

Name: Sangarange Sandanayake

Student Reference Number: 10899486

Module Code: PUSL 2018

Module Name: computational theory and statistics for computing

Coursework Title: Computational Theory STAT coursework (Group Assignment)

Deadline Date: 02/01/2024

Member of staff responsible for coursework: Dr. Rasika Ranaweera

Program: BSc.(Hons) Data Science

Please note that University Academic Regulations are available under Rules and Regulations on the University website www.plymouth.ac.uk/studenthandbook.

Group work: please list all names of all participants formally associated with this work and state whether the work was undertaken alone or as part of a team. Please note you may be required to identify individual responsibility for component parts.

Group Number 22 - Members

- * Weerasinghe Dissanayake - Task 2 and report
- * Gosisa Jinasena - Task 2 and report
- * Sangarange Sandanayake - Task 1 and report
- * Urulugastenne Amarakone - Task 1 and report

We confirm that we have read and understood the Plymouth University regulations relating to Assessment Offences and that we are aware of the possible penalties for any breach of these regulations. We confirm that this is the independent work of the group.

Signed on behalf of the group: S.chamara

Individual assignment: ***I confirm that I have read and understood the Plymouth University regulations relating to Assessment Offences and that I am aware of the possible penalties for any breach of these regulations. I confirm that this is my own independent work.***

Signed:

Use of translation software: failure to declare that translation software or a similar writing aid has been used will be treated as an assessment offence.

I *have not used translation software.

If used, please state name of software.....

Overall mark _____%

Assessors Initials _____

Date_____



UNIVERSITY OF
PLYMOUTH

PUSL2018 Computational Theory and Statistics for Computing
(23/AU/M)

Final Report

PUSL 2018 - Group Assignment

Group 22

Group Members Details

PU Index No.	Student Name	Degree Program
10899177	Urulugastenne Amarakone	Data science
10899302	Weerasinghe Dissanayaka	Data science
10899478	Gosisa Jinasena	Data science
10899486	Sangarange Sandanayake	Data science

CONTENTS

Task 1: Monte Carlo Simulation	4
1. Question.....	4
2.Introduction	4
3. Discussions and Mathematics.....	4
4. Pseudocode.....	10
5. Python Code with an explanation	11
6.Results Analyze	14
7. Generated plots	16
8. Summary and conclusion	17
Task 2: Statistics.....	18
1. Question.....	18
2.Introduction	18
3) Task 2.1	18
3.1) Discussions and Mathematics.....	18
4) Task 2.2	21
4.1) Discussions and Mathematics.....	21
5. Pseudocode.....	22
6. Full Python code and explanation.....	23
7. Results Analyze	26
8. Summary and conclusion	26
Group contribution	28
Appendices	30
References.....	31

Task 1: Monte Carlo Simulation

1. Question

1. Your task is to write a program to simulate a game of dart in which the dartists either hit or miss the dartboard. Compute the probability of hitting for any dartist. You may consider the radius as one (1) and the center as (0,0). Note that hitting can be considered as hitting anywhere in the circular area.
2. Explain how you can use this simulation to estimate "pi".
3. Run your simulation for 1K (thousand), 10K, 100K, 100K, 1000K times (N) and plot the value of pi in an Excel sheet. Show in a graph the value of pi against N becoming closer to the exact when the simulation runs for many times.
4. You need to run each experiment for 10 times, log the values, and compute the mean and mode. All values and computations should be recorded in the same Excel file.

2.Introduction

In this question, we have to estimate the numerical $\pi(\text{pi})$ value of a circle by using a dart-throwing game simulation. The main theories we had to use were Monte Carlo simulation and probability for develop the Python program to solve this question. A proper discussion of the mathematical and theoretical approach that we used to calculate the estimated pi value of a circular dartboard is explained below. Finally, we calculated the probability of hitting the dart board by a dartist, estimated Pi values, Mean $\pi(\text{pi})$ value, and mode Pi value, generated plots and graphs, and finally logged the results to an Excel file.

3. Discussions and Mathematics

1. What is Monte Carlo simulation?

A statistical technique called the Monte Carlo simulation uses random sampling to get numerical values. This approach allows us to describe and evaluate intricate processes or systems that might contain uncertain probabilistic components. It can estimate the outcomes and probability of obtaining the desired result by generating different random inputs.

The problem in this question is solved by, estimating the value of Pi using the above-described method of throwing darts at a square-shaped dart board with a circular target area. A proper visualization of solving the question is mentioned below in the question discussion.

2. Law of Large Numbers:

The Monte Carlo approach makes use of the law of large numbers which means when the number of experiments increases the estimated value converges to the true value. For this scenario, the observed values (probabilities of hitting the target) converge to the true values (the actual probabilities). This is the basic principle behind why Monte Carlo simulations work so effectively for this type of question.

3. Geometry and Probability:

We determine $\pi(\pi)$ value by making use of the geometric relationship between the circle and square using a probabilistic approach. We are using probability theory to do the calculations. (IBM, 2022)

Question discussion

As the question says, we need to consider the radius of the circle as '1' and the center as '0,0'. According to that, we have assumed a Square shaped dart board and inside that square, we place the dart circle.

The circle radius (r) is equal to '1' and therefore, the length of one quadrant of the square is also equal to '1' or ' r '. The length (L) of the square is equal to ' $2r$ '. The assumed dartboard looks as follows (figure 2).



Figure 1: Image of a dart board

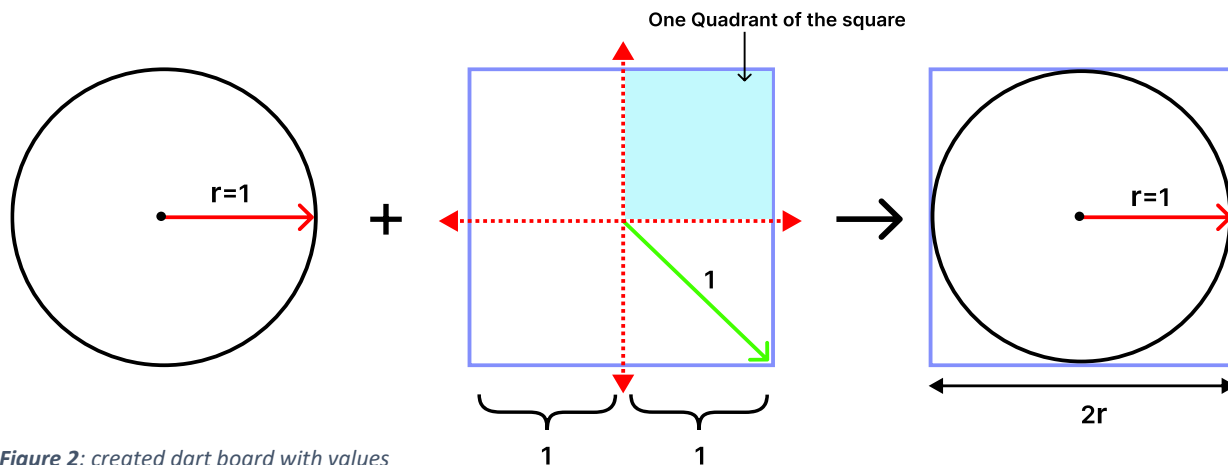


Figure 2: created dart board with values

Before moving on to the theories of predicting the value of π (pi) utilizing the theories in Monte Carlo simulation let's examine the mathematical foundation of the question using the figure.

Calculation,

The radius of the inner circle (r) = 1

Length of the square = $2r$

Length / 2 of the square = r

The calculations are shown below.

Area of the circle (A_c) = πr^2

Area of the square (A_s) = $4r^2$

If $r=1$,

$$\begin{aligned}\text{Area of the circle (}A_c\text{)} &= \pi r^2 \\ &= \pi 1^2 \\ &= \pi\end{aligned}$$

$$\begin{aligned}\text{Area of the square (}A_s\text{)} &= 4r^2 \\ &= 4*1^2 \\ &= 4\end{aligned}$$

The ratio between the areas,

$$A_c : A_s = \pi / 4$$

The π is equal to 4 times of ratio between the areas.

$$\begin{aligned}\pi &= A_c/A_s * 4 \\ \pi &= \pi/4 * 4 \\ \pi &= \pi\end{aligned}$$

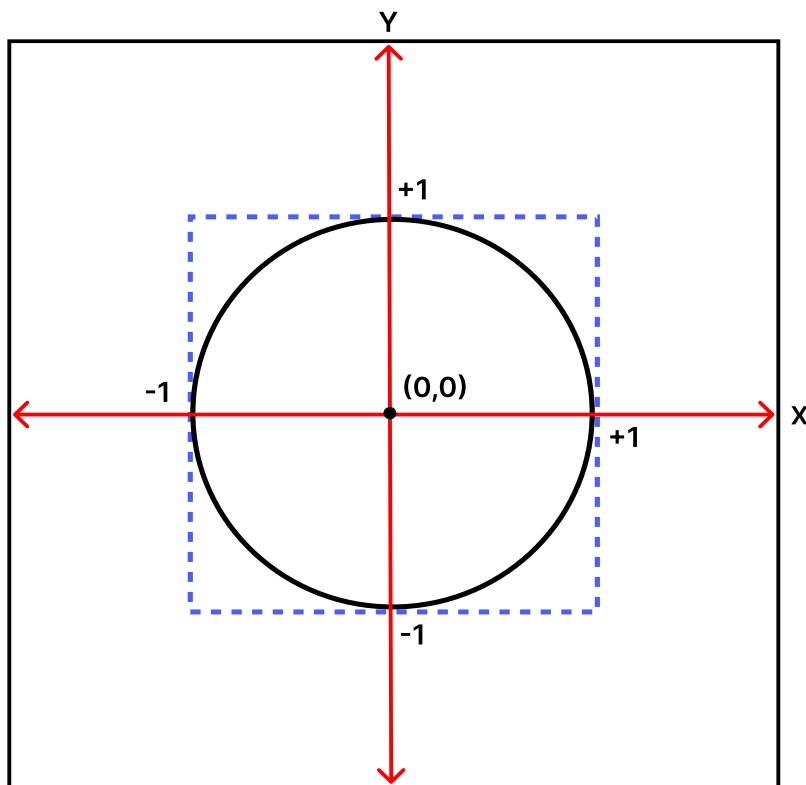


Figure 3: full dart board placing with axis notation

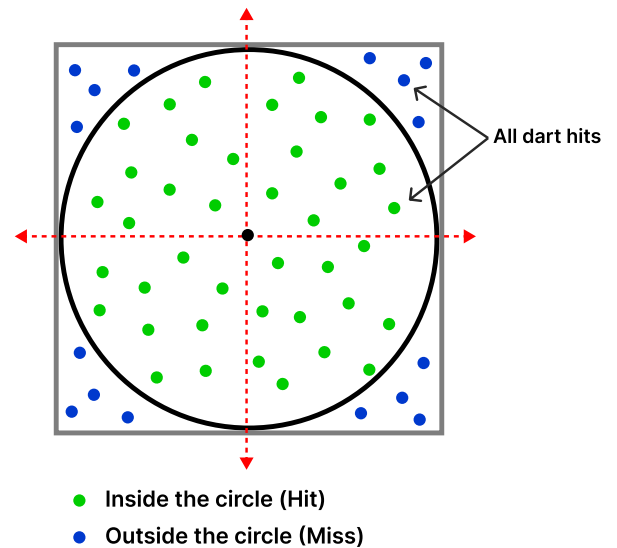


Figure 4: Show all dart hits

Above figure 3 shows the positioning of the dartboard, its coordinates placed with the x and y axes like a graph. It samples the x and y coordinates from negative 1 to 1. The x and y are both independently going to be numbers that are sampled from between negative 1 and 1 uniformly y between negative 1 and 1.

Figure 4 shows how the dart throws can fall into the dart board. Green spots denote the darts that hit inside the circle and blue spots represent the missed darts that land outside the circle. We need a total number of dart hits and a total number of dart throws (**N**) to calculate the probability. The steps are described below by using these images to calculate the (**p**) value.

Estimating the π (pi) value of the circle

Let's imagine that we are just going to throw darts to the board. The darts can be placed anywhere on the board as shown in figure 4. Every one of those Dart hits can fall either inside the circle which is this region or it can fall outside of the circle.

How can we calculate the Pi value? We assume a dart randomly lands in the upper right quadrant of the square. The dart can be placed inside or outside the circle. The probability that a dart lands inside the circle is proportional to the area of the Circle out of the total area of the square.

$$\frac{\text{Number of darts hit on the circle}}{\text{Total number of darts}} = \frac{\text{Area of the circular part (Ac)}}{\text{Area of the square part (As)}} = (P)$$

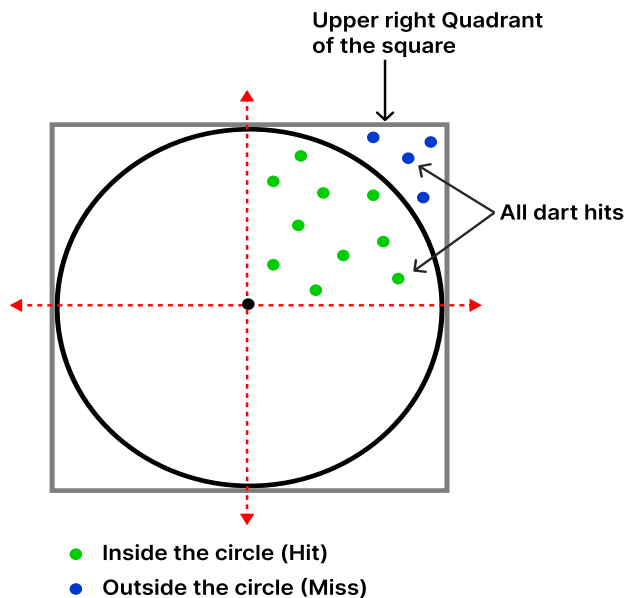


Figure 5: quadrant image

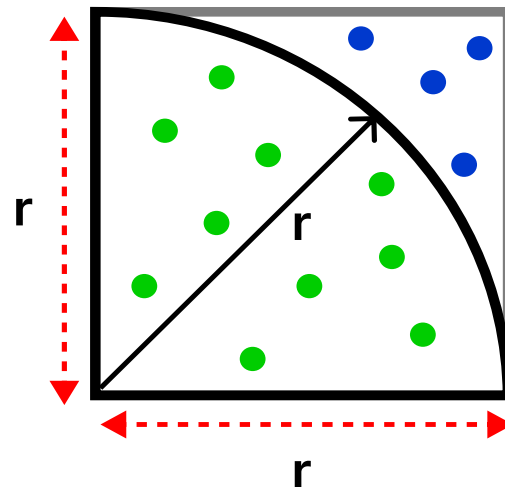


Figure 6: quadrant square (1/4)

Calculate the probability that darts land inside the circle according to the above images

In the above image if a dart lands in the circular part. The probability is the area of the circular part divided by the total area of the square part now the area of the circular part is going to be 1/4 the area of a total Circle because we only selected the upper right quadrant. So, the area is $\pi r^2 / 4$.

Area of the square is just the length* width and in this case, the length is equal to 'r' or '1' (figure 6). The area is r^2 or 1.

$$\frac{\text{Number of darts hit on the circle}}{\text{Total number of darts}} = \frac{\text{Area of the circular part (Ac)}}{\text{Area of the square part (As)}} = (P)$$

$$\text{The probability of lands in the circular part (P)} = \frac{\text{Number of darts hit on the circle}}{\text{Total number of darts}}$$

$$\text{The probability of lands in the circular part (P)} = \frac{\text{Area of the circular part (Ac)}}{\text{Area of the square part (As)}}$$

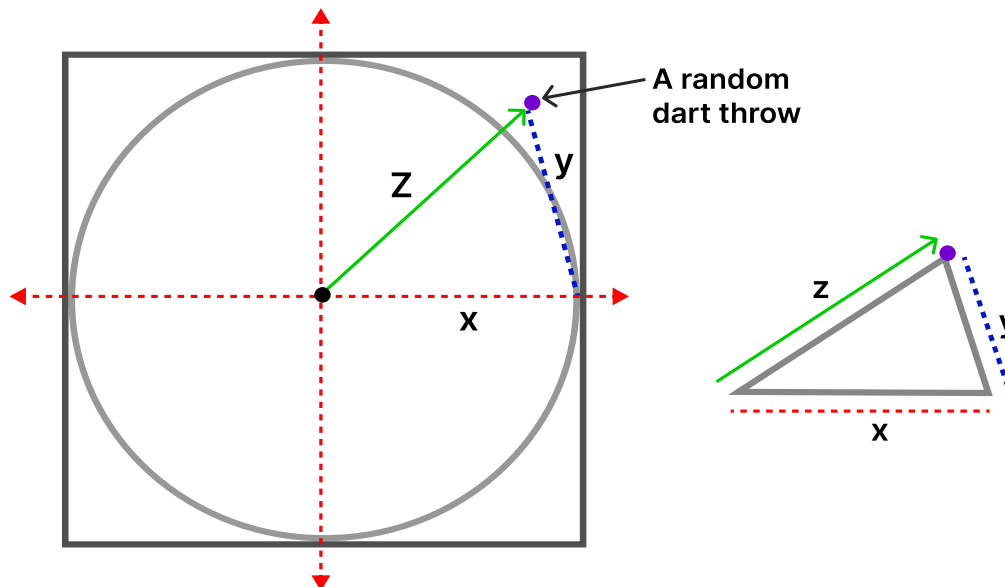
$$P = \frac{\pi r^2 / 4}{r^2}$$

$$P = \frac{\pi}{4}$$

$$\pi = P * 4$$

The answer of $\pi = 4P$. The estimated pi value is equal to 4 times probability (p). The probability value is approximately 0.78539. The p value is always high because more area was occupied by the circle.

Identify the place where the dart lands



How can we identify if a point lands inside the circle or outside? First of all, assume any dart landing point. Let's say the dart lands here in the purple point, this point has an **X** coordinate and it has a **y** coordinate. We need to draw an imaginable line from the origin point **(0,0)** to point **Z**. The distance from the origin to the point can be calculated using the Pythagorean theorem.

According to the Pythagorean theorem,

$$\begin{aligned} Z &= x^2 + y^2 \\ Z &= \sqrt{x + y} \\ Z &= \sqrt{x + y} \leq 1 \end{aligned}$$

- If **Z** is higher than '1' that means **Z** is larger than the radius of the circle which means the dart is located outside the circle. And it counts as a 'miss'.
- If **Z** is smaller than '1' that means the dart is located inside the circle because it's less than the radius of the circle. And this one counts as a 'Hit'.
- The condition for a random Dart throw being inside the circle is $Z \leq 1$ or $x^2 + y^2 \leq 1$.

(youtube, 2022)

4. Pseudocode

```
pseudo dart.py ×
1 Function throwing_dart():
2     x = random.uniform(-1, 1)
3     y = random.uniform(-1, 1)
4     distance = sqrt(x^2 + y^2)
5     return distance <= 1
6
7 Function simulates_dart_game(num_of_darts):
8     hits = 0
9     For i in range(num_of_darts):
10         If throwing_dart() is True:
11             hits += 1
12     Return hits / num_of_darts
13
14 Function estimate_pi(num_experiments, num_darts_per_experiment):
15     pi_values = []
16     For i in range(num_experiments):
17         probability = simulates_dart_game(num_darts_per_experiment)
18         pi_estimate = 4 * probability
19         Append pi_estimate to pi_values
20     Return pi_values
21
22 Function main():
23     Parse command line argument num_experiments
24
25     List num_darts_per_experiment = [1000, 10000, 100000, 1000000]
26     Initialize experiment_results as an empty list
27
28     For each num_darts in num_darts_per_experiment:
29         probability = simulates_dart_game(num_darts)
30         Print "Probability of hitting with num_darts darts: probability"
31
32         pi_values = estimate_pi(num_experiments, num_darts)
33
34         Print "Estimated Pi values for num_experiments experiments with num_darts darts: pi_values"
35
36 If script is executed directly:
37     If length of command line arguments is less than 2:
38         Print "Usage: python dart_simulation.py <num_experiments>"
39         Exit with status code 1
40     Call main()
```

5. Python Code with an explanation

1) importing the libraries

```
1 import argparse
2 import random
3 import math
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 import sys
7
```

This code was used to import the necessary libraries needed to run the Python script. Such as for parsing command line arguments, generating random numbers, providing math functions, data manipulation storing data, and generating plots. The libraries are argparse, random, math, pandas, matplotlib, and sys.

2) Simulating Dart throwing using the throwing dart function

```
7 def throwing_dart():
8     x = random.uniform(-1, b: 1)
9     y = random.uniform(-1, b: 1)
10    distance = math.sqrt(x**2 + y**2)
11    return distance <= 1
12
```

To put it simply, this function imitates tossing a dart at random points on a two-dimensional plane and determines whether the dart landed within or outside of a unit circle with the origin at its center. If the result is a hit, the function returns True; otherwise, it returns False. When estimating probabilities or computing numerical solutions, Monte Carlo techniques frequently employ this type of simulation.

3) Simulating the Dart game

```
12
13 def simulates_dart_game(num_of_darts):
14     hits = 0
15     for _ in range(num_of_darts):
16         if throwing_dart():
17             hits += 1
18     return hits / num_of_darts
19
```

This function calls the throwing Dart() method for the given number of darts, so simulating a dart-throwing game. To estimate the chance of hitting the target in this simulation, it counts the number of darts that hit it and returns the ratio of hits to the total amount of darts. When estimating probabilities using Monte Carlo techniques, this kind of simulation is frequently used.

4) Estimate Pi Function

```
20 def estimate_pi(num_experiments, num_darts_per_experiment):
21     pi_values = []
22     for _ in range(num_experiments):
23         probability = simulates_dart_game(num_darts_per_experiment)
24         pi_estimate = 4 * probability
25         pi_values.append(pi_estimate)
26     return pi_values
27
```

This function is defined for performing various experiments with a specific number of darts per experiment. Then it calculates the estimated pi value by multiplying the probability of hitting the circle by 4 and finally collects the results in a list.

5) Logging Results Function

```
27
28 def log_experiment_results(experiment_results, filename):
29     df = pd.DataFrame(experiment_results, columns=['Number of Darts', 'Mean Pi', 'Mode Pi'])
30     df.to_excel(filename, index=False)
31     print(f"Experiment results logged in {filename}")
32
```

This function takes the experiment results (number of darts, mean π , mode π) and logs them in an Excel file. This code generates an Excel file automatically with the mean and mode values for each dart throw.

6) Plotting Function

```
33 def plot_pi_estimates(pi_values, filename):
34     plt.figure(figsize=(10, 6))
35     plt.plot(pi_values)
36     plt.axhline(y=math.pi, color='r', linestyle='--', label='True Pi')
37     plt.xlabel('Number of Experiments')
38     plt.ylabel('Pi Estimate')
39     plt.title('Convergence of Estimated Pi with Increasing Number of Experiments')
40     plt.legend()
41     plt.grid(True)
42     plt.savefig(filename)
43     plt.show()
44
```

This function creates a plot of the estimated π values over multiple experiments and directly saves it as an image file.

7) Main Function

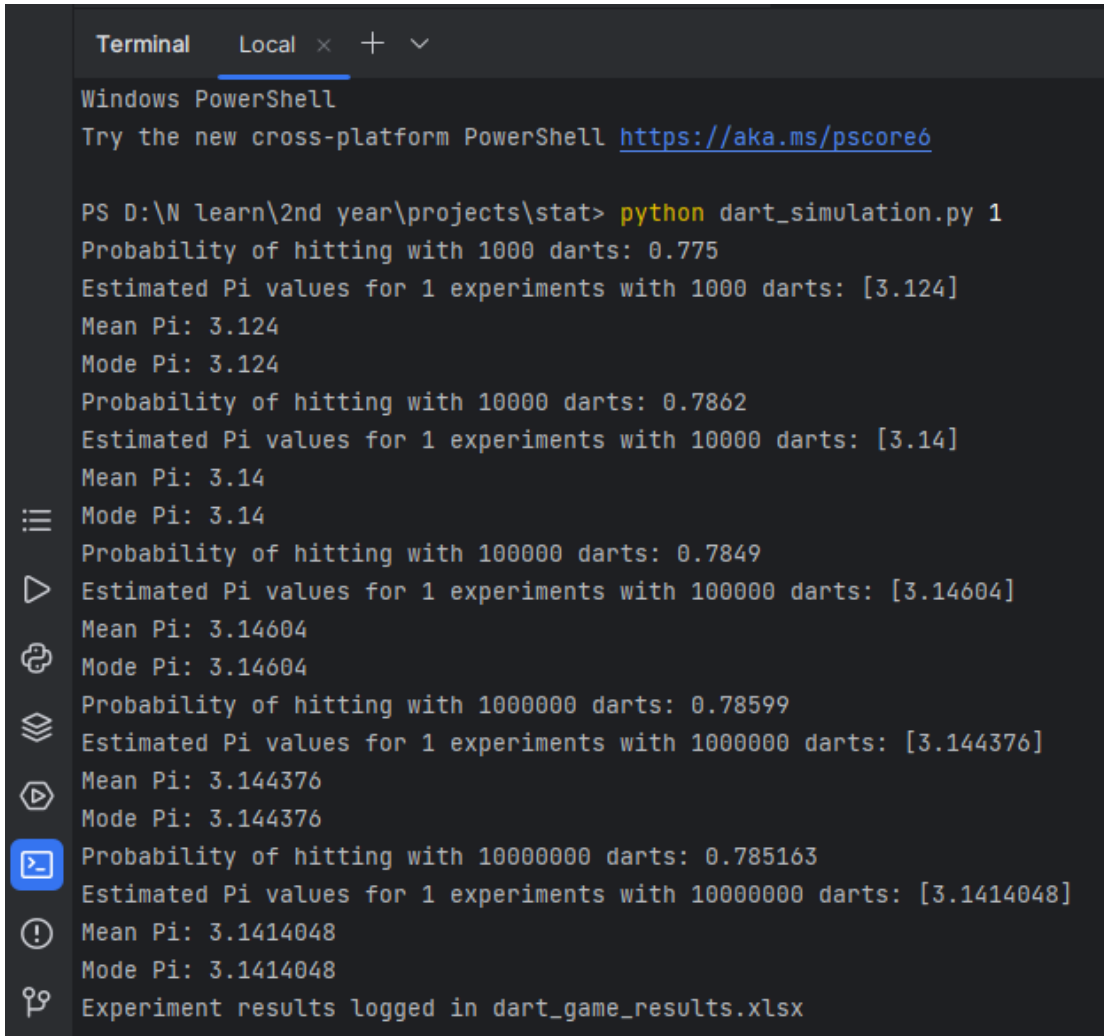
```
46 def main():
47     # command line argument to change the number of experiments need to be run the simulation
48     parser = argparse.ArgumentParser(description='Monte Carlo Simulation for Dart Game')
49     parser.add_argument('name_or_flags: 'num_experiments', type=int, help='Number of experiments to run')
50     args = parser.parse_args()
51
52     num_darts_per_experiment = [1000, 10000, 100000, 1000000, 10000000]
53     experiment_results = []
54
55     for num_darts in num_darts_per_experiment:
56         # calculate the probability of hitting in each number of darts per experiment
57         probability = simulates_dart_game(num_darts)
58         print(f"Probability of hitting with {num_darts} darts: {probability}")
59
60         # get the pi values and calculate mean and mode of estimated pi values
61         pi_values = estimate_pi(args.num_experiments, num_darts)
62         mean_pi = sum(pi_values) / len(pi_values)
63         mode_pi = max(set(pi_values), key=pi_values.count)
64         experiment_results.append((num_darts, mean_pi, mode_pi))
65
66         # print the calculated pi, mean, and mode values
67         print(f"Estimated Pi values for {args.num_experiments} experiments with {num_darts} darts: {pi_values}")
68         print(f"Mean Pi: {mean_pi}")
69         print(f"Mode Pi: {mode_pi}")
70
71     # log the results in an Excel sheet
72     log_experiment_results(experiment_results, filename='dart_game_results.xlsx')
73
74     # create a plot of estimated pi values
75     all_pi_values = estimate_pi(args.num_experiments, max(num_darts_per_experiment))
76     plot_pi_estimates(all_pi_values, filename='pi_estimates_plot.png')
77
78     if __name__ == "__main__":
79         if len(sys.argv) < 2:
80             print("Usage: python dart_simulation.py <num_experiments>")
81             sys.exit(1)
82         main()
```

The main function is the entry point of the script. It will parse the command-line argument specifying the number of experiments to run (**Num_experiments**). It then iterates over different numbers of darts per experiment, calculates probabilities, and estimates π . The results are printed and logged into an Excel file, and a plot is generated automatically, and update image and Excel file each time the code runs.

The script can be run from the command line, providing the number of experiments as an argument, e.g., **python dart_simulation.py 100**. The results are logged in an Excel file, and a plot is saved as an image. If no command-line argument is provided, the script prints a usage message and exits.

6.Results Analyze

As mentioned in the coursework deliverables this Python program can be executed through a terminal after giving the file directory path. The program facilitates parsing arguments so that the simulation can be tested for any number (**N**) of experiments specified by the user at the end of the command line. ex:- **python dart_simulation.py <Num_experiments>**



```
Terminal Local x + v
Windows PowerShell
Try the new cross-platform PowerShell https://aka.ms/powershell

PS D:\N learn\2nd year\projects\stat> python dart_simulation.py 1
Probability of hitting with 1000 darts: 0.775
Estimated Pi values for 1 experiments with 1000 darts: [3.124]
Mean Pi: 3.124
Mode Pi: 3.124
Probability of hitting with 10000 darts: 0.7862
Estimated Pi values for 1 experiments with 10000 darts: [3.14]
Mean Pi: 3.14
Mode Pi: 3.14
Probability of hitting with 100000 darts: 0.7849
Estimated Pi values for 1 experiments with 100000 darts: [3.14604]
Mean Pi: 3.14604
Mode Pi: 3.14604
Probability of hitting with 1000000 darts: 0.78599
Estimated Pi values for 1 experiments with 1000000 darts: [3.144376]
Mean Pi: 3.144376
Mode Pi: 3.144376
Probability of hitting with 10000000 darts: 0.785163
Estimated Pi values for 1 experiments with 10000000 darts: [3.1414048]
Mean Pi: 3.1414048
Mode Pi: 3.1414048
Experiment results logged in dart_game_results.xlsx
```

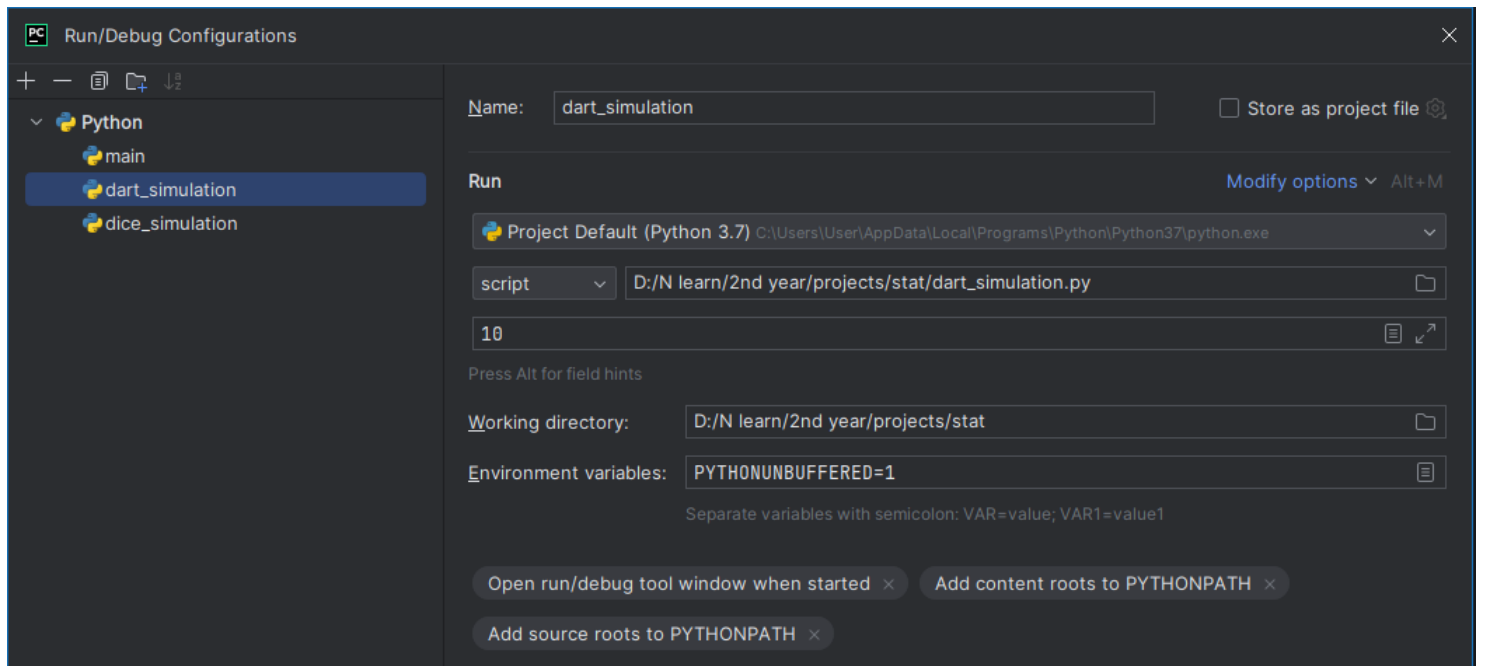
This is the **1st output** of the script that runs through the terminal. The command line argument given through the terminal is **python dart_simulation.py 1**. This simulation can be tested for any number of experiments by simply changing the value “.py 1 ” to any experiment times like 2, 5, or 10 experiments. As an example, **python dart_simulation.py 10**.

Here, you can see the output shows the probability of hitting with the desired number of total dart throws, the estimated Pi value for a given number of darts (**N**), the Mean value, and mode value. Here the program executed an output only for 1 experiment. Because of that, it gave only one estimated pi value. Also, the mean and mode values are the same. The reason was when the number of experiments was low, the random values that are generated were not diverse enough to give different mode and mean values. The program outputs the probability for various numbers (**N**) of experiments. (**P**) = 0.7851 approximately.

```
Terminal Local x Local (2) x + v
Windows PowerShell
Try the new cross-platform PowerShell https://aka.ms/powershell

PS D:\N learn\2nd year\projects\stat> python dart_simulation.py 10
Probability of hitting with 1000 darts: 0.774
Estimated Pi values for 10 experiments with 1000 darts: [3.132, 3.096, 3.092, 3.104, 3.236, 3.152, 3.16, 3.144, 3.152, 3.08]
Mean Pi: 3.1348
Mode Pi: 3.152
Probability of hitting with 10000 darts: 0.7859
Estimated Pi values for 10 experiments with 10000 darts: [3.1444, 3.1524, 3.1212, 3.1596, 3.1328, 3.1272, 3.1224, 3.1432, 3.1592, 3.1444]
Mean Pi: 3.14068
Mode Pi: 3.1444
Probability of hitting with 100000 darts: 0.78461
Estimated Pi values for 10 experiments with 100000 darts: [3.13852, 3.1428, 3.14176, 3.13812, 3.13964, 3.14, 3.13868, 3.14316, 3.14072, 3.13688]
Mean Pi: 3.1400280000000005
Mode Pi: 3.1428
Probability of hitting with 1000000 darts: 0.785024
Estimated Pi values for 10 experiments with 1000000 darts: [3.140232, 3.141224, 3.140328, 3.141624, 3.143404, 3.142236, 3.145244, 3.138932, 3.14164, 3.140396]
Mean Pi: 3.141526
Mode Pi: 3.141224
Probability of hitting with 10000000 darts: 0.7854029
Estimated Pi values for 10 experiments with 10000000 darts: [3.1414616, 3.1411592, 3.1420936, 3.1427904, 3.1415496, 3.1418012, 3.1417368, 3.1416676, 3.1416416, 3.1421564]
Mean Pi: 3.1418057999999998
Mode Pi: 3.1411592
Experiment results logged in dart_game_results.xlsx
```

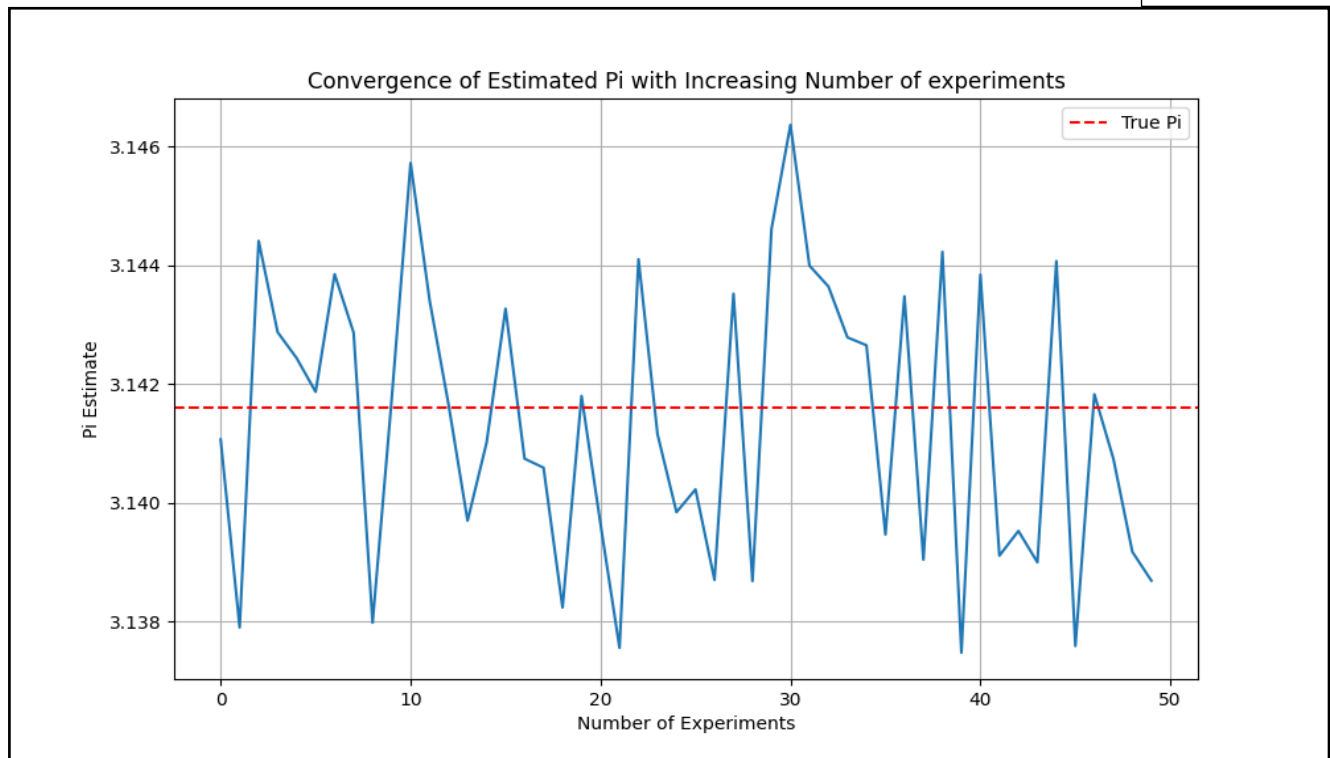
This is the **2nd output** taken through the terminal. Here you can observe the difference between mean and mode values, because of the higher number of random values generated diverse enough to give different mode and mean values under 10 experiments. Lastly, there are 10 estimated Pi value outputs given for each number of (**N**) dart throws under each 10 experiments.



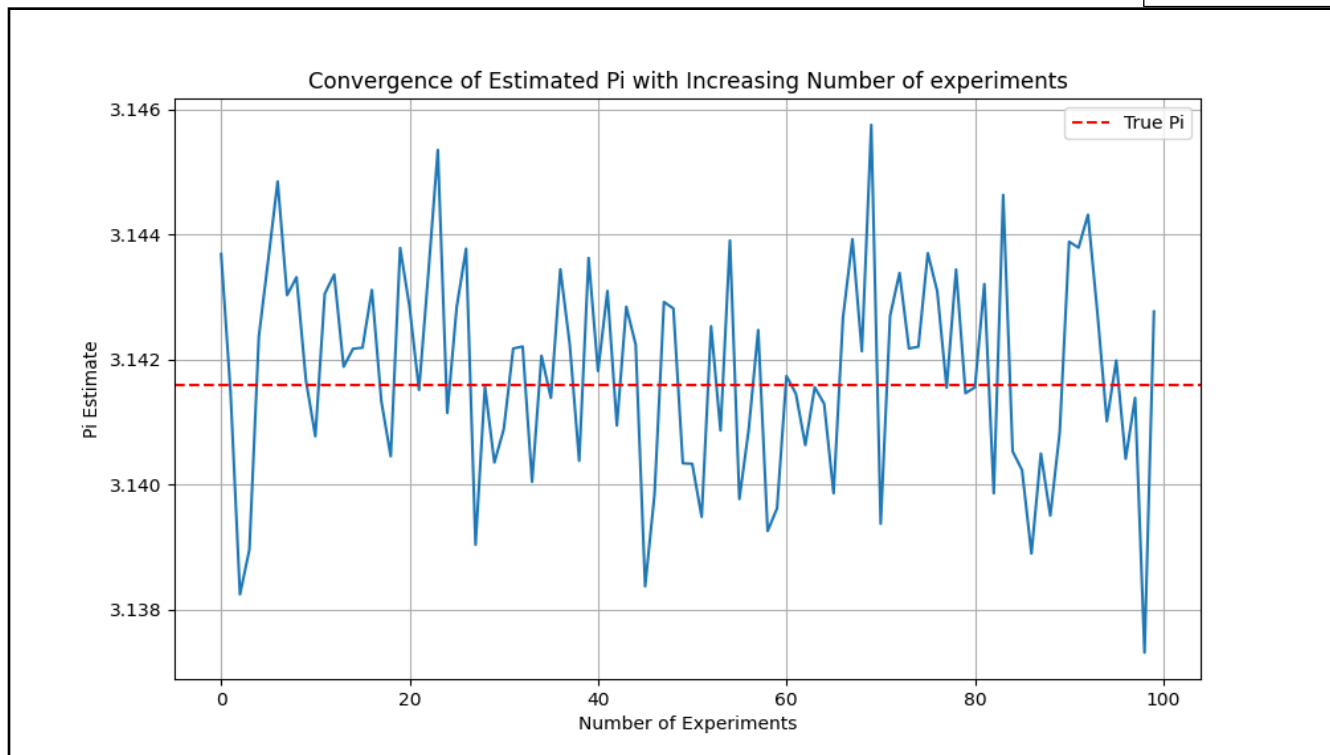
The user also can change the number of experiments by creating a debug configuration file in the IDE.

7. Generated plots

50 experiments



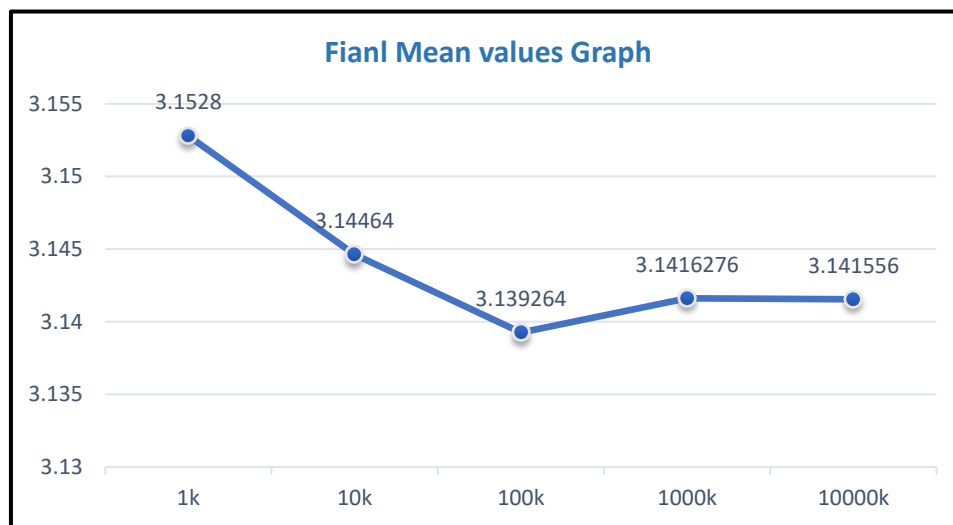
100 experiments



The above 2 graphs show how the estimated pi value changes along with the number of experiments. When the number of experiments and dart throws (**N**) increases, the estimated pi value becomes very close/accurate to the actual Pi value. Apart from the above plots, the excel file contains more detailed plots for every experiment and also for the mean values. (link is attached at end of the 1st task)

8. Summary and conclusion

This Python script combines elements of probability, geometry, and statistical sampling to estimate π using a Monte Carlo simulation of dart throwing. It applies mathematical concepts such as the law of large numbers and geometric properties to provide accurate estimations of π (Pi) through the random sampling method. The code's structure facilitates experimentation with different numbers of darts, allowing for a deeper understanding of the estimates using plots and graphs.



The conclusion is we can get a very close estimated Pi value by using Monte Carlo simulation and mathematical parts. When the number of dart throws (**N**) increases the probability value and the Pi value become very close to the actual pi value (actual Pi = 3.1415). In the above image, you can observe when there are ten million (10 000K) dart throws (**N**) the value of **Pi** is equal to **3.141556**. So, the value is the same as the actual.

For more detailed information, please refer to the created excel sheet. The link is attached below.

Excel sheet link – (please click “ view raw ” to download the Excel file after opening the page)

<https://github.com/scssandanayake/computational-theory-and-stat-coursework/blob/main/dart%20game%20results.xlsx>

Task 2: Statistics

1. Question

What is the probability that 10 dice throws add up exactly to 32?

1. Calculate this exactly by counting all possible ways of making 32 from 10 dice.
2. Simulate throwing the dice (say 500 times), count the number of times the results add up to 32, and divide this by 500.

2.Introduction

In this task, we have a statistic & probability-based problem to solve. We divided this task into two main parts (task 2.1, task 2.2) and included a discussion, a mathematical explanation, a pseudo code, a Python program of the solved problem & an explanation of the Python program for each part respectively.

3) Task 2.1

3.1) Discussions and Mathematics

1. Discussion

In this part, we need to calculate the probability that 10 dice throws add up exactly to 32 by calculating all the possible ways that it can happen. A dice has 6 faces. We need to do 10 dice throws or throw 10 dice at the same time. Then we can get 10 displays.

The minimum sum we can get from 10 dice = minimum face value × number of dice

$$= 1 \times 10$$

$$= \underline{\underline{10}}$$

$$\text{Minimum Sum} = [1] + [1] + [1] + [1] + [1] + [1] + [1] + [1] + [1] + [1] = 10$$

And the maximum sum we can obtain from 10 dice = maximum face value × number of dice

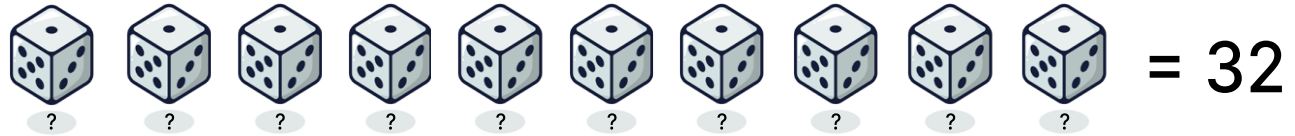
$$= 6 \times 10$$

$$= \underline{\underline{60}}$$

$$\text{Maximum Sum} = [6] + [6] + [6] + [6] + [6] + [6] + [6] + [6] + [6] + [6] = 60$$

If the sum of 10 dice are equals to 32, It's counted as a possible way of making 32 from 10 dice.

$$[?] + [?] + [?] + [?] + [?] + [?] + [?] + [?] + [?] + [?] = 32$$



Using the above information,

We can make a simple equation. Let ' T ' be the sum of 10 dice throws. Our target is to find $P(T = 32)$.

$P(T = 32)$ means the probability of getting 32 as the total summation.

- Total number of instances that will occur = number of faces in a die ^ (number of dices)
- Therefore, the total number of instances, say $n(t) = 6^{10}$
- The number of instances that the throwing of dice adds up to 32, Say $n(x)$.

Therefore, the probability that 10 dice throws adds up to 32 = $n(x) / n(t)$

$$P(T = 32) = n(x) / n(t)$$

The value of $n(t) = 6^{10} = 60,466,176$

Getting the value of $n(t)$ is not a hard task. However, calculating the $n(x)$ value manually can be difficult. As an alternative, we can use a Python program or a computed algebra system. The computation's theory is based on solving this type of question are "Generating functions and Combinatorics".

Also, we can use permutations and combinations to calculate the number of ways to get the summation of 32. But manually doing such a task could be very complex because we need to count each combination and simplify it. (Smith, 2020)

2. Mathematics

How can we calculate the value of $n(x)$

We select the functions method to solve this issue. For a single die, the generating function can be calculated using this equation “ $x + x^2 + x^3 + x^4 + x^5 + x^6$ ”. The number of all possible ways to get the value of T with a single die is represented by the coefficient of ‘ x^T ’ In the expansion of this generating function. The $T = 32$ (total summation from 10 dice). So, the $x^T = x^{32}$.

For the total of ten dice, the generating function is “ $(x + x^2 + x^3 + x^4 + x^5 + x^6)^{10}$ ”.

In the final step have to find the coefficient of x^{32} .

The coefficient of $x^{32} = (x + x^2 + x^3 + x^4 + x^5 + x^6)^{10}$.

The value of $n(x) = x^{32}$.

we used a coefficient calculator to find the value of x^{32} .

(Brozius, 2019)

3.Calculations

Polynomial Coefficients

This calculator computes the coefficients of a Polynomial that you input below.

Answer

Input

Variable = x

Polynomial = $(x + x^2 + x^3 + x^4 + x^5 + x^6)^{10}$

$x^{32} : 3801535$

(calculus, n.d.)

So now we know the values for $n(x)$ and $n(t)$,

$$P(T = 32) = n(x) / n(t)$$

= number of instances that the throwing of dice adds up to 32 / total number of instances

$$= 3801535 / 604661146$$

$$= \underline{0.06287043}$$

the probability that 10 dice throws add up exactly to 32 = **0.06287043** .

finally, we create a Python script for calculate the $P(T = 32)$ value.

4) Task 2.2

4.1) Discussions and Mathematics

1. Discussion

In the second part of the question, we need to calculate the probability of 10 dice throws adding up exactly for 32 in **N** number of simulations. (N is given as 500)

The **number of dies** used in 1 simulation is equal to 10. because the question says that **10 dice are thrown** in each time. So, a random 10 numbers will be generated for 10 dice.

According to that, we know 1 simulation is equal to 1 instance. It could be a successful event or not.

Now 10 dice throwing 500 times and get the number of successful instances to calculate the probability.

2. Mathematics

$n(t)$ (number of simulations) = 500.

Let $n(x)$ be the number of successful instances that 10 dice throws that add up exactly to sum as 32.

To find out how many times we'll get the sum as 32 out of 500 times these steps are followed.

- Simulate throwing 10 dice.
- Calculate the sum of the numbers obtained.
- Count how many times the sum is equal to 32.
- Calculate the probability.

Calculate the estimated Probability

To calculate the probability, we divide the number of successful outcomes in the simulations by the specified number of simulations (500).

P = number of successful instances/number of simulations

$$P = n(x) / n(t)$$

$$P = n(x) / n(t)$$

finally, we code the Python script for calculate the probability value for the simulation. Python codes are explained in the below section.

5. Pseudocode

```
pseudo dice.py x
57 Function count_ways(target_sum, num_dice, memo):
58     If num_dice is 1:
59         Return 1 if 1 <= target_sum <= 6, else return 0
60
61     If (target_sum, num_dice) in memo:
62         Return memo[(target_sum, num_dice)]
63
64     Initialize ways to 0
65     For i from 1 to 6:
66         If target_sum - i >= 0:
67             Increment ways by count_ways(target_sum - i, num_dice - 1, memo)
68
69     Memoize ways as memo[(target_sum, num_dice)]
70     Return ways
71
72 Function probability(target_sum, num_dice):
73     Calculate total_ways using count_ways(target_sum, num_dice)
74     Calculate total_outcomes as 6 raised to the power of num_dice
75     Calculate probability as total_ways / total_outcomes
76     Return probability
77
78 Function simulate(num_simulations, num_dice, target_sum):
79     Initialize count_success to 0
80
81     Repeat num_simulations times:
82         Generate a list of num_dice random integers between 1 and 6
83         Calculate total_sum as the sum of the list
84
85         If total_sum equals target_sum:
86             Increment count_success
87
88     Calculate probability as count_success / num_simulations
89     Return probability
90
91 If length of command line arguments is not 2:
92     Print "Usage: python dice_simulation.py <num_simulations>"
93     Exit with status code 1
94 Parse num_simulations from command line argument
95
96 Initialize num_dice to 10
97 Initialize target_sum to 32
```

6. Full Python code and explanation

We created only one Python script for get the desired output for each task as mentioned in the coursework guidelines.

1) Importing the libraries

```
1 import random
2 import sys
```

This code was used to import the necessary libraries needed to run the Python script. Such as for parsing command line arguments, and generating random numbers. The libraries are random and sys.

2) Counting the possible ways to get the target sum

```
4  ## Task 2 part 1 calculations ##
5  def count_ways_to_get_sum(target_sum, num_dice, memo={}):
6      if num_dice == 1:
7          # Base case: one die, the sum must be between 1 and 6
8          return 1 if 1 <= target_sum <= 6 else 0
9
10     if (target_sum, num_dice) in memo:
11         return memo[(target_sum, num_dice)]
12
```

This code defines a function (**count_ways_to_get_sum**) that calculates the number of ways to get a specific sum using a given number of dice. The function uses memoization (caching) to store previously computed results improve efficiency and avoid redundant calculations.

The base scenario. If there is only one die, the sum must be between 1 and 6. So we return 1 if the target sum is between 1 and 6, and 0 otherwise. Then we check to see if the results for the given parameters are already in the Memoization cache, and if it is we return it.

3) Adding the total number of ways

```
13     ways = 0
14     for i in range(1, 7):
15         if target_sum - i >= 0:
16             ways += count_ways_to_get_sum(target_sum - i, num_dice - 1, memo)
17
18     memo[(target_sum, num_dice)] = ways
19     return ways
20
```

These code lines within the (**count_ways_to_get_sum**) function handle base cases and recursive calls to calculate the ways to get the target sum. The code returns the total number of ways.

4) Calculates the probability of getting the target sum

```
21 def probability_of_sum(target_sum, num_dice):
22     total_ways = count_ways_to_get_sum(target_sum, num_dice)
23     total_outcomes = 6 ** num_dice
24     probability = total_ways / total_outcomes
25     return probability ## part 1 end ##
```

This function (**probability_of_sum**) calculates the probability of getting the specified sum with the given number of dice by dividing the total number of ways by the total number of outcomes.

5) Task 2.2 simulation function

```
27 ## Task 2 part 2 calculations ##
28 def simulate_dice_throws(num_simulations, num_dice, target_sum):
29     count_success = 0
30
```

This function (**simulate_dice_throws**) simulates throwing a number of dice and a number of simulations times for each simulation.

6) Find the probability of specified simulation times (N)

```
30
31     for _ in range(num_simulations):
32         # Simulate throwing num_dice and summing the results
33         dice_results = [random.randint(a:1, b:6) for _ in range(num_dice)]
34         total_sum = sum(dice_results)
35
36         # Check if sum matches the target_sum
37         if total_sum == target_sum:
38             count_success += 1
39
40     probability = count_success / num_simulations
41     return probability ## part 2 end ##
42
```

The above code lines are within the (**simulate_dice_throws**) function perform the simulation using random dice throws, and count the number of successful throws where the sum matches the target sum. Finally, calculate the simulation probability.

7) Main function

```
44 # Check if the number of CL arguments is correct
45 if len(sys.argv) != 2:
46     print("Usage: python dice_simulation.py <num_simulations>")
47     sys.exit(1)
48
49 # Parse CL argument
50 num_dice = 10
51 target_sum = 32
52 num_simulations = int(sys.argv[1])
53
54 #Calculate the exact probability
55 result = probability_of_sum(target_sum, num_dice)
56 print(f"probability of getting a sum of {target_sum} from {num_dice} dice throws is: {result:.10f}")
57
58 # Perform the simulation
59 result = simulate_dice_throws(num_simulations, num_dice, target_sum)
60 print(f"Simulated probability of getting a sum of {target_sum} from "
61       f"{num_dice} dice throws in {num_simulations} simulations: {result:.4f}")
62
```

1.Command Line Handling

Inside the main the first section checks if the correct number of command-line arguments is provided (only 1 argument, which is the number of simulations). If not, it prints a usage message and exits.

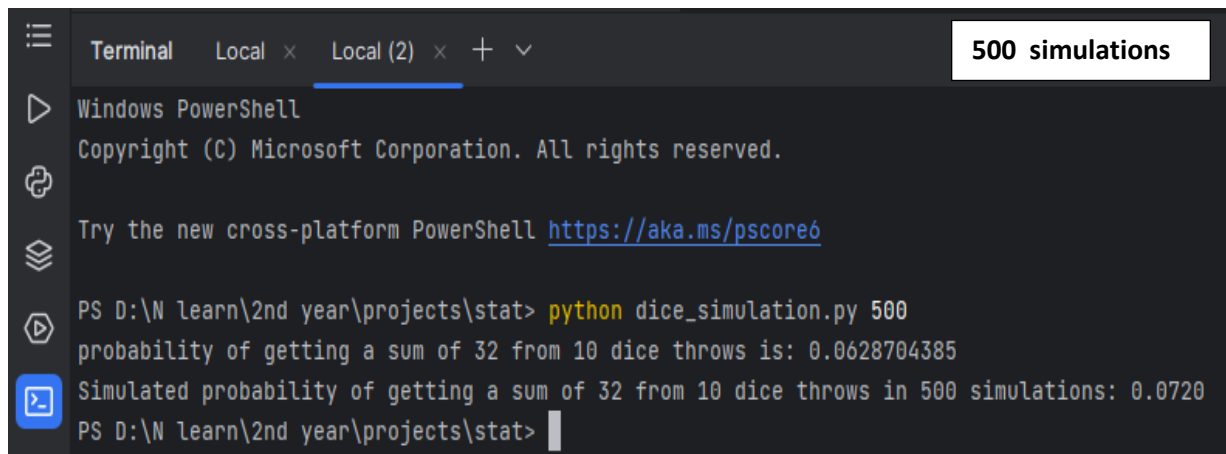
It then parses a number of dice, target sum, and number of simulations in a command line argument.

2. Print Results

This part calculates and prints the exact probability using the **(probability_of_sum)** function. Then, it performs the simulation using the **(simulate_dice_throws)** function and prints the simulated probability.

7. Results Analyze

As mentioned in the coursework deliverables this Python program can be executed through a terminal after giving the file directory path. The program facilitates parsing arguments so that the simulation can be tested for any number (N) of simulations specified by the user at the end of the command line using this **python dice_simulation.py <Num_simulations>** . The outputs are shown below for different simulations.



The screenshot shows a Windows PowerShell terminal window with the title bar 'Terminal Local x Local (2) x + v'. A tab on the right indicates '500 simulations'. The terminal output is as follows:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\N learn\2nd year\projects\stat> python dice_simulation.py 500
probability of getting a sum of 32 from 10 dice throws is: 0.0628704385
Simulated probability of getting a sum of 32 from 10 dice throws in 500 simulations: 0.0720
PS D:\N learn\2nd year\projects\stat> |
```

Here we show the probability outputs for 500 simulations in 3 continuous tries.

1st try,

number of iterations: 500

Probability = 0.0680

2nd try,

number of iterations: 500

Probability = 0.0628

3rd try,

number of iterations: 500

Probability = 0.0720

As we can see, the probabilities of the simulation (N=500) are approximately in the range of 0.060 – 0.075 when we run the code several times.

The probability(P) of getting a sum of 32 from 10 dice throws is = 0.06287043

8. Summary and conclusion

As you can see in the above images shown in the results part of the probability of getting the sum of 32 from 10 dice throws is always 0.06287043 (P) and it's unchanged. The reason is there are constant possible ways of getting 32 as the summation of 10 dice throws. So, if we do this practically for infinity times the number of possible ways of getting 32 as summation is not changed but continues until it ends. So, the probability value remains unchanged and it is proven in math calculation part. The goal is to achieve a sum of 32. Since there is only a limited number of ways to obtain a sum of 32 with 10 dice, the probability of success becomes stable over unlimited simulations.

Let's move to the output and see what number of ways and outcomes are calculated randomly,

```
C:\Users\User\AppData\Local\Programs\Python\Python37\python.exe "D:\N lea
3801535
60466176
probability of getting a sum of 32 from 10 dice throws is: 0.0628704385
Process finished with exit code 0
```

Number of ways that the sum of dice can add up exactly to 32 = 3801535

Total number of outcomes = 60466176

The probability that 10 dice throws add up exactly to 32 is,

Number of ways 32 can be added up / Total number of outcomes =

$3801535 / 60466176 = 0.06287 \approx (0.0629)$

But when we give an exact number of times(N) as the simulation value the probability will change according to that number. So, the value will change each time with the impact of random number generation but it varies in a range according to the given simulation value(N).

- 300 simulations = 0.03 - 0.05
- 500 simulations = 0.06 - 0.07

When we increase the number of simulations like 10K, 15K, and 20K the simulated probability value becomes closer to the actual value (P) 0.06287074. An example output is given below.

```
Run  dice_simulation  x
C:\Users\User\AppData\Local\Programs\Python\Python37\python.exe "D:\N learn\2nd year\projects\
probability of getting a sum of 32 from 10 dice throws is: 0.0628704385
Simulated probability of getting a sum of 32 from 10 dice throws in 10000 simulations: 0.0630
Process finished with exit code 0
```

Group contribution

PU Index No	Student Name	Task	Contribution
10899486	Sangarange Sandanayake	Task 1	<ul style="list-style-type: none">• Program implementation Preparing the Python code for task 1. Writing the code according to the mathematical calculations.• Program testing And Rewriting Checking the accuracy of Python code and error management and correcting it with re-writing.• Simulation Execution Decide the number of iterations, add a command line argument, and validate the results.• Data logging Creating Excel sheets, Checking Excel sheet data, finalizing data, and implementing that data to visualization.• Data Analysis & Graphical Drawings All the graphs drawing and picture designing for the final report. Data identification using graphs and implementation added to the documentation.• Report finalizing Report writing for the task 1. Check the overall report. error correction, summary writing, cover page designing using Figma, and assuming tasks to members.• Mathematical Calculations Doing all the mathematical calculations and mathematical implementation for task 1.

10899177	Urulugastenne Amarakone	Task 1	<ul style="list-style-type: none"> • Simulation Refinement Identifying the areas in which the simulation code can be optimized for good performance. • Program implementation Preparing the Python code according to the simulation and writing the code according to the mathematical calculations. • Excel Sheet setup Define columns for different variables, run multiple experiments, and check that all the necessary data is recorded correctly. • Data analysis, Graphical Representation Mean and mode calculation using a program checking the accuracy with manual calculations, graphs describing, and graph report writing. • Report writing Python code documentation, code visualization, and summary writing.
10899478	Gosisa Jinasena	Task 2	<ul style="list-style-type: none"> • Creating the logic Calculating the minimum sum and maximum sum, how to count possible ways equation. • Mathematical Operations Making a mathematical equation for finding the probability, Finding the probability. • Program implementation Preparing the Python code for task two part one • Program testing Giving inputs by command line argument in the terminal for testing the Python program. • Report writing Introduction, Python code explanation, Mathematical explanation and discussion.

10899302	Weerasinghe Dissanayaka	Task 2	<ul style="list-style-type: none"> • Report writing Python code explanation, Discussion, Conclusion, and pseudocode • Creating the pseudo-code for the scenario Identify the whole scenario and make a pseudo-code • Program implementation Preparing the Python code for task two part two • Code error fixing Recheck the code of task 2 and error recorection. • Report finalizing Finalize the report of task 2 and design images and other image outputs.
----------	-------------------------	--------	--

Appendices

1)GitHub repository link (full project file):-

<https://github.com/scssandanayake/computational-theory-and-stat-coursework.git>

2)Excel sheet link:- (please click “ view raw ” to download the Excel file after opening the page)

<https://github.com/scssandanayake/computational-theory-and-stat-coursework/blob/main/dart%20game%20results.xlsx>

3)task 1 source code –

https://github.com/scssandanayake/computational-theory-and-stat-coursework/blob/main/task%201%20code%20file/dart_simulation.py

4)task 2 source code –

https://github.com/scssandanayake/computational-theory-and-stat-coursework/blob/main/task%202%20code%20file/dice_simulation.py

References

Brozius, H., 2019. *quora*. [Online]

Available at: <https://www.quora.com/What-is-the-probability-that-10-dice-throws-add-up-exactly-to-32>

calculus, 1., n.d. [Online]

Available at: <https://www.123calculus.com/en/polynomial-coefficients-page-1-60-110.html>

IBM, 2022. [Online]

Available at: <https://www.ibm.com/topics/monte-carlo-simulation>

Smith, S., 2020. *quora*. [Online]

Available at: <https://www.quora.com/What-is-the-probability-that-10-dice-throws-add-up-exactly-to-32>

youtube, 2022. [Online]

Available at: https://www.youtube.com/results?search_query=find%2Bpi%2Busing%2Bmonte%2Bcarlo