DS Masters Statistical Computing Assignment 2

Ndivhuwo Nyase!

06 March 2023

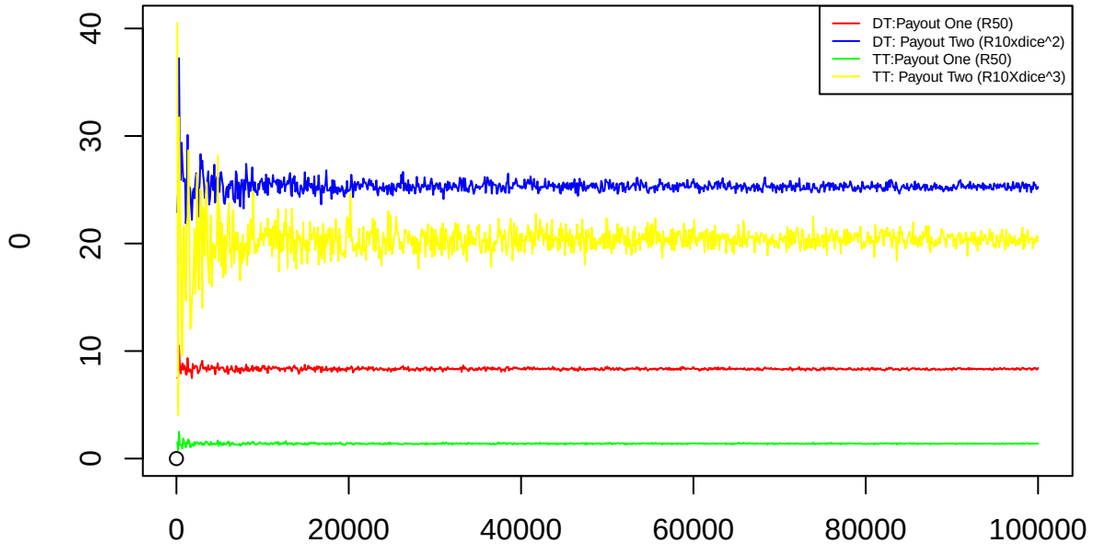
Question 1 [12]

Should you play the 'Double trouble' or 'Triple threat' dice game? Both games cost R1 to play. To play Double trouble, you roll two six-sided dice at the same time. To play Triple threat, you roll three six-sided dice at the same time. You win whenever all of the dice display the same number e.g. (1,1) or (4,4) for Double trouble, or (2,2,2) or (3,3,3) for Triple threat.

There are two payment options for both of these games:

- R50 each time you win, or
- R10 x (product of numbers on die) e.g. if you win in Double trouble by rolling two 4's, your payout is R10 x 4x4 = R160. If you win by rolling three 2's in Triple threat, your payment is R10 x 2x2x2 = R80.

a. Write functions to simulate each of these games `nsim` times, calculating the two payouts for each game for each time you simulate the game (i.e. you should end up with four payout vectors, two for each game, with lengths equal to `nsim`). Replicate simulations of these games, for a sequence of `nsim` values from 100 to 100 000, in increments of 100. Calculate the mean of the payout for each of these replications. For example, when `nsim` = 1000 that means you play each game 1000 times and calculate the mean of the two payout options for each game, and end up with 1000 sets of mean payouts for each of the two payout options for each game. Make a line plot that displays the mean of each payout option across the two games as a function of the number of times each game is played. Your plot should clearly indicate which line corresponds to which combination of game and payout option. Comment on what can be deduced about the two games and payout options. [6]




```
## [1] 20.34597
```

The double trouble and triple threat payout of R50 have the smallest expected returns at all n_simulations as compared to the other payouts. Therefore, one should not play the games with the R50 payout for both games.

With that being said, the games in which the payout is multiplied by the dices are more profitable.For the Triple Threat payout at lower n_sims it is possible to get more lucky or unlucky and as the means payouts vary from 10 to 32. Thus illustrating a higher risk and higher reward stakes game. However due to the central limit theorem that states the mean of a sample of data will be closer to the mean of the overall population in question, as the sample size increases. This is shown small sample sizes of the triple threat payout (R10XDICE^3) can vary from 10 to 32. The sample size seems to converge to around 20.

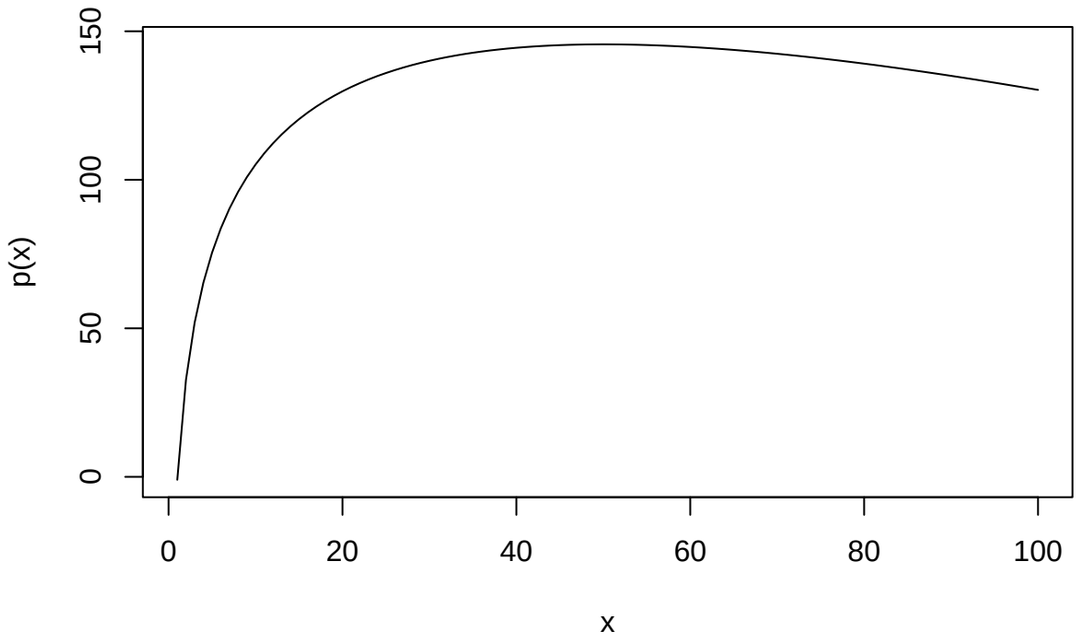
This makes the best game with the best payout double trouble(R10*DICE^2) as the expected average for all n_sim hovers around 25 which is the highest expected payout for all games.

Now suppose that the payout for Double trouble is a function:

$$p(x) = 50log(x) - x$$

where x is the number of throws of the dice.

b. Graph the function, for $x \epsilon [1; 100]$ [1]



c. Use `optim` to determine the optimal number of throws to maximise the payout. Start with x = 1 throws. [1]

```
## $par
## [1] 50
##
## $value
## [1] 145.6012
##
## $counts
## function gradient
##       48       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
## [1] "Optimal Amount of throws to maximize payout is :  50"
```

d. Write a function to use the Newton-Raphson method to find the optimum number of throws to maximise payout. Start with x = 1 throws. Your function should return the number of throws, optimal value and number of iterations taken to converge to the optimal solution. [4]

*Hints: $\frac{d}{dx}50log(x+1) - x = \frac{50}{x+1} - 1$ and $\frac{d}{dx}\frac{50}{x+1} - 1 = \frac{-50}{(x+1)^2}$

```
## [1] "Optimal Amount of throws to maximize payout is :  49"
```

```
## [1] "Optimal Value :  145.591"
```

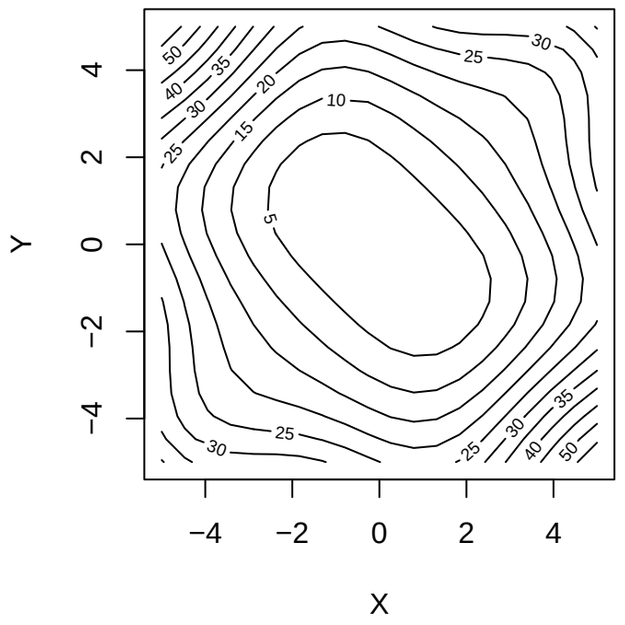
```
## [1] "Number of Iterations :  10"
```

Question 2 [13]

Suppose that you are trying to minimize the following function:

$$f(x, y) = x^2 + y^2 + xsin(y) + ysin(x)$$

a. Graph this function over the domains $x \epsilon [-5; 5]$ and $y \epsilon [-5, 5]$ using the `outer` and `contour` functions and an increment of 0.1 between successive values of x and y [2]



b. Find the minimum of the function using `optim` with the Nelder-Mead method. Use (-4,-4) as the starting values for x and y. [2]

```
## $par
## [1]  0.03595196 -0.03618544
##
## $value
## [1] 0.0000006186339
##
## $counts
## function gradient
##       75       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Gradient descent is a numerical optimization routine that is often used to obtain the parameters that minimize a multi-parameter function. The algorithm is an iterative one. If we define

$$Param_i = (a^{(i)}, b^{(i)})^T$$

as the value of a 2×1 vector that contains the estimated optimal values of x and y at iteration i, the algorithm updates the parameter estimates using the vector $Param_i$ as well as the gradient of the function `f(x,y)` at $Param_i$ (i.e. $f'(Param_i)$) using the following update rule:

$$Param_{i+1} = Param_i - \alpha f'(Param_i),$$

where α is some constant representing the learning rate.

c. Write an R function to calculate the gradients of `f`, where $\frac{df}{dx} = 2x + siny + ycosx$ and $\frac{df}{dy} = 2y + xcosy + sinx$. Return the answers as a two-dimensional vector. Name the function `grad`. [2]

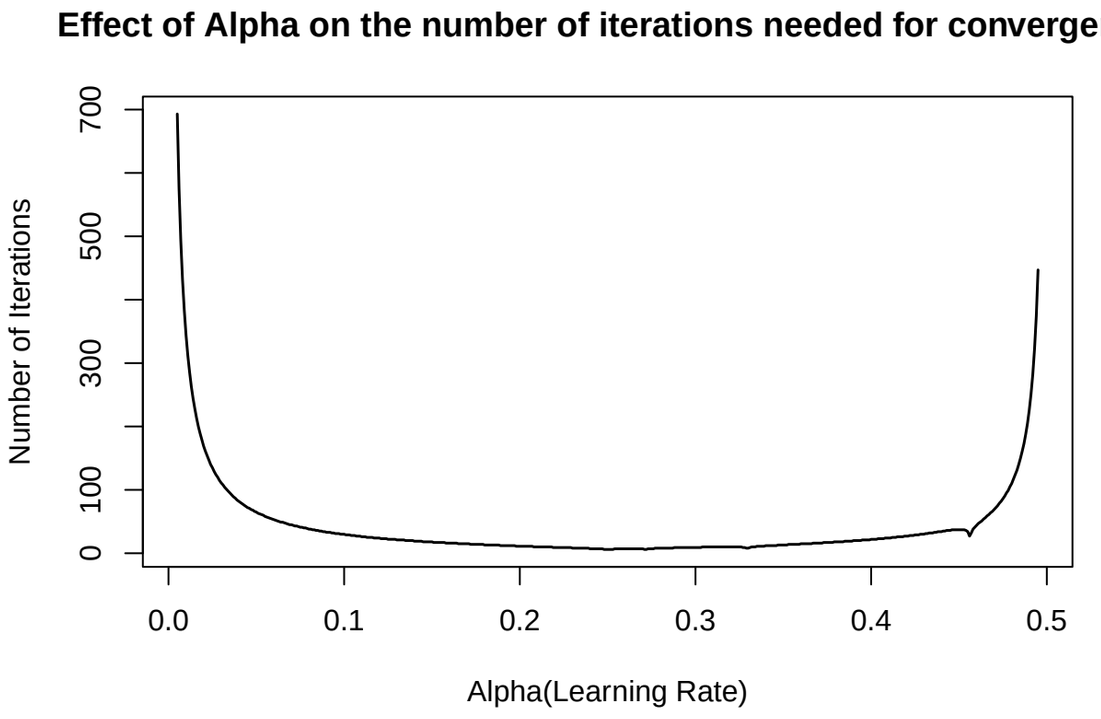
d. Now, write your own gradient descent algorithm. Name the function `grad_desc`. Set $\alpha = 0.1$. Your function should contain the following arguments, `params`, `max_iters`, `tol` and `alpha`, where `max_iters` is the maximum allowable

number of iterations used to obtain the optimal parameter vector estimates. Include a stopping criterion that terminates the algorithm once both gradients are within `tol` = 0.0001 of 0. Set the maximum number of iterations as 1000. Use (-4,-4) as the starting parameter values. Print out the optimal parameter values, the value of the function at these parameter values, the gradients, and the number of iterations taken for the algorithm to converge. If the algorithm does not converge within `max_iters`, terminate the algorithm and print a message that indicates that the algorithm did not converge within `max_iters` iterations. [4]

```
## [1] "Optimal Parameter Values = x:  -0.00001553248 , y:  -0.00001553248"
## [1] "Optimal Value of the function:  0.00000000097"
## [1] "Gradient of xy at optimal values = x: -0.0000621299 , y: -0.0000621299"
## [1] "Number of iterations to optimal parameters :  30"
```

```
## [1] 30
```

e. Using your gradient descent algorithm above, write a program to investigate the effect of using different learning rates, ranging from $\alpha = 0.005$ to $\alpha = 0.495$ in increments of 0.001. Use the same values for all the other parameters of the algorithm as above. Record the number of iterations taken to reach convergence for each value of α. Plot a line graph of the number of iterations to reach convergence as a function of the learning rate. Title the graph appropriately. [3]



Question 3 [10]

The figure below is of the game 'tic tac toe' or 'noughts and crosses'. The game is played by two players, one who enters 'X's and another who enters '0's. The players take turns in choosing which of the nine blocks to enter their symbol, with the goal being to win the game by having three of their symbols in a row (either in a straight line or along a diagonal). If neither player manages to get three of their symbols in a row before all nine blocks have symbols, the game ends in a draw.

Write an R program in order to simulate the above game 10000 times.Your program should do the following:

- Randomly select which player starts first
- Randomly populate the blocks, one at a time, with each player's symbol
- Stop the game as soon as one of the players has won, or when all blocks have been filled (i.e. the game has been drawn)
- Record which player has won or whether the game ended in a draw

Print the proportion of times each of the three results of the game happens.

*Hint: denote each of the blocks with a number, from 1 to 9, as in the diagram below:

```
counter<-tictactoe(10000)
data.frame(table(counter)/10000)
```

```
##         counter   Freq
## 1         Draws 0.0339
## 2 Player 1 Wins 0.4802
## 3 Player 2 Wins 0.4859
```