Nama : Nanda Zayyan Nurirfan
NIM : 21120122130079
Metode Numerik B

Implementasi Integrasi Numerik untuk Menghitung Estimasi Nilai Pi

1. Ringkasan

   Proyek ini bertujuan untuk menghitung nilai π (pi) secara numerik dengan mengintegrasikan fungsi $f(x)=41+x2f(x)=1+x24$ dari 0 sampai 1 menggunakan metode integrasi Riemann. Proses dilakukan dengan berbagai nilai N untuk melihat pengaruh jumlah segmen terhadap akurasi hasil dan waktu eksekusi.

2. Konsep

   Metode integrasi Riemann menghitung integral dengan membagi area di bawah kurva menjadi segmen-segmen kecil berbentuk persegi panjang dan menjumlahkan luasnya. Semakin banyak segmen (N) yang digunakan, semakin kecil lebar masing-masing segmen dan semakin akurat hasilnya.

3. Implementasi Kode

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

def riemann_integral(f, a, b, N):
    width = (b - a) / N
    total_area = 0
    for i in range(N):
        total_area += f(a + i * width) * width
    return total_area

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

# Nilai referensi pi
pi_reference = 3.14159265358979323846

# Variasi nilai N
N_values = [10, 100, 1000, 10000]

# Hasil perhitungan, galat RMS, dan waktu eksekusi
results = []
errors = []
execution_times = []
```

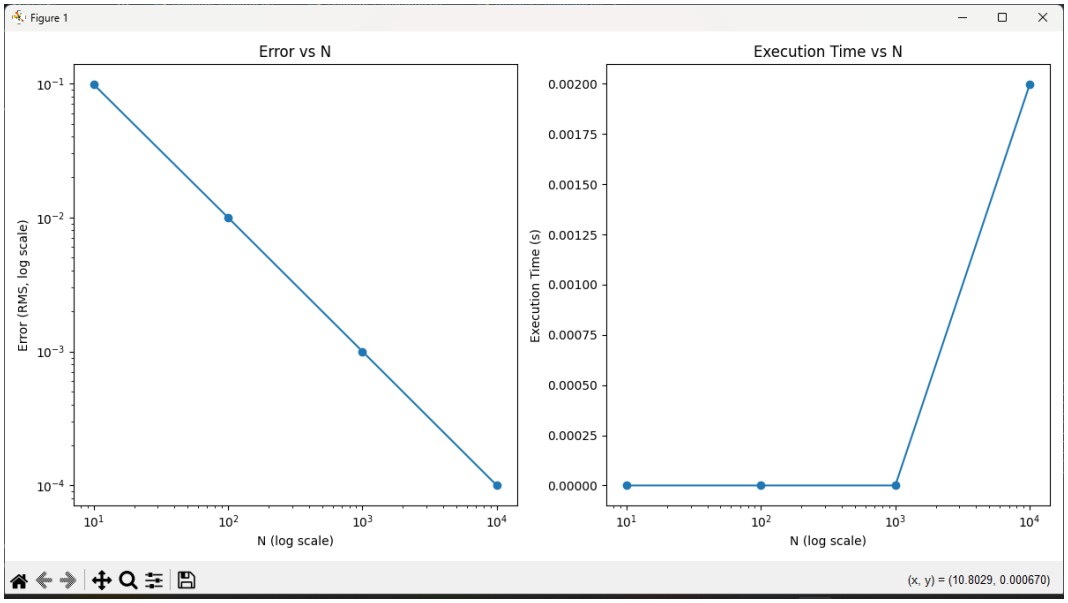
```
for N in N_values:
    start_time = time.time()
    pi_estimate = riemann_integral(f, 0, 1, N)
    end_time = time.time()

    execution_time = end_time - start_time
    error = np.sqrt((pi_estimate - pi_reference)**2)

    results.append(pi_estimate)
    errors.append(error)
    execution_times.append(execution_time)

# Output hasil
print("N\t\tπ estimate\t\tError (RMS)\t\tExecution Time (s)")
for i, N in enumerate(N_values):

print(f"{N}\t\t{results[i]:.15f}\t{errors[i]:.15f}\t{execution_tim
es[i]:.8f}")

# Plotting
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(N_values, errors, marker='o')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('N (log scale)')
plt.ylabel('Error (RMS, log scale)')
plt.title('Error vs N')

plt.subplot(1, 2, 2)
plt.plot(N_values, execution_times, marker='o')
plt.xscale('log')
plt.xlabel('N (log scale)')
plt.ylabel('Execution Time (s)')
plt.title('Execution Time vs N')

plt.tight_layout()
plt.show()
```

4. Hasil Pengujian

a. Hasil running pada terminal

N	π estimate	Error (RMS)	Execution Time (s)
10	3.239925988907159	0.098333335317366	0.00000000
100	3.151575986923127	0.009983333333334	0.00000000
1000	3.142592486923122	0.000999833333329	0.00000000
10000	3.141692651923117	0.000099998333324	0.00199747

b. Grafik hasil running



5. Alur Source Code

Alur kode dimulai dengan mendefinisikan fungsi `riemann_integral`, yang bertanggung jawab untuk menghitung integral menggunakan metode Riemann. Fungsi ini menerima parameter fungsi `f`, batas bawah `a`, batas atas `b`, dan jumlah segmen `N`. Lebar setiap segmen dihitung sebagai `(b - a) / N`, dan variabel `total_area` diinisialisasi dengan nilai 0. Sebuah loop kemudian berjalan dari 0 hingga N-1, di mana pada setiap iterasi, nilai fungsi `f` pada titik tengah segmen dikalikan dengan lebar segmen dan ditambahkan ke `total_area`. Fungsi `riemann_integral` mengembalikan `total_area` sebagai hasil integral.

Selanjutnya, fungsi `f` didefinisikan untuk merepresentasikan fungsi matematika yang akan diintegrasikan, yaitu \(f(x) = \frac{4}{1 + x^2} \). Variabel `pi_reference` menyimpan nilai referensi π untuk perbandingan hasil.

Kemudian, daftar `N_values` berisi variasi nilai N yang akan digunakan untuk pengujian, yaitu 10, 100, 1000, dan 10000.

Program kemudian masuk ke loop yang iterasi melalui setiap nilai N. Pada setiap iterasi, waktu mulai eksekusi dicatat menggunakan `time.time()`. Estimasi π dihitung dengan memanggil fungsi `riemann_integral` dengan parameter fungsi `f`, batas bawah 0, batas atas 1, dan nilai N. Setelah integral dihitung, waktu akhir eksekusi dicatat dan durasi eksekusi dihitung. Galat RMS dihitung sebagai akar kuadrat dari kuadrat selisih antara estimasi π dan nilai referensi π. Hasil estimasi, galat, dan waktu eksekusi disimpan dalam daftar `results`, `errors`, dan `execution_times` masing-masing.

Setelah loop selesai, hasil untuk setiap N ditampilkan dalam format tabel yang menunjukkan nilai N, estimasi π, galat RMS, dan waktu eksekusi. Kode juga membuat dua plot: satu untuk galat RMS terhadap N dalam skala logaritmik dan satu lagi untuk waktu eksekusi terhadap N dalam skala logaritmik. Kedua plot memberikan visualisasi dari hubungan antara jumlah segmen, galat, dan waktu eksekusi.

Dengan alur ini, kode secara sistematis menghitung nilai π menggunakan metode Riemann untuk berbagai nilai N, mengukur akurasi hasil dengan galat RMS, dan mengevaluasi efisiensi komputasi melalui waktu eksekusi, menyediakan wawasan tentang trade-off antara akurasi dan efisiensi dalam metode numerik ini.

6. Analisis Hasil
    - Error (RMS): Semakin besar nilai N, semakin kecil galat RMS, menunjukkan bahwa hasil estimasi semakin mendekati nilai referensi π.
    - Waktu Eksekusi: Waktu eksekusi meningkat seiring dengan bertambahnya N, sesuai dengan kompleksitas komputasi yang meningkat.
    - Plot: Plot menunjukkan bahwa galat berkurang secara eksponensial dengan peningkatan N, dan waktu eksekusi meningkat hampir secara linier dalam skala log.

7. Kesimpulan

Metode integrasi Riemann efektif untuk menghitung nilai π dengan akurasi yang meningkat seiring bertambahnya jumlah segmen N. Namun, peningkatan akurasi ini disertai dengan peningkatan waktu komputasi, sehingga diperlukan keseimbangan antara akurasi dan efisiensi.