TUGAS METODE NUMERIK

IMPLEMENTASI INTEGRAL RIEMANN

ZEKA EMO

21120122130075

2024/2025

Link GitHub: https://github.com/zekaemo/Implementasi-Integrasi_Metode-Numerik

Regresi Linear

Metode integrasi Riemann adalah teknik untuk menghitung integral dari suatu fungsi dengan membagi interval di bawah kurva menjadi m subinterval sama besar. Pada setiap subinterval dibentuk persegi panjang setinggi kurva pada setiap titik tengah persegi panjang tersebut. Area setiap subinterval diperoleh dengan mengalikan panjang dan lebar masing-masing persegi panjang, kemudian menjumlahkan area tersebut untuk menaksir integral fungsi dalam interval tertentu. Fungsi proses integrasi menggunakan metode titik tengah dapat dituliskan pada persamaan :

$$\int_a^b f(x)\ dx \approx \sum_{\{i=1\}}^{\{m\}} f\left(i\ \frac{|b-a|}{m} - \frac{|b-a|}{2m}\right)$$

dimana (b) dan (a) masing-masing merupakan batas atas dan bawah interval kurva yang hendak dihitung integralnya.

Error dari metode ini dapat diestimasi menggunakan Persamaan (9.9):

$$\int_a^b h(x), dx = -\frac{\{(b-a)^3\}}{\{24\ m^2\}} f''(\xi)$$

dimana (ξ) merupakan nilai antara (a) dan (b). Metode ini memberikan pendekatan numerik untuk integral, dan akurasinya meningkat dengan memperkecil lebar subinterval.

I. Source Code

```
import numpy as np
import time

pi_const = 3.14159265358979323846

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

# Metode integrasi Riemann
def riemann_integral(N):
    start_time = time.time()
    x = np.linspace(0, 1, N + 1)
    dx = 1 / N  # Lebar tiap subinterval
    mid_points = (x[:-1] + x[1:]) / 2
    integral_value = np.sum(f(mid_points) * dx)
    end_time = time.time()
    execution_time = end_time - start_time
    rms_error = np.sqrt(np.mean((integral_value - np.pi)**2))
    return integral_value, rms_error, execution_time

# Nilai N yang akan digunakan
N_values = [10, 100, 1000, 10000]

# Kalkulasi integral, galat RMS, dan waktu eksekusi untuk tiap N
results = {}
for N in N_values:
    integral_value, rms_error, execution_time = riemann_integral(N)
    results[N] = {
        'Integral Value': integral_value,
        'RMS Error': rms_error,
        'Execution Time': execution_time
    }
```

```
# Tampilkan hasil
for N, result in results.items():
    print(f"N = {N}")
    print(f"  Nilai pi: {result['Integral Value']}")
    print(f"  Galat RMS: {result['RMS Error']}")
    print(f"  Waktu Eksekusi: {result['Execution Time']} detik")
    print()
```

II. Alur Kode

1. Import library

```
import numpy as np
import time
```

- Library numpy digunakan untuk menjalankan operasi-operasi numerik.
- Library time untuk mengukur waktu eksekusi program.

2. Mendeklarasikan nilai pi dan fungsi persamaan

```
pi_const = 3.14159265358979323846
def f(x):
 return 4 / (1 + x**2)
```

- Mendefinisikan nilai pi = 3.14159265358979323846 dalam variabel pi_const.
- Menyimpan fungsi persamaan yang akan diproses dalam sebuah fungsi bernama f dengan parameter x

3. Deklarasi fungsi Integrasi Riemann

```
def riemann_integral(N):
    start_time = time.time()
    x = np.linspace(0, 1, N + 1)
    dx = 1 / N
    mid_points = (x[:-1] + x[1:]) / 2
    integral_value = np.sum(f(mid_points) * dx)
    end_time = time.time()
    execution_time = end_time - start_time
    rms_error = np.sqrt(np.mean((integral_value - pi_const)**2))
    return integral_value, rms_error, execution_time
```

- Mendefinisikan fungsi `riemann_integral` dengan parameter N

- Mencatat waktu mulai eksekusi dalam variabel `start_time` dan akhir masa eksekusi dengan `end_time`. Lama proses akan dihitung dari pengurangan nilai dua variabel ini.
- Mengatur batas serta lebar interval melalui instruksi `np.linspace` dan `dx = 1 / N`.
- Menghitung nilai integral dengan menggunakan titik tengah yang didapat dari variabel `mid_points` melalui instruksi np.sum(f(mid_points) * dx)
- Menghitung galat RMS melalui instruksi `np.sqrt(np.mean((integral_value - pi_const)**2))` yang disimpan pada variabel `rms_error`.

4. Memasukkan nilai N yang digunakan

```
N_values = [10, 100, 1000, 10000]
```

- Mendefinisikan nilai N yang akan digunakan dalam proses eksekusi

5. Kalkulasi integral, galat RMS, dan waktu eksekusi untuk tiap N

```
results = {}
for N in N_values:
    integral_value,       rms_error,       execution_time       =
riemann_integral(N)
    results[N] = {
        'Integral Value': integral_value,
        'RMS Error': rms_error,
        'Execution Time': execution_time
    }
```

- Membuat dictionary kosong bernama `results`.
- Memanggil fungsi `riemann_integral` dan melakukan proses perhitungan integral, galat RMS, dan waktu eksekusi
- Menjalankan perulangan (*loop*) sebanyak N kali

6. Menampilkan hasil

```
for N, result in results.items():
    print(f"N = {N}")
    print(f"  Nilai pi: {result['Integral Value']}")
    print(f"  Galat RMS: {result['RMS Error']}")
    print(f"  Waktu Eksekusi: {result['Execution Time']} detik")
```

```
    print()
```

- Menampilkan hasil integral, galat RMS, dan lama eksekusi

III. Analisis Hasil

```
N = 10
  Nilai pi: 3.142425985001098
  Galat RMS: 0.0008333314113047052
  Waktu Eksekusi: 0.0023078918457031 25 detik

N = 100
  Nilai pi: 3.141600986923125
  Galat RMS: 8.333333331833614e-06
  Waktu Eksekusi: 0.0001487731933593 75 detik

N = 1000
  Nilai pi: 3.1415927369231262
  Galat RMS: 8.333333122936 89e-08
  Waktu Eksekusi: 0.0001137256622314 4531 detik

N = 10000
  Nilai pi: 3.1415926544231265
  Galat RMS: 8.333340228364 25e-10
  Waktu Eksekusi: 0.0005283355712890 625 detik
```

1. **Hubungan antara N dan Galat RMS**:

Semakin besar nilai N, galat RMS cenderung menurun secara signifikan. Ini menunjukkan bahwa dengan meningkatnya jumlah iterasi (N), kesalahan dalam pendekatan nilai pi juga berkurang secara signifikan. Ini sesuai dengan harapan karena semakin banyak titik yang digunakan dalam pendekatan, semakin dekat pendekatan tersebut ke nilai sebenarnya. Hal ini terlihat dari penurunan eksponensial dalam galat RMS seiring dengan peningkatan nilai N.

2. **Hubungan antara N dan Waktu Eksekusi**:

Waktu eksekusi cenderung meningkat secara linier dengan peningkatan nilai N. Hal ini disebabkan oleh fakta bahwa semakin banyak iterasi yang diperlukan untuk menghitung nilai pi, semakin lama

waktu yang diperlukan untuk menyelesaikan komputasi. Namun, waktu eksekusi masih relatif singkat bahkan untuk nilai N yang cukup besar seperti 10000, yang menunjukkan efisiensi model yang digunakan, yaitu integrasi reinmann.

3. **Hubungan antara N dan Nilai Pi yang Dihasilkan**:

Nilai pi yang dihasilkan tampaknya konvergen ke nilai sebenarnya (3.14159265358979323846) seiring dengan peningkatan nilai N. Meskipun ada variasi kecil dalam nilai pi yang dihasilkan untuk setiap iterasi, tetapi secara keseluruhan, nilai pi yang dihasilkan mendekati nilai sebenarnya seiring dengan peningkatan nilai N. Ini menunjukkan bahwa metode yang digunakan untuk mengestimasi nilai pi cukup akurat, terutama ketika jumlah iterasi (N) ditingkatkan.

Kesimpulannya, dengan meningkatnya nilai N, galat dalam pendekatan nilai pi menurun, waktu eksekusi meningkat, dan nilai pi yang dihasilkan mendekati nilai sebenarnya. Metode yang digunakan cukup efisien dalam menghasilkan perkiraan yang akurat untuk nilai pi.