Nama : Annisa Charisma Wijayanti
NIM : 211220122140086
Mata Kuliah : Metode Numerik – Kelas D
Jurusan : Teknik Komputer
Link Github : https://github.com/annisacharisma/Implementasi-Integrasi_Annisa-Charisma-Wijayanti_Metode-Numerik

Implementasi Integrasi Numerik untuk Menghitung Estimasi nilai Pi

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.

INTEGRASI SIMPSON 1/3

Aturan Simpson adalah salah satu metode numerik yang digunakan untuk mengevaluasi integral tertentu. Metode integrasi Simpson membagi interval [*a*,*b*] menjadi *n* subinterval, di mana *n* harus merupakan bilangan genap. Setiap pasangan subinterval digunakan untuk mengaproksimasi luas area di bawah kurva fungsi *f*(*x*) dengan menggunakan polinomial kuadrat.

Metode Simpson 1/3:

$$\int_a^b f(x)dx = \frac{h}{3}\left(f_0 + 4\sum_{i=1,3,5,\ldots}^{n-1} f_i + 2\sum_{i=2,4,6,\ldots}^{n-1} f_i + f_n\right)$$

Sehingga jika dijabarkan menjadi:

$$\int_a^b f(x)dx = \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \ldots + 2f_{n-2} + 4f_{n-1} + f_n)$$

Dimana *h* adalah lebar segmen:

$$h = \frac{b-a}{n}$$

Keterangan: b = batas atas integral

a = batas bawah integral

n = jumlah subinterval, harus genap

h = lebar segmen.

Persamaan tersebut mudah diingat dengan mengingat pola koefisien persamaan tersebut adalah 1,4,2,4,2,…,2,4,1. Namun penggunaan kaidah 1/3 Simpson mengharuskan jumlah subinterval n genap. Pada Aturan Simpson 1/3 merupakan perpanjangan dari aturan trapesium di mana integral didekati dengan polinomial orde kedua.

Konsep Algoritma Metode Simpson 1/3:

- Tentukan fungsi *f*(*x*) dan selang integrasinya yaitu batas bawah *b* dan batas atas *a* integrasi
- Tentukan jumlah subinterval *n*
- Hitung lebar segmen
- Tentukan awal integrasi $x_0 = a$ dan akhir $x_n = b$, hitung juga nilai *f*(*a*) serta *f*(*b*)
- Untuk *x* = 1, 2, 3, . . . , *n* – 1,

- Jika ganjil, hitung: $4 \times f(x)$
- Jika genap, hitung: $2 \times f(x)$

- Hitung hasil akhir penjumlahan dengan rumus yang telah disebutkan diatas, $Hasil = \frac{h}{3} * sum$

Source Code:

```
import numpy as np
import time

# Mendefinisikan fungsi Integral
def f(x):
    return 4 / (1 + x**2)

def simpson_1_3(a, b, N):
    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-1, 2):
        integral += 2 * f(a + i * h)

    integral *= h / 3
    return integral

def rms_error(estimate, reference):
    error = (estimate - reference)**2
    return np.sqrt(error)

# Nilai referensi pi
reference_pi = 3.14159265358979323846

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

# Menyimpan hasil estimasi, galat RMS, dan waktu eksekusi
results = []
execution_times = []
rms_errors = []

for N in N_values:
    start_time = time.time()
    estimate = simpson_1_3(0, 1, N)
    end_time = time.time()

    results.append(estimate)
    execution_times.append(end_time - start_time)

    # Menghitung galat RMS untuk nilai N saat ini
    rms_errors.append(rms_error(estimate, reference_pi))

# Hasil yang akan ditampilkan
for N, result, error, exec_time in zip(N_values, results, rms_errors,
execution_times):
    print(f"n={N}")
```

```
    print(f"nilai pi= {result}")
    print(f"ralat rms= {error}")
    print(f"waktu= {exec_time} detik\n")
```

Penjelasan Kode:

Kode tersebut bertujuan untuk menghitung estimasi nilai Pi menggunakan metode Simpson 1/3 dan menganalisis galat RMS (Root Mean Square Error) serta waktu eksekusi untuk berbagai jumlah subinterval *n*-nya.

1. Import Modul yang diperlukan. `Import numpy as np` digunakan untuk operasi numerik sedangkan `import time` digunakan untuk mengukur waktu eksekusinya.
2. Tahap selanjutnya adalah mendefinisikan fungsi integral. Fungsi $f(x) = \frac{4}{1+x^2}$ didefinisikan pada kode menggunakan bahasa python menjadi:

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

3. Menghitung lebar segmen dengan menggunakan rumus `h = (b - a) / N`
4. Menentukan awal integrasi $x_0 = a$ dan akhir $x_n = b$, serta menghitung nilai *f*(*a*) serta *f*(*b*) dengan menggunakan kode `integral = f(a) + f(b)`
5. Selanjutnya adalah menghitung jumlah dari *i* = 1 hingga *i* = *n* – 1. Jika *i* = ganjil, hitung dengan rumus $4 \times f(x), dan$ jika i = genap, hitung dengan rumus $2 \times f(x)$. Jika diimplementasikan pada kode menjadi:

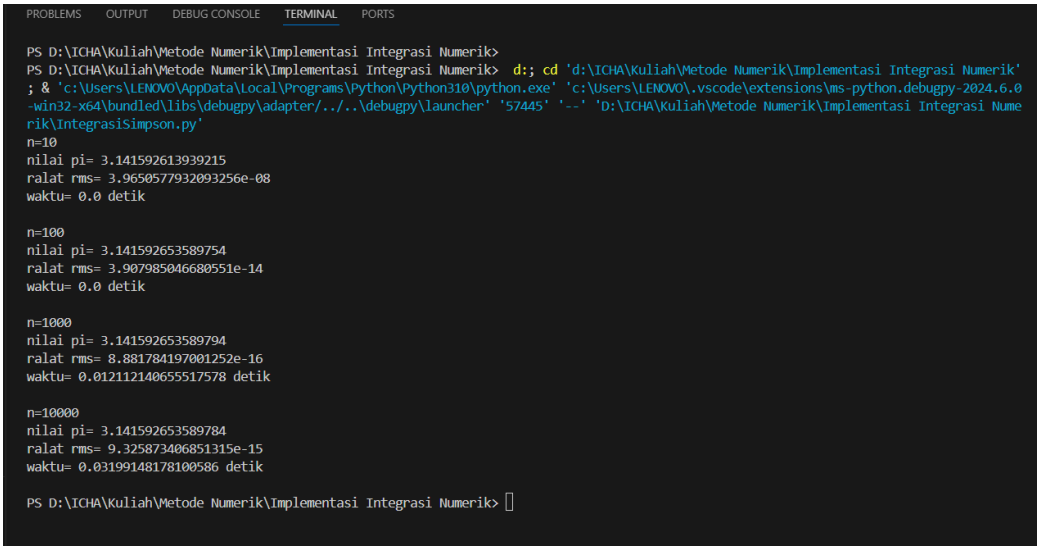
```
for i in range(1, N, 2):
        integral += 4 * f(a + i * h)
for i in range(2, N-1, 2):
        integral += 2 * f(a + i * h)
```

6. Kode `integral *= h / 3` digunakan untuk menghitung hasil akhir penjumlahan dengan rumus aturan simpson 1/3.
7. Tahap selanjutnya adalah menggunakan fungsi `rms_error(estimate, reference)` untuk menghitung galat RMS antara estimasi dan nilai referensi Pi yang telah ditetapkan, yaitu `reference_pi = 3.14159265358979323846`
8. Menetapkan berbagai variasi nilai *n*, `N_values = [10, 100, 1000, 10000]` yang akan digunakan untuk perhitungan.
9. Kemudian loop untuk menghitung estimasi, galat RMS, dan waktu eksekusi untuk berbagai jumlah subinterval *n* yang telah ditentukan. Proses dimulai dengan melakukan iterasi melalui list `N_values`, yang berisi nilai-nilai *n*. Untuk setiap nilai *n*, waktu mulai dicatat menggunakan `time.time`. Estimasi nilai Pi kemudian dihitung dengan fungsi

`simpson_1_3(0, 1, N)`, dengan 0 dan 1 sebagai batas integrasinya. Lalu waktu eksekusi ini disimpan dalam list `execution_times`. Estimasi nilai Pi disimpan dalam list `results`, dan galat RMS antara estimasi dan nilai referensi Pi dihitung menggunakan fungsi `rms_error(estimate, reference_pi)`, kemudian disimpan dalam list `rms_errors`.

10. Tahap terakhir adalah menampilkan hasil menggunakan loop untuk menampilkan hasil estimasi nilai Pi, galat RMS, dan waktu eksekusi untuk setiap nilai *n*.

Hasil:



Penjelasan Hasil:

Estimasi Nilai Pi, Galat RMS, dan Waktu Eksekusi:

- *n* = 10
  - nilai pi = 3.141592613939215
  - ralat rms = 3.9650577932093256e-08
  - waktu = 0.0 detik
- *n* = 100
  - nilai pi = 3.141592653589754
  - ralat rms = 3.907985046680551e-14
  - waktu = 0.0 detik
- *n* = 1000
  - nilai pi = 3.141592653589794
  - ralat rms = 8.881784197001252e-16
  - waktu = 0.0 detik

- n = 10000
  - nilai pi = 3.141592653589784
  - ralat rms = 9.325873406851315e-15
  - waktu = 0.020061731338500977 detik

Estimasi nilai Pi didapatkan dari perhitungan menggunakan rumus simpson 1/3, Ralat RMS didapatkan dari rumus $RMS\ Error = \sqrt{(Estimate - Reference)^2}$. Waktu eksekusi dihitung sebagai selisih antara waktu mulai dan waktu selesai eksekusi fungsi menggunakan rumus Waktu Eksekusi = end time – start time.

Berdasarkan data hasil dari perhitungan menggunakan metode Simpson 1/3, dengan meningkatnya nilai *n*, hasil dari nilai Pi semakin mendekati nilai sebenarnya atau nilai referensi yang digunakan yaitu 3.14159265358979323846. Lalu untuk Galat RMS nya, semakin besar nilai *n* yang digunakan, semakin menurun galat RMS yang dihasilkan sehingga menunjukkan bahwa hasilnya akan semakin mendekati nilai referensi dari pi. Namun, dalam sisi waktu eksekusi, dengan meningkatnya nilai *n* maka waktu eksekusi yang dibutuhkan akan semakin besar. Contohnya Pada *n*=10, galat RMS masih relatif besar, tetapi berkurang drastis ketika *n* meningkat menjadi 100, 1000, dan 10000, menunjukkan peningkatan akurasi. Waktu eksekusi untuk jumlah subinterval yang kecil hingga menengah (10, 100, 1000) hampir tidak terdeteksi, menandakan efisiensi metode ini. Namun, untuk *n*=10000, waktu eksekusi mulai terukur (sekitar 0.0162 detik), meskipun masih dalam batas yang wajar mengingat peningkatan ketepatan yang diperoleh.