

TUGAS 2 – IMPLEMENTASI SIMULASI MONTE CARLO

Nama Lengkap	Mohammad Wijdan Arrosyid
NIM	202210370311497
Mata Kuliah	Pemodelan dan Simulasi Data 6B

Deskripsi : “Implementasi Simulasi Monte Carlo untuk menentukan nilai π ”

Skema :

1. Kelereng yang dimiliki berjumlah (N) 10.000 buah
2. Semua kelereng dilempar secara **acak** dalam kotak persegi yang didalamnya terdapat lingkaran
3. Kelereng yang berada **di dalam** lingkaran berwarna **hijau** dan **di luar** berwarna **orange**
4. Setelah semua kelereng dilempar, kita ukur jarak posisi kelereng dari pusat, menghitung rasio kelereng untuk menghasilkan estimasi π .
5. Rumus yang digunakan:

- Menghitung Jarak Kelereng dari Pusat (Teorema Pythagoras):

$$d = \sqrt{x^2 + y^2}$$

- Menentukan Kelereng di Dalam Lingkaran:

$$d \leq r$$

- Estimasi Nilai π (Pi)

$$\pi \approx 4 \times \left(\frac{N_{\text{inside}}}{N_{\text{total}}} \right)$$

- Probabilitas Kelereng Masuk ke Dalam Lingkaran

$$P = \frac{N_{\text{inside}}}{N_{\text{total}}}$$

Komponen :

1. Bahasa Python
2. Simulasi Monte Carlo
3. Pustaka Lib. numpy
 - Menghasilkan titik acak untuk posisi kelereng dalam wadah.
 - Menghitung jarak kelereng ke pusat lingkaran
 - Menghitung jumlah kelereng di dalam lingkaran
 - Menghitung estimasi π
 - Menentukan kelereng di luar dan di dalam untuk divisualisasikan dengan plot.
4. Pustaka Lib. matplotlib.pyplot
 - Membuat persegi dan lingkaran
 - Visualisasi simulasi monte carlo dengan kelereng di dalam dan diluar dengan warna yang berbeda.

Source Code : *dapat diakses di GitHub ()*

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def generate_marbles(num_marbles, radius):
5     return np.random.uniform(-radius, radius, num_marbles), np.random.uniform(-radius, radius, num_marbles)
6
7 def calculate_distances(x, y):
8     return np.sqrt(x**2 + y**2)
9
10 def count_inside_circle(distances, radius):
11     return np.sum(distances <= radius)
12
13 def estimate_pi(num_inside, num_marbles):
14     return (num_inside / num_marbles) * 4, num_inside / num_marbles
15
16 def plot_simulation(num_marbles, radius, x, y, inside_circle, estimated_pi, probability):
17     fig, ax = plt.subplots(figsize=(6, 6))
18     ax.set_xlim(-radius, radius)
19     ax.set_ylim(-radius, radius)
20
21     ax.add_patch(plt.Rectangle((-radius, -radius), 2*radius, 2*radius, fill=False, color='blue', linewidth=2))
22     ax.add_patch(plt.Circle((0, 0), radius, fill=False, color='red', linewidth=2))
23
24     ax.scatter(x[inside_circle], y[inside_circle], color='green', s=5, label='Inside Circle')
25     ax.scatter(x[~inside_circle], y[~inside_circle], color='orange', s=5, label='Outside Circle')
26
27     text_str = f"Inside: {np.sum(inside_circle)}\nOutside: {num_marbles - np.sum(inside_circle)}"
28     ax.text(-radius, radius * 0.8, text_str, fontsize=12, color='black', bbox=dict(facecolor='white', alpha=0.7))
29
30     ax.legend()
31     ax.set_title(f"Monte Carlo Simulation (N={num_marbles})\nEstimated pi: {estimated_pi:.4f}, Probability: {probability:.4f}")
32     plt.show()
33
34 def monte_carlo_marbles(num_marbles):
35     radius = 0.5
36     x, y = generate_marbles(num_marbles, radius)
37     distances = calculate_distances(x, y)
38     num_inside = count_inside_circle(distances, radius)
39     estimated_pi, probability = estimate_pi(num_inside, num_marbles)
40     inside_circle = distances <= radius
41
42     plot_simulation(num_marbles, radius, x, y, inside_circle, estimated_pi, probability)
43     return estimated_pi, num_inside, num_marbles - num_inside, probability
44
45
46 num_marbles = 10000
47 estimated_pi, num_inside, num_outside, probability = monte_carlo_marbles(num_marbles)
48
49 print(f"Estimasi nilai pi: {estimated_pi}")
50 print(f"Jumlah kelereng dalam lingkaran: {num_inside}")
51 print(f"Jumlah kelereng di luar lingkaran: {num_outside}")
52 print(f"Probabilitas masuk dalam lingkaran: {probability}")
```

Fungsi yang dibuat:

1. generate_marbles(num_marbles, radius)
 - melempar kelereng dengan acak dalam persegi dengan panjang sisi 2 x radius.
2. calculate_distances(x, y)
 - menghitung jarak tiap elemen dari pusat (0,0).
3. count_inside_circle(distances, radius)
 - menghitung kelereng yang ada dalam lingkaran

4. `estimate_pi(num_inside, num_marbles)`
 - menghitung estimasi dari nilai π
5. `plot_simulation()`
 - tampilan grafik simulasi kelereng dengan pyplot
6. `monte_carlo_marbles(num_marbles)`
 - Menjalankan seluruh proses dari simulasi

Setelah di run dalam code python, maka dihasilkan output:

- Kita bandingkan dengan kelereng yang berjumlah 10.000 (kiri) dan 5.000 (kanan)



Maka, dari hasil percobaan yang dilakukan dalam simulasi monte carlo, didapati:

“Semakin banyak jumlah kelereng yang dilibatkan, maka semakin akurat hasil dari π dengan mengarah pada 3,14”