Implementasi Integrasi Numerik untuk

Menghitung Estimasi Nilai Pi

Nama : Rachel Savitri

NIM : 21120122140111

Kelas : C

Link GitHub : https://github.com/aaceelll/Metode-Integrasi-Trapezoid---Rachel-Savitri---21120122140111

Source Code:

```
import numpy as np
import time

# Fungsi yang akan diintegralkan
def f(x):
    return 4 / (1 + x**2)

# Metode Integrasi Trapezoid
def trapezoid_integration(a, b, N):
    x = np.linspace(a, b, N+1)
    y = f(x)
    h = (b - a) / N #Lebar setiap subinterval
    integral = (h / 2) * (y[0] + 2 * np.sum(y[1:-1]) + y[-1])
    return integral

# Menghitung RMS galat
def rms_error(estimated_pi, true_pi):
    return np.sqrt((estimated_pi - true_pi)**2)

# Nilai referensi pi
true_pi = 3.14159265358979323846

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

# Pengujian dan pengukuran waktu eksekusi
results = []

for N in N_values:
    start_time = time.time()
    estimated_pi = trapezoid_integration(0, 1, N)
    end_time = time.time()

    rms = rms_error(estimated_pi, true_pi)
    exec_time = end_time - start_time

    results.append((N, estimated_pi, rms, exec_time))

# Hasil yang Akan Ditampilkan
for result in results:
    N, estimated_pi, rms, exec_time = result
```

```
    print(f"N = {N}, Estimated Pi = {estimated_pi}, RMS Error = {rms}, 
Execution Time = {exec_time} seconds")
```

Output:

```
N = 10, Estimated Pi = 3.1399259889071587, RMS Error = 0.0016666646826344333, Execution Time = 0.00021767616271972656 seconds
N = 100, Estimated Pi = 3.141575986923129, RMS Error = 1.6666666664111318e-05, Execution Time = 9.894371032714844e-05 seconds
N = 1000, Estimated Pi = 3.141592486923127, RMS Error = 1.6666666624587378e-07, Execution Time = 0.00045609474182128906 seconds
N = 10000, Estimated Pi = 3.1415926519231263, RMS Error = 1.666666804567285e-09, Execution Time = 0.0003120899200439453 seconds
```

Alur Kode:

```
import numpy as np
import time

# Fungsi yang akan diintegralkan
def f(x):
    return 4 / (1 + x**2)

# Metode Integrasi Trapezoid
def trapezoid_integration(a, b, N):
    x = np.linspace(a, b, N+1)
    y = f()
    h = (b - a) / N #Lebar setiap subinterval
    integral = (h / 2) * (y[0] + 2 * np.sum(y[1:-1]) + y[-1])
    return integral
```

Pada kode tersebut, fungsi f(x) didefinisikan sebagai fungsi yang akan diintegralkan, dalam hal ini `f(x):return 4 / (1 + x**2)`. Fungsi trapezoid integration (a, b, N) untuk menghitung integral menggunakan metode trapesium. Dalam metode ini, interval `(b - a)` dibagi menjadi N subinterval dengan lebar h. Kemudian, nilai x dan y dihitung untuk setiap titik dalam subinterval menggunakan numpy.

```
# Menghitung RMS galat
def rms_error(estimated_pi, true_pi):
    return np.sqrt((estimated_pi - true_pi)**2)

# Nilai referensi pi
true_pi = 3.14159265358979323846

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

# Pengujian dan pengukuran waktu eksekusi
results = []
```

Root Mean Square (RMS) dihitung dari galat antara nilai perkiraan pi yang dihasilkan dengan nilai pi yang sebenarnya. Fungsi `rms_error` akan mengambil 2 parameter yang disebutkan dan kemudian akan mengembalikkan nilai RMS galat antar keduanya. Setelah itu, program akan menetapkan nilai acuan untuk pi `(true_pi)` dengan angka seperti pada kode. Selanjutnya, variasi nilai N `(N_values)` dipilih untuk pengujian. Nantinya, setiap hasil pengujian dan pengukuran waktu eksekusi dari setiap nilai N akan dicatat dalam daftar `results`.

```
for N in N_values:
    start_time = time.time()
    estimated_pi = trapezoid_integration(0, 1, N)
    end_time = time.time()

    rms = rms_error(estimated_pi, true_pi)
    exec_time = end_time - start_time

    results.append((N, estimated_pi, rms, exec_time))

# Hasil yang Akan Ditampilkan
for result in results:
    N, estimated_pi, rms, exec_time = result
    print(f"N = {N}, Estimated Pi = {estimated_pi}, RMS Error = {rms},
Execution Time = {exec_time} seconds")
```

Perhitungan integral menggunakan metode trapesium untuk mengestimasi nilai π (pi) dengan berbagai jumlah iterasi yang ditentukan oleh nilai N dalam daftar N_values. kode menghitung nilai kesalahan rata-rata kuadrat (RMS) antara estimasi π dan nilai π sebenarnya, serta menghitung waktu eksekusi. Hasil perhitungan, termasuk nilai N, estimasi π, RMS error, dan waktu eksekusi, disimpan dalam daftar hasil.

Hasil Pengujian:

Dari hasil pengujian dengan nilai N, didapatkan data sebagai berikut:

1. **N = 10**
   - Estimasi Pi : 3.1399259889071587
   - Galat RMS : 0.0016666646826344333
   - Waktu Eksekusi : 0.00021767616271972656 seconds
2. **N = 100**
   - Estimasi Pi : 3.141575986923129
   - Galat RMS : 1.6666666664111318e-05
   - Waktu Eksekusi : 9.894371032714844e-05 seconds
3. **N = 1000**
   - Estimasi Pi : 3.141592486923127
   - Galat RMS : 1.6666666624587378e-07
   - Waktu Eksekusi : 0.00045609474182128906 seconds

4. **N = 10000**

- Estimasi Pi : 3.1415926519231263
- Galat RMS : 1.666666804567285e-09
- Waktu Eksekusi : 0.0003120899200439453 seconds

Analisis Hasil:

1. Galat RMS secara signifikan akan menurun seiring dengan peningkatan nilai N. Jadi, dapat dikatakan bahwa semakin kecil nilai galat RMS, maka akan semakin akurat estimasinya.
2. Pertimbangan pada nilai N untuk memilih nilai N yang optimal tergantung pada kebutuhan spesifik dari aplikasi. Misalnya, nilai N = 10000 memberikan estimasi yang sangat dekat dengan nilai pi sebenarnya, tetapi membutuhkan waktu eksekusi yang jauh lebih lama dibandingkan dengan nilai N yang lebih kecil.
3. Waktu eksekusi meningkat secara keseluruhan seiring meningkatnya nilai N, namun terdapat variasi waktu eksekusi pada tiap nilai N yang perkembangannya tidak selalu meningkat secara linear. Misalnya, waktu eksekusi pada nilai N = 100 lebih lama disbanding N = 1000.

Kesimpulan:

Semakin besar nilai N yang digunakan, semakin mendekati nilai π yang dihasilkan dengan nilai π sebenarnya. Hal ini ditunjukkan dengan penurunan galat RMS yang menunjukkan peningkatan akurasi estimasi. Namun, perlu diingat bahwa waktu eksekusi umumnya meningkat seiring dengan kenaikan nilai N. Variasi waktu eksekusi di antara nilai N yang berbeda menunjukkan adanya kompleksitas dalam proses komputasi. Oleh karena itu, pemilihan nilai N yang optimal perlu mempertimbangkan keseimbangan antara akurasi estimasi dan waktu eksekusi. Nilai N tertentu mungkin memberikan keseimbangan yang baik, tergantung pada kebutuhan spesifik aplikasi atau analisis yang dilakukan.