Bios 6301: Assignment 5

Megan Taylor

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

```r
fx = function(x){
  cos(x)-x
}
# need arguments of two initial guesses and the function
secant = function(x0, x1, fun){
  # initialize a data frame to store results of each iteration
  sectab <- data.frame(i=as.numeric(),x2 = as.numeric())
  for (i in 1:100){
    # formula for secant method
    x2 = x1 - fun(x1)*(x1-x0)/(fun(x1)-fun(x0))
    # add result of this iteration to data frame
    sectab[nrow(sectab) + 1,] = c(i, round(x2, digits = 7))
    # after the first iteration, compare to previous iteration
    if (i>1) {
      # if the previous iteration is the same (with rounding to 7 digits), end
      if (sectab[i-1, "x2"]==sectab[i, "x2"]){
        break
        }
    }
    # x0 is now x1 and x1 is now x2
    x0 = x1
    x1 = x2
  }
  # return the data frame
  sectab
}

# example output and time to run
start.time.sec = Sys.time()
secant(-4,4,fx)
```

```
##   i         x2
## 1 1 -0.6536436
## 2 2  0.4504461
## 3 3  0.9482131
## 4 4  0.7252078
## 5 5  0.7385077
## 6 6  0.7390869
## 7 7  0.7390851
## 8 8  0.7390851
```

```
end.time.sec = Sys.time()
difftime(end.time.sec, start.time.sec)
```

```
## Time difference of 0.07878709 secs
```

```
sectab = secant(-4,4,fx)
root = sectab[nrow(sectab),2]
# root of example function
root
```

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

```
# testing against Newton's method
nsim = 1000
secits <- data.frame(its=1:1000)
avg.time.sec = 0
x0 = 0
x1 = 1
for (n in 1:nsim){
    start.time.sec = Sys.time()
    secsim <- secant(x0, x1, fx)
    end.time.sec = Sys.time()
    secits[n,] = nrow(secsim)
    avg.time.sec = avg.time.sec + (difftime(end.time.sec, start.time.sec)/nsim)
    x0 = x0 - 0.01
    x1 = x1 + 0.01
}
# average number of iterations for secant method to converge with
# difference between initial guesses gradually growing over 1000 runs
mean(secits[,1])
```

```
## [1] 7.105
```

```
# average time to run
avg.time.sec
```

```
## Time difference of 0.001381468 secs
```

```
# Newton-Raphson Method (similat code to secant method)
fxprime = function(x){
  -sin(x)-1
}
NR = function(x0, fun, funprime){
  NRtab <- data.frame(i=as.numeric(),x1 = as.numeric())
  for (i in 1:100){
    x1 = x0 - fun(x0)/funprime(x0)
    NRtab[nrow(NRtab) + 1,] = c(i, round(x1, digits = 7))
    if (i>1) {
      if (NRtab[i-1, "x1"]==NRtab[i, "x1"]){
        break
        }
      }
    x0 = x1
  }
  NRtab
}

# example output and time to run
start.time.NR = Sys.time()
NR(-4, fx, fxprime)
```

```
##   i         x1
## 1 1 -2.0952006
## 2 2  9.7706387
## 3 3 -6.4343983
## 4 4  2.3050812
## 5 5  0.5974991
## 6 6  0.7442094
## 7 7  0.7390909
## 8 8  0.7390851
## 9 9  0.7390851
```

```
end.time.NR = Sys.time()
difftime(end.time.NR, start.time.NR)
```

```
## Time difference of 0.01691604 secs
```

```
# testing NR for many
NRits <- data.frame(its=1:1000)
```

```
avg.time.NR = 0
x0 = -10
for (n in 1:nsim){
    start.time.NR = Sys.time()
    NRsim <- NR(x0, fx, fxprime)
    end.time.NR = Sys.time()
    NRits[n,] = nrow(NRsim)
    avg.time.NR = avg.time.NR + (difftime(end.time.NR, start.time.NR)/nsim)
    x0 = x0 + 0.02
}
# average number of iterations for convergence of Newton's method as initial
# guess goes from -10 to 10, excluding cases where the method did not converge
# within 100 iterations
mean(NRits[which(NRits[,1]!=100),1])
```

```
## [1] 17.3568
```

```
# aveage time for Newton's method to run
avg.time.NR
```

```
## Time difference of 0.005006438 secs
```

```
# Secant method converges faster. Based on these tests, it converges an average
# of 2.44 (17.3568/7.105) times faster. Secant method also took less time to
# run on my computer on average (0.0014 seconds vs. 0.0041 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:

```
x <- 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(){
  x <- sum(ceiling(6*runif(2)))
  print(paste("First roll is",x))
  if (x==7 | x ==11){
    print(paste("You win!"))
    win = 1
  } else {
    y = 0
    i = 2
    while(y != x){
      y = sum(ceiling(6*runif(2)))
      print(paste("Roll", i, "is", y))
      i = i+1
      if (y == 7 |y == 11){
        print("You lose!")
        break
      }
    }
    if (y==x){
      print("You matched the first roll; you win!")
      win = 1
    }
  }
}
craps()
```

```
## [1] "First roll is 4"
## [1] "Roll 2 is 5"
## [1] "Roll 3 is 6"
## [1] "Roll 4 is 8"
## [1] "Roll 5 is 6"
## [1] "Roll 6 is 10"
## [1] "Roll 7 is 5"
## [1] "Roll 8 is 10"
## [1] "Roll 9 is 5"
## [1] "Roll 10 is 8"
## [1] "Roll 11 is 9"
## [1] "Roll 12 is 9"
## [1] "Roll 13 is 5"
## [1] "Roll 14 is 11"
## [1] "You lose!"
```

```
craps()
```

```
## [1] "First roll is 6"
## [1] "Roll 2 is 9"
## [1] "Roll 3 is 9"
## [1] "Roll 4 is 11"
## [1] "You lose!"
```

```
craps()
```

```
## [1] "First roll is 6"
## [1] "Roll 2 is 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)

```
# function that will calculate number of wins for a given number of games
# and RNG seed
craps.mult <-function(games, seed){
  set.seed(seed)
  wins = 0
  for(g in 1:games){
    x <- sum(ceiling(6*runif(2)))
    if (x==7 | x ==11){
      wins = wins + 1
    } else {
      y = 0
      i = 2
      while(y != x){
        y = sum(ceiling(6*runif(2)))
        i = i+1
        if (y == 7 |y == 11){
          break
        }
      }
    if (y==x){
      wins = wins + 1
    }
  }
  }
  wins}

# testing seed numbers 1-1000 until finding one that wins 10/10 games
for (i in 1:1000){
  if(craps.mult(games = 10, seed = i)==10){
    i
    break
  }
}

# confirming
craps.mult(10, 880)
```

```
## [1] 10
```

```
# set to new seed
set.seed(880)

# display output of 10 games using seed 880
for (i in 1:10){
  cat("\nGame", i, "\n")
  craps()
}
```

```
##
## Game 1
## [1] "First roll is 7"
## [1] "You win!"
##
## Game 2
## [1] "First roll is 8"
## [1] "Roll 2 is 9"
## [1] "Roll 3 is 3"
## [1] "Roll 4 is 10"
## [1] "Roll 5 is 6"
## [1] "Roll 6 is 8"
## [1] "You matched the first roll; you win!"
##
## Game 3
## [1] "First roll is 10"
## [1] "Roll 2 is 10"
## [1] "You matched the first roll; you win!"
##
## Game 4
## [1] "First roll is 9"
## [1] "Roll 2 is 9"
## [1] "You matched the first roll; you win!"
##
## Game 5
## [1] "First roll is 11"
## [1] "You win!"
##
## Game 6
## [1] "First roll is 8"
## [1] "Roll 2 is 8"
## [1] "You matched the first roll; you win!"
##
## Game 7
## [1] "First roll is 5"
## [1] "Roll 2 is 5"
## [1] "You matched the first roll; you win!"
##
## Game 8
## [1] "First roll is 7"
## [1] "You win!"
##
## Game 9
## [1] "First roll is 9"
## [1] "Roll 2 is 9"
## [1] "You matched the first roll; you win!"
##
## Game 10
## [1] "First roll is 7"
## [1] "You win!"
```

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)

```
# initialize a previous length variable and index variable
lenprev = 0
index = 1
# for loop that runs over each function listed in funs
for (i in funs){
  # number of arguments (subtract 1 because the list of args always
  # includes NULL at the end)
  len <- length(as.list(args(i)))-1
  # if the number of arguments for the current function is more than the
  # previous function with the most arguments, its length becomes the new
  # standard and the index is stored as a variable
  if (len > lenprev){
    lenprev <- len
    num <- index
  }
  index <- index+1
}
# Show index of funs with the highest number of arguments
funs[num]
```

```
## $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: 0x0000000017b254a8>
## <environment: namespace:base>
```

```
# This shows that the scan function has the most arguments
```

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

```
# Initialize a counter for functions with zero arguments
zeros = 0
funlist = c()
```

```r
# run for each function in funs
for (i in funs){
  len <- length(as.list(args(i)))-1
  # if there are no arguments, add 1 to the zero counter
  if (len == 0){
    zeros = zeros + 1
    funlist = c(funlist, i)
  }
}
# There are 48 functions with no arguments (used funlist to confirm)
zeros
```

```
## [1] 48
```

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