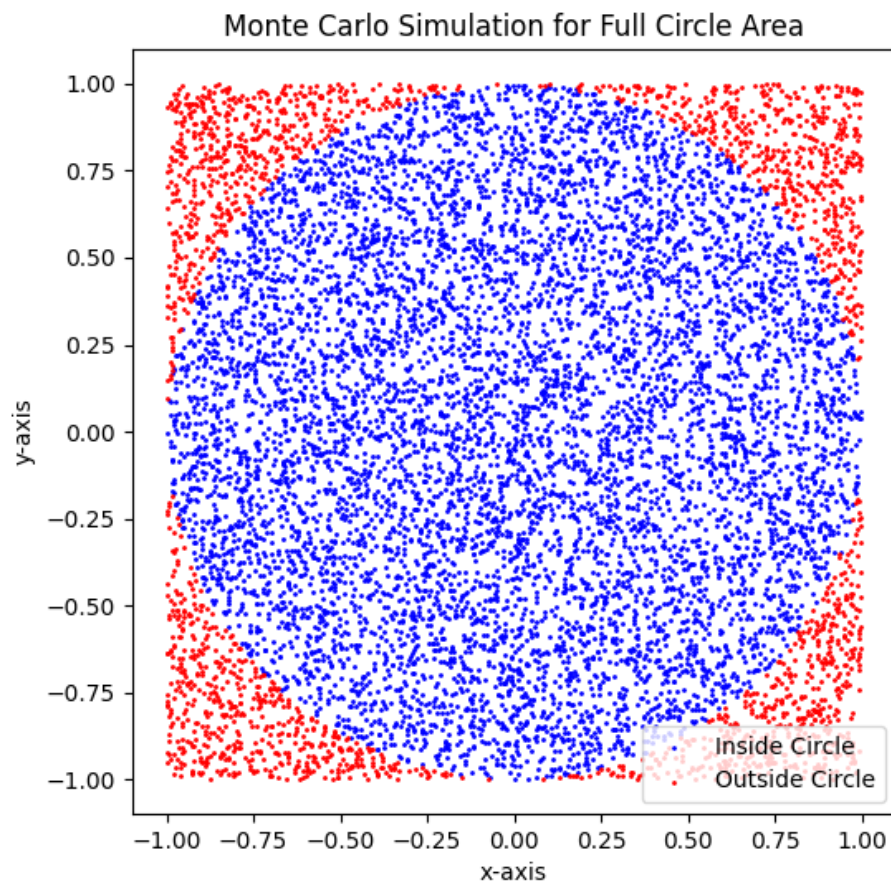


Ahmad Rifaldi A.

D121211057

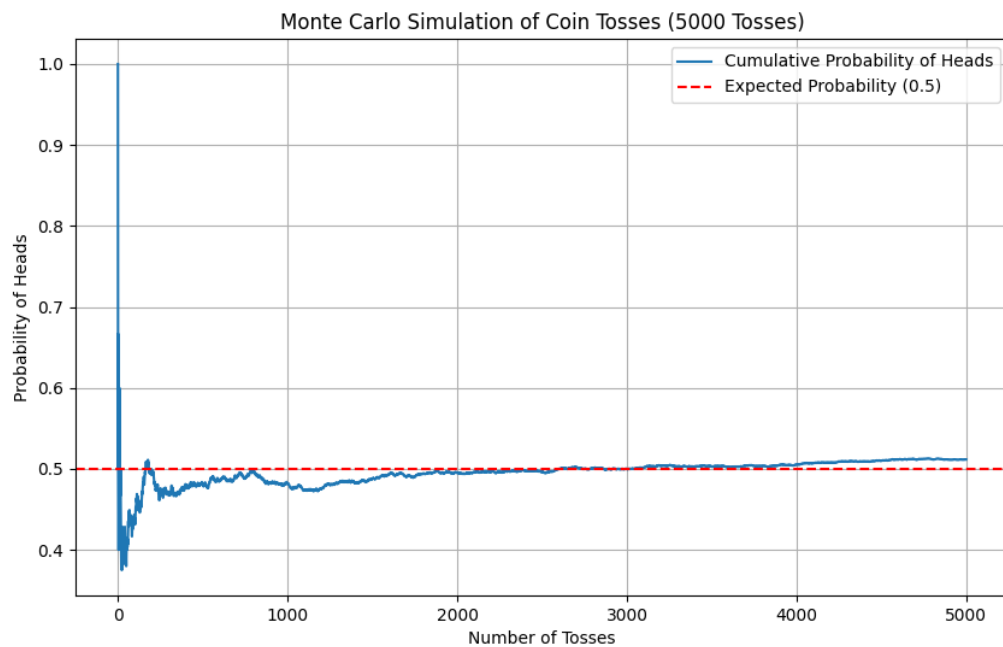
1.

```
probabilitas > D121211051_ahmad zacky fayiz roesdi_tugas 3 > task 1 > ...
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def is_inside_circle(x, y):
5     return x**2 + y**2 <= 1
6
7 np.random.seed(42)
8 n_points = 10000
9 x = np.random.uniform(-1, 1, n_points)
10 y = np.random.uniform(-1, 1, n_points)
11
12 inside_circle = is_inside_circle(x, y)
13
14 plt.figure(figsize=(6,6))
15 plt.scatter(x[inside_circle], y[inside_circle], color='blue', s=1, label='Inside Circle')
16 plt.scatter(x[~inside_circle], y[~inside_circle], color='red', s=1, label='Outside Circle')
17 plt.gca().set_aspect('equal', adjustable='box')
18
19 plt.title('Monte Carlo Simulation for Full Circle Area')
20 plt.xlabel('x-axis')
21 plt.ylabel('y-axis')
22
23 plt.legend(loc="lower right")
24 plt.show()
25
```



2.

```
probabilitas > D121211051_ahmad zacky fayiz roesdi_tugas 3 > task 2 > ...
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def lemparkoin(n_tosses):
5     return np.random.choice([0, 1], size=n_tosses) # 0 for tails, 1 for heads
6
7 def cumulative_probabilities(tosses):
8     return np.cumsum(tosses) / np.arange(1, len(tosses) + 1)
9
10 n_tosses = 5000
11 tosses = lemparkoin(n_tosses)
12 cumulative_prob = cumulative_probabilities(tosses)
13
14 plt.figure(figsize=(10, 6))
15 plt.plot(cumulative_prob, label="Cumulative Probability of Heads")
16 plt.axhline(y=0.5, color='r', linestyle='--', label='Expected Probability (0.5)')
17 plt.title(f"Monte Carlo Simulation of Coin Tosses ({n_tosses} Tosses)")
18 plt.xlabel("Number of Tosses")
19 plt.ylabel("Probability of Heads")
20 plt.legend()
21 plt.grid(True)
22 plt.show()
23
```



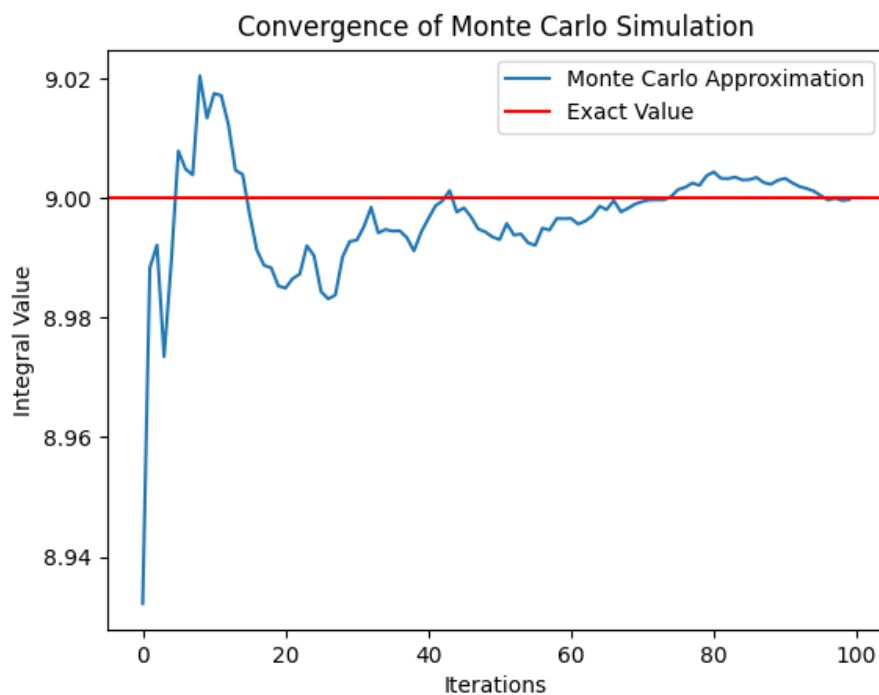
Tugas kelompok

Ahmad Zacky Fayiz Roesdi (D121211051)

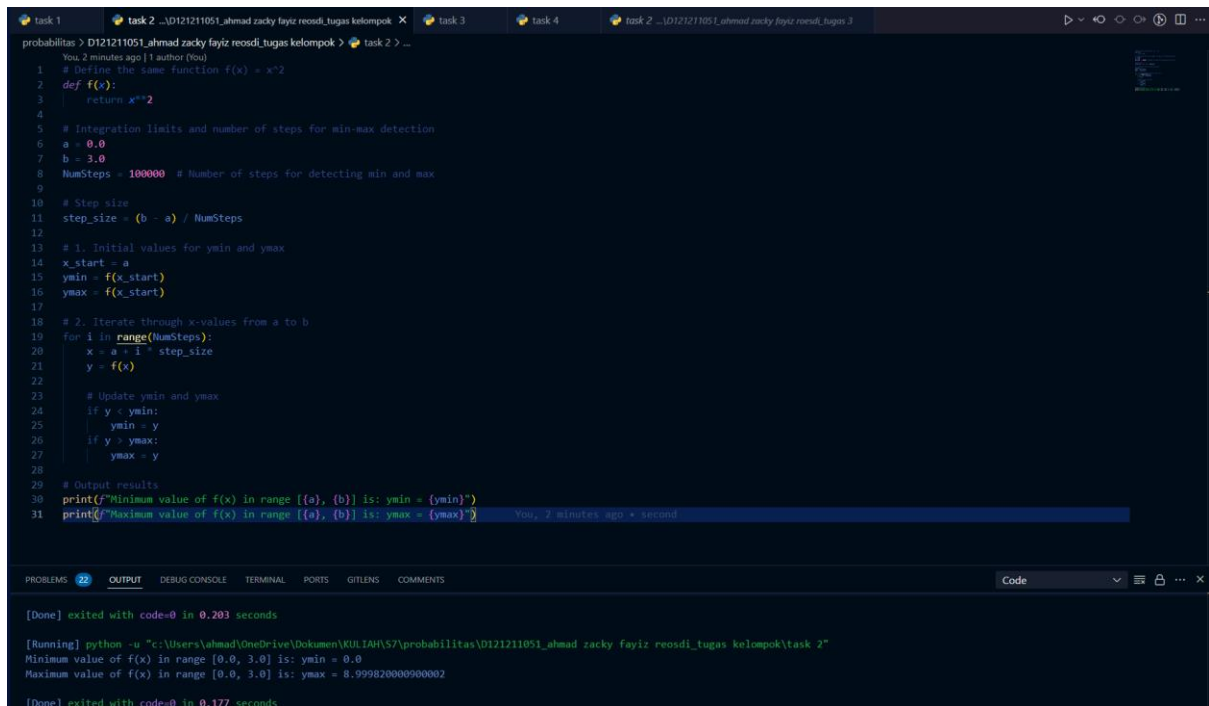
Ahmad Rifaldi A. (D121211057)

1.

```
You, 49 seconds ago | 1 author (You)
1 import random
2 import matplotlib.pyplot as plt
3
4 # 1. PRNG Initialization
5 random.seed(2)
6
7 # 2. Define the function f(x) = x^2
8 def f(x):
9     return x**2
10
11 # 3. Integration limits and number of steps
12 a = 0.0
13 b = 3.0
14 n = 1000000 # Number of random points
15
16 # Monte Carlo Integration
17 sum_f = 0
18 samples = []
19 for i in range(n):
20     x_rand = random.uniform(a, b)
21     sum_f += f(x_rand)
22     if (i + 1) % (n // 100) == 0:
23         samples.append((b - a) * sum_f / (i + 1))
24
25 # Final approximation of the integral
26 monte_carlo_result = (b - a) * sum_f / n
27
28 # 4. Exact solution
29 exact_solution = (b**3) / 3 - (a**3) / 3
30
31 # 5. Output results
32 print(f"Monte Carlo Approximation: {monte_carlo_result}")
33 print(f"Exact Solution: {exact_solution}")
34 print(f"Error: {abs(monte_carlo_result - exact_solution)}")
35
36 # 6. Plotting
37 plt.plot(samples, Label="Monte Carlo Approximation")
38 plt.axhline(y=exact_solution, color='r', linestyle='--', Label="Exact Value")
39 plt.xlabel("Iterations")
40 plt.ylabel("Integral Value")
41 plt.title("Convergence of Monte Carlo Simulation")
42 plt.legend()
43 plt.show()
```



2.



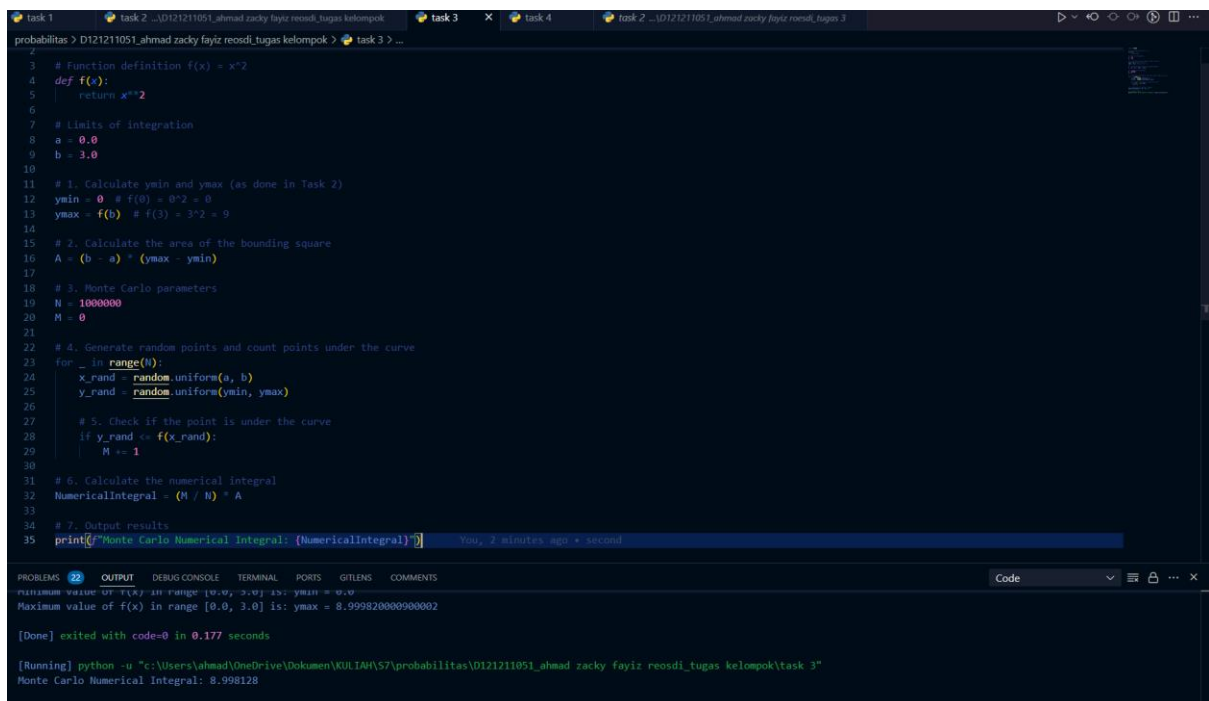
```
task 1 task 2 ...D121211051_ahmad zacky fayiz reosdi_tugas kelompok X task 3 task 4 task 2 ...D121211051_ahmad zacky fayiz reosdi_tugas 3
probabilitas > D121211051_ahmad zacky fayiz reosdi_tugas kelompok > task 2 > ...
You 2 minutes ago | 1 author (You)
1 # Define the same function f(x) = x^2
2 def f(x):
3     return x**2
4
5 # Integration limits and number of steps for min-max detection
6 a = 0.0
7 b = 3.0
8 NumSteps = 100000 # Number of steps for detecting min and max
9
10 # Step size
11 step_size = (b - a) / NumSteps
12
13 # 1. Initial values for ymin and ymax
14 x_start = a
15 ymin = f(x_start)
16 ymax = f(x_start)
17
18 # 2. Iterate through x-values from a to b
19 for i in range(NumSteps):
20     x = a + i * step_size
21     y = f(x)
22
23     # Update ymin and ymax
24     if y < ymin:
25         ymin = y
26     if y > ymax:
27         ymax = y
28
29 # Output results
30 print(f"Minimum value of f(x) in range [{a}, {b}] is: ymin = {ymin}")
31 print(f"Maximum value of f(x) in range [{a}, {b}] is: ymax = {ymax}")
You, 2 minutes ago • second

PROBLEMS 22 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS COMMENTS Code
[Done] exited with code=0 in 0.203 seconds

[Running] python -u "c:\Users\ahmad\OneDrive\Dokumen\KULIAH\S7\probabilitas\D121211051_ahmad zacky fayiz reosdi_tugas kelompok\task 2"
Minimum value of f(x) in range [0.0, 3.0] is: ymin = 0.0
Maximum value of f(x) in range [0.0, 3.0] is: ymax = 8.999820000900002

[Done] exited with code=0 in 0.177 seconds
```

3.



```
task 1 task 2 ...D121211051_ahmad zacky fayiz reosdi_tugas kelompok X task 3 task 4 task 2 ...D121211051_ahmad zacky fayiz reosdi_tugas 3
probabilitas > D121211051_ahmad zacky fayiz reosdi_tugas kelompok > task 3 > ...
2
3 # Function definition f(x) = x^2
4 def f(x):
5     return x**2
6
7 # Limits of integration
8 a = 0.0
9 b = 3.0
10
11 # 1. Calculate ymin and ymax (as done in Task 2)
12 ymin = 0 # f(0) = 0^2 = 0
13 ymax = f(b) # f(3) = 3^2 = 9
14
15 # 2. Calculate the area of the bounding square
16 A = (b - a) * (ymax - ymin)
17
18 # 3. Monte Carlo parameters
19 N = 1000000
20 M = 0
21
22 # 4. Generate random points and count points under the curve
23 for _ in range(N):
24     x_rand = random.uniform(a, b)
25     y_rand = random.uniform(ymin, ymax)
26
27     # 5. Check if the point is under the curve
28     if y_rand <= f(x_rand):
29         M += 1
30
31 # 6. Calculate the numerical integral
32 NumericalIntegral = (M / N) * A
33
34 # 7. Output results
35 print(f"Monte Carlo Numerical Integral: {NumericalIntegral}")
You, 2 minutes ago • second

PROBLEMS 22 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS COMMENTS Code
Minimum value of f(x) in range [0.0, 3.0] is: ymin = 0.0
Maximum value of f(x) in range [0.0, 3.0] is: ymax = 8.999820000900002

[Done] exited with code=0 in 0.177 seconds

[Running] python -u "c:\Users\ahmad\OneDrive\Dokumen\KULIAH\S7\probabilitas\D121211051_ahmad zacky fayiz reosdi_tugas kelompok\task 3"
Monte Carlo Numerical Integral: 8.998128
```

4.

```

1 You 4 minutes ago | 1 author (You)
2 import random
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 # Define the function f(x) = x^2
7 def f(x):
8     return x**2
9
10 # Limits of integration
11 a = 0.0
12 b = 3.0
13
14 # 1. Generate data for function curve
15 x_vals = np.linspace(a, b, 500) # 500 points between a and b
16 y_vals = f(x_vals)
17
18 # 2. Monte Carlo parameters
19 N = 10000
20 M = 0
21 ymin = 0
22 ymax = f(b) # f(3) = 9
23
24 # Lists to store points under and above the curve
25 points_under = []
26 points_above = []
27
28 # 3. Generate random points
29 for _ in range(N):
30     x_rand = random.uniform(a, b)
31     y_rand = random.uniform(ymin, ymax)
32
33     # Check if the point is under the curve
34     if y_rand <= f(x_rand):
35         points_under.append((x_rand, y_rand)) # Points under the curve
36     else:
37         points_above.append((x_rand, y_rand)) # Points above the curve
38
39 # 4. Plotting the function curve and random points
40 plt.plot(x_vals, y_vals, color='red', label=r'$f(x) = x^2$') # Function curve in red
41
42 # Unpack points for plotting
43 if points_under:
44     x_under, y_under = zip(*points_under)
45     plt.scatter(x_under, y_under, color='blue', s=1, label='Points under the curve')
46
47 if points_above:
48     x_above, y_above = zip(*points_above)
49     plt.scatter(x_above, y_above, color='yellow', s=1, label='Points above the curve')
50
51 plt.title("Monte Carlo Simulation: f(x) = x^2")
52 plt.xlabel("x")
53 plt.ylabel("y")
54 plt.legend()
55
56 plt.show()
57
58

```

