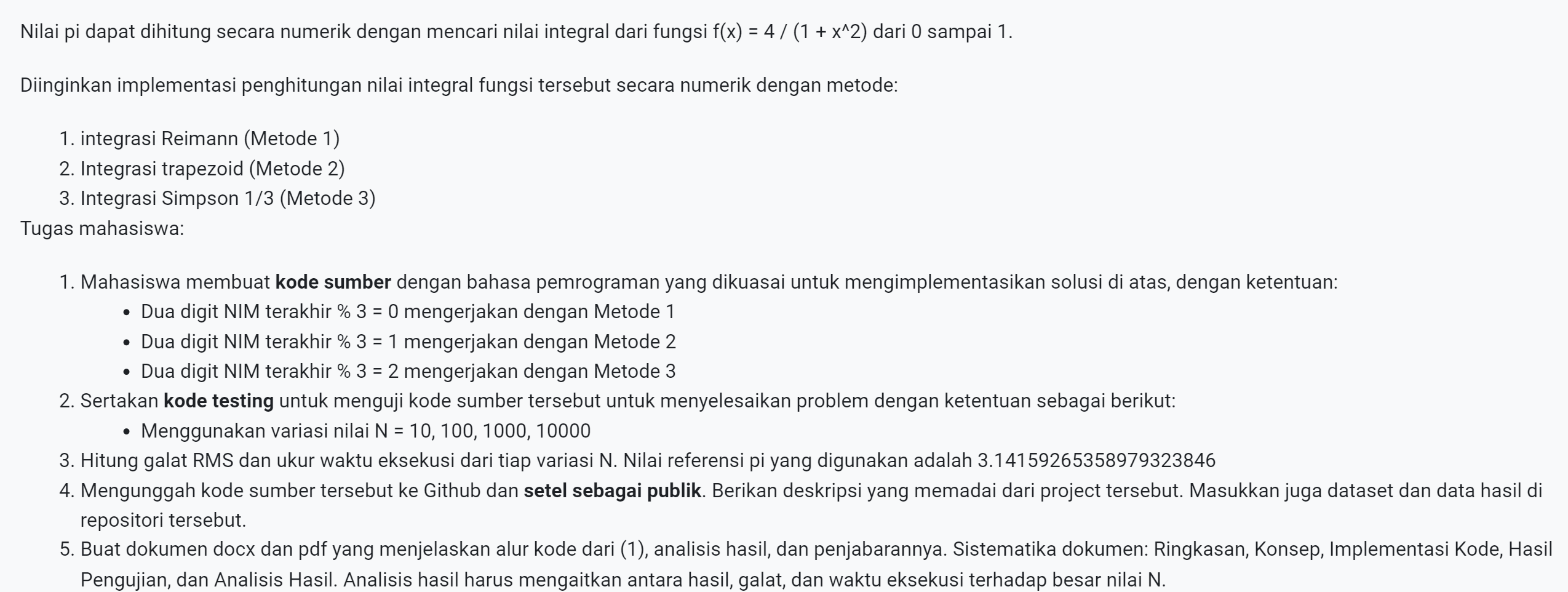
METODE NUMERIK

SOLUSI SISTEM PERSAMAAN LINEAR

ASSIGNMENT – 4

1 Juni 2024

|  |  |  |
| --- | --- | --- |
| Nama | NIM | Kelas |
| Mutiara Sabrina R | 21120122140129 | Metode Numerik - C |

****

**Metode Integrasi Simpson 1/3 (Metode 3)**

Link Github : <https://github.com/Mutiara1626/-Implementasi-Integrasi-Numerik-untuk-Menghitung-Estimasi-nilai-Pi-Mutiara-Sabrina-R-21120122140129>

1. **Ringkasan**

Proyek ini bertujuan untuk menghitung nilai π secara numerik menggunakan metode integrasi Simpson 1/3. Fungsi yang diintegrasikan adalah dengan batas integrasi dari 0 hingga 1. Metode Simpson 1/3 adalah salah satu metode numerik yang digunakan untuk menghitung integral dengan membagi interval integrasi menjadi sejumlah partisi genap. Hasil integral yang dihitung kemudian dibandingkan dengan nilai referensi π, dan galat RMS (Root Mean Square Error) serta waktu eksekusi untuk berbagai nilai partisi N dianalisis. Variasi nilai N yang digunakan dalam pengujian adalah 10, 100, 1000, dan 10000. Selain itu, grafik visualisasi disertakan untuk menunjukkan galat RMS, waktu eksekusi, dan hasil integral terhadap variasi nilai N.

1. **Konsep**

Integral numerik digunakan untuk menghitung nilai integral suatu fungsi ketika integral analitik sulit atau tidak mungkin dilakukan. Salah satu metode integral numerik yang umum digunakan adalah metode Simpson 1/3. Metode Simpson 1/3 adalah teknik integral numerik yang mengaproksimasi integral dari fungsi dengan membagi interval integrasi menjadi sejumlah partisi genap dan menggunakan parabola untuk menghampiri kurva fungsi.

1. **Implementasi Kode**

*Source Code :*

|  |
| --- |
| import numpy as np  import time  import matplotlib.pyplot as plt  # Definisikan fungsi f(x)  def f(x):  return 4 / (1 + x\*\*2)  # Implementasi metode Simpson 1/3  def simpson\_1\_3(a, b, N):  if N % 2 != 0:  raise ValueError("N harus genap.")  h = (b - a) / N  integral = f(a) + f(b)    for i in range(1, N, 2):  integral += 4 \* f(a + i \* h)    for i in range(2, N, 2):  integral += 2 \* f(a + i \* h)    integral \*= h / 3  return integral  # Nilai referensi pi  pi\_ref = 3.14159265358979323846  # Fungsi untuk menguji integrasi dengan berbagai nilai N  def test\_integration(N\_values):  a = 0  b = 1  results = []  for N in N\_values:  start\_time = time.time()  integral = simpson\_1\_3(a, b, N)  end\_time = time.time()    # Hitung galat RMS  error\_rms = np.sqrt((integral - pi\_ref)\*\*2)  exec\_time = end\_time - start\_time  results.append((N, integral, error\_rms, exec\_time))  return results  N\_values = [10, 100, 1000, 10000]  results = test\_integration(N\_values)  for res in results:  print(f"N = {res[0]:5}, Integral = {res[1]:.15f}, RMS Error = {res[2]:.15e}, Execution Time = {res[3]:.6f} seconds")  # Ekstrak data untuk grafik  N\_values, integrals, errors, exec\_times = zip(\*results)  # Plot grafik  plt.figure(figsize=(18, 6))  # Plot galat RMS  plt.subplot(1, 3, 1)  plt.plot(N\_values, errors, marker='o')  plt.xscale('log')  plt.yscale('log')  plt.xlabel('N')  plt.ylabel('RMS Error')  plt.title('RMS Error vs N')  plt.grid(True)  # Plot waktu eksekusi  plt.subplot(1, 3, 2)  plt.plot(N\_values, exec\_times, marker='o')  plt.xscale('log')  plt.xlabel('N')  plt.ylabel('Execution Time (seconds)')  plt.title('Execution Time vs N')  plt.grid(True)  # Plot hasil integral  plt.subplot(1, 3, 3)  plt.plot(N\_values, integrals, marker='o')  plt.xscale('log')  plt.axhline(y=pi\_ref, color='r', linestyle='--', label='Reference Pi')  plt.xlabel('N')  plt.ylabel('Integral Value')  plt.title('Integral Value vs N')  plt.legend()  plt.grid(True)  plt.tight\_layout()  plt.show() |

Penjelasan :

1. import numpy as np : library untuk operasi numerik dan manipulasi data dalam bentuk array.

import matplotlib.pyplot as plt : library untuk membuat visualisasi data dalam bentuk grafik.

1. def f(x):

return 4 / (1 + x\*\*2)

Fungsi yang digunakan untuk menghitung nilai π secara numerik. Fungsi ini dipilih karena integral dari fungsi ini dari 0 hingga 1 adalah π.

1. # Implementasi metode Simpson 1/3

def simpson\_1\_3(a, b, N):

if N % 2 != 0:

raise ValueError("N harus genap.")

h = (b - a) / N

integral = f(a) + f(b)

for i in range(1, N, 2):

integral += 4 \* f(a + i \* h)

for i in range(2, N, 2):

integral += 2 \* f(a + i \* h)

integral \*= h / 3

return integral

Fungsi simpson\_1\_3 menghitung nilai integral dari fungsi 𝑓(𝑥) menggunakan metode Simpson 1/3 pada interval [𝑎,𝑏] dengan N partisi. Pertama, fungsi memastikan bahwa N adalah genap, karena metode ini memerlukan jumlah partisi yang genap. Kemudian, lebar setiap partisi h dihitung dengan membagi panjang interval dengan N. Nilai awal integral diinisialisasi dengan menjumlahkan nilai fungsi pada titik awal dan titik akhir interval. Selanjutnya, dua loop digunakan untuk menambahkan nilai 4×𝑓(𝑥) pada titik-titik ganjil dan 2×𝑓(𝑥) pada titik-titik genap dalam partisi. Setelah semua penjumlahan selesai, hasil integral dikalikan dengan sesuai dengan aturan Simpson 1/3 dan kemudian nilai integral dikembalikan sebagai output dari fungsi.

1. # Nilai referensi pi

pi\_ref = 3.14159265358979323846

Dalam kode ini, menetapkan nilai referensi π yang sangat presisi dengan banyak digit desimal untuk digunakan dalam perhitungan dan analisis..

1. # Fungsi untuk menguji integrasi dengan berbagai nilai N

def test\_integration(N\_values):

a = 0

b = 1

results = []

for N in N\_values:

start\_time = time.time()

integral = simpson\_1\_3(a, b, N)

end\_time = time.time()

# Hitung galat RMS

error\_rms = np.sqrt((integral - pi\_ref)\*\*2)

exec\_time = end\_time - start\_time

results.append((N, integral, error\_rms, exec\_time))

return results

Fungsi test\_integration menguji metode integrasi Simpson 1/3 dengan berbagai nilai partisi N yang diberikan dalam N\_values. Batas integrasi ditetapkan dari 0 hingga 1. List results diinisialisasi untuk menyimpan hasil pengujian. Untuk setiap nilai N dalam N\_values, waktu mulai dicatat, dan nilai integral dihitung menggunakan metode Simpson 1/3. Waktu selesai kemudian dicatat untuk menghitung waktu eksekusi. Galat RMS antara nilai integral yang dihitung dan nilai referensi π dihitung. Hasil pengujian, termasuk nilai N, hasil integral, galat RMS, dan waktu eksekusi, ditambahkan ke dalam results. Akhirnya, fungsi mengembalikan list results yang berisi hasil pengujian untuk berbagai nilai N.

1. N\_values = [10, 100, 1000, 10000]

results = test\_integration(N\_values)

for res in results:

print(f"N = {res[0]:5}, Integral = {res[1]:.15f}, RMS Error = {res[2]:.15e}, Execution Time = {res[3]:.6f} seconds")

Pertama, list N\_values didefinisikan dengan berbagai nilai partisi N yang akan digunakan untuk menguji metode integrasi Simpson 1/3, yaitu 10, 100, 1000, dan 10000. Fungsi test\_integration kemudian dipanggil dengan N\_values sebagai argumen, dan hasil pengujiannya disimpan dalam list results. Loop for digunakan untuk memproses setiap hasil dalam results. Untuk setiap hasil, nilai N, nilai integral yang dihitung, galat RMS, dan waktu eksekusi dicetak dengan format yang teratur. Output menunjukkan kinerja metode Simpson 1/3 dalam menghitung nilai integral dengan berbagai nilai N.

1. # Ekstrak data untuk grafik

N\_values, integrals, errors, exec\_times = zip(\*results)

Dalam kode ini, digunakan untuk mengekstrak data dari list results ke dalam empat variabel terpisah yang akan digunakan untuk membuat grafik. Fungsi zip(\*results) mengelompokkan elemen-elemen dari setiap tuple dalam results menjadi beberapa tuple baru berdasarkan posisi elemen. Kemudian, masing-masing hasil dari zip tersebut dipisah ke dalam variabel N\_values, integrals, errors, dan exec\_times. N\_values akan berisi nilai N, integrals berisi hasil integral yang dihitung, errors berisi galat RMS, dan exec\_times berisi waktu eksekusi untuk setiap nilai N. Ini memungkinkan data untuk diproses dan divisualisasikan secara lebih mudah dalam grafik..

1. # Plot grafik

plt.figure(figsize=(18, 6))

# Plot galat RMS

plt.subplot(1, 3, 1)

plt.plot(N\_values, errors, marker='o')

plt.xscale('log')

plt.yscale('log')

plt.xlabel('N')

plt.ylabel('RMS Error')

plt.title('RMS Error vs N')

plt.grid(True)

# Plot waktu eksekusi

plt.subplot(1, 3, 2)

plt.plot(N\_values, exec\_times, marker='o')

plt.xscale('log')

plt.xlabel('N')

plt.ylabel('Execution Time (seconds)')

plt.title('Execution Time vs N')

plt.grid(True)

# Plot hasil integral

plt.subplot(1, 3, 3)

plt.plot(N\_values, integrals, marker='o')

plt.xscale('log')

plt.axhline(y=pi\_ref, color='r', linestyle='--', label='Reference Pi')

plt.xlabel('N')

plt.ylabel('Integral Value')

plt.title('Integral Value vs N')

plt.legend()

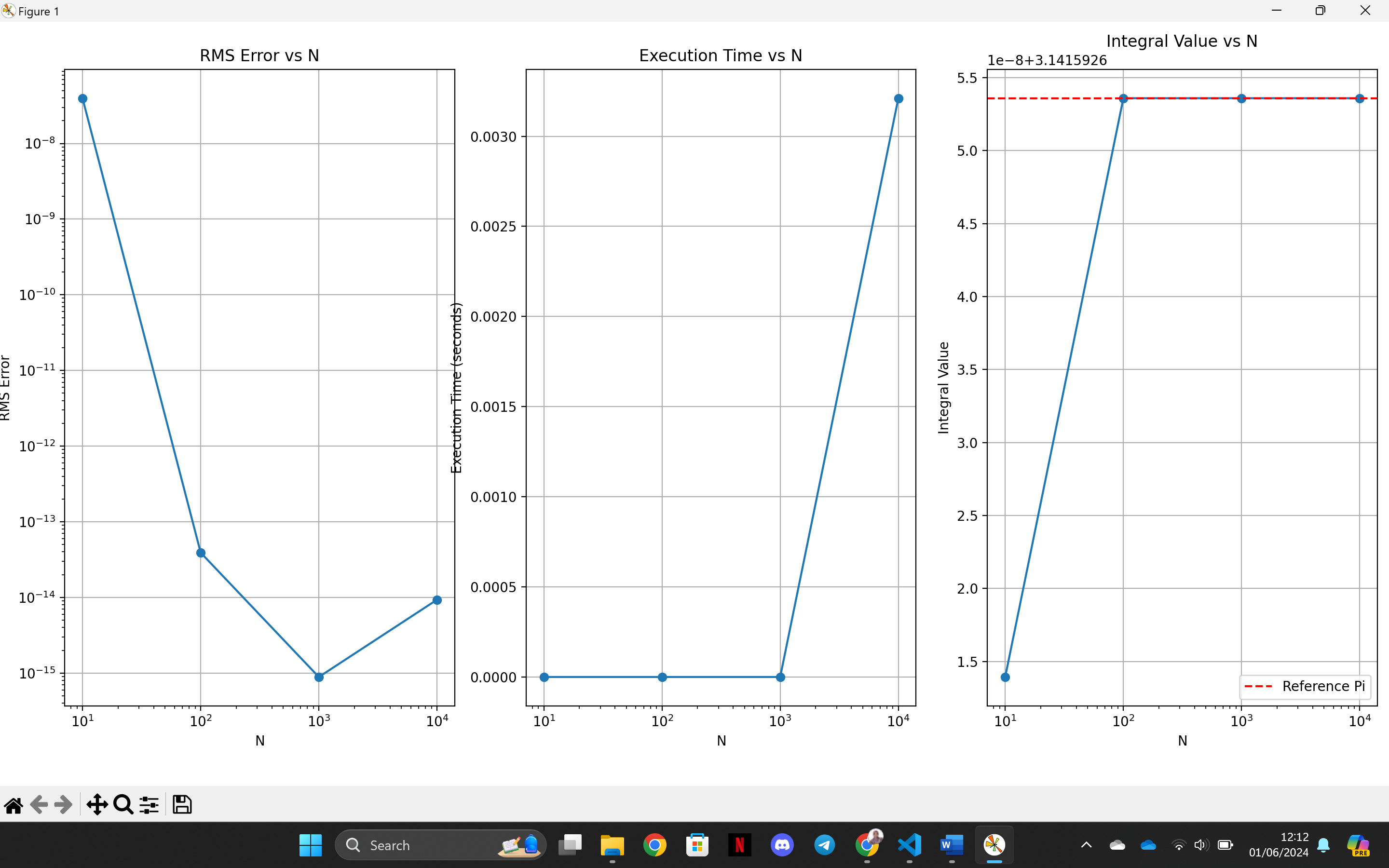
plt.grid(True)

plt.tight\_layout()

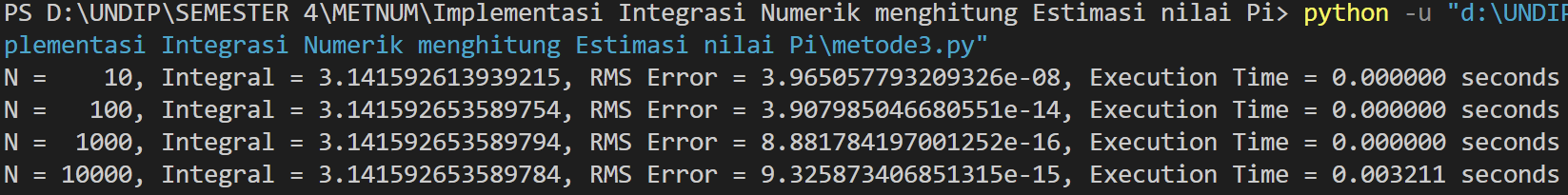
plt.show().

Untuk menampilkan galat RMS, waktu eksekusi, dan hasil integral terhadap berbagai nilai N dalam bentuk grafik

1. **Hasil Pengujian**



Gambar 1 1 Grafik hasil pengujian



Gambar 1 2 Hasil pengujian ketika di run

1. **Analisis Hasil**
2. RMS Error vs N

Grafik pertama menunjukkan bahwa galat RMS menurun secara signifikan saat jumlah partisi N meningkat. Penurunan ini mengikuti pola eksponensial pada skala log-log, menunjukkan bahwa metode Simpson 1/3 semakin akurat dengan semakin banyaknya partisi.

1. Execution Time vs N

Grafik kedua menunjukkan waktu eksekusi yang meningkat seiring dengan bertambahnya jumlah partisi N. Pada skala logaritmik, terlihat bahwa peningkatan N meningkatkan kompleksitas komputasi dan waktu yang dibutuhkan, yang diharapkan karena lebih banyak partisi memerlukan lebih banyak perhitungan.

1. Integral Value vs N

Grafik ketiga menunjukkan bahwa nilai integral mendekati nilai referensi π saat N meningkat. Nilai integral konvergen ke nilai yang benar dengan peningkatan N, sementara dengan nilai N kecil, hasil integral dapat menyimpang dari nilai referensi.

1. **Kesimpulan**

Metode Simpson 1/3 terbukti efektif dalam menghitung nilai integral secara numerik dengan peningkatan akurasi yang sebanding dengan peningkatan jumlah partisi. Namun, perlu mempertimbangkan peningkatan waktu komputasi saat memilih nilai N yang optimal.