Science 2 - Assignment 2

Name: Sreeja Guduri
Roll number: 2021102007

QUESTION 1:

Science 2 - Assignment 2

Roll number - 2021102007
$\therefore$ option 3

3 (a) e^x in the range $[-3, 3]$

SOLVING BY HAND :-

$f(x) = e^x$

$$\int_{-3}^{3} f(x) = \int_{-3}^{3} e^x \, dx$$

$$= e^x \Big|_{-3}^{3}$$

$$= e^3 - e^{-3}$$

$$\approx 20.035$$

MONTE CARLO METHOD :-

In the Monte Carlo method, we first select and sample the function randomly such that the samples are uniformly distributed over the desired range. Then we find the average of the sum of the areas of these samples. This value should converge to the value of the definite integral.

This method is done using python code. We can see that for N = 10000, the value comes out to be 20.136 which is very close to the desired value.
Hence, verified.

3(b) $e^x \cos x$ in the range $[-1, 1]$

SOLVING BY HAND:

$f(x) = e^x \cos x$

$$\int f(x)\,dx = \int e^x \cos x\,dx$$

$$= \cos x\, e^x - \int -\sin x\, e^x\,dx \quad \text{[Integration by parts]}$$

$$= \cos x\, e^x + \int e^x \sin x\,dx$$

$$= e^x \cos x + e^x \sin x - \int \cos x\, e^x\,dx$$

$$I = e^x(\cos x + \sin x) - I$$

$$2I = e^x(\cos x + \sin x)$$

$$\Rightarrow I = \int e^x \cos x\,dx = \frac{e^x(\cos x + \sin x)}{2}$$

Applying the limits,

$$\int_{-1}^{1} e^x \cos x\,dx = \left[\frac{e^x(\cos x + \sin x)}{2}\right]_{-1}^{1}$$

$$= \frac{3.75 - (-0.11)}{2}$$

$$= 1.93$$

MONTE CARLO METHOD:

We can see from the code used, when N = 10000, the value obtained is 1.927 which is very close our desired value of 1.93.

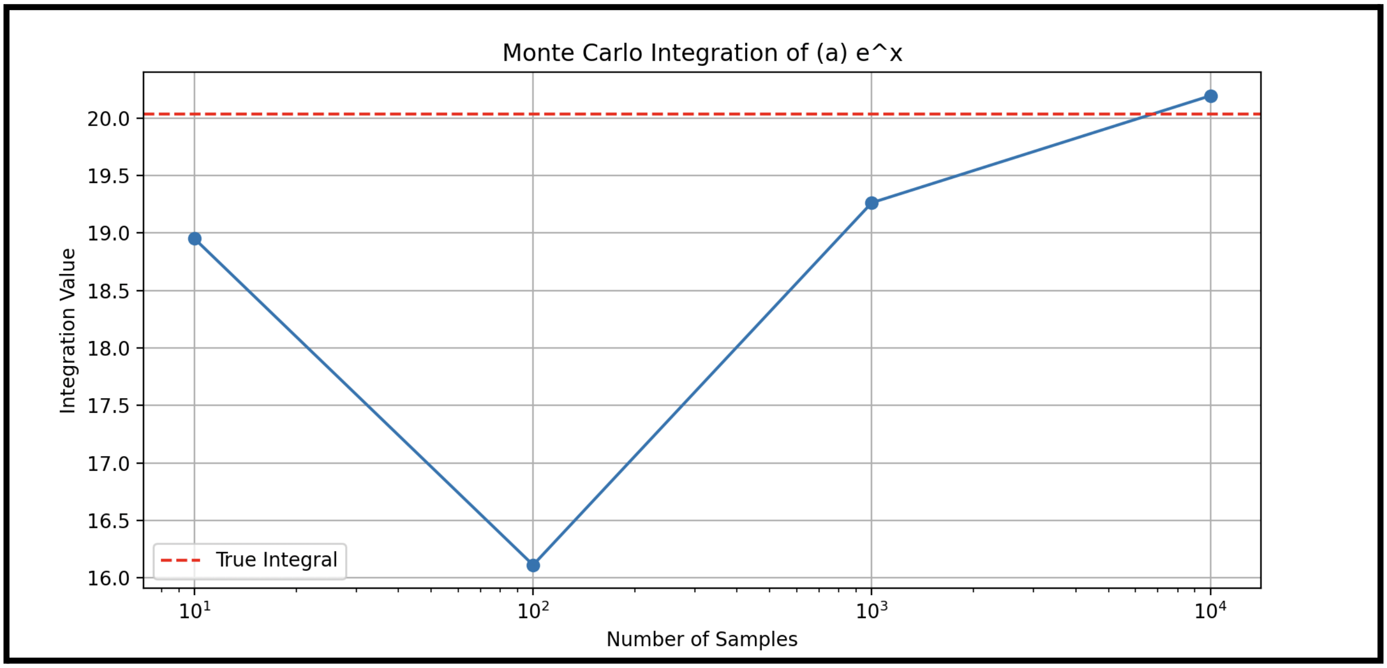
Hence, verified.

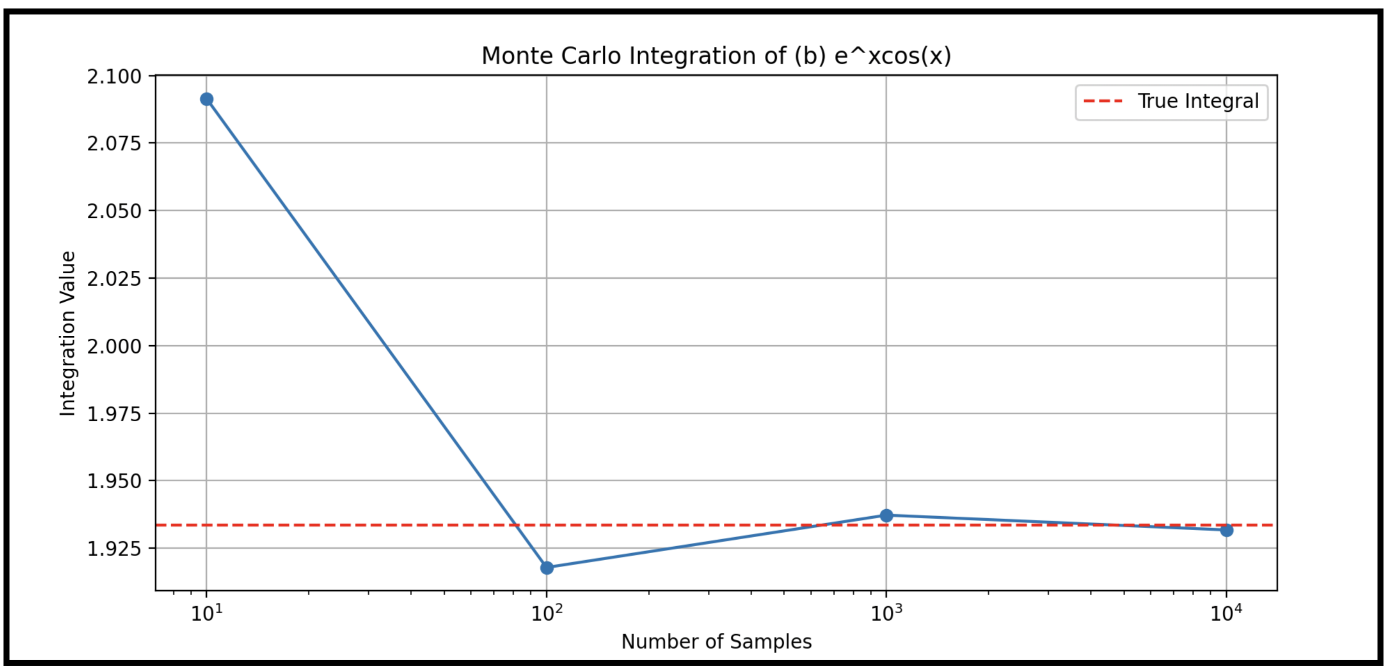
Code:

```python
import numpy as np
import matplotlib.pyplot as plt

samples = [10, 100, 1000, 10000]

#PART A
# Calculate integration value for different sample sizes
a = -3
b = 3
result = []
print("PART A:\n")
for N in samples:
  # random values uniformly distributed between a and b
  x_values = np.random.uniform(a, b, N)

  y_values = np.exp(x_values)

  # Monte Carlo
  MC = np.mean(y_values) * (b - a)
  result.append(MC)

  # Print the results
  print(f"N = {N}   MC estimate = {MC:.3f}")
```

```
true_value = np.exp(b) - np.exp(a)
# Plotting
plt.figure()
plt.plot(samples, result, marker='o')
plt.xscale('log')
plt.axhline(y=true_value, color='r', linestyle='--', label='True Integral')
plt.xlabel('Number of Samples')
plt.ylabel('Integration Value')
plt.title('Monte Carlo Integration of (a) e^x')
plt.legend()
plt.grid(True)
plt.show()


#part b
print("PART B:\n")
a = -1
b = 1
result = []
for N in samples:
  # random values uniformly distributed between a and b
  x_values = np.random.uniform(a, b, N)

  y_values = np.exp(x_values)*np.cos(x_values)

  # Monte Carlo
  MC = np.mean(y_values) * (b - a)
  result.append(MC)

  # Print the results
  print(f"N = {N}   MC estimate = {MC:.3f}")

true_value = ((np.exp(b) * (np.cos(b) + np.sin(b))) - (np.exp(a) * (np.cos(a) + np.sin(a))))/ 2
# Plotting
plt.figure()
plt.plot(samples, result, marker='o')
plt.xscale('log')
plt.axhline(y=true_value, color='r', linestyle='--', label='True Integral')
plt.xlabel('Number of Samples')
plt.ylabel('Integration Value')
plt.title('Monte Carlo Integration of (b) e^xcos(x)')
plt.legend()
plt.grid(True)
plt.show()
```

<u>Results:</u>


Monte Carlo Integration of (a) e^x
Integration Value
20.0
19.5
19.0
18.5
18.0
17.5
17.0
16.5
16.0
True Integral
10^1
10^2
10^3
10^4
Number of Samples


Monte Carlo Integration of (b) e^xcos(x)
Integration Value
2.100
2.075
2.050
2.025
2.000
1.975
1.950
1.925
True Integral
10^1
10^2
10^3
10^4
Number of Samples

QUESTION 2:

a)

$a = -(2021102007 \,\%\, 5 + 1)$

$= -3$

14:00

$b = 2 * (2021102007 \,\%\, 10)$

15:00 $= 14$

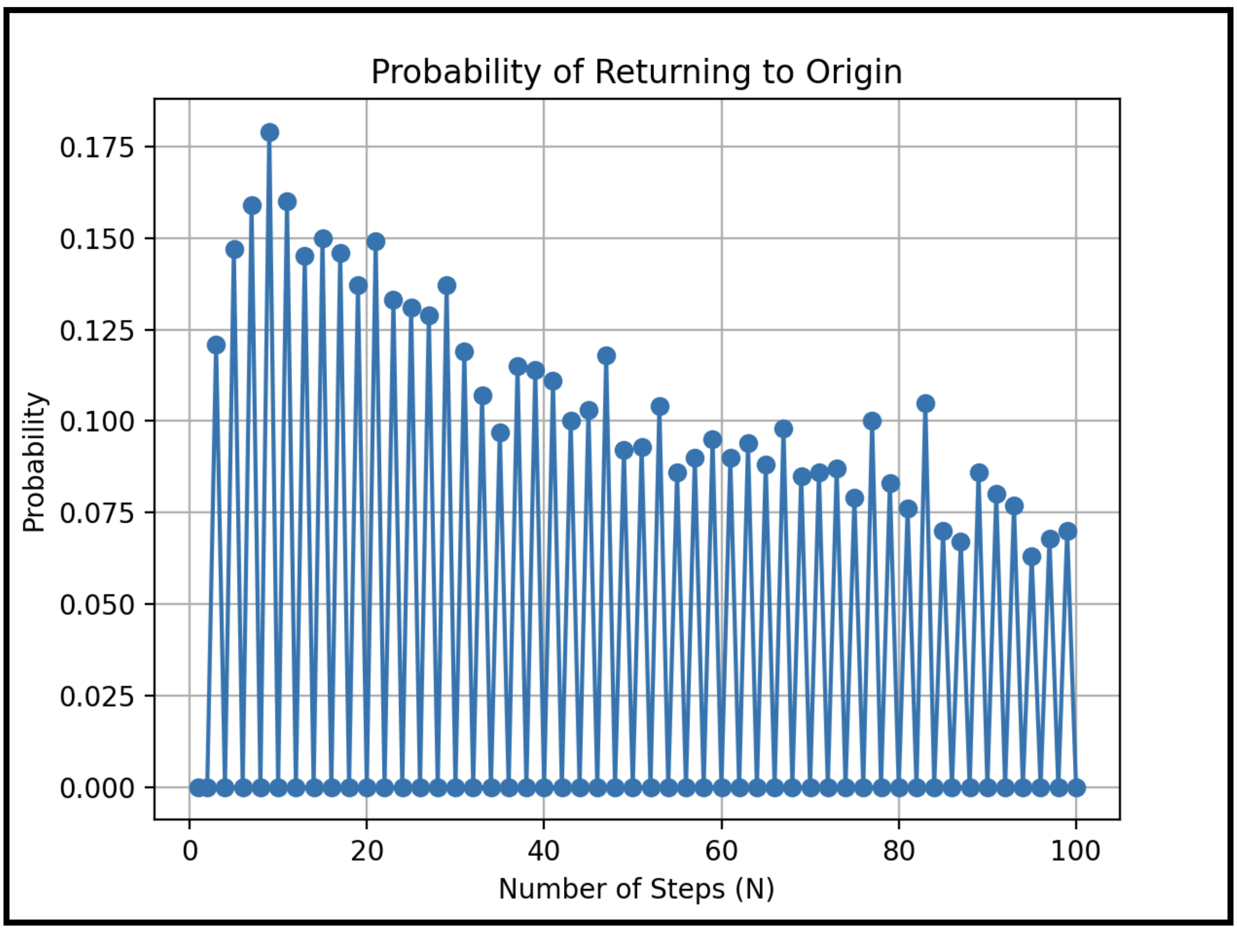
For this part, we write code to simulate a random walk where we can either take 1 step forward or backward (1 or -1 respectively). We then calculate the probability of reaching back to the origin (0) by simulating the experiment at least a 100 times.

<u>Code:</u>

```python
import numpy as np
import matplotlib.pyplot as plt

def probability(N, n):
  count = 0

  for i in range(n): # number of simulations
    position = -3 #initial position
    for j in range(N):
      step = np.random.choice([-1, 1])
      position += step
```

```
    if position == 0:
      count += 1

  return count / n

N_values = list(range(1, 101))
probabilities = [probability(N, 1000) for N in N_values]

plt.plot(N_values, probabilities, marker='o', linestyle='-')
plt.title('Probability of Returning to Origin')
plt.xlabel('Number of Steps (N)')
plt.ylabel('Probability')
plt.grid(True)
plt.show()
```

Result:

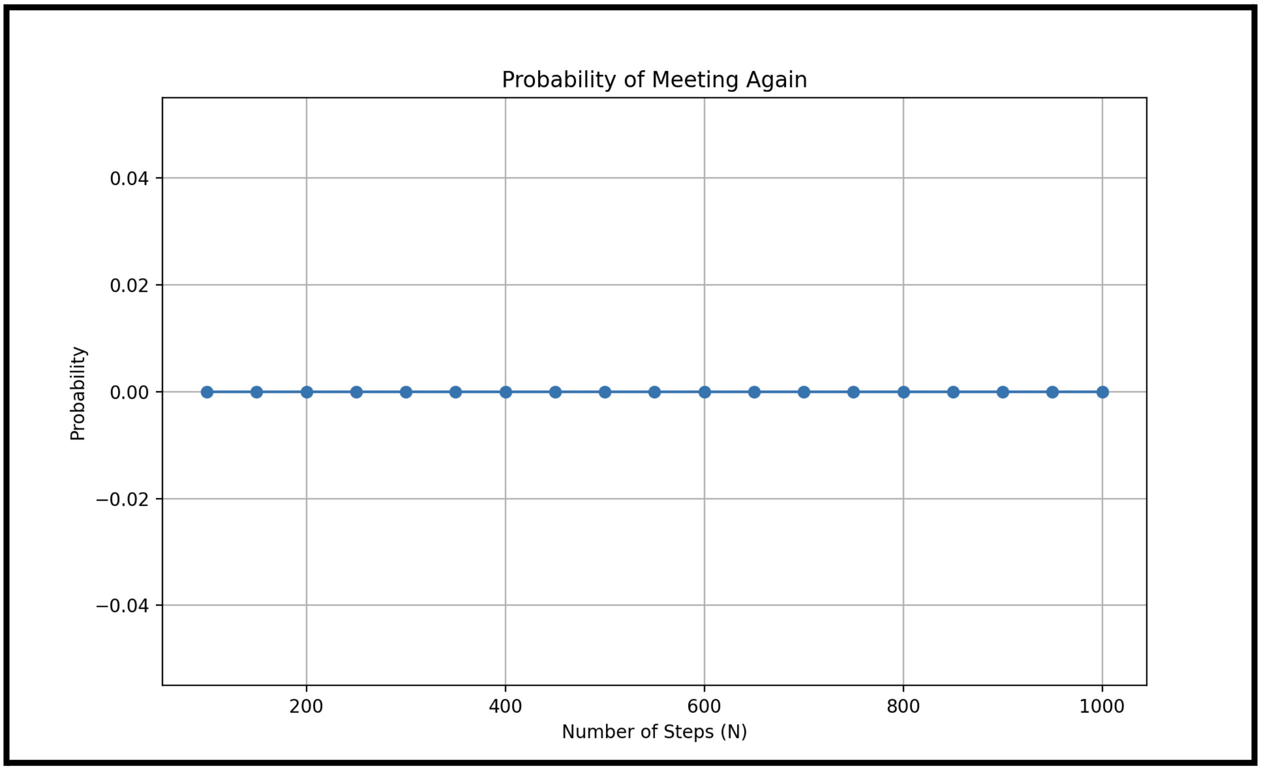


b) This is implemented very similar to part (a) except that now we simulate two random walks and see if they meet after N timesteps. We notice that the probability of the two people meeting is zero (because the difference between their initial points is odd)

Code:

```
Python

import numpy as np
import matplotlib.pyplot as plt

def probability(N, n):
  count = 0
  for i in range(n):
    pos1 = -3
    pos2 = 14

    for j in range(N):
      step = np.random.choice([-1, 1])
      pos1 += step

    for j in range(N):
      step = np.random.choice([-1, 1])
      pos2 += step

    if pos1 == pos2:
      count += 1
  return count / n

N_values = list(range(100, 1001, 50))
print(N_values)
probabilities = [probability(N,1000) for N in N_values]

plt.plot(N_values, probabilities, marker='o', linestyle='-')
plt.title('Probability of Meeting Again')
plt.xlabel('Number of Steps (N)')
plt.ylabel('Probability')
plt.grid(True)
plt.show()
```

Result:



c) For the final part of the question, we calculate the mean displacement traveled by calculating the mean displacement for each value of N.

Code:

```python
import numpy as np
import matplotlib.pyplot as plt

def mean(N, n):
  disp = []
  mean_disp = []
  for i in range(n):
    position = 0 # start at origin
    for j in range(N):
```

```
            step = np.random.choice([-1, 1])
            position += step
            disp.append(abs(position))

        temp = np.mean(disp)
        mean_disp.append(temp)
    return np.mean(mean_disp) # mean of all the mean displacements for a particular N

N_values = list(range(1, 101))
mean_displacements = [mean(N, 1000) for N in N_values]

plt.plot(N_values, mean_displacements, marker='o', linestyle='-')
plt.title('Mean Displacement after N Steps')
plt.xlabel('Number of Steps (N)')
plt.ylabel('Mean Displacement')
plt.grid(True)
plt.show()
```

Result: