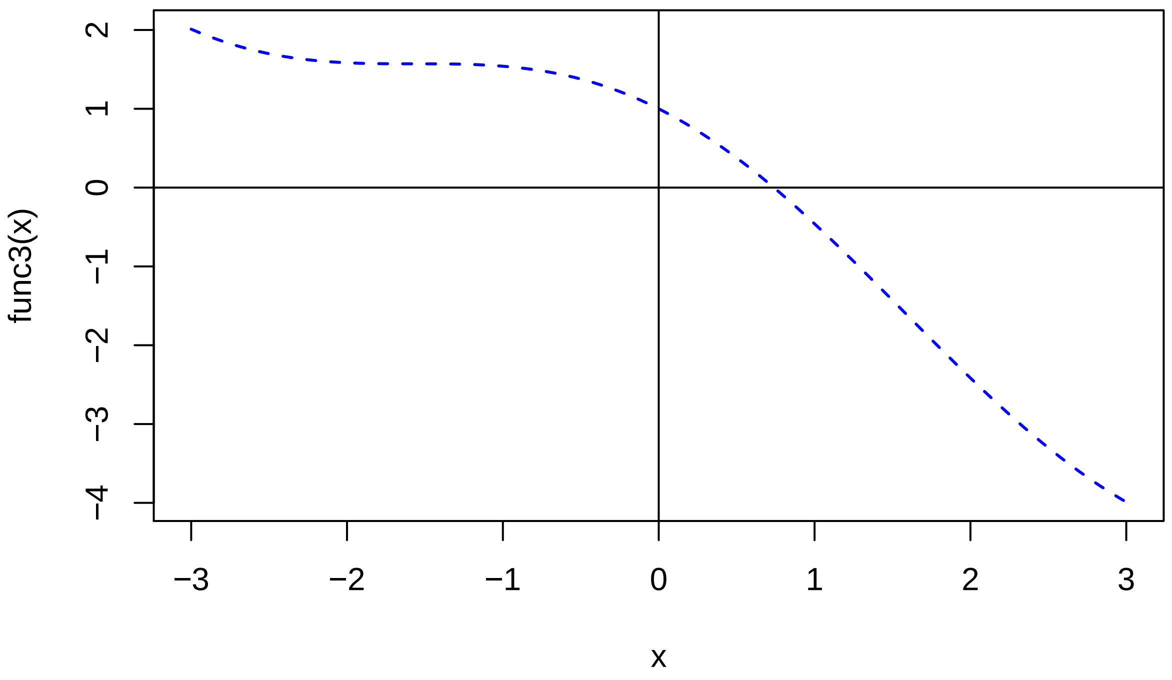
The general form of the recurrence equation for the secant method is:

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

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

```r
# example function
func3 <- function(q)  {
  cos(q) - q
}
# plot the function to see how the equation behaves
# and determine the interval bounds for arguments used in secant.method
curve(func3, xlim=c(-3,3), col='blue', lwd=1.5, lty=2)
abline(h=0)
abline(v=0)
```



Secant Method

```r
## secant method
secant.method <- function(f, x0, x1, tol = 1e-9, n = 500) {
  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.
      return(x2)
    }
    # If the root was not determined in the previous iteration,
    # update the values and proceed to the next iteration.
    x0 <- x1
    x1 <- x2
  }
}
secant.method(func3,0,2)
```

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

Secant method found the root of $f(x) = \cos(x) - x$ is 0.7391.

Newton-Raphson Method

```r
##Newton-Raphson method
newton.raphson <- function(f, a, tol = 1e-9, n = 500) {
  x0 <- a # Set start value to supplied lower bound
for (i in 1:n) {
    dx <- -sin(x0) - 1 #f'(x0)
    x1 <- x0 - (f(x0) / dx) # Calculate next value x1
    if (abs(x1 - x0) < tol) {
      return(x1)
    }
    x0 <- x1
  }
}
newton.raphson(func3,0)
```

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

Newton raphson method gives the same result of 0.7391.

Compare the time use of these two methods

```r
# time difference
newraphtim <- system.time(replicate(1e5,newton.raphson(func3,-5)))
secanttim <- system.time(replicate(1e5,secant.method(func3,-5,2)))
secanttim / newraphtim
```

```
##      user    system   elapsed
## 0.4589818 0.3870968 0.4576535
```

In comparison,Secant method is much faster than New Raphson method.
"User CPU time" gives the CPU time spent by the current process and "system CPU time" gives the CPU time spent by the kernel (the operating system) on behalf of the current process. Elapsed Time is the time charged to the CPU(s) for the expression. Secant method uses half of the time that New Raphson method requires[1].

Question 2

20 points

The game of craps is played as follows. 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)

```r
set.seed(100)

# crap function
crap <- function(){
  d2 <- 99 # impossible roll
  # 1st roll
  d1 <- sum(ceiling(6*runif(2)))
  if(d1 %in% c(7,11)){  #won
    cat(d1,"You won\n")
    return(1)
  }  else {
  while (d2 != d1) {
    d2 <- sum(ceiling(6*runif(2)))
    if(d2 == 7 | d2 == 11){ #lost
      cat(d1, d2, "You lost\n" )
      return(0)
      } else if(d2 == d1){ #won
        cat(d1, d2, "You won\n" )
        return(1)
        }
  }
  }
}
replicate(3,crap())
```

```
## 4 11 You lost
## 6 11 You lost
## 6 7 You lost
```

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

2. 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)

```r
# find a seed that will win ten straight games
seedfind <- NULL
for (i in 500:1000) {
  set.seed(i)
  capture.output(seedfind[i] <- sum(replicate(10,crap())))
}
which(seedfind == 10)
```

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

With seed 880, we can win ten straight games.

show the output of the ten games

```r
# show the output of the ten games
set.seed(880)
replicate(10,crap())
```

```
## 7 You won
## 8 8 You won
## 10 10 You won
## 9 9 You won
## 11 You won
## 8 8 You won
## 5 5 You won
## 7 You won
## 9 9 You won
## 7 You won
```

```
##  [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:

```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
# find number of arguments for each functions
objs <- mget(ls("package:base"), inherits = TRUE)
funs <- Filter(is.function, objs)
args.number <- sapply(funs, function(x){length(formalArgs(x))})
max(args.number)
```

```
## [1] 22
```

```r
# find the function with the most arguments
funs[which.max(args.number)]
```

```
## $scan
## function (file = "", what = double(), nmax = -1L, n = -1L, sep = "",
##     quote = if (identical(sep, "\n")) "" else "'\"", dec = ".",
##     skip = 0L, nlines = 0L, na.strings = "NA", flush = FALSE,
##     fill = FALSE, strip.white = FALSE, quiet = FALSE, blank.lines.skip = TRUE,
##     multi.line = TRUE, comment.char = "", allowEscapes = FALSE,
##     fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
## {
##     na.strings <- as.character(na.strings)
##     if (!missing(n)) {
##         if (missing(nmax))
##             nmax <- n/pmax(length(what), 1L)
##         else stop("either specify 'nmax' or 'n', but not both.")
##     }
##     if (missing(file) && !missing(text)) {
##         file <- textConnection(text, encoding = "UTF-8")
##         encoding <- "UTF-8"
##         on.exit(close(file))
##     }
##     if (is.character(file))
##         if (file == "")
##             file <- stdin()
##         else {
##             file <- if (nzchar(fileEncoding))
##                 file(file, "r", encoding = fileEncoding)
##             else file(file, "r")
##             on.exit(close(file))
##         }
##     if (!inherits(file, "connection"))
##         stop("'file' must be a character string or connection")
##     .Internal(scan(file, what, nmax, sep, dec, quote, skip, nlines,
##         na.strings, flush, fill, strip.white, quiet, blank.lines.skip,
##         multi.line, comment.char, allowEscapes, encoding, skipNul))
## }
## <bytecode: 0x7fb3c37a4340>
## <environment: namespace:base>
```

Function “scan” has the most arguments.

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

```r
no.args <- (args.number == 0)
sum(no.args)
```

```
## [1] 226
```

There are 226 functions have no arguments.

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

Reference: [1] https://stackoverflow.com/questions/5688949/what-are-user-and-system-times- measuring-in-r-system-timeexp-output