Adrian Loekman
UID: 105785820
CEE/MAE M20
November 9, 2021

HOMEWORK 6

1. Monte Carlo Simulation for Decision Making

1.1. Introduction

In this problem, we are introduced a scenario of a basketball. The main question from this problem is: if "my team" was down by 3 points with 30 seconds remaining, which decision provides the more chance of winning, taking a three point and go to overtime or take the two quickly and foul the opponent, hope the opponent doesn't make at least 1 free throw and get the rebound, and do a 2-point buzzer beater? This question can be answered using Monte Carlo Simulation where we will use 300, 1000, 2000 repeating scenarios. Using those and the given parameters, we can solve the win percentage from both decision from the number of simulations. We can compare those and choose the best decision.

1.2. Model and Methods

In this problem, I will use a pseudocode in order to make the illustration of the problem easier. The main model is to use two functions, one for the three point strategy and another for the two point strategy. We need to make the initial assumption that only three point attempts are made during the first strategy and two point attempts on the second strategy.

Main Script:

```
List all the percentages for part a(shooting, rebounding, freethrow, overtime)
List all simulations as an array

win1 = calculate win by calling function takeThree.m using given parameters
winPercentage1 = win1./simulations * 100%
Print Results

win2 = calculate win using two point strategy using given parameters
winPercentage2 = win2./simulations * 100%
print results.

For part b,
List new percentages, same parameters need not to be written twice
Call functions takeThree and takeTwo, then calculate win percentage using each strategy
and print the outputs similar to part a.
```

Function takeThree.m:

```
Win,lose = function with parameters
```

```
Initiate array for wins and losses for each simulation (1,3)
For I = 1:3
        Simulation(1/2/3)
n=1
start while loop as n <= simulation
if three point made
        go to overtime
        win in overtime or lose in overtime
else lose if three point is not made
end
update iteration
end
```

Function takeTwo.m :

```
Function handles
Initiate arrays for wins and losses
For I = 1:3
Simulation
m = 1 start iteration
start while loop as m <= simulation
time remaining
does our team have possession? (true/false)
start while loop for time remaining is more than 0
if possession is ours (true)
        downBy (initially 3)
        if two point is made
                team downBy – 2
                time reduced to either 24 seconds or subtract the time by 5 depending on
         time
         possession goes to opponent (false)
         end
end
if down by <=2, we go for the two point buzzer beater, end
if opponent has possession (false)
time is subtracted by 5 (foul)
opponent takes free throw
if 2nd free throw is not good, get the chance to rebound
        possession either true or false depending on who gets rebound
end

calculate wins depending on downBy (>0 is a loss, <0 is a win, = 0 goes to overtime and
either win or loss).
Update win/lose
Update iteration
```

From the pseudocode above, it can have even more “ifs” since the game of basketball can vary and have more options for decisions. A couple of assumptions are made to ease the model of the functions and script. More details about the model, calculations, and result will be discussed in section 1.4.

1.3. Calculations and Results

The code produces a set of statements when it’s being run. The numbers and the simulations here are key to choosing the decision.

```
The Results From Part a are:
Win From Three Point with 300 Simulations: 15.67%
Win From Three Point with 1000 Simulations: 16.50%
Win From Three Point with 2000 Simulations: 17.55%

Win From Two Point with 300 Simulations: 3.33%
Win From Two Point with 1000 Simulations: 4.40%
Win From Two Point with 2000 Simulations: 4.30%

The Results From Part b are:
Win From Three Point with 300 Simulations: 10.67%
Win From Three Point with 1000 Simulations: 14.60%
Win From Three Point with 2000 Simulations: 12.65%

Win From Two Point with 300 Simulations: 20.00%
Win From Two Point with 1000 Simulations: 15.70%
Win From Two Point with 2000 Simulations: 14.60%
```

It is very interesting when the parameters are changed slightly, the outcome of the match becomes very different and thus might change the decision when this event occurs.

1.4. Discussion

First, to answer the question stated in the problem statement, the probability of winning using the parameters in part a is much higher using the three-point strategy than the two point. However, when the two-point, three point, and free throw percentage is changed for part b, it looks like taking the two point has a slightly higher chance of winning than the three point. The reason behind this might be the increase in two point percentage and decrease in three point and opponent free throw, giving the team more probability in winning the match using two point strategy.

An interesting point in this simulation is the number of simulation it takes. In other words, when a basketball team is losing by 3 points with 30 seconds left, the probability is already very small (no higher than 20%). Also note that there are 300, 1000, and 2000 simulations, which in practice means this is with a ratio of 300, 1000, or 2000 matches. This looks insane considering a team winning the NBA finals can only play a maximum of much less than those numbers.

The strategy might seem important in this scenario but there are many assumptions being made to the simulation. For example, we do not know how the opponent is defending or if the opponent shoots when they get an offense. If this happens than our team may be down by more than 30 and just lose before taking the chance. So, this case is much simplified otherwise would be cumbersome to code it.

In conclusion, the three point strategy works better with parameters in part a but the two point strategy has a slightly better chance of winning in part b. Note that many assumptions are being made and these percentages may vary in real life. This is a good example from a real life situation that can be calculated mathematically through probability. This is very interesting since it already provides us with an idea. In addition to that, I learn to set attention to details since even simplified, there's so much scenarios I need to take into account to make the simulation work.

2. Implementing Customized Probability Distributions

2.1. Introduction

In this problem, we are given a function p(x) for a certain domain. From the given function, we can implement a method to draw samples from the method. The random sample will be repeated for n simulations and output a histogram of the customized probability density function, which is not in uniform nor normal distribution. In other words, when we draw a random sample from p(x), which should be in the domain, what is the probability of getting each sample in the domain?

2.2. Model and Methods

First, we need to do some calculations by hand. We need to find the value of P(x) using the information from p(x). We know the value of P(x) is the integral of p(x) from -Inf to x. Using cumulative probability density functions, we can find P(x<0), P(0<=x<=3), and P(x>3). If we add these up, we should get the value 1. This aligns with the requirement where $\int_{-\infty}^{\infty} p(x)dx = 1$. From this, we can evaluate $P(x) = \int p(0 \leq x \leq 3)dx$ which is $P(x) = \int -\frac{2}{9}x + \frac{2}{3}dx = -\frac{1}{9}x^2 + \frac{2}{3}x$. Take the inverse of the integral so we have the form x = P^{-1}(y).
Next, we can write the function `myRand.m` in part a. This function takes requires no input (just like rand) because it automatically draws a random sample from the customized probability distribution function. Inside the function we first need to draw a random sample on [0,1] and set this to y. Then, using x = P^{-1}(y) we can find x, which is our output. Since $P(x) = -\frac{1}{9}x^2 + \frac{2}{3}x$ we can find the inverse which is $3 \pm 3\sqrt{1-x}$. We will use $3 - 3\sqrt{1-x}$ because if we test run myRand function, it will correctly give us a random sample at [0,3]. The other would output a random sample not in the correct domain.
We can write up our main script now. This simulation uses 11000 samples so we need to preallocate an array containing 11000 data. Then, use a for loop with 11000 iterations where each loop generates a random sample from myRand.

After all the data is obtained, a histogram can be created using histogram with pdf normalization property

```
h = histogram(P,'Normalization','pdf');
```

In addition to that, we can also look at the main function p(x) and see how the distribution is affected by the main function. The plot of the function and histogram can be plotted in a single graph for better observations

2.3. Calculations and Results

When the main script is run, a plot of histogram and line is generated. We can observe the histogram and the plot of the function p(x)

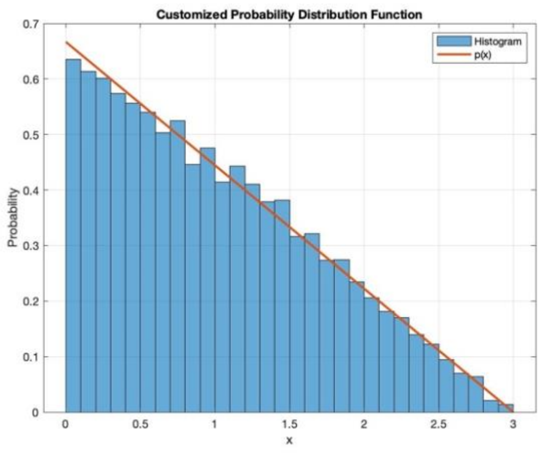

Figure 1 Customized Probability Function from p(x)

From the observation, we can see that the plot of histogram and the function is closely related. We can discuss this in the section below.

2.4. Discussion

From the output given in section 2.3, we can see the probability density distribution of the function is very closely related to the function itself. To understand better, I assume we ask the computer to generate a number from 0 to 3. What is the probability of each number from that range to be outputted? Unlike uniform distribution, where the probability gets higher in the middle, we have a customized distribution from a function. This function would then tell us how the probability distribution would look like. We can estimate by taking extra attention to the function at range 0 to 3. The point when x(0) is 2/3 and x(3) is 0. In addition to that, we can see from the equation that this function has a negative slope. Thus, we can estimate that the probability it would output a number close to zero is 2/3 and zero probability to output a number close to 3. Using the histogram, we can see that the estimate matches the probability distribution function. We can observe that the probability of the random sample outputting a value close to 0 is much more likely than a value close to 3 (It is not exactly zero, but the probability is less than 0.1). This tells us that the probability function will determine the probability distribution function and thus would determine the probability of an output being displayed from the random sample.

This histogram is a good approximation of the probability density function since it closely tells readers, with the help of the histogram, that the probability decreases along each random sample. In other words, the probability is 0 to generate a value from -Inf to 0, decreases from 2/3 to 0 at [0,3], and 0 probability to generate a value from 3 to Inf.

3. Birthday Paradox

3.1. Introduction

The birthday paradox tells us that from 23 random people in a cup match, two of them had the same birthday. The question then arises, how many people is needed to have at least the probability of 50% two of those people have the same birthday? We assume that there are 365 days in one year. This random drawing will be done from 10 – 100 people for a number of cycles (3000). From the plot, we can verify the number of people with probability of 50% using Poisson Approximation.

3.2. Model and Method

First, we can put the number of simulations under the variable `sim`. The reason is that whenever we need to change the number of simulation cycles, we would only need to change one value and does not disturb the overall code. Variable n is the number of people, we don't know how many we need to get 0.5 probability so we can scale up from 10 to 100. For each n (this requires a for loop), we will do a for loop for the number of simulation cycles. Inside the for loop, a random number from 1 to 365 is generated in n columns.

```
birthday = unidrnd(365,people,1);
```

The code above draws a random integer sample from 1 to 365 for people needed. Then, from n data, is there a number that is outputted twice? If there is then we assume the 2 persons have the same birthday. In order to check that, we will use a function called `unique`

```
sameDay = unique(birthday(:)); % generate a list of people with
the same birthday
        if numel(birthday) ~= numel(sameDay) % if sameDay is not
equal to birthday, 2 or more integers are the same
            sameBirthday(1,i) = sameBirthday(1,i) + 1; % update
frequency
        end
```

the variable `sameDay` is the variable `birthday` listed from the smallest to greatest. The reason to use `unique` is because if the same integer is outputted in `birthday` then the same integer would only be outputted once instead of twice. This creates a difference in the number of elements in `birthday` and `unique`. Thus, an if statement with the conditions when the two arrays have not the same elements, we update the variable `sameBirthday`. Then, we can calculate the probability from each set of people, n, with the simulation cycles. We can plot the probability and n where n is the independent variable and probability is the dependent variable.

We can see later in section 3.3 that the number of people to have at least 50% probability lies about 20-25, but we can approximate this further using *Poisson's Approximation*. In order to support my verification, I found an article from Cornell University Math Department explaining the birthday problem and *Poisson's Approximation*. The article provides a formula to calculate the probability.

$$\bar{p}(n) = e^{-\frac{n(n-1)/2}{365}}$$

From the formula above, we can calculate the number of people n, since we know the probability is 0.5. So, to find n, the inverse of the function is needed. We then found the equation $n^2 - n + 730 * ln(\bar{p}(n)) = 0$. $\bar{p}(n)$is the probability so we can substitute it with 0.5. I set this as an anonymous function f = @(n) $n^2 - n + 730 * ln(\bar{p}(n))$ and use a root finding method to solve for n, fzero, with an initial guess at 20. The result of the n gives out 22.999 or rounded to 23.

3.3. Calculations and Results

When the code is run, a graph showing the probability of two person sharing the same birthday corresponding to the number of people is plotted. From the formula derived, we also obtain that 23 people is the number of people needed to at least have 0.5 probability that 2 people have the same birthday.

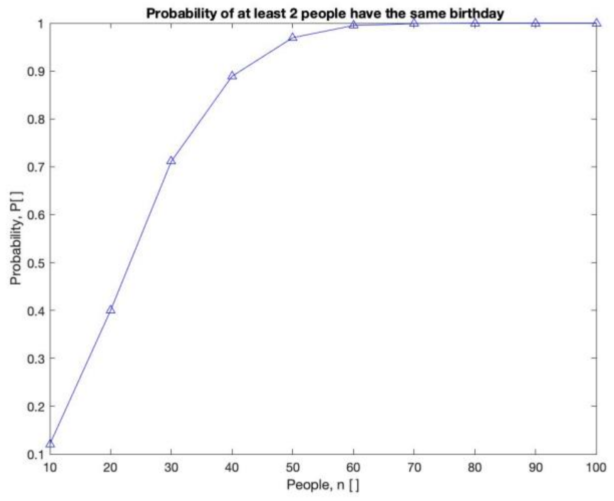

Figure 2 Probability vs People Figure for Birthday Probelm

```
From Poisson Approximation, 23 people is needed for two people to have
the same birthday with probability of at least 0.5
```

The details from the results will be discussed in the section below

3.4. Discussion

In the section above, we obtained that 23 people is needed to be in a room such that there is at least 50% probability that at least two people have the same birthday. Note that the plot subtly shows readers that the probability 0.5 lies at around 20-25 and the *Poisson Approximation* or Probability Theory strengthens that claim using the formula to get 22.999 or 23 people when rounded. This seems to be a good approximation since it gives the readers the amount of probability that matches the graph.

The application of math in MATLAB is quite interesting. Scientists and mathematicians are able to calculate the probability of an event happening using data. There must be some assumptions needed to be made prior to the probability, but nonetheless still gives an approximation with a relatively small error. This is a takeaway lesson I got from learning all about probability. It is interesting and also a mystery since the distribution of the data seems to not be normal when it comes to these. I would like to know more about this!

Works Cited

Zheng, Tianyi. “Birthday Problem.” *Birthday Problem*, 2009, http://pi.math.cornell.edu/~mec/2008-2009/TianyiZheng/Birthday.html#:~:text=Birthday%20Problem&text=As%20an%20application%20of%20the,problem%2C%20which%20is%20quite%20interesting.&text=The%20event%20that%20at%20least,1%20minus%20the%20above%20probability.