Bios 301: Assignment 3

Utsav Kumar

Due Tuesday, 11 November, 1:00 PM

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

50 points total.

Submit a single knitr file (named `homework3.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.

Question 1

15 points

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:

```
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. (15 points)

```
set.seed(100)
game_craps <- function (print.out)
# set print.out as 0 if don't want to print values for each dice roll
{
  counter <- 1
  # If the player wins then games.won is changed to else kept 0
  games.won <- 0
  moves = c()
  x <-sum(ceiling(6*runif(2)))
  x0 <- x
  moves[counter] = x0
  if (print.out != 0)
  {cat ("For dice roll ",counter , " the value is ", moves[counter], "\n")}
  if (x0 == 7 || x0 == 11)
  {
    if (print.out != 0)
    {cat("you win in your first move \n \n" )}
    counter = counter + 1
    games.won <- games.won + 1
  }
  else
```

```r
  {
    while (1)
    {
      x <- sum(ceiling(6*runif(2)))
      counter <- counter +1
      moves[counter] = x
      if (print.out != 0)
      {cat ("For dice roll ",counter , " the value is ", moves[counter], "\n")}
      if (x == x0)
      {
        if (print.out != 0)
        {cat ("you win \n \n")}
        games.won <- games.won + 1
        break
      }
      if (x == 7 || x == 11)
      {
        if (print.out != 0)
        {cat ("you loose \n \n")}
        break
      }
    }
  }
  return (games.won)
}

# Results for first 10 games for the set.seed = 100
play <- c()
no_games <- 3
for (i in 1:no_games)
{
  play[i] <- game_craps(1)
}
```

```
## For dice roll  1  the value is  4
## For dice roll  2  the value is  5
## For dice roll  3  the value is  6
## For dice roll  4  the value is  8
## For dice roll  5  the value is  6
## For dice roll  6  the value is  10
## For dice roll  7  the value is  5
## For dice roll  8  the value is  10
## For dice roll  9  the value is  5
## For dice roll  10  the value is  8
## For dice roll  11  the value is  9
## For dice roll  12  the value is  9
## For dice roll  13  the value is  5
## For dice roll  14  the value is  11
## you loose
##
## For dice roll  1  the value is  6
## For dice roll  2  the value is  9
## For dice roll  3  the value is  9
## For dice roll  4  the value is  11
```

```
## you loose
##
## For dice roll  1  the value is  6
## For dice roll  2  the value is  7
## you loose
##
```

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. (5 points)

```r
# Using the same function as above mentioned and trying to find the seed value
# for which first ten plays results in win
seed.value <- 0
# stores the seed value for which total games won = 10
for (i in 1:1000)
# Loop to find the seed value running till 1000
{
  total.games.won <- 0
  # stroring total games won for the first 10 play
  set.seed(i)
  for (j in 1:10)
  # First ten games are played for selected seed value
  {
    games.won <- game_craps (0)
    # returns 1 if game is won
    total.games.won = total.games.won + games.won
    # counting total games won out of 10
  }
  if (total.games.won == 10)
  # if all games for the first 10 games are won
  {
    seed.value = i
    break
  }
}

cat ("Value of the seed for which first 10 straight games are won =", seed.value)
```

```
## Value of the seed for which first 10 straight games are won = 880
```

```r
set.seed (seed.value)
play.10straight.win.game <- c ()
# setting up the seed for which all first 10 games are won
for (i in 1:10)
{
  # printing the game proceedings so print.out is 1
  play.10straight.win.game[i] <- game_craps (1)
}
```

```
## For dice roll  1  the value is  7
## you win in your first move
##
## For dice roll  1  the value is  8
```

```
## For dice roll  2  the value is  9
## For dice roll  3  the value is  3
## For dice roll  4  the value is  10
## For dice roll  5  the value is  6
## For dice roll  6  the value is  8
## you win
##
## For dice roll  1  the value is  10
## For dice roll  2  the value is  10
## you win
##
## For dice roll  1  the value is  9
## For dice roll  2  the value is  9
## you win
##
## For dice roll  1  the value is  11
## you win in your first move
##
## For dice roll  1  the value is  8
## For dice roll  2  the value is  8
## you win
##
## For dice roll  1  the value is  5
## For dice roll  2  the value is  5
## you win
##
## For dice roll  1  the value is  7
## you win in your first move
##
## For dice roll  1  the value is  9
## For dice roll  2  the value is  9
## you win
##
## For dice roll  1  the value is  7
## you win in your first move
##
```

Question 2

20 points

Code a function that does golden section search, and use this function to find all of the global maxima on the following function:

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ |x| \log\left(\frac{|x|}{2}\right) e^{-|x|} & \text{otherwise} \end{cases}$$

on the interval [-10, 10].

To get an idea of what the function looks like, it might be helpful to plot it.

```
# Defining function for which all maxima has to be found in the range [-10,10]
fx <- function (x)
```

```r
{
  if (x == 0)
  {
    0
  }
  else
  {
    abs(x)*log((abs(x)/2))*exp(-abs(x))
  }
}

# Golden section search method to find maxima
golden.section.search = function (f, minx, maxx, tolerance)
{
   golden.ratio = 2/(sqrt(5) + 1)
   minx0 <- minx
   maxx0 <- maxx
   # Initial points for iteration
   x1 = maxx - golden.ratio*(maxx - minx)
   x2 = minx + golden.ratio*(maxx - minx)

   f1 = f(x1)
   f2 = f(x2)

   iteration = 0

   while (abs(maxx - minx) > tolerance)
   {
      iteration = iteration + 1

      if (f1 > f2)
      # If f1 > f2 it means maxima is between minx and x2
      {
         # Set the new maxx
         maxx = x2
         # Getting the updated iteration points and respective function values
         x2 = x1
         f2 = f1
         x1 = maxx - golden.ratio*(maxx - minx)
         f1 = f(x1)
      }
      else
      {
         # else maximum is between x1 and maxx
         # setting up the new minx and maxx
         minx = x1
         # Getting the updated iteration points and respective function values
         x1 = x2
         f1 = f2
         x2 = minx + golden.ratio*(maxx - minx)
         f2 = f(x2)
      }
   }
```

```r
        # Using the middle point of the final minx and maxx for the answer
        cat('total number of iterations = ', iteration, '\n')
        cat('final minx =', minx, '\n')
        cat('final maxx =', maxx, '\n')
        finalx = (minx + maxx)/2
        cat('point of maxima for function f for the range [',minx0,',',maxx0,' ]
            exists at x =', finalx, '\n')
    }

    # Since the function is even and plotting the graphs shows maxima exists
    # on either side of origin

        golden.section.search (fx, -10, 0, 5e-8)
```

```
## total number of iterations =  40
## final minx = -3.17
## final maxx = -3.17
## point of maxima for function f for the range [ -10 , 0  ]
##            exists at x = -3.17
```

```r
        golden.section.search (fx, 0, 10, 5e-8)
```

```
## total number of iterations =  40
## final minx = 3.17
## final maxx = 3.17
## point of maxima for function f for the range [ 0 , 10  ]
##            exists at x = 3.17
```

Question 3

10 points

Obtain the code for using Newton's Method to estimate logistic regression parameters (`logistic.r`) and modify it to predict `death` from `weight`, `hemoglobin` and `cd4baseline` in the HAART dataset. Use complete cases only. Report the estimates for each parameter, including the intercept.

Note: The original script `logistic_debug.r` is in the exercises folder. It needs modification, specifically, the logistic function should be defined:

```r
logistic <- function(x) 1 / (1 + exp(-x))
```

```r
    # Reading data
    data <- read.table("~/Documents/BIOS301/Bios301/datasets/haart.csv", sep=",", head=T)

    # Logistic function
    logistic <- function(x) 1 / (1 + exp(-x))

    # extracting the required columns
    x1 <- data[,c("cd4baseline","weight","hemoglobin")]
    y1 <- data[,c("death")]

    # finding dimensions
```

```r
  n <- dim(x1)[1]
  k <- dim(x1)[2]

  x1 <- as.matrix(x1)
  y1 <- as.matrix(y1)

  # removing those datasets which are not complete
  x <- na.omit(x1)
  # defining y to the column size of x
  y <- rep(1,nrow(x))

  estimate_logistic <- function(x, y, MAX_ITER=10)
  {

    n <- dim(x)[1]
    k <- dim(x)[2]

    x <- as.matrix(cbind(rep(1, n), x))
    y <- as.matrix(y)

    # Initialize fitting parameters
    theta <- rep(0, k+1)

    J <- rep(0, MAX_ITER)

    for (i in 1:MAX_ITER)
    {

      # Calculate linear predictor
      z <- x %*% theta
      # Apply logit function
      h <- logistic(z)

      # Calculate gradient
      grad <- t((1/n)*x) %*% as.matrix(h - y)
      # Calculate Hessian
      H <- t((1/n)*x) %*% diag(array(h)) %*% diag(array(1-h)) %*% x

      # Calculate log likelihood
      J[i] <- (1/n) %*% sum(-y * log(h) - (1-y) * log(1-h))

      # Newton's method
      theta <- theta - solve(H) %*% grad
    }

    return(theta)
}

estimate_logistic(x, y)
```

```
##                  [,1]
##             1.120e+01
## cd4baseline -5.378e-17
```

```
## weight       1.127e-14
## hemoglobin   3.819e-14
```

Question 4

5 bonus points

Import the `addr.txt` file from the GitHub repository. This file contains a listing of names and addresses (thanks google). Parse each line to create a data.frame with the following columns: lastname, firstname, streetno, streetname, city, state, zip. Keep middle initials or abbreviated names in the firstname column. Print out the entire data.frame.

```
 nc<-max(count.fields("~/Documents/BIOS301/Bios301/datasets/addr.txt", sep=""))
x<-read.table("~/Documents/BIOS301/Bios301/datasets/addr.txt", sep="",
              col.names=paste("v",1:nc,sep="."),fill=T, )
x.df <- data.frame(x)
print (x.df)
```