

Bios 6301: Assignment 5

Yeji Ko

Due Thursday, 14 October, 1:00 PM

$5^{n=\text{day}}$ 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 x_2 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, f){
  x2 <- x1 - f(x1)*{(x1-x0)/(f(x1)-f(x0))}
  while(abs(x2 - x1) < 0.0001){
    x0 <- x1
    x1 <- x2
  }
  return(x2)
```

```

}

secant(1,2, function(x) cos(x) - x)

## [1] 0.7650347

system.time(secant(1,2, function(x) cos(x) - x))

##      user      system elapsed
##    0.003      0.000      0.003

newton<- function(x1,f,fdev){
  x2 <- x1 - (f(x1)/fdev(x1))
  while (abs(x2 - x1) < 0.0001) {
    x1 <- x2
  }
  return(x2)
}

newton(1, function(x){cos(x) - x}, function(x) {-sin(x)-1})

## [1] 0.7503639

system.time(newton(1, function(x){cos(x) - x}, function(x) {-sin(x)-1}))

##      user      system elapsed
##         0         0         0

```

The results from the secant function and the Newton-Raphson function are very similar, 0.7650 and 0.7503 respectively. Their running time was also very close, but the Newton-Raphson function was faster by 0.002.

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:

```

craps <- function(n, m, k){
  result <- c(rep(NA,100))
  for(i in 1:k){
    x <- sum(ceiling(6*runif(2)))
    result <- x
    if(x==n || x==m){
      print(cat("Game", i, ":", "result is win"))
      i <- i + 1
    }
    else{
      newroll <- sum(ceiling(6*runif(2)))
      result <- c(result, newroll)
      while(newroll!=x && newroll!=n && newroll!=m){
        newroll <- sum(ceiling(6*runif(2)))
        result <- c(result, newroll)
      }
    }
  }
}

```

```

    }
    if(newroll==x){
      print(cat("Game", i, ":", "result is win"))
      i <- i + 1
    }
    else if(newroll==n || newroll==m){
      print(cat("Game", i, ":", "result is lose"))
      i <- i + 1
    }
  }
  print(result)
}
}

```

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(7,11,3)

```

```

## Game 1 : result is loseNULL
## [1] 4 5 6 8 6 10 5 10 5 8 9 9 5 11
## Game 2 : result is loseNULL
## [1] 6 9 9 11
## Game 3 : result is loseNULL
## [1] 6 7

```

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)

```

craps <- function(n, m, k){
  result <- c(rep(NA,100))
  game <- data.frame(seq(1:k),NA)
  for(i in 1:k){
    x <- sum(ceiling(6*runif(2)))
    result <- x
    if(x==n || x==m){
      game[i,2] <- 1 # 1 for "win"
      i <- i + 1
    }
    else{
      newroll <- sum(ceiling(6*runif(2)))
      result <- c(result, newroll)
      while(newroll!=x && newroll!=n && newroll!=m){
        newroll <- sum(ceiling(6*runif(2)))
        result <- c(result, newroll)
      }
      if(newroll==x){
        game[i,2] <- 1
        i <- i + 1
      }
      else if(newroll==n || newroll==m){
        game[i,2] <- 0 # 0 for "lose"
        i <- i + 1
      }
    }
  }
}

```

```

    }
  }
}
#print(game)
sum(game[,2])
}

for(n in 1:10000){
set.seed(n)
x <- craps(7,11,10)
  if(x ==10){
    print(cat("seed:", n, "10 straight games"))
  }
  else{
    n <- n+1
  }
}

```

```

## seed: 880 10 straight gamesNULL
## seed: 1639 10 straight gamesNULL
## seed: 4352 10 straight gamesNULL
## seed: 4411 10 straight gamesNULL
## seed: 8839 10 straight gamesNULL
## seed: 9085 10 straight gamesNULL

```

```

set.seed(880)
craps(7,11,10)

```

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

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)
length(funs)

```

```
## [1] 1233
```

Using this list, write code to answer these questions.

1. Which function has the most arguments? (3 points)

```

x <- data.frame(seq(1:length(funs)), NA)
names(x) <- c("num", "nargs")
formals(unlist(funs[100])[[1]]) # tryout

```

```

## $target
##
##
## $current
##
##
## $...
##

```

```
##
## $check.attributes
## [1] TRUE

length(formals(unlist(funs[100]))[[1]]))

## [1] 4

for(i in 1:length(funs)){
  x[i,2] <- length(formals(unlist(funs[i]))[[1]]))
  i <- i+1
}

head(x,10)

##      num nargs
## 1      1      0
## 2      2      2
## 3      3      2
## 4      4      0
## 5      5      2
## 6      6      2
## 7      7      0
## 8      8      1
## 9      9      1
## 10    10      0

x$num[which(x$nargs== max(x$nargs))]

## [1] 961

funs[960] # a function named 'scan' has 22 arguments, which is the largest number of arguments.

## $scale.default
## function (x, center = TRUE, scale = TRUE)
## {
##     x <- as.matrix(x)
##     nc <- ncol(x)
##     if (is.logical(center)) {
##         if (center) {
##             center <- colMeans(x, na.rm = TRUE)
##             x <- sweep(x, 2L, center, check.margin = FALSE)
##         }
##     }
##     else {
##         if (!is.numeric(center))
##             center <- as.numeric(center)
##         if (length(center) == nc)
##             x <- sweep(x, 2L, center, check.margin = FALSE)
##         else stop("length of 'center' must equal the number of columns of 'x'")
##     }
##     if (is.logical(scale)) {
##         if (scale) {
##             f <- function(v) {
##                 v <- v[!is.na(v)]
##                 sqrt(sum(v^2)/max(1, length(v) - 1L))
##             }
##         }
##     }
## }
```

```
##           scale <- apply(x, 2L, f)
##           x <- sweep(x, 2L, scale, "/", check.margin = FALSE)
##       }
##   }
##   else {
##       if (!is.numeric(scale))
##           scale <- as.numeric(scale)
##       if (length(scale) == nc)
##           x <- sweep(x, 2L, scale, "/", check.margin = FALSE)
##       else stop("length of 'scale' must equal the number of columns of 'x'")
##   }
##   if (is.numeric(center))
##       attr(x, "scaled:center") <- center
##   if (is.numeric(scale))
##       attr(x, "scaled:scale") <- scale
##   x
## }
## <bytecode: 0x7f9dd3110468>
## <environment: namespace:base>
```

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

```
sum(x$nargs ==0) # 225 functions have no arguments.
```

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

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