Bios 6301: Assignment 5

Haoyang Yi

10/3/2022

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

```
library(numDeriv)
# fx
fx = function(x){
  cos(x)-x
}
xn = NULL
# secant algorithm
secant = function(f,a,b,n,tol){
  x0 = a
  fx0 = f(x0)
  if (fx0 == 0) {
    return(a)
  }
  x1  = b
  fx1 = f(b)
  if (fx1 == 0) {
    return(b)
```

```
  }
  x = rep(0,n)
  x[1] = x0
  x[2] = x1
  fx = rep(0,n)
  fx[1] = fx0
  fx[2] = fx1
  for (i in 3:n){
    d = fx[i-1]*(x[i-1]-x[i-2])/(fx[i-1]-fx[i-2])
    x[i]=x[i-1]-d
    newx = x[i]
    fx[i]=f(newx)
    if (abs(x[i]-x[i-1])<tol){
      res = list('root approximation'=x[i],'iteration'=i)
      return(res)
    }
  }
  print('need more iterations')
}
secant(fx,0,1,1000,1e-5)
```

```
## $`root approximation`
## [1] 0.7390851
##
## $iteration
## [1] 7
```

```
cos(0.7390851)-0.7390851 # close enough to 0
```

```
## [1] 5.558929e-08
```

```
# newton raphson algorithm
newton.raphson = function(f,a,n=1000,tol=1e-5){
  x0 = a
  fx0 = f(x0)
  if (fx0 == 0) {
    return(a)
  }
  x = NULL
  x[1] = x0
  for (i in 2:n){
    dx = genD(func = f,x=x[i-1])$D[1]
    x[i]=x[i-1]-f(x[i-1])/dx
    if (abs(x[i]-x[i-1])<tol){
      res = list('root approximation'=x[i],'iteration'=i)
      return(res)
    }
  }
  print('need more iterations')
}
newton.raphson(fx,0,1000,1e-5) # correct root of f(x)
```

```
## $`root approximation`
## [1] 0.7390851
##
```

```
## $iteration
## [1] 6
```

```
microbenchmark::microbenchmark(secant(fx,0,1,1000,1e-5),newton.raphson(fx,0,1000,1e-5))
```

```
## Unit: microseconds
##                                expr   min     lq     mean median      uq    max
##       secant(fx, 0, 1, 1000, 1e-05)  22.9  38.80   82.551  47.40   91.30  488.8
##  newton.raphson(fx, 0, 1000, 1e-05) 514.4 732.15 1031.141 861.85 1162.45 4237.0
##  neval cld
##    100  a
##    100   b
```

In this case, the initial guess in secant method is that x0 = 0 and x1 = 1, the initial guess in Newton Raphson method is x0 = 0. Based on the output of microbenchmark(), secant method has shorter computing time, also secant method needs less iteration for computing, secant method has better performance than Newton Raphson's method in this case.

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)
fcraps = function() {
  x = sum(ceiling(6*runif(2)))
  if (x==7|x==11) return("win")
  while (TRUE) {
    x1 = sum(ceiling(6*runif(2)))
    if (x1 == x) return("win")
    if (x1 == 7|x1==11) return("lose")
  }
}
ncraps = function(f=fcraps,n){
  res = NULL
  for (i in 1:n){
    res[i]=f()
  }
  return(res)
}
ncraps(n=3)
```

```
## [1] "lose" "lose" "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)

```
seed = 1

while (seed<10000){
  set.seed(seed)
  res = ncraps(n=10)
  a = rep('win',10)
  if (length(unique(res))==1){
    if ((res %in% 'win')[1]==TRUE) break
  }
  seed=seed+1
}
seed
```

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

```
set.seed(seed)
ncraps(n=10)
```

```
##  [1] "win" "win" "win" "win" "win" "win" "win" "win" "win" "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)

```
res = NULL
for (i in 1:length(funs)){
  arguments = length(formals(funs[[i]]))
  res[i]=arguments
  }
funs[which.max(res)] # the scan() function has the most arguments
```

```
## $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: 0x0000021031a1ef48>
## <environment: namespace:base>
```

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

```
length(which(res==0)) # 227 functions have no arguments
```

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

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