Implementasi Integrasi Trapezoid Menggunakan Python

Nama : Muhammad Nio Hastungkoro
NIM : 21120122140155
Matkul : Metode Numerik (A)

1. Ringkasan

Dokumen ini menjelaskan implementasi program yang menggunakan metode integrasi trapesium untuk menghitung nilai π (pi) dengan beberapa jumlah sub-interval yang berbeda. Program ini mengukur waktu eksekusi dan galat RMS berdasarkan nilai referensi pi yang sudah diberikan sebelumnya untuk setiap jumlah sub-interval, kemudian memvisualisasikan hasilnya. Kami akan membahas konsep, implementasi kode, hasil pengujian, dan analisis hasil.

2. Konsep

Integrasi trapesium adalah metode numerik untuk menghitung integral tertentu. Metode ini mendekati area di bawah kurva dengan membagi kurva menjadi sejumlah trapesium dan menjumlahkan luasnya.

Rumus dasar untuk integrasi trapesium adalah:

$$\int_a^b f(x)dx \approx \frac{\Delta x}{2}\left[f(x_0) + 2\sum_{i=1}^{N-1} f(x_i) + f(x_N)\right]$$

Keterangan:

- a dan b adalah batas bawah dan atas integral.
- N adalah jumlah sub-interval.
- $\Delta x = \frac{b-a}{N}$ adalah lebar setiap sub-interval.
- x_i =a$a + i\Delta x$ adalah titik-titik yang membagi interval $[a, b]$.

3. Galat RMS

Galat RMS (*Root Mean Square Error*) adalah ukuran galat antara hasil estimasi dan nilai yang sebenarnya. Galat RMS dihitung dengan mengambil akar kuadrat dari rata-rata dari selisih kuadrat antara nilai estimasi dan nilai yang sebenarnya. Untuk rumusnya, diasumsikan terdapat n pasangan nilai yang diamati dan diprediksi, yang direpresentasikan sebagai $(y_i, \hat{y}_i)$ untuk $i = 1, 2, \ldots, n$. Galat RMS dihitung dengan rumus berikut:

$$RSME = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i, \hat{y}_i)^2}$$

Keterangan:

- RSME adalah singkatan dari *Root Mean Squared Error* (Galat RMS)
- y_i adalah adalah nilai sebenarnya.
- $\hat{y}_i$ adalah adalah nilai yang diprediksi oleh model.
- n adalah jumlah observasi atau pengukuran.

Namun, karena nilai y_i yang diberikan hanya berjumlah satu, dan perhitungan galat RMS dilakukan pada setiap nilai sub-interval secara individual, maka masing-masing jumlah data pada y_i, y_i, dan n hanya berjumlah satu. Oleh karena itu, rumus galat RMS yang digunakan pada implementasi ini adalah sebagai berikut:

$$RSME = \sqrt{(y - \hat{y})^2}$$

Atau lebih sederhananya:

$$RSME = |y - \hat{y}|$$

keterangan:

- RSME adalah singkatan dari *Root Mean Squared Error* (Galat RMS)
- y adalah adalah nilai sebenarnya yang diberikan sebelumnya, yaitu referensi pi = 3.14159265358979323846
- $\hat{y}$ adalah adalah nilai yang diprediksi oleh model.

4. Implementasi Kode

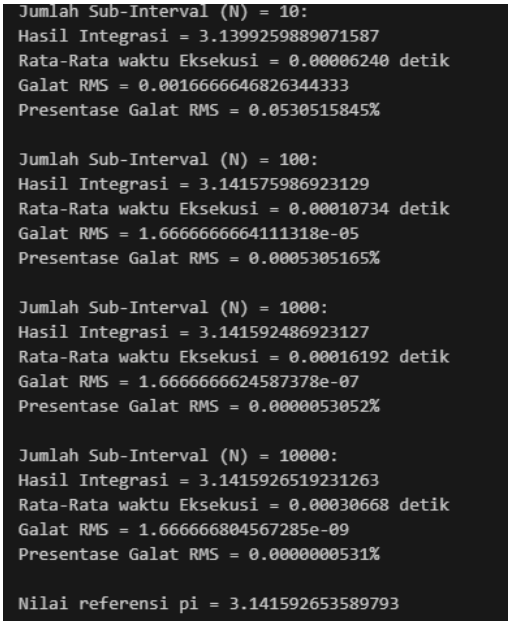
```
import timeit
import numpy as np
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from matplotlib.widgets import Button

class PiApproximation:
    def __init__(self):
        self.pi_reference = 3.14159265358979323846
        self.N_values = np.array([10, 100, 1000, 10000])
```

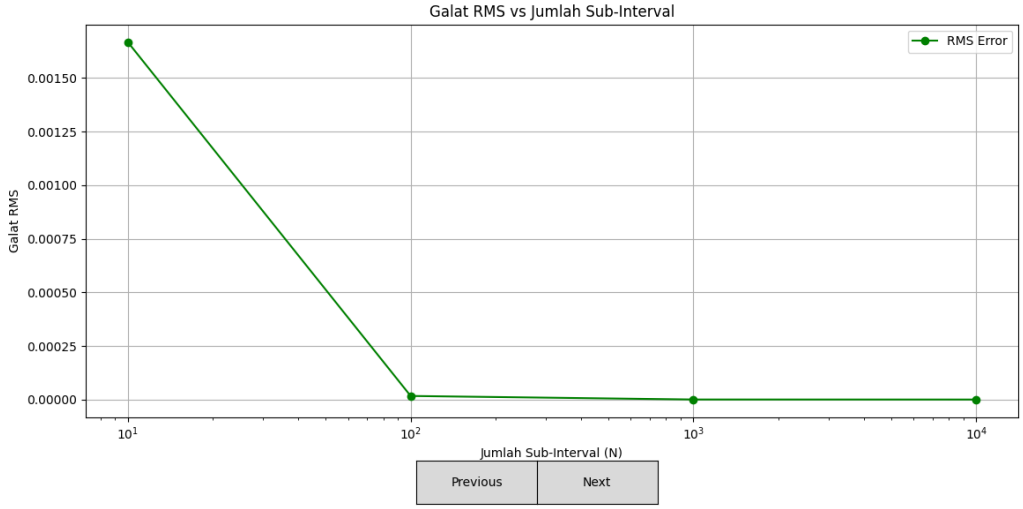
```
        self.pi_approximations = []
        self.execution_times = []
        self.rms_errors = []

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

    def trapezoid_integration(self, a, b, N):
        delta_x = (b - a) / N
        x = np.linspace(a, b, N+1)
        y = self.f(x)
        integral = (delta_x / 2) * (y[0] + 2 * np.sum(y[1:-1]) + y[-1])
        return integral

    def calculate_rms_error(self, pi_approx):
        return np.sqrt(mean_squared_error([self.pi_reference] *
len(pi_approx), pi_approx))

    def run(self):
        num_repeats = 1000
        for N in self.N_values:
            timer = timeit.Timer(lambda: self.trapezoid_integration(0, 1,
N))
            execution_times = timer.repeat(repeat=num_repeats, number=1)
            avg_execution_time = np.mean(execution_times)
            pi_approx = self.trapezoid_integration(0, 1, N)

            self.pi_approximations.append(pi_approx)
            self.execution_times.append(avg_execution_time)

            rms_error = self.calculate_rms_error([pi_approx])
            self.rms_errors.append(rms_error)

            print(f"N = {N}:")
            print(f"pi Approximation = {pi_approx}")
            print(f"Average Execution Time = {avg_execution_time:.8f}
detik")
            print(f"Error RMS = {rms_error}")
            rms_error_percent = (abs(self.pi_reference - pi_approx) /
self.pi_reference) * 100
```

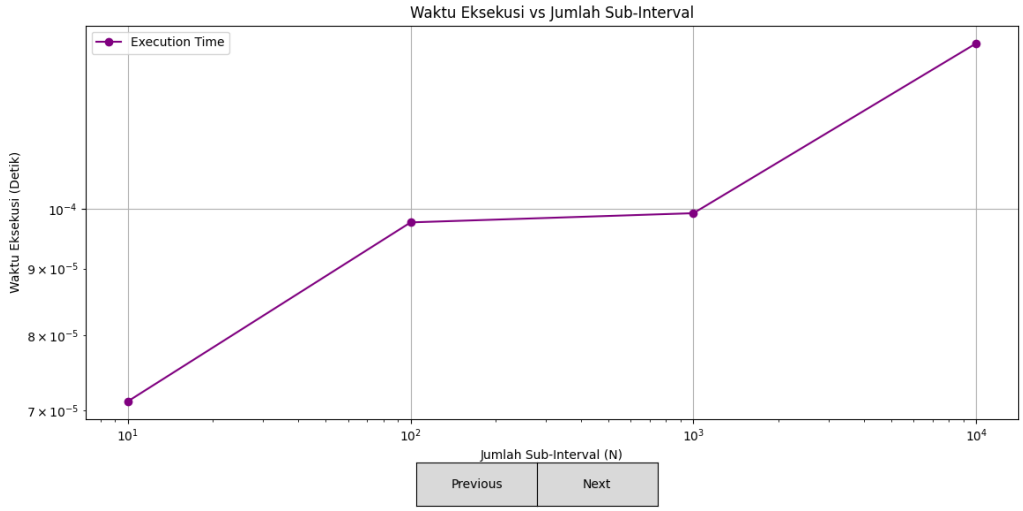
```
            print(f"Error RMS in percentage =
{rms_error_percent:.10f}%\n")

        print(f"pi Reference = {self.pi_reference}")

class Plotting:
    def __init__(self, pi_app):
        self.pi_app = pi_app
        self.fig, self.axs = plt.subplots(figsize=(13, 6))
        self.plot_index = 0
        self.plot_types = ['RMS Error', 'Execution Time']
        self.update_plot()

        plt.subplots_adjust(bottom=0.2)

        axprev = plt.axes([0.4, 0.05, 0.1, 0.075])
        axnext = plt.axes([0.5, 0.05, 0.1, 0.075])
        self.prev_button = Button(axprev, 'Previous')
        self.next_button = Button(axnext, 'Next')

        self.prev_button.on_clicked(self.show_previous_plot)
        self.next_button.on_clicked(self.show_next_plot)

        plt.show()

    def update_plot(self):
        self.axs.clear()

        if self.plot_types[self.plot_index] == 'RMS Error':
            self.axs.plot(self.pi_app.N_values, self.pi_app.rms_errors,
marker='o', color='green', label='RMS Error')
            self.axs.set_xscale('log')
            self.axs.set_xlabel('Jumlah Sub-Interval (N)')
            self.axs.set_ylabel('Galat RMS')
            self.axs.set_title('Galat RMS vs Jumlah Sub-Interval')
            self.axs.legend()
            self.axs.grid(True)
        elif self.plot_types[self.plot_index] == 'Execution Time':
            self.axs.plot(self.pi_app.N_values,
self.pi_app.execution_times, marker='o', color='purple', label='Execution
Time')
```

```
            self.axs.set_xscale('log')
            self.axs.set_yscale('log')
            self.axs.set_xlabel('Jumlah Sub-Interval (N)')
            self.axs.set_ylabel('Waktu Eksekusi (Detik)')
            self.axs.set_title('Waktu Eksekusi vs Jumlah Sub-Interval')
            self.axs.legend()
            self.axs.grid(True)

        self.fig.canvas.draw_idle()

    def show_previous_plot(self, event):
        self.plot_index = (self.plot_index - 1) % len(self.plot_types)
        self.update_plot()

    def show_next_plot(self, event):
        self.plot_index = (self.plot_index + 1) % len(self.plot_types)
        self.update_plot()

# Running the program
pi_app = PiApproximation()
pi_app.run()
plotting = Plotting(pi_app)
```

5. Hasil Pengujian







6. Analisis Hasil

Berdasarkan hasil pengujian, berikut adalah analisis dari program integrasi trapesium:

a. Akurasi Integrasi:

Akurasi integrasi meningkat seiring dengan bertambahnya jumlah sub-interval (N). Ketika N bertambah, hasil perkiraan nilai π mendekati nilai referensi (π = 3.14159265358979323846). Selain itu, galat RMS menurun secara signifikan saat N meningkat. Hal ini menunjukkan bahwa metode integrasi trapesium semakin akurat dengan jumlah sub-interval yang lebih besar.

b. Waktu Eksekusi:

Waktu eksekusi meningkat secara logaritmik seiring dengan bertambahnya jumlah sub-interval. Hal ini karenak semakin banyak sub-interval yang digunakan, semakin banyak perhitungan yang dilakukan, sehingga memakan waktu lebih lama. Meskipun waktu eksekusi meningkat, peningkatan tersebut tidak signifikan jika dibandingkan dengan peningkatan akurasi yang diperoleh.

Link GitHub:
https://github.com/NioHast/Implementasi-Integrasi-Trapezoid