Nama	: Muhammad Hisyam Kamil
NIM	: 202210370311060
Mata Kuliah	: Pemodelan dan Simulasi Data
Kelas	: 6B
Source Code	: https://github.com/hisyam99/python-monte-carlo-simulation
Google Colab (IPYNB)	: https://mil.kamil.my.id/data-modelling-task2-hisyam99

LAPORAN SIMULASI MONTE CARLO UNTUK ESTIMASI NILAI π

1. Pendahuluan

Dalam bidang matematika, π (pi) merupakan konstanta fundamental yang dikenal sebagai rasio keliling lingkaran terhadap diameternya, dengan nilai eksak sekitar 3,14159. Sifat irasional dari konstanta tersebut, yang tidak dapat dinyatakan sebagai pecahan sederhana, telah mendorong berbagai upaya penghitungan sejak zaman kuno. Mulai dari metode poligon Archimedes hingga pendekatan komputasi modern, perkembangan teknik estimasi π terus berlangsung hingga kini.

Metode Monte Carlo, yang berbasis pada simulasi probabilitas dan bilangan acak, menawarkan pendekatan berbeda dalam menghitung nilai π. Teknik tersebut pertama kali dikembangkan pada pertengahan abad ke-20 oleh ilmuwan seperti Stanislaw Ulam dan John von Neumann untuk kebutuhan proyek Manhattan. Nama "Monte Carlo" diambil dari kasino di Monako, mengacu pada elemen acak yang menjadi inti metode tersebut. Dalam konteks mata kuliah Pemodelan dan Simulasi Data, pendekatan tersebut menjadi alat penting untuk memahami aplikasi simulasi stokastik dalam menyelesaikan permasalahan matematika secara numerik.

2. Landasan Teori

2.1. Definisi Metode Monte Carlo

Metode Monte Carlo merujuk pada kelompok algoritma komputasi yang memanfaatkan pengacakan untuk memperoleh solusi numerik terhadap permasalahan tertentu. Pendekatan tersebut melibatkan pembuatan sampel acak dalam jumlah besar, diikuti dengan analisis berdasarkan aturan tertentu untuk menghasilkan perkiraan statistik. Prinsip tersebut didasarkan pada hukum bilangan besar, yang menyatakan bahwa peningkatan jumlah sampel akan mengurangi penyimpangan dari nilai ekspektasi sebenarnya.

Secara historis, metode tersebut muncul dari kebutuhan simulasi proses kompleks pada era Perang Dunia II, khususnya dalam pemodelan reaksi nuklir. Kesederhanaan dan fleksibilitasnya menjadikan teknik tersebut relevan untuk berbagai aplikasi, termasuk estimasi nilai π dalam konteks geometri probabilitas.

2.2. Mekanisme Estimasi π dengan Monte Carlo

Penerapan metode Monte Carlo untuk estimasi π didasarkan pada hubungan geometris antara lingkaran dan persegi. Diasumsikan terdapat lingkaran dengan jari-jari 1 yang berada di dalam persegi berpusat di titik (0,0), dengan sisi sepanjang 2 (rentang koordinat dari -1 hingga 1 pada sumbu x dan y). Perhitungan luas dilakukan sebagai berikut:

- Luas lingkaran: $Luas = \pi r^2 = \pi \times 1^2 = \pi$
- Luas persegi: $Luas = 2^2 = 4$
- Rasio luas lingkaran terhadap persegi: $\frac{\pi}{4}$

Ketika titik-titik acak disebarkan secara seragam di dalam persegi, probabilitas sebuah titik berada di dalam lingkaran sama dengan rasio luas tersebut, yaitu $\frac{\pi}{4}$. Dengan menghasilkan N titik acak dan menghitung jumlah titik yang berada di dalam lingkaran (N_{dalam}), rasio tersebut dapat diperkirakan:

$$\frac{N_{dalam}}{N} \approx \frac{\pi}{4}$$

Berdasarkan persamaan tersebut, estimasi nilai π diperoleh melalui:

$$\pi \approx 4 \times \frac{N_{dalam}}{N}$$

Peningkatan jumlah titik (N) akan menghasilkan estimasi yang lebih akurat karena fluktuasi acak menjadi semakin kecil relatif terhadap total sampel.

2.3. Penentuan Posisi Titik dalam Lingkaran

Posisi sebuah titik (x, y) di dalam lingkaran dengan jari-jari 1 ditentukan menggunakan persamaan:

$$x^2 + y^2 \leq 1$$

Jika nilai $x^2 + y^2$ kurang dari atau sama dengan 1, titik tersebut berada di dalam atau pada batas lingkaran. Jika lebih besar dari 1, titik tersebut berada di luar lingkaran. Proses tersebut menjadi dasar penghitungan jumlah titik dalam lingkaran pada simulasi Monte Carlo.

2.4. Kelebihan dan Keterbatasan

Metode Monte Carlo memiliki kelebihan dalam hal kesederhanaan implementasi dan kemampuan menangani permasalahan kompleks yang sulit diselesaikan secara analitis. Namun, sifat probabilitasnya menyebabkan hasil yang tidak selalu konsisten antar simulasi. Selain itu, tingkat akurasi bergantung pada jumlah titik yang digunakan, sehingga peningkatan jumlah sampel dapat memperpanjang waktu komputasi.

3. Metodologi

Simulasi untuk mengestimasi nilai π dengan metode Monte Carlo dilakukan menggunakan Python dengan pustaka NumPy dan Matplotlib. Implementasi tersebut dirancang dengan pendekatan berorientasi objek untuk memastikan struktur yang terorganisir dan efisien. Langkah-langkah yang diterapkan diuraikan sebagai berikut:

1. **Desain Struktur Kode**
   Kelas MonteCarloPi dibuat dengan method-method berikut:
   - __init__: Menginisialisasi jumlah titik.
   - generate_points: Menghasilkan titik acak dan menghitung jumlah titik dalam lingkaran.
   - calculate_pi: Menghitung estimasi π.
   - get_point_coordinates: Memisahkan koordinat untuk keperluan visualisasi.
   - plot_results: Menghasilkan scatter plot.

   Fungsi estimate_pi() digunakan sebagai titik masuk utama simulasi.
2. **Pembuatan Titik Acak**
   Titik-titik dihasilkan menggunakan np.random.uniform(-1, 1, (N, 2)) untuk membentuk N pasang koordinat secara simultan, memanfaatkan efisiensi vektorisasi NumPy.
3. **Penentuan Titik dalam Lingkaran**
   Jarak kuadrat dari pusat dihitung dengan operasi array:

$$distances = np.sum(points^2, axis = 1)$$

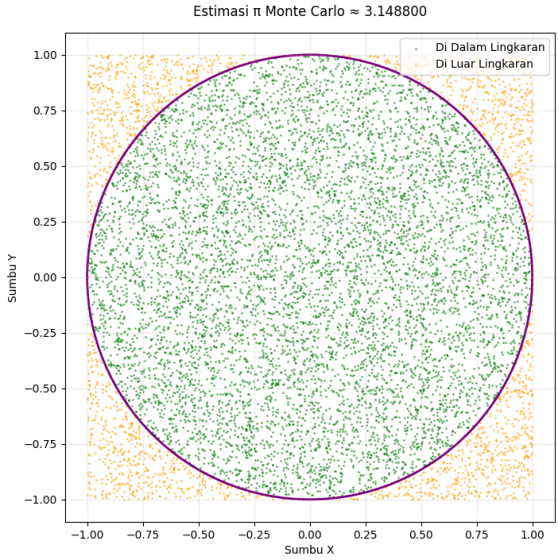
   Jumlah titik dalam lingkaran diperoleh melalui:

$$np.sum(distances <= 1)$$

4. **Perhitungan Estimasi π**
   Estimasi π dihitung dengan rumus:

$$\pi \approx 4 \times \frac{N_{dalam}}{N}$$

5. **Visualisasi Hasil**



Scatter plot dibuat dengan konfigurasi:

- Titik dalam lingkaran: warna hijau, transparansi 0.5.
- Titik luar lingkaran: warna oranye, transparansi 0.5.
- Lingkaran referensi: warna ungu, tebal garis 2.
- Elemen tambahan: grid, label sumbu, legenda, dan judul dengan nilai estimasi π.

6. **Source Code**

```
import numpy as np

import matplotlib.pyplot as plt

from typing import Tuple


class MonteCarloPi:

"""Kelas untuk menghitung estimasi π menggunakan metode Monte Carlo."""


def __init__(self, num_points: int):

"""Inisialisasi dengan jumlah titik simulasi."""

self.num_points = num_points

self.inside_circle = 0

self.points = None


def generate_points(self) -> None:

"""Menghasilkan titik acak dan mengklasifikasikannya."""

# Hasilkan semua titik sekaligus untuk efisiensi lebih baik
```

```
self.points = np.random.uniform(-1, 1, (self.num_points, 2))

# Hitung titik di dalam lingkaran menggunakan operasi vektor

distances = np.sum(self.points ** 2, axis=1)

self.inside_circle = np.sum(distances <= 1)


def calculate_pi(self) -> float:

"""Menghitung estimasi nilai π."""

return 4 * (self.inside_circle / self.num_points)


def get_point_coordinates(self) -> Tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]:

"""Mengembalikan koordinat titik dalam dan luar lingkaran."""

mask = np.sum(self.points ** 2, axis=1) <= 1

inside_points = self.points[mask]

outside_points = self.points[~mask]

return (inside_points[:, 0], inside_points[:, 1],

outside_points[:, 0], outside_points[:, 1])


def plot_results(self, estimated_pi: float) -> None:

"""Membuat visualisasi hasil simulasi."""

fig, ax = plt.subplots(figsize=(8, 8))


# Ambil koordinat untuk plotting

x_in, y_in, x_out, y_out = self.get_point_coordinates()


# Plot titik-titik

ax.scatter(x_in, y_in, c='green', s=1, label='Di Dalam Lingkaran', alpha=0.5)

ax.scatter(x_out, y_out, c='orange', s=1, label='Di Luar Lingkaran', alpha=0.5)


# Tambahkan lingkaran

circle = plt.Circle((0, 0), 1, color='purple', fill=False, lw=2)

ax.add_patch(circle)


# Konfigurasi plot

ax.set_xlim(-1.1, 1.1)

ax.set_ylim(-1.1, 1.1)
```

```
ax.set_aspect('equal', adjustable='box')
ax.set_xlabel('Sumbu X')
ax.set_ylabel('Sumbu Y')
ax.legend(loc='upper right')
ax.set_title(f'Estimasi π Monte Carlo ≈ {estimated_pi:.6f}', pad=15)

# Tambahkan grid
plt.grid(True, linestyle='--', alpha=0.3)
plt.show()

def estimate_pi(num_points: int = 10000) -> float:
"""Fungsi utama untuk menjalankan simulasi Monte Carlo."""
simulator = MonteCarloPi(num_points)
simulator.generate_points()
pi_estimate = simulator.calculate_pi()
simulator.plot_results(pi_estimate)
return pi_estimate

if __name__ == '__main__':
# Jalankan simulasi
estimated_pi = estimate_pi(10000)
print(f'Nilai estimasi π: {estimated_pi:.6f}')
print(f'Nilai aktual π: {np.pi:.6f}')
print(f'Error absolut: {abs(np.pi - estimated_pi):.6f}')
```

4. Hasil dan Pembahasan

4.1. Hasil Simulasi Python

Simulasi menggunakan Python dengan 10.000 titik menghasilkan contoh representatif sebagai berikut:

- Jumlah titik dalam lingkaran: 7.854
- Estimasi π:

$$4 \times \frac{7854}{10000} = 3,1416$$

- Nilai aktual π: 3,14159
- Error absolut:

$$|3,14159 - 3,1416| = 0,00001$$

Visualisasi scatter plot dilengkapi dengan grid dan label sumbu untuk mempermudah interpretasi distribusi titik.

4.2. Analisis Hasil

Hasil simulasi menunjukkan estimasi π yang sangat mendekati nilai aktual, dengan error absolut sebesar 0,00001. Tingkat presisi tersebut dicapai berkat pemrosesan numerik yang konsisten melalui vektorisasi NumPy. Waktu komputasi untuk 10.000 titik berlangsung singkat, menunjukkan efisiensi pendekatan tersebut. Visualisasi yang dihasilkan memberikan gambaran jelas tentang distribusi titik, dengan elemen tambahan seperti grid dan legenda yang mendukung analisis.

Hasil simulasi Monte Carlo dipengaruhi oleh kualitas generator bilangan acak. NumPy menggunakan algoritma Mersenne Twister yang andal, memastikan distribusi titik yang seragam dan konsisten. Pendekatan OOP memungkinkan modularitas, mempermudah penambahan fitur seperti analisis statistik tanpa mengubah struktur utama. Variasi hasil antar simulasi merupakan karakteristik metode stokastik, yang dapat diatasi dengan pengulangan simulasi dan perhitungan rata-rata. Peningkatan jumlah titik hingga 100.000 atau lebih dapat dilakukan dengan mudah, menunjukkan skalabilitas pendekatan tersebut.

5. Kesimpulan dan Saran

Metode Monte Carlo terbukti efektif dalam menghasilkan estimasi nilai π melalui simulasi probabilitas. Implementasi berbasis Python dengan pendekatan berorientasi objek dan vektorisasi menunjukkan efisiensi, kecepatan, dan kemampuan visualisasi yang unggul. Estimasi yang dihasilkan mendekati nilai aktual π (3,14159), dengan error kecil yang dapat diperkecil melalui peningkatan jumlah titik.