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 *pi*"

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 *pi.*
5. Rumus yang digunakan:
   - Menghitung Jarak Kelereng dari Pusat (Teorema Pythagoras):

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

   - Menentukan Kelereng di Dalam Lingkaran:

$$\boldsymbol{d} \leq \boldsymbol{r}$$

   - Estimasi Nilai π (Pi)

$$\boldsymbol{\pi} \approx \mathbf{4} \times \left(\frac{\boldsymbol{N}_{\mathbf{inside}}}{\boldsymbol{N}_{\mathbf{total}}}\right)$$

   - Probabilitas Kelereng Masuk ke Dalam Lingkaran

$$\boldsymbol{P} = \frac{\boldsymbol{N}_{\mathbf{inside}}}{\boldsymbol{N}_{\mathbf{total}}}$$

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 *pi*
   - 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* (https://github.com/miegoyeng/Model-dan-Simulasi-Data/tree/main/Tugas%202)

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

def generate_marbles(num_marbles, radius):
    return np.random.uniform(-radius, radius, num_marbles), np.random.uniform(-radius, radius, num_marbles)

def calculate_distances(x, y):
    return np.sqrt(x**2 + y**2)

def count_inside_circle(distances, radius):
    return np.sum(distances <= radius)

def estimate_pi(num_inside, num_marbles):
    return (num_inside / num_marbles) * 4, num_inside / num_marbles

def plot_simulation(num_marbles, radius, x, y, inside_circle, estimated_pi, probability):
    fig, ax = plt.subplots(figsize=(6, 6))
    ax.set_xlim(-radius, radius)
    ax.set_ylim(-radius, radius)

    ax.add_patch(plt.Rectangle((-radius, -radius), 2*radius, 2*radius, fill=False, color='blue', linewidth=2))
    ax.add_patch(plt.Circle((0, 0), radius, fill=False, color='red', linewidth=2))

    ax.scatter(x[inside_circle], y[inside_circle], color='green', s=5, label='Inside Circle')
    ax.scatter(x[~inside_circle], y[~inside_circle], color='orange', s=5, label='Outside Circle')

    text_str = f"Inside: {np.sum(inside_circle)}\nOutside: {num_marbles - np.sum(inside_circle)}"
    ax.text(-radius, radius * 0.8, text_str, fontsize=12, color='black', bbox=dict(facecolor='white', alpha=0.7))

    ax.legend()
    ax.set_title(f"Monte Carlo Simulation (N={num_marbles})\nEstimated π: {estimated_pi:.4f}, Probability: {probability:.4f}")
    plt.show()

def monte_carlo_marbles(num_marbles):
    radius = 0.5
    x, y = generate_marbles(num_marbles, radius)
    distances = calculate_distances(x, y)
    num_inside = count_inside_circle(distances, radius)
    estimated_pi, probability = estimate_pi(num_inside, num_marbles)
    inside_circle = distances <= radius

    plot_simulation(num_marbles, radius, x, y, inside_circle, estimated_pi, probability)
    return estimated_pi, num_inside, num_marbles - num_inside, probability


num_marbles = 10000
estimated_pi, num_inside, num_outside, probability = monte_carlo_marbles(num_marbles)

print(f"Estimasi nilai pi: {estimated_pi}")
print(f"Jumlah kelereng dalam lingkaran: {num_inside}")
print(f"Jumlah kelereng di luar lingkaran: {num_outside}")
print(f"Probabilitas masuk dalam lingkaran: {probability}")
```

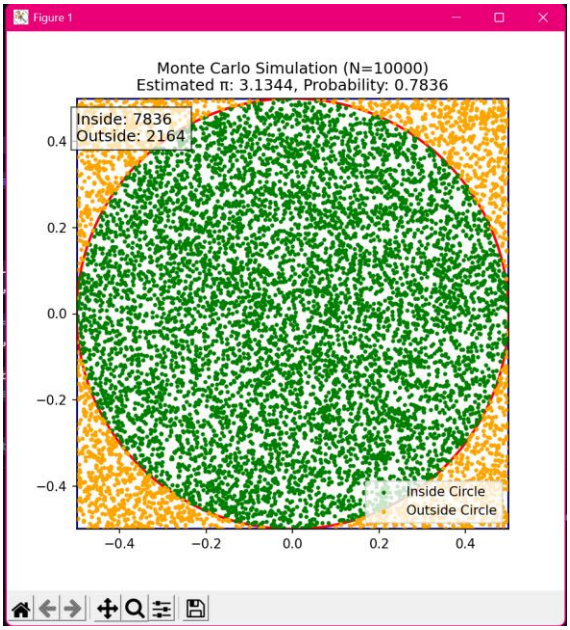
Fungsi yang dibuat:

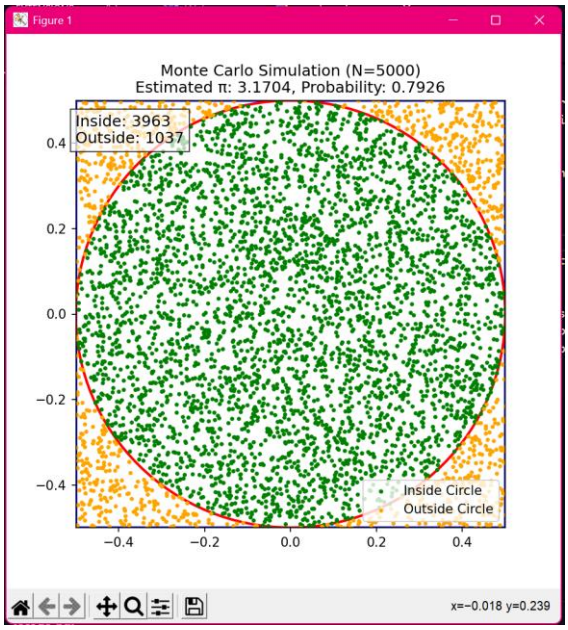
1. generate_marbles(num_marbles, radius)
   - melempar kelereng dengan acak dalam persegi dengan panjang sisi 2 x radius.
2. calculate_distances(x, y)
   - menghitung jarak tiap elemen dari pusat (0,0).
3. count_inside_circle(distances, radius)
   - menghitung kelereng yang ada dalam lingkaran

4. estimate_pi(num_inside, num_marbles)
   - menghitung estimasi dari nilai *pi*\
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 pi dengan mengarah pada 3,14"