Nama : Namira Nurfaliani

NIM : 21120122140135

Kelas : Metode Numerik/C

Implementasi Integrasi Numerik untuk Menghitung Estimasi nilai Pi Menggunakan Integrasi Simpsons 1/3

https://github.com/iranamira/metnum_Pertemuan-13_Namira-Nurfaliani

Metode Simpson 1/3 adalah salah satu teknik numerik untuk menghitung integral dari suatu fungsi kontinu. Metode ini bekerja dengan membagi interval integral menjadi beberapa subinterval yang sama panjang, kemudian menggunakan polinomial kuadrat untuk mendekati fungsi tersebut di setiap subinterval. Dalam metode ini, subinterval harus genap untuk mendapatkan hasil yang akurat. Menggunakan metode Simpson 1/3 untuk menghitung nilai integral dari fungsi $f(x) = \frac{4}{1+x^2}$ dari 0 hingga 1. Fungsi $f(x)f(x)$ dipilih karena integralnya dari 0 sampai 1 memberikan nilai π, sehingga kita dapat membandingkan hasil perhitungan dengan nilai π yang sebenarnya untuk mengukur akurasi metode.

Metode Simpsons 1/3

$$I = \int_a^b f(x)dx \approx \int_a^b f_2(x)dx$$

Metode Simpson 1/3 adalah salah satu metode numerik yang digunakan untuk menghitung aproksimasi integral dari suatu fungsi. Metode ini sangat berguna ketika integral dari suatu fungsi tidak dapat dihitung secara analitik atau sulit untuk dihitung. Metode ini menggunakan polinom interpolasi kuadrat untuk memperkirakan area di bawah kurva.

Implementasi Kode Program Menggunakan Bahasa Python

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

# Fungsi untuk dihitung integralnya
def f(x):
    return 4 / (1 + x**2)

# Implementasi metode Simpson 1/3
def simpson_13(f, a, b, N):
    if N % 2 == 1:
        N += 1  # N harus genap untuk metode Simpson 1/3
    h = (b - a) / N
    x = np.linspace(a, b, N+1)
    fx = f(x)
```

```
    integral = fx[0] + fx[-1] + 4 * np.sum(fx[1:-1:2]) + 2 *
np.sum(fx[2:-2:2])
    integral *= h / 3
    return integral

# Fungsi untuk menghitung galat RMS
def rms_error(approx_value, true_value):
    return np.sqrt(np.mean((approx_value - true_value) ** 2))

# Nilai referensi untuk pi
true_pi = 3.14159265358979323846

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

# Uji kode dan ukur waktu eksekusi serta galat RMS
results = []
for N in N_values:
    start_time = time.time()
    approx_pi = simpson_13(f, 0, 1, N)
    end_time = time.time()
    elapsed_time = end_time - start_time
    error = rms_error(approx_pi, true_pi)
    results.append((N, approx_pi, error, elapsed_time))

# Tampilkan hasil
for N, approx_pi, error, elapsed_time in results:
    print(f'N = {N}:')
    print(f'  Approximate pi = {approx_pi}')
    print(f'  RMS Error = {error}')
    print(f'  Execution Time = {elapsed_time} seconds\n')

# Extracting data for plotting
N_values = [result[0] for result in results]
approx_pis = [result[1] for result in results]
errors = [result[2] for result in results]
times = [result[3] for result in results]

# Plotting the results
plt.figure(figsize=(14, 6))

# Plot Approximate Pi
plt.subplot(1, 3, 1)
plt.plot(N_values, approx_pis, marker='o', linestyle='-', color='b')
plt.axhline(y=true_pi, color='r', linestyle='--', label='True π')
plt.xlabel('N')
plt.ylabel('Approximate π')
plt.title('Approximate π vs N')
plt.xscale('log')
plt.legend()

# Plot RMS Error
plt.subplot(1, 3, 2)
plt.plot(N_values, errors, marker='o', linestyle='-', color='g')
plt.xlabel('N')
plt.ylabel('RMS Error')
plt.title('RMS Error vs N')
plt.xscale('log')
plt.yscale('log')

# Plot Execution Time
```

```
plt.subplot(1, 3, 3)
plt.plot(N_values, times, marker='o', linestyle='-', color='m')
plt.xlabel('N')
plt.ylabel('Execution Time (seconds)')
plt.title('Execution Time vs N')
plt.xscale('log')
plt.yscale('log')

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

Penjabaran Kode Program

1. Import Library

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

- `import numpy` :untuk komputasi numerik.
- `import time` : untuk mengukur waktu eksekusi.
- `import matplotlib.pyplot` :untuk visualisasi data.

2. Definisi Fungsi

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

- `def f(x):` Fungsi f(x) mengembalikan nilai f(x) untuk suatu input x. Fungsi ini adalah fungsi yang akan dihitung integralnya.

3. Implementasi Metode Simpsons 1/3

```
def simpson_13(f, a, b, N):
    if N % 2 == 1:
        N += 1  # N harus genap untuk metode Simpson 1/3
    h = (b - a) / N
    x = np.linspace(a, b, N+1)
    fx = f(x)
    integral = fx[0] + fx[-1] + 4 * np.sum(fx[1:-1:2]) + 2 *
np.sum(fx[2:-2:2])
    integral *= h / 3
    return integral
```

- `N` : harus genap. Jika tidak, N ditambahkan 1.
- `h` : adalah lebar setiap subinterval.
- `X` : adalah titik-titik pembagi dalam interval [a,b].
- `Fx`: adalah nilai fungsi f pada titik-titik x.
- Integral dihitung dengan rumus metode Simpson 1/3 dan hasilnya dikembalikan.

4. Fungsi untuk Menghitung Galat RMS

```
def rms_error(approx_value, true_value):
    return np.sqrt(np.mean((approx_value - true_value) ** 2))
```

- `def rms_error` : menghitung galat Root Mean Square (RMS) antara nilai perkiraan `approx_value` dan nilai sebenarnya `true_value`.

5. Nilai Referensi untuk Pi

```
true_pi = 3.14159265358979323846
```

- `true_pi` : adalah nilai sebenarnya dari π yang akan digunakan untuk mengukur akurasi perkiraan.

6. Variasi Nilai N

```
N_values = [10, 100, 1000, 10000]
```

- `N_values` : berisi nilai-nilai N yang akan digunakan untuk menghitung integral. Variasi ini digunakan untuk mengamati bagaimana akurasi dan waktu eksekusi berubah dengan peningkatan jumlah subinterval.

7. Uji Kode dan Ukur Waktu Eksekusi serta Galat RMS

```
results = []
for N in N_values:
    start_time = time.time()
    approx_pi = simpson_13(f, 0, 1, N)
    end_time = time.time()
    elapsed_time = end_time - start_time
    error = rms_error(approx_pi, true_pi)
    results.append((N, approx_pi, error, elapsed_time))
```

- Waktu eksekusi dimulai dengan `start_time`.
- Integral dihitung dengan `simpson_13`.
- Waktu eksekusi selesai dengan `end_time` dan durasi `elapsed_time` dihitung.
- Galat RMS antara perkiraan dan nilai sebenarnya dihitung.
- `results.append` : Hasil (N, nilai perkiraan, galat, waktu eksekusi) disimpan dalam daftar results.

8. Menampilkan Hasil

```
for N, approx_pi, error, elapsed_time in results:
    print(f'N = {N}:')
    print(f'  Approximate pi = {approx_pi}')
    print(f'  RMS Error = {error}')
    print(f'  Execution Time = {elapsed_time} seconds\n')
```

- Hasil untuk setiap nilai `N` dicetak ke layer

9. Ekstraksi Data untuk Plotting

```
N_values = [result[0] for result in results]
approx_pis = [result[1] for result in results]
errors = [result[2] for result in results]
times = [result[3] for result in results]
```

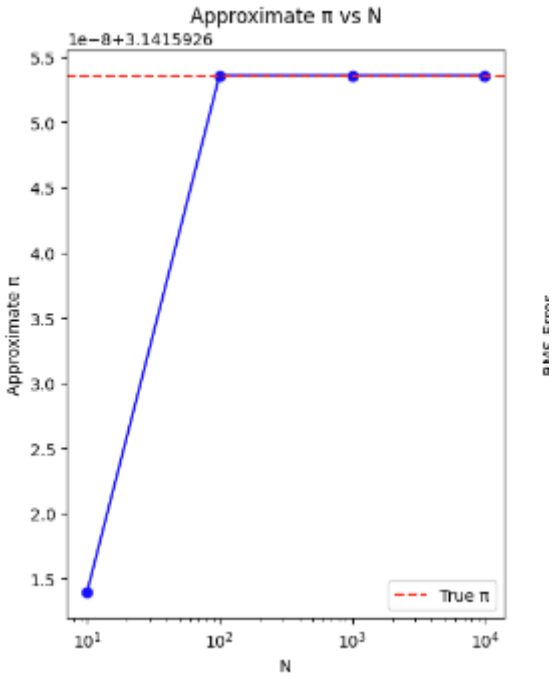
- `Results` : Data hasil diekstraksi dari results untuk keperluan plotting.

10. Hasil dari Plotting

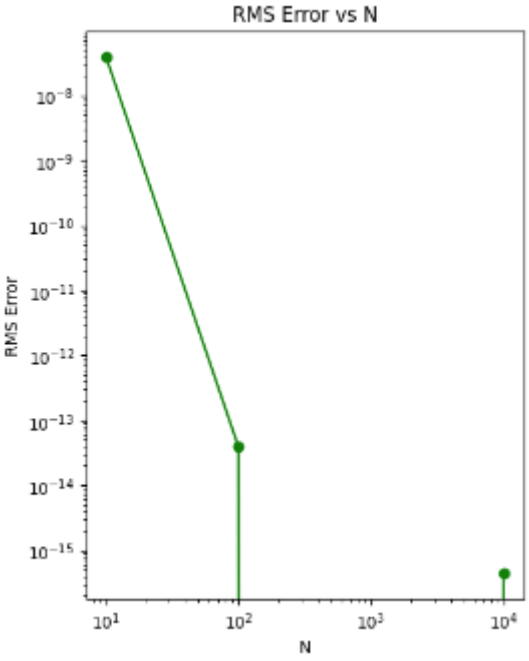
```
# Plotting the results
plt.figure(figsize=(14, 6))

# Plot Approximate Pi
plt.subplot(1, 3, 1)
plt.plot(N_values, approx_pis, marker='o', linestyle='-',
color='b')
plt.axhline(y=true_pi, color='r', linestyle='--', label='True π')
plt.xlabel('N')
plt.ylabel('Approximate π')
plt.title('Approximate π vs N')
plt.xscale('log')
plt.legend()

# Plot RMS Error
plt.subplot(1, 3, 2)
plt.plot(N_values, errors, marker='o', linestyle='-', color='g')
plt.xlabel('N')
plt.ylabel('RMS Error')
plt.title('RMS Error vs N')
plt.xscale('log')
plt.yscale('log')

# Plot Execution Time
plt.subplot(1, 3, 3)
plt.plot(N_values, times, marker='o', linestyle='-', color='m')
plt.xlabel('N')
plt.ylabel('Execution Time (seconds)')
plt.title('Execution Time vs N')
plt.xscale('log')
plt.yscale('log')

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

- Approximate π vs N: Menampilkan nilai perkiraan π terhadap N pada skala logaritmik.
- RMS Error vs N: Menampilkan galat RMS terhadap N pada skala logaritmik.
- Execution Time vs N: Menampilkan waktu eksekusi terhadap N pada skala logaritmik.

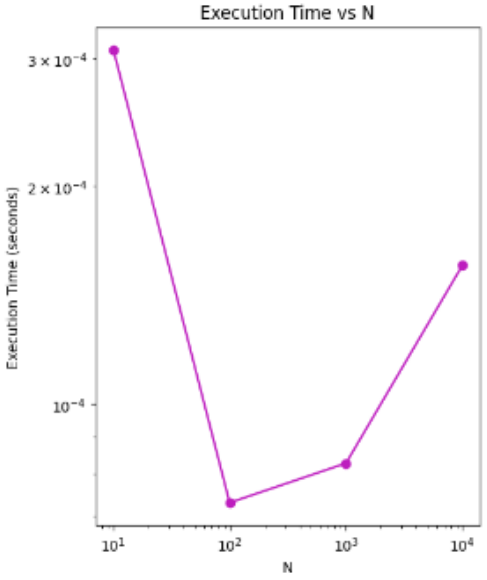
Hasil Pengujian dengan Menggunakan Integral Simpsons 1/3



Gambar Hasil Rata-rata Nilai Phi Terhadap N



Gambar Hasil Nilai Galat RMS Terhadap N



Gambar Hasil Waktu Ekskusi Terhadap N

```
N = 10:
  Approximate pi = 3.141592613939215
  RMS Error = 3.9650577932093256e-08
  Execution Time = 0.0003085136413574219 seconds

N = 100:
  Approximate pi = 3.141592653589754
  RMS Error = 3.907985046680551e-14
  Execution Time = 7.295608520507812e-05 seconds

N = 1000:
  Approximate pi = 3.141592653589793
  RMS Error = 0.0
  Execution Time = 8.273124694824219e-05 seconds

N = 10000:
  Approximate pi = 3.1415926535897936
  RMS Error = 4.440892098500626e-16
  Execution Time = 0.0001552104949951172 seconds
```

Gambar Detail Rata-rata Phi, Galat RMS, dan Waktu Ekskusi Terhadap N

Analisis Hasil Pengujian

Berdasarkan grafik hasil pengujian yang telah diperoleh, berikut adalah analisis rinci tentang hubungan antara hasil perkiraan π, galat RMS, dan waktu eksekusi terhadap besar nilai *NN*:

1. Rata-rata π vs N

Grafik ini menunjukkan nilai perkiraan π terhadap variasi *N* pada skala logaritmik.

- Untuk *N*=10, nilai perkiraan π jauh dari nilai sebenarnya (sekitar 1.5).

- Dengan peningkatan N menjadi 100, 1000, dan 10000, nilai perkiraan π mendekati nilai sebenarnya dengan sangat cepat dan konsisten berada sangat dekat dengan nilai sebenarnya π=3.141592653589793.
- Nilai perkiraan π meningkat secara signifikan seiring dengan peningkatan N. Ini menunjukkan bahwa metode Simpson 1/3 menjadi lebih akurat dengan jumlah subinterval yang lebih besar.
- Setelah N≥100, nilai perkiraan π stabil dan sangat dekat dengan nilai sebenarnya, menunjukkan konvergensi yang baik dari metode Simpson 1/3.

2. Galat RMS vs N

Grafik ini menunjukkan galat RMS terhadap variasi N pada skala logaritmik untuk kedua sumbu.

- Galat RMS mulai dari nilai yang relatif besar untuk N=10 dan menurun drastis saat N meningkat.
- Galat RMS mencapai nilai yang sangat kecil (hampir mendekati nol) ketika N≥100.
- Penurunan galat RMS secara eksponensial menunjukkan bahwa metode Simpson 1/3 sangat sensitif terhadap jumlah subinterval yang digunakan.
- Setelah N≥100, galat RMS hampir nol, mengindikasikan bahwa metode ini menghasilkan nilai yang sangat akurat untuk integral dari $f(x)$.

3. Waktu Ekskusi vs N

Grafik ini menunjukkan waktu eksekusi terhadap variasi N pada skala logaritmik untuk kedua sumbu.

- Waktu eksekusi bervariasi secara non-linier dengan peningkatan N.
- Waktu eksekusi menurun dari N=10 ke N=100, kemudian sedikit meningkat untuk N=1000dan meningkat lebih lanjut untuk N=10000
- Penurunan waktu eksekusi dari N=10 ke N=100 mungkin disebabkan oleh overhead dalam inisialisasi array yang lebih besar dibandingkan dengan perhitungan integral itu sendiri untuk N kecil.
- Peningkatan waktu eksekusi untuk N≥1000N≥1000 disebabkan oleh peningkatan jumlah perhitungan yang perlu dilakukan, meskipun masih dalam waktu yang dapat diterima (dalam orde milidetik).

Kesimpulan

1. Akurasi: Metode Simpson 1/3 memberikan nilai perkiraan yang sangat akurat untuk integral dari fungsi f(x) dengan peningkatan jumlah subinterval N. Nilai perkiraan π konvergen dengan cepat ke nilai sebenarnya setelah N≥100.
2. Galat RMS: Galat RMS menurun drastis seiring dengan peningkatan N. Setelah N≥100, galat RMS sangat kecil, menunjukkan bahwa metode ini sangat efektif dalam menghasilkan nilai integral yang akurat.
3. Waktu Eksekusi: Waktu eksekusi bervariasi secara non-linier dengan peningkatan N. Meskipun waktu eksekusi meningkat untuk N yang sangat besar, peningkatan ini masih

dalam rentang yang wajar dan menunjukkan bahwa metode Simpson 1/3 efisien untuk nilai N yang besar.