

Nama : Femas Arianda Rizki

NIM : 21120122130080

Kelas : Metode Numerik – B

Dua digit NIM terakhir % 3 = 80 % 3 = 2

Jadi mengerjakan dengan Metode 3 (Integrasi Simpson 1/3)

## A. Ringkasan

Nilai pi dapat dihitung secara numerik dengan mencari nilai integral dari fungsi  $f(x) = 4 / (1 + x^2)$  dari 0 hingga 1. Untuk menyelesaikan perhitungan dari nilai integral fungsi tersebut salah satunya yaitu menggunakan metode integrasi simpson 1/3. Di sini saya mengimplementasi solusi permasalahan di atas dengan membuat kode sumber dengan bahasa pemrograman python. Di situ juga disertakan kode testing untuk menguji kode sumber dengan variasi nilai N = 10, 100, 1000, dan 10000. Serta juga dilakukan perhitungan galat RMS dan pengukuran waktu eksekusi dari tiap variasi N. Nilai referensi pi yang digunakan adalah 3.14159265358979323846. Jadi hasil akhir dari kode ini berupa estimasi nilai pi beserta galat RMS dan waktu eksekusi dari tiap variasi N.

## B. Konsep

Metode Simpson 1/3 adalah teknik numerik untuk menghitung integral dari suatu fungsi yang memiliki partisi genap. Konsep dasarnya adalah membagi interval integral menjadi beberapa subinterval yang sama panjang, kemudian menggunakan polinomial kuadrat untuk mengaproksimasi fungsi pada setiap subinterval. Untuk setiap dua subinterval, integralnya dihitung menggunakan formula:

$$\int_a^b f(x) dx \approx \frac{h}{3} \left[ f(x_0) + 4 \sum_{i=1,3,5,\dots}^{n-1} f(x_i) + 2 \sum_{i=2,4,6,\dots}^{n-2} f(x_i) + f(x_n) \right]$$

Kode di atas menggunakan metode Simpson 1/3 untuk menghitung estimasi nilai pi dengan cara mengintegrasikan fungsi  $f(x) = 4 / (1 + x^2)$  dari 0 hingga 1. Fungsi `simpson_13(a, b, N)` bertanggung jawab untuk melakukan perhitungan integral menggunakan metode Simpson 1/3 dengan partisi N. Fungsi `calculate_pi(N_values)` menghitung estimasi nilai pi untuk berbagai nilai N dan mencatat waktu eksekusi serta galat RMS dibandingkan dengan nilai referensi pi. Hasil dari setiap perhitungan untuk variasi nilai N dicetak untuk memberikan gambaran

mengenai akurasi dan efisiensi waktu dari metode Simpson 1/3 dalam menghitung integral untuk estimasi nilai pi.

### C. Implementasi Kode

#### Kode Sumber, Kode Testing, Galat RMS, dan Waktu Eksekusi:

```
# Nama : Femas Arianda Rizki
# NIM : 21120122130080
# Kelas : Metode Numerik - B

# Dua digit NIM terakhir % 3 = 80 % 3 = 2
# Jadi mengerjakan dengan Metode 3 (Integrasi Simpson 1/3)

# Kode sumber
import numpy as np
import time
from decimal import Decimal, getcontext

# Menetapkan presisi ke 24 untuk memastikan kita mendapatkan 20
digit setelah titik desimal dan hasil yang lebih presisi
getcontext().prec = 24

# Fungsi yang akan diintegrasikan,  $f(x) = 4 / (1 + x^2)$ 
def f(x):
    return Decimal(4) / (Decimal(1) + x**2)

# Metode integrasi Simpson 1/3 untuk menghitung integral fungsi
f dari a ke b dengan N partisi
def simpson_13(a, b, N):
    if N % 2 == 1:
        N += 1 # Make N even if it is odd
    h = (b - a) / N
    integral = f(Decimal(a)) + f(Decimal(b)) # Menghitung nilai
f pada batas integrasi
    for i in range(1, N):
        k = Decimal(a) + i * Decimal(h) # Menghitung nilai k pada
partisi ke-i
        if i % 2 == 0:
            integral += 2 * f(k) # Jika i genap, kalikan dengan
2
        else:
            integral += 4 * f(k) # Jika i ganjil, kalikan dengan
4
    integral *= Decimal(h) / Decimal(3) # Kalikan hasil dengan
h/3 sesuai dengan rumus Simpson 1/3
    return integral
```

```

# Fungsi untuk menghitung nilai pi untuk berbagai nilai N
def calculate_pi(N_values):
    pi_ref = Decimal('3.14159265358979323846') # Nilai referensi
    pi dengan 20 digit setelah titik desimal
    results = []

    for N in N_values:
        start_time = time.time() # Catat waktu mulai
        pi_est = simpson_13(Decimal(0), Decimal(1), N) # Hitung
        estimasi pi menggunakan metode Simpson 1/3
        end_time = time.time() # Catat waktu selesai
        elapsed_time = end_time - start_time # Hitung waktu yang
        diperlukan
        rms_error = np.sqrt((pi_ref - pi_est) ** 2) # Hitung galat
        RMS (Root Mean Square Error)
        results.append((N, pi_est, rms_error, elapsed_time)) #
        Simpan hasil dalam daftar

    return results

# Kode testing
N_values = [10, 100, 1000, 10000] # Variasi nilai N
results = calculate_pi(N_values) # Hitung nilai pi untuk masing-
masing N

for result in results:
    N, pi_est, rms_error, elapsed_time = result
    print(f"N = {N}: Pi Estimate = {pi_est}, RMS Error =
    {rms_error}, Time Elapsed = {elapsed_time} seconds")

```

### Penjelasan Alur Kode:

#### a. Mengimpor pustaka

```

import numpy as np
import time
from decimal import Decimal, getcontext

```

- Mengimpor `numpy` untuk perhitungan numerik.
- Mengimpor `time` untuk mengukur waktu eksekusi.
- Mengimpor `Decimal` dan `getcontext` dari modul `decimal` untuk presisi tinggi dalam perhitungan angka desimal.

#### b. Menetapkan Presisi

```

# Menetapkan presisi ke 24 untuk memastikan kita mendapatkan
20 digit setelah titik desimal dan hasil yang lebih presisi
getcontext().prec = 24

```

- Menetapkan presisi ke 24 digit untuk memastikan hasil perhitungan pi presisi hingga 20 digit setelah titik desimal.

#### c. Fungsi $f(x)$

```
# Fungsi yang akan diintegrasikan,  $f(x) = 4 / (1 + x^2)$ 
def f(x):
    return Decimal(4) / (Decimal(1) + x**2)
```

- Mendefinisikan fungsi yang akan diintegrasikan:  $f(x) = 4 / (1 + x^2)$ .
- Menggunakan tipe Decimal untuk menjaga presisi tinggi.

d. Fungsi `simpson_13(a, b, N)`

```
# Metode integrasi Simpson 1/3 untuk menghitung integral fungsi
# f dari a ke b dengan N partisi
def simpson_13(a, b, N):
    if N % 2 == 1:
        N += 1 # Jika N ganjil, tambahkan 1 untuk membuatnya
        genap
    h = (b - a) / N # Lebar tiap partisi
    integral = f(Decimal(a)) + f(Decimal(b)) # Menghitung
    nilai f pada batas integrasi
    for i in range(1, N):
        k = Decimal(a) + i * Decimal(h) # Menghitung nilai k
        pada partisi ke-i
        if i % 2 == 0:
            integral += 2 * f(k) # Jika i genap, kalikan dengan
            2
        else:
            integral += 4 * f(k) # Jika i ganjil, kalikan
            dengan 4
    integral *= Decimal(h) / Decimal(3) # Kalikan hasil dengan
    h/3 sesuai dengan rumus Simpson 1/3
    return integral
```

- Mengimplementasikan metode integrasi Simpson 1/3 untuk menghitung integral fungsi  $f(x)$  dari  $a$  ke  $b$  dengan  $N$  partisi.
- Jika  $N$  ganjil, tambahkan 1 agar  $N$  menjadi genap.
- Hitung lebar tiap partisi  $h$ .
- Hitung nilai fungsi pada batas integrasi dan setiap partisi.
- Gunakan koefisien 2 dan 4 sesuai aturan Simpson 1/3.
- Kalikan hasil akhir dengan  $h/3$ .

e. Fungsi `calculate_pi(N_values)`

```
# Fungsi untuk menghitung nilai pi untuk berbagai nilai N
def calculate_pi(N_values):
    pi_ref = Decimal('3.14159265358979323846') # Nilai
    referensi pi dengan 20 digit setelah titik desimal
    results = []

    for N in N_values:
        start_time = time.time() # Catat waktu mulai
```

```

        pi_est = simpson_13(Decimal(0), Decimal(1), N) #
Hitung estimasi pi menggunakan metode Simpson 1/3
        end_time = time.time() # Catat waktu selesai
        elapsed_time = end_time - start_time # Hitung waktu
yang diperlukan
        rms_error = np.sqrt((pi_ref - pi_est) ** 2) # Hitung
galat RMS (Root Mean Square Error)
        results.append((N, pi_est, rms_error, elapsed_time)) #
Simpan hasil dalam daftar

    return results

```

- Fungsi ini menghitung nilai pi untuk berbagai nilai N.
- Menggunakan nilai referensi pi dengan 20 digit setelah titik desimal.
- Untuk setiap nilai N, catat waktu mulai dan waktu selesai, hitung waktu eksekusi.
- Hitung estimasi pi menggunakan metode Simpson 1/3.
- Hitung galat RMS (Root Mean Square Error) antara estimasi dan nilai referensi pi.
- Simpan hasil estimasi pi, galat RMS, dan waktu eksekusi dalam daftar hasil.

#### f. Kode Testing

```

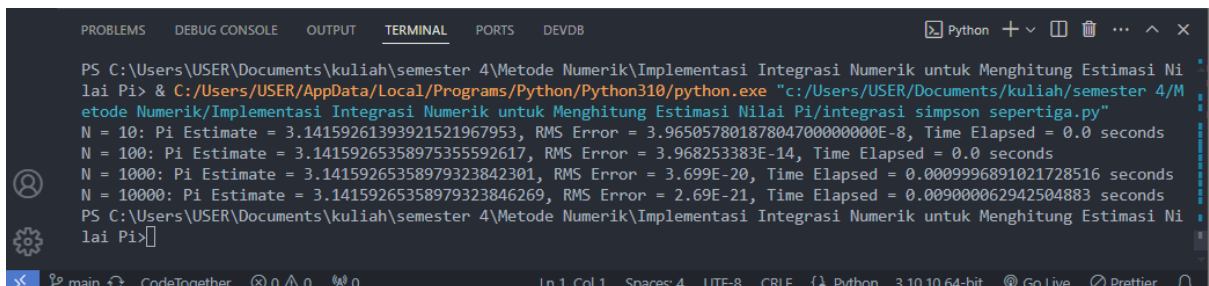
# Kode testing
N_values = [10, 100, 1000, 10000] # Variasi nilai N
results = calculate_pi(N_values) # Hitung nilai pi untuk
masing-masing N

for result in results:
    N, pi_est, rms_error, elapsed_time = result
    print(f"N = {N}: Pi Estimate = {pi_est}, RMS Error =
{rms_error}, Time Elapsed = {elapsed_time} seconds")

```

- Mendefinisikan variasi nilai N yang akan digunakan: 10, 100, 1000, dan 10000.
- Menghitung nilai pi untuk masing-masing nilai N menggunakan fungsi calculate\_pi.
- Menampilkan hasil estimasi pi, galat RMS, dan waktu eksekusi untuk setiap nilai N.

## D. Hasil Pengujian



```

PS C:\Users\USER\Documents\kuliah\semester 4\Metode Numerik\Implementasi Integrasi Numerik untuk Menghitung Estimasi Nilai Pi> & C:/Users/USER/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/USER/Documents/kuliah/semester 4/Metode Numerik/Implementasi Integrasi Numerik untuk Menghitung Estimasi Nilai Pi/integrasi simpson sepertiga.py"
N = 10: Pi Estimate = 3.14159261393921521967953, RMS Error = 3.96505780187804700000000E-8, Time Elapsed = 0.0 seconds
N = 100: Pi Estimate = 3.14159265358975355592617, RMS Error = 3.968253383E-14, Time Elapsed = 0.0 seconds
N = 1000: Pi Estimate = 3.14159265358979323842301, RMS Error = 3.699E-20, Time Elapsed = 0.0009996891021728516 seconds
N = 10000: Pi Estimate = 3.14159265358979323846269, RMS Error = 2.69E-21, Time Elapsed = 0.009000062942504883 seconds
PS C:\Users\USER\Documents\kuliah\semester 4\Metode Numerik\Implementasi Integrasi Numerik untuk Menghitung Estimasi Nilai Pi>

```

## E. Analisis

Setelah dilakukan penulisan kode sumber untuk mencari estimasi nilai pi beserta galat RMS dan waktu eksekusi dari tiap variasi N, maka didapat hasilnya. Sebelumnya, dalam kode tersebut digunakan 24 digit angka desimal, jadi hasil estimasi pi yang didapat 23 digit angka desimal di belakang koma. Dalam mencari solusi tersebut digunakan beberapa variasi nilai N, yaitu 10, 100, 1000, 10000. Karena hal itu, hasil estimasi nilai pi beserta galat RMS dan waktu eksekusi dari tiap variasi N akan berbeda-beda. Semakin tinggi nilai N, estimasi nilai pi yang didapat semakin sesuai dengan nilai pi referensi yang ada. Berikut penjabarannya, N = 10 memiliki kesesuaian hingga digit ke-7 di belakang koma, N = 100 memiliki kesesuaian hingga digit ke-13 di belakang koma, N = 1000 memiliki kesesuaian hingga digit ke-19 di belakang koma, dan N = 10000 memiliki kesesuaian hingga digit ke-20 di belakang koma atau bisa dikatakan sesuai dengan nilai referensi pi yang ada.

Selain estimasi nilai pi, juga dilakukan pencarian galat RMS dan waktu eksekusi tiap variasi nilai N. Antara beberapa hal tersebut terdapat hubungan satu sama lain. Semakin sesuai estimasi nilai pi yang didapat terhadap nilai pi referensi atau semakin tinggi nilai variasi N maka nilai galat RMS akan semakin kecil, dan begitu sebaliknya. Dan juga semakin tinggi nilai N maka waktu eksekusi yang dibutuhkan maka akan semakin tinggi, dalam hal ini nilai variasi N = 10000 memiliki waktu eksekusi yang paling tinggi. Detail galat RMS dan waktu eksekusi tiap variasi nilai N dapat dilihat pada gambar hasil pengujian. Jadi kesimpulan akhirnya, nilai variasi N berbanding lurus terhadap kesesuaian estimasi nilai pi terhadap nilai pi referensi dan waktu eksekusi yang dibutuhkan, namun berbanding terbalik dengan nilai galat RMS.