STU22004 - Applied Probability I · Group Project

Member and Contribution

- Prab Hiranayachatri, 20336871

  Solve and write the report on the question 1 and 4

  Format and check the overall report

- Dmitry Kryukov, 20336877

  Set up discord and start discussing the work

  Solve and write the report on the question 2 and 3

- Kostiantyn Ohorodnyk, 20336895

  Resolve and check the answers

  Add the part beyond the module, Additional Computations

Question 1

A 12-sided die is rolled continuously until all the possible outcome have occurred at least once

Estimate the expected number of dice rolls needed using a simulation study.

Simulation

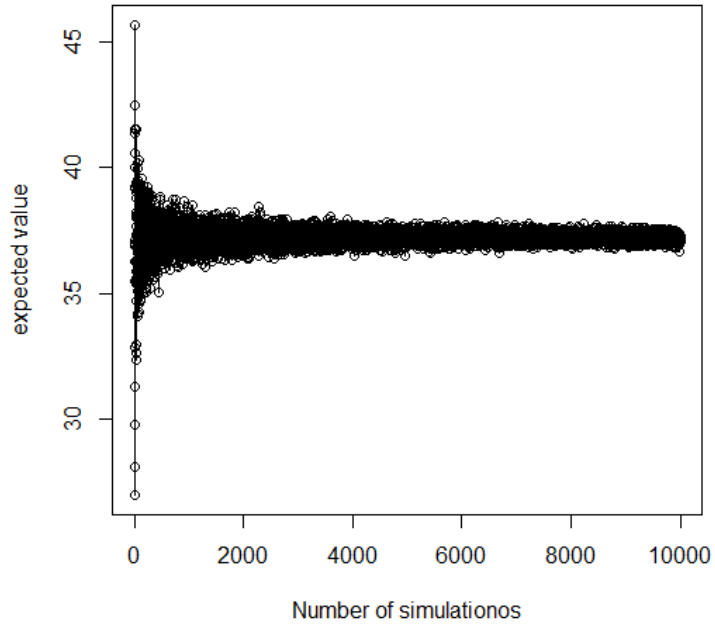
For the simulation, we decide to use R as the simulation tool. We simulated situation and get the average number of rolls needed as commented in the code.

```
nSimulation <- 10000 # assign the number of simulations wanted
sum <- 0
for (i in 1:nSimulation){
  faceTurned <- vector("logical", 12) # initialised 12 false boolean
  dice <- 1:12 # make a 12-faced dice (1 to 12)
  count <- 0
  while(sum(faceTurned) != 12){
    count <- count + 1 # count the number of rolls done
    rolledFace <- sample(dice, 1) # roll the die
    faceTurned[rolledFace] = TRUE # switch the boolean of the face which been rolled to be
  }
  sum <- sum + count # add the number of rolls used to the sum of every simulation
}
expectedNumber <- sum / nSimulation # get the average number of rolls by dividing the sum
print(expectedNumber)
```

Additional Computation

To confirm that the value given by running 10000 repetition is reliable, we run simulations with different number of repetitions and compare the result by graph.

As shown below, the results converge to a point in the range between 35 and 40, which is not clearly shown in 10,000 simulations. From which, it can be concluded that the true value, which the results converge to, is in the range.



Compute the expected number of dice rolls needed analytically.

We compute the expected number of rolls to be taken to see a new face when different numbers of faces have been seen.
As the case falls into geometric distribution, find the probability of the first success to be in a specific order when each trial has a constant independent probability, the expected value of each is $\frac{1}{p}$

Face seen	Probability	Expected value
0	$^{12}/_{12}$	$^{12}/_{12}$
1	$^{11}/_{12}$	$^{12}/_{11}$
2	$^{10}/_{12}$	$^{12}/_{10}$
3	$^{9}/_{12}$	$^{12}/_{9}$
4	$^{8}/_{12}$	$^{12}/_{8}$
5	$^{7}/_{12}$	$^{12}/_{7}$
6	$^{6}/_{12}$	$^{12}/_{6}$
7	$^{5}/_{12}$	$^{12}/_{5}$
8	$^{4}/_{12}$	$^{12}/_{4}$
9	$^{3}/_{12}$	$^{12}/_{3}$
10	$^{2}/_{12}$	$^{12}/_{2}$
11	$^{1}/_{12}$	$^{12}/_{1}$

$$sum = \frac{12}{12} + \frac{12}{11} + \frac{12}{10} + ... + \frac{12}{1} = 37.2385 \tag{1}$$

A pair of 6-sided fair dice are rolled continuously until all the possible outcome (i.e. all possible sums of two dice, 2, 3, ..., 12) have occurred at least once. Estimate the expected number of dice rolls needed using a simulation study.

For the simulation,

```
nSimulation <- 10000

# assign a vector value representing 6-sided to the two dice
```

```r
dice1 <- c(1:6) # dice1
dice2 <- c(1:6) # dice2

sumCount <- 0 # declare a variable to count the number of times used
for (i in 1:nSimulation){
  sumSeen <- vector("logical", 11) # declare 12 boolean to note if the sum has been seen
  count <- 0
  while(sum(sumSeen) != 11){ # loop until al the 12 sums are seen
    count <- count  + 1 #iterate count
    rolled1 <- sample(dice1, 1) # roll dice 1
    rolled2 <- sample(dice2, 1) # roll dice 2
    sumSeen[rolled1 + rolled2] = TRUE # set the sum to be seen
  }
  sumCount <- sumCount + count # add the number of times used to the sum
}

expectedValue <- sumCount/nSimulation # compute the average rolls
print(expectedValue)
```

```r
nSimulation <- 10000

# assign a vector value representing 6-sided to the two dice
dice1 <- c(1:6) # dice1
dice2 <- c(1:6) # dice2

sumCount <- 0 # declare a variable to count the number of times used
for (i in 1:nSimulation){
   # declare 12 boolean to note if the sum has been seen
  sumSeen <- vector("logical", 11)
  count <- 0
  while(sum(sumSeen) != 11){ # loop until al the 12 sums are seen
    count <- count  + 1 #iterate count
    rolled1 <- sample(dice1, 1) # roll dice 1
    rolled2 <- sample(dice2, 1) # roll dice 2
    sumSeen[rolled1 + rolled2] = TRUE # set the sum to be seen
  }
  sumCount <- sumCount + count # add the number of times used to the sum
}

expectedValue <- sumCount/nSimulation # compute the average rolls
print(expectedValue)
```

Question 2

A deck of 100 cards - numbered 1, 2, ..., 100 - is shuffled and then turned over one card at a time. We say that a "hit"occurs whenever card i is the ith card to be turned over, $i = 1, ..., 100$. Simulate 10000 repetitions of the game to estimate the expectation and variance of the total number of hits.

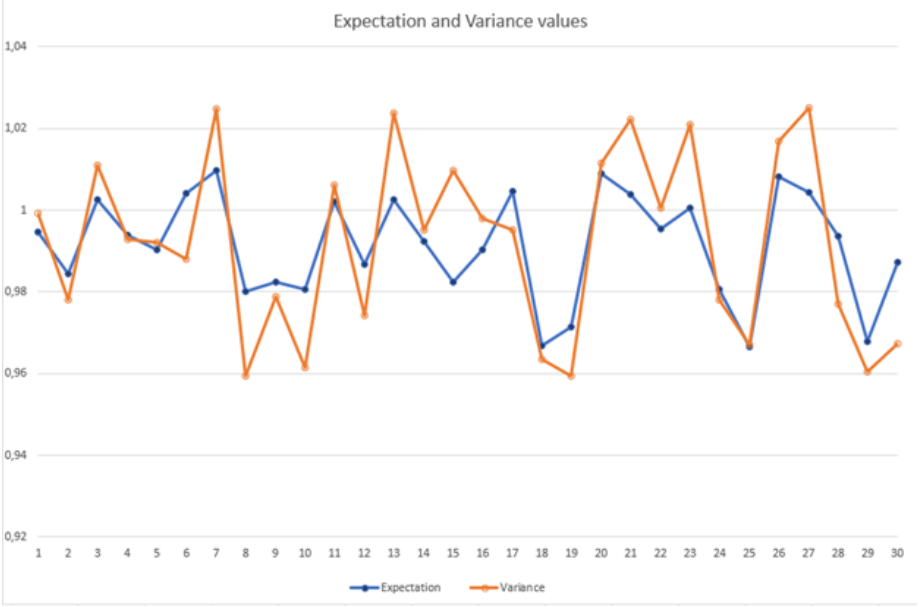
Simulation

As written in the task, we are simulating 10000 repetitions of the game in a for loop. Then, we are creating a deck of 100 cards and a variable which holds an amount of hits found. Then the second loop is created to go through the deck; there we take a card 1 by 1 and check whether "hit" occurs or not, and if so, incrementing nHit variable by 1. Then, to estimate the expectation and variance of the total number of hits, we need to store total number of hits in a vector nHitVector. In the end of both loops, firstly, we are computing the average of hits get (mean(nHitVector), which is our expectation, and secondly, variance is being calculated using var(nHitVector). The average expectation fluctuates between 0.94 – 1.01, while variance being in range of 0.95-1.03.

```
nHitVector <- c()
# run the simulation 10000 times
for (i in 1:10000){
  deck <- 1:100 # make a deck of 100
  nHit <- 0 # count the number of times hit is found
  for (j in 1:100){ # go through the deck
    turnedCard <- sample(deck, 1) # get one card from the deck
    deck <- setdiff(deck, turnedCard) # take the card out of the deck
    if (turnedCard == j){ # check if there is a hit
      nHit <- nHit + 1
    }
  }
  nHitVector <- c(nHitVector, nHit) # add hit to the record
}
expectedHit <- mean(nHitVector) # get the average of hits get
varianceHit <- var(nHitVector)
print(expectedHit)
print(varianceHit)
```

Additional Computations

To compare and visualize the difference in final variance and expectation got after running our code several times, we can plot those variables on the line chart. In this graph, the code was run 30 times, outputting different values.



In the graph above, we can see both values being in the approximate range of 0.95 – 1.03, and the more instances being run, the more precise would be the plotted graph. In addition, it will be correct to notice that on the graph both values have no more than 0.3 difference in a certain iteration chosen.

Question 3

Consider the following game: you begin with $20. You flip a coin, winning $10 if the coin lands heads and losing $10 if the coin lands tails. The game is played continuously unttil you either go broke or have $100(i.e. a new profit of $80). Estimate using simulation studies

Simulation

For this question we ran the simulation 1000 times and counted the number of coin flips made throughout the game, setting it to 0 every new simulation. If the flipped side that we get after the sample() function is HEAD, then we add 20 to the player's wallet, opposite for TAIL. If money goes above or reaches 100, the game ends and the counter of won games is increased by 1. If it reaches 0, then the game ends. Before the start of the new game, we check if the counter of coin flips is less or equal to 10 and increase the counter for FinishWithin10 by 1.

The probability you win the game.

After all the simulations finished, we divide the number of won games by the number of simulations, therefore getting the probability of winning the game, which across several runs is always around 0.2

The probability the game ends within ten coin flips.

Same computations were made for the probability the game ends withing ten coin flips. We divided the FinishWithin10 counter by the number of simulations made. In the end, after several runs, we get around 0.56

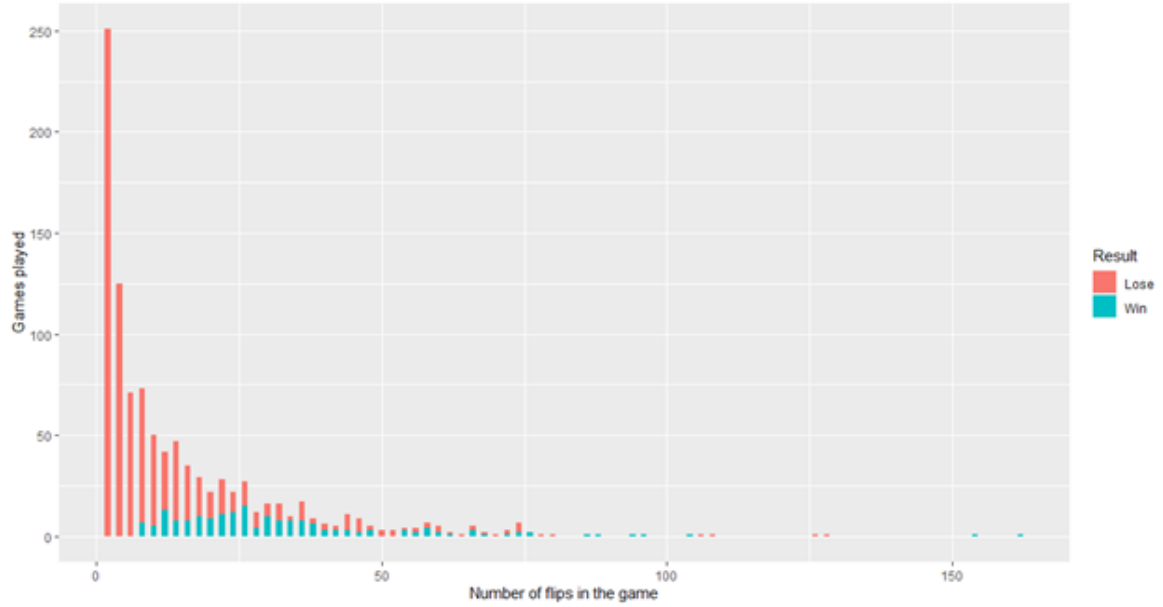
Code

```
library(ggplot2)

nSimulation <- 1000
BET <- 10
HEAD <- 1
TAIL <- 2
coin <- c(1,2)
nFlipVector <- c()
Result <- c()
nFinishWithin10 <- 0
for (i in 1:nSimulation){
  nCoinFlip <- 0
  endGame <- FALSE
  money <- 20
  while (!endGame){
    nCoinFlip <- nCoinFlip + 1
    flipedSide <- sample(coin, 1)
    if (flipedSide == HEAD){
      money <- money + BET
    }
```

```
    else if (flipedSide == TAIL){
      money <- money - BET
    }
    if (money >= 100){
      Result <- c(Result, "Win")
      nFlipVector <- c(nFlipVector, nCoinFlip)
      endGame = TRUE
    }
    else if (money <= 0){
      Result<- c(Result, "Lose")
      nFlipVector <- c(nFlipVector, nCoinFlip)
      endGame = TRUE
    }
  }
  if (nCoinFlip <= 10){
    nFinishWithin10 <- nFinishWithin10 + 1
  }
}
winningProb <- (sum(Result == "Win")/nSimulation)
finishIn10Prob <- (nFinishWithin10/nSimulation)
print(winningProb)
print(finishIn10Prob)

qplot(nFlipVector, fill = Result, xlab = "Number of flips in the game",
      ylab = "Games played", binwidth = 1)
```

Additional Computations

We plot the histogram using R to see the number of flips it takes for each game, and the results of the games. The histogram width is set to be one. So, each number of flips is plotted individually. However, there are only 25 bars shown in the range between 0 and 50, as it is not possible for the game to be ended within odd number of flips. According to the histogram, it is suggested that if the game is finished in a few number of flips, less than 26, it is more likely to be a loss.



Question 4

Consider a coin tossing game between two players. The first player picks a combination of heads (H) and tails (T) of length 3 (e.g. HHH, HTH) that they think will come up. The second player subsequently picks a different combination of heads and tails of the same length. The coin is then flipped continuously until a player completes their combination of heads and tails. For each of the 8 combinations player 1 picks (i.e. HHH, HTH, HHT, etc.), which combinations should. player 2 pick to maximise the chance of winning? For each of the 8 combinations player 1 picks, estimate the chance of winning for player 2 using the optimal strategy.

Solution

In order to find the best combination for the player 2 we checked 1000 simulations for each possible combination of two different combinations from both players and showed everything in a table.
First, we made an array of all possible combinations. Then, for each item in the array we made another array, without the original item and ran 1000 simulations with it. Each simulation has an array with 3 items, shifting the array to the left for each coin flip and adding the result to the end of it. If at any point the sequence is the same as the chosen combination of any player, the game ends, and if the winner is Player 2, then we increase the counter.
After every pair, we add the probability of player 2 winning to the table.

Player 1	Choice to pick
HHH	THH
HHT	THH
HTH	HHT
HTT	HHT
THH	TTH
THT	TTH
TTH	HTT
TTT	HTT

Probability Table

Below is the probability table which shows the probability of the second player winning where the rows represent the first player's choice and the columns represent the second's player as shown in the key table below it.

	1	2	3	4	5	6	7	8
1	0.000	0.523	0.588	0.585	0.865	0.590	0.712	0.503
2	0.486	0.000	0.324	0.345	0.730	0.383	0.511	0.277
3	0.383	0.655	0.000	0.514	0.500	0.485	0.630	0.449
4	0.405	0.672	0.497	0.000	0.513	0.512	0.234	0.128
5	0.123	0.237	0.522	0.491	0.000	0.510	0.635	0.385
6	0.432	0.628	0.497	0.512	0.516	0.000	0.662	0.393
7	0.294	0.506	0.385	0.721	0.326	0.334	0.000	0.482
8	0.491	0.702	0.575	0.881	0.586	0.584	0.505	0.000

Key	Choice
1	HHH
2	HHT
3	HTH
4	HTT
5	THH
6	THT
7	TTH
8	TTT

Code

```
nSimulation <- 1000

coin  <- c(1,2)
HEAD <- 1
TAIL <- 2

choices <- list()
choices[[1]] <- c(HEAD, HEAD, HEAD)
choices[[2]] <- c(HEAD, HEAD, TAIL)
choices[[3]] <- c(HEAD, TAIL, HEAD)
choices[[4]] <- c(HEAD, TAIL, TAIL)
choices[[5]] <- c(TAIL, HEAD, HEAD)
choices[[6]] <- c(TAIL, HEAD, TAIL)
choices[[7]] <- c(TAIL, TAIL, HEAD)
choices[[8]] <- c(TAIL, TAIL, TAIL)

probTable <- matrix(rep(0, times = 64), ncol = 8, byrow = TRUE)
colnames(probTable) <- c(1:8)
rownames(probTable) <- c(1:8)

for (i in 1:length(choices)){
  choiceLeft <- setdiff(c(1:8), i)
  for (j in choiceLeft){
    nPlayer2Win <- 0
    for (k in 1:nSimulation){
      endGame <- FALSE
      flipedSeq <- c(0,0,0)
      while(!endGame){
        flipedSide <- sample(coin, 1)
        flipedSeq <- c(flipedSeq[2:3], flipedSide)
        if (identical(choices[[i]], flipedSeq)){
          endGame <- TRUE
        }
        else if (identical(choices[[j]], flipedSeq)){
          nPlayer2Win <- nPlayer2Win + 1
          endGame <- TRUE
        }
      }
```

```
    }
    player2WinningProb <- nPlayer2Win/nSimulation
    probTable[i,j] <- player2WinningProb
  }
}
```