METODE NUMERIK
SOLUSI SISTEM PERSAMAAN LINEAR
ASSIGNMENT – 4

1 Juni 2024

Nama	NIM	Kelas
Mutiara Sabrina R	21120122140129	Metode Numerik - C

Nilai pi dapat dihitung secara numerik dengan mencari nilai integral dari fungsi f(x) = 4 / (1 + x^2) dari 0 sampai 1.

Diinginkan implementasi penghitungan nilai integral fungsi tersebut secara numerik dengan metode:

1. integrasi Reimann (Metode 1)
2. Integrasi trapezoid (Metode 2)
3. Integrasi Simpson 1/3 (Metode 3)

Tugas mahasiswa:

1. Mahasiswa membuat **kode sumber** dengan bahasa pemrograman yang dikuasai untuk mengimplementasikan solusi di atas, dengan ketentuan:
   - Dua digit NIM terakhir % 3 = 0 mengerjakan dengan Metode 1
   - Dua digit NIM terakhir % 3 = 1 mengerjakan dengan Metode 2
   - Dua digit NIM terakhir % 3 = 2 mengerjakan dengan Metode 3
2. Sertakan **kode testing** untuk menguji kode sumber tersebut untuk menyelesaikan problem dengan ketentuan sebagai berikut:
   - Menggunakan variasi nilai N = 10, 100, 1000, 10000
3. Hitung galat RMS dan ukur waktu eksekusi dari tiap variasi N. Nilai referensi pi yang digunakan adalah 3.14159265358979323846
4. Mengunggah kode sumber tersebut ke Github dan **setel sebagai publik**. Berikan deskripsi yang memadai dari project tersebut. Masukkan juga dataset dan data hasil di repositori tersebut.
5. Buat dokumen docx dan pdf yang menjelaskan alur kode dari (1), analisis hasil, dan penjabarannya. Sistematika dokumen: Ringkasan, Konsep, Implementasi Kode, Hasil Pengujian, dan Analisis Hasil. Analisis hasil harus mengaitkan antara hasil, galat, dan waktu eksekusi terhadap besar nilai N.

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

A. Ringkasan

Proyek ini bertujuan untuk menghitung nilai π secara numerik menggunakan metode integrasi Simpson 1/3. Fungsi yang diintegrasikan adalah $f(x) = \frac{4}{1+x^2}$ 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.

B. 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.

C. 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.

2. ```
   def f(x):
       return 4 / (1 + x**2)
   ```

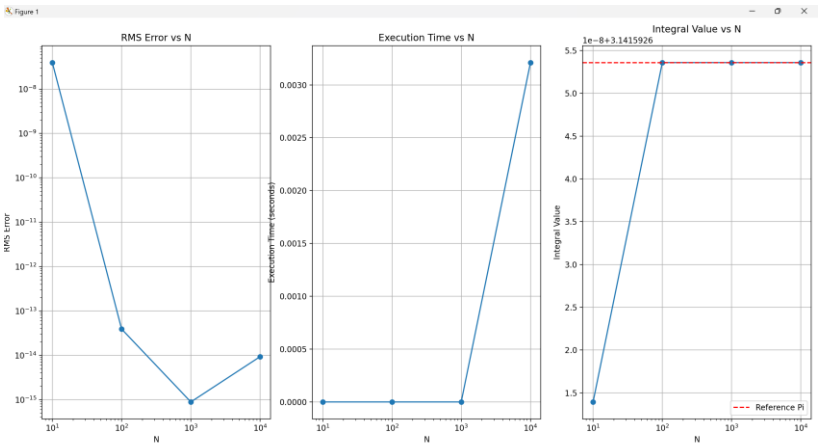
   Fungsi yang digunakan untuk menghitung nilai π secara numerik. Fungsi ini dipilih karena integral dari fungsi ini dari 0 hingga 1 adalah π.
   ```

3.
```
# 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 $4\times f(x)$ pada titik-titik ganjil dan $2\times f(x)$ pada titik-titik genap dalam partisi. Setelah semua penjumlahan selesai, hasil integral dikalikan dengan $\frac{h}{3}$ sesuai dengan aturan Simpson 1/3 dan kemudian nilai integral dikembalikan sebagai output dari fungsi.

4.
```
# 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..

5.
```
# 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.

6. ```
   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.

7. ```
   # 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..

8. ```
   # 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

D. Hasil Pengujian



Gambar 1 1 Grafik hasil pengujian

```
PS D:\UNDIP\SEMESTER 4\METNUM\Implementasi Integrasi Numerik menghitung Estimasi nilai Pi> python -u "d:\UNDI
plementasi Integrasi Numerik menghitung Estimasi nilai Pi\metode3.py"
N =    10, Integral = 3.141592613939215, RMS Error = 3.965057793209326e-08, Execution Time = 0.000000 seconds
N =   100, Integral = 3.141592653589754, RMS Error = 3.907985046680551e-14, Execution Time = 0.000000 seconds
N =  1000, Integral = 3.141592653589794, RMS Error = 8.881784197001252e-16, Execution Time = 0.000000 seconds
N = 10000, Integral = 3.141592653589784, RMS Error = 9.325873406851315e-15, Execution Time = 0.003211 seconds
```

Gambar 1 2 Hasil pengujian ketika di run

E. Analisis Hasil

a. 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.

b. 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.

c. 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.

F. 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.