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
import math

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

# Metode Simpson 1/3
def simpson_integration(a, b, n):
    h = (b - a) / n
    integral = f(a) + f(b)
    for i in range(1, n):
        x = a + i * h
        if i % 2 == 0:
            integral += 2 * f(x)
        else:
            integral += 4 * f(x)
    integral *= h / 3
    return integral

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

# Nilai referensi pi
pi_ref = 3.14159265358979323846

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

# Loop untuk menghitung integral, galat RMS, dan waktu eksekusi untuk
setiap N
results = []
for N in N_values:
    start_time = time.time()
    estimated_pi = simpson_integration(0, 1, N)
    end_time = time.time()
    rms = rms_error(estimated_pi, pi_ref)
    execution_time = end_time - start_time
    results.append((N, estimated_pi, rms, execution_time))
```

```
# Menampilkan hasil
for result in results:
    N, estimated_pi, rms, execution_time = result
    print(f'N = {N}:')
    print(f'  Estimated Pi = {estimated_pi}')
    print(f'  RMS Error = {rms}')
    print(f'  Execution Time = {execution_time} seconds')
    print()
```

Konsep yang digunakan :

Integrasi Simpson 1/3 yang merupakan metode numerik untuk menghitung integral yang menggunakan polinomial kuadrat untuk mengaproksimasi fungsi yang diintegrasikan.

Hasil Pengujian:

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

1. **N = 10**
   - Estimasi Pi : 3.1415926139392147
   - Galat RMS : 3.9650578376182466e-08
   - Waktu Eksekusi : 2.7418136596679688e-05 seconds
2. **N = 100**
   - Estimasi Pi : 3.1415926535897545
   - Galat RMS : 3.863576125695545e-14
   - Waktu Eksekusi : 7.319450378417969e-05 seconds
3. **N = 1000**
   - Estimasi Pi : 3.1415926535897913
   - Galat RMS : 1.7763568394002505e-15
   - Waktu Eksekusi : 0.0007622241973876953 seconds
4. **N = 10000**
   - Estimasi Pi : 3.141592653589807
   - Galat RMS : 1.3766765505351941e-14
   - Waktu Eksekusi : 0.007527828216552734 seconds

Analisis Hasil:

Kode di atas menghitung integral dari fungsi $f(x) = \frac{4}{1+x^2}$ dari 0 sampai 1 dengan menggunakan metode Simpson 1/3 untuk variasi nilai N = 10, 100, 1000, dan 10000. Hasil dari estimasi pi, galat RMS, dan waktu eksekusi untuk setiap N adalah pada gambar berikut.

```
N = 10:
  Estimated Pi = 3.1415926139392147
  RMS Error = 3.9650578376182466e-08
  Execution Time = 2.7418136596679688e-05 seconds

N = 100:
  Estimated Pi = 3.1415926535897545
  RMS Error = 3.863576125695545e-14
  Execution Time = 7.319450378417969e-05 seconds

N = 1000:
  Estimated Pi = 3.1415926535897913
  RMS Error = 1.7763568394002505e-15
  Execution Time = 0.0007622241973876953 seconds

N = 10000:
  Estimated Pi = 3.141592653589807
  RMS Error = 1.3766765505351941e-14
  Execution Time = 0.007527828216552734 seconds
```

Dari gambar tersebut dapat disimpulkan bahwa:

1. Pertimbangan pada nilai N untuk memilih nilai N yang optimal tergantung pada kebutuhan spesifik dari aplikasi. Misalnya, nilai N = 100 memberikan estimasi yang sangat dekat dengan nilai pi sebenarnya, tetapi membutuhkan waktu eksekusi yang jauh lebih lama dibandingkan dengan nilai N yang lebih kecil.
2. 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.
3. waktu eksekusi metode ini akan meningkat seiring dengan bertambahnya jumlah subinterval.

Alur Kode:

```
import numpy as np
import time
import math

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

Pada kode tersebut, fungsi f(x) didefinisikan untuk mewakili fungsi `f(x):return 4 / (1 + x**2)` yang merupakan fungsi yang akan diintegrasikan. Metode Simpson 1/3 akan menggunakan fungsi ini untuk mengevaluasi nilai fungsi pada titik-titik tertentu dalam interval integrasi untuk mendekati nilai integral sebenarnya.

```
# Metode Simpson 1/3
def simpson_integration(a, b, n):
    h = (b - a) / n
```

```
    integral = f(a) + f(b)
    for i in range(1, n):
        x = a + i * h
        if i % 2 == 0:
            integral += 2 * f(x)
        else:
            integral += 4 * f(x)
    integral *= h / 3
    return integral

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

Potongan kode tersebut menghitung nilai perkiraan integral definite dari fungsi f pada interval [a, b] menggunakan metode Simpson 1/3. Pertama, fungsi ini menghitung lebar tiap subinterval (h) dengan membagi (b - a) dengan jumlah subinterval (n). Kemudian inisialisasi nilai integral dengan nilai fungsi f di titik a dan b. Pengecekan dilakukan pada setiap iterasi (i) untuk menentukan apakah i genap atau ganjil.

- Jika i genap, maka kontribusi subinterval tersebut adalah 2 kali nilai fungsi f di titik x.
- Jika i ganjil, maka kontribusi subinterval tersebut adalah 4 kali nilai fungsi f di titik x.

Setelah iterasi selesai, nilai integral dikalikan dengan h / 3 untuk mendapatkan hasil akhir sesuai rumus metode Simpson 1/3. Kemudian menghitung galat RMS (Root Mean Square Error) antara nilai perkiraan (estimated) dan nilai aktual (actual) dari integral definite.

```
# Loop untuk menghitung integral, galat RMS, dan waktu eksekusi untuk
setiap N
results = []
for N in N_values:
    start_time = time.time()
    estimated_pi = simpson_integration(0, 1, N)
    end_time = time.time()
    rms = rms_error(estimated_pi, pi_ref)
    execution_time = end_time - start_time
    results.append((N, estimated_pi, rms, execution_time))

# Menampilkan hasil
for result in results:
    N, estimated_pi, rms, execution_time = result
    print(f'N = {N}:')
    print(f'  Estimated Pi = {estimated_pi}')
    print(f'  RMS Error = {rms}')
    print(f'  Execution Time = {execution_time} seconds')
    print()
```

Kode ini terlebih dahulu menyiapkan daftar kosong `results` untuk menyimpan hasil perhitungan. Iterasi akan dilakukan berdasarkan nilai-nilai yang ada pada list `N_values`. Setiap nilai N merepresentasikan jumlah subinterval yang akan digunakan.

Di dalam iterasi:

- Fungsi `time.time()` dipanggil untuk mencatat waktu mulai perhitungan (`start_time`). Kemudian (`estimated_pi`) dihitung menggunakan fungsi `simpson_integration` dengan argumen `0` sebagai batas bawah, `1` sebagai batas atas, dan nilai `N` sebagai jumlah subinterval.
- Fungsi `time.time()` dipanggil kembali untuk mencatat waktu selesai perhitungan (`end_time`).
- Waktu eksekusi (`execution_time`) dihitung dengan selisih `end_time` dan `start_time`.
- Setelah iterasi selesai, setiap hasil perhitungan yang terdiri dari `N`, `estimated_pi`, `rms`, dan `execution_time` disimpan sebagai tuple ke dalam list `results`.

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.