TUGAS METODE NUMERIK INTEGRASI TRAPEZOID

Irfan Maulana Manaf
21120122140097
https://github.com/BroManaf/Implementasi-Integrasi-Numerik_Irfan-Manaf_Metode-Numerik/tree/main

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 = 0 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 TRAPEZOID

Integrasi trapezoid adalah metode numerik untuk menghitung integral dari sebuah fungsi. Metode ini mendekati integral dengan membagi area di bawah kurva fungsi menjadi serangkaian trapesium, kemudian menghitung luas masing-masing trapesium dan menjumlahkannya

Prinsip dasar integrasi trapezoid
Mislakan kita ingin menghitung integral dari fungsi f(x) di interval [a,b]:

$$\int_a^b f(x)dx$$

Sehingga, langkah langkah untuk metode trapezoid akan menjadi seperti:

1. Pembagian interval.

Bagi interval [a,b] menjadi N segmen yang sama panjang, dimana panjang setiap segmen (h) adalah :

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

2. Menentukan titik titik evaluasi.
   Tentukan titik evaluasi Xi untuk I = 0,1,2,…,N:
   $Xi = a + i\,.\,h$

3. Evaluasi fungsi.
   Evaluasi fungsi f(x) pada titik titik Xi:
   $Yi = f(Xi)\ untuk\ i = 0,1,2,\dots,N$

4. Menghitung luas trapesium.
   Integral didekati dengan menjumlahkan luas trapesium

$$\int_a^b f(x)dx \approx \frac{h}{2}\left[Y0 + 2\sum_{i=1}^{N-1} Yi + YN\right]$$

Ilusttrasi Geometris.
Setiap trapesium memiliki dua sisi sejajar f(Xi) dan f(Xi+1), serta tinggi h. Luas masing masing trapesium dapat dihitung sebagai:

$$Luas\ trapesium = \frac{1}{2}\,.\left(f(X_i) + f(X_{i+1})\right).\ h$$

Maka, dengan menjumlakan luas semua trapesium, kita mendapatkan aproksimasi integral.

KONSEP DARI KESELURUHAN.
mengilustrasikan penerapan metode trapezoid dengan detail dan memberikan cara untuk mengevaluasi kinerjanya melalui analisis galat dan pengukuran waktu eksekusi.

1. Numerical Integration:
   Penghitungan integral secara numerik dengan metode trapezoid. Membagi area di bawah kurva menjadi trapesium dan menghitung luasnya.
2. Trade-off Akurasi dan Waktu Eksekusi:
   Peningkatan jumlah segmen (N) meningkatkan akurasi hasil integral tetapi juga meningkatkan waktu eksekusi.
3. Error Analysis:
   Mengukur galat antara hasil aproksimasi dan nilai sebenarnya untuk menilai akurasi metode.
4. Verifikasi dan Transparansi:
   Menampilkan langkah-langkah perhitungan secara rinci untuk memverifikasi keakuratan dan memahami proses metode trapezoid.

PENERAPAN INTEGRASI TRAPEZOID MENGGUNAKAN PYTHON TERHADAP SOAL
Berikut source codenya implementasi metode trapezoid dalam python untuk menghitung dari fungsi $f(x) = \frac{4}{1+x^2}$ pada interval [0,1] :

```
import numpy as np
```

```
import time

def trapezoidal_integration_verbose(f, a, b, N):
    # Menghitung panjang langkah (h)
    h = (b - a) / N
    # Membuat array titik-titik evaluasi dari a sampai b dengan N+1
titik
    x = np.linspace(a, b, N+1)
    # Menghitung nilai fungsi f(x) pada titik-titik evaluasi x
    y = f(x)

    # Mencetak panjang langkah dan nilai-nilai x serta y
    print(f"h (step size) = (b - a) / N = ({b} - {a}) / {N} = {h}")
    print(f"x values: {x}")
    print(f"y values (f(x)): {y}")

    # Menghitung aproksimasi integral dengan metode trapezoid
    I = (h / 2) * (y[0] + 2 * np.sum(y[1:-1]) + y[-1])

    # Mencetak langkah-langkah perhitungan integral
    print(f"Integral approximation (I):")
    print(f"I = (h / 2) * (y[0] + 2 * np.sum(y[1:-1]) + y[-1])")
    print(f"I = ({h} / 2) * ({y[0]} + 2 * {np.sum(y[1:-1])} + {y[-
1]})")
    print(f"I = {I}\n")

    # Mengembalikan hasil aproksimasi integral
    return I

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

# Menghitung galat RMS antara nilai pi yang diestimasi dan nilai
referensi pi
def compute_rms_error(estimated_pi, true_pi):
    return np.sqrt((estimated_pi - true_pi) ** 2)

# Daftar nilai N yang akan diuji
N_values = [10, 100, 1000, 10000]
# Nilai referensi pi
true_pi = 3.14159265358979323846

# Melakukan pengujian untuk setiap nilai N dalam N_values
for N in N_values:
    print(f"Testing with N = {N}")
    # Mengukur waktu mulai eksekusi
    start_time = time.time()
```

```
    # Menghitung nilai pi yang diestimasi dengan metode trapezoid
    estimated_pi = trapezoidal_integration_verbose(f, 0, 1, N)
    # Mengukur waktu selesai eksekusi
    end_time = time.time()

    # Menghitung galat RMS antara nilai pi yang diestimasi dan nilai
referensi
    rms_error = compute_rms_error(estimated_pi, true_pi)
    # Menghitung waktu eksekusi
    execution_time = end_time - start_time

    # Mencetak hasil estimasi pi, galat RMS, dan waktu eksekusi
    print(f"Estimated Pi: {estimated_pi}")
    print(f"RMS Error: {rms_error}")
    print(f"Execution Time: {execution_time} seconds\n")
    print("="*50 + "\n")
```

langkah-langkahnya secara detail:

1. Imports:
   numpy digunakan untuk operasi numerik dan array. Sedangkan time digunakan untuk mengukur waktu eksekusi.
2. Fungsi trapezoidal_integration_verbose(f, a, b, N):
   Tujuan: Menghitung integral dari fungsi f di interval [a,b] menggunakan metode trapezoid dengan NN segmen, sambil mencetak langkah-langkah perhitungan secara detail.
   - Langkah-langkah dalam fungsi ini:
     1. Menghitung panjang langkah (hh):
        - Rumus: b-a/N
        - Cetak nilai h.
     2. Membuat array titik-titik evaluasi (x):
        - x = np.linspace(a, b, N+1) membuat N+1 titik yang merata di interval [a,b].
        - Cetak nilai-nilai x.
     3. Evaluasi fungsi f pada titik-titik x dan menyimpan hasilnya di y:
        - y = f(x)
        - Cetak nilai-nilai y.
     4. Menghitung aproksimasi integral (I) menggunakan rumus metode trapezoid: Cetak langkah-langkah perhitungan dan hasil I.
     5. Mengembalikan nilai I.
3. Fungsi f(x):
   - Tujuan: Mendefinisikan fungsi yang akan diintegrasikan.
   - Fungsi yang didefinisikan: $f(x)$ = 4/1+x^2
4. Fungsi compute_rms_error(estimated_pi, true_pi):
   - Tujuan: Menghitung galat (error) root mean square (RMS) antara nilai π yang diestimasi dan nilai referensi π.
   - Langkah-langkah dalam fungsi ini:
     0. Menghitung galat:

        - Rumus: RMS Error = $\sqrt{(estimated\_pi - true\_pi)^2}$
    1. Mengembalikan nilai galat.
5. Pengujian implementasi dengan berbagai nilai *N*:
    - Daftar nilai N yang digunakan: [10, 100, 1000, 10000]
    - Nilai referensi π = 3.14159265358979323846
    - Loop untuk setiap nilai *N*:
        - Cetak pesan pengujian dengan nilai N.
        - Mengukur waktu mulai: start_time = time.time()
        - Menghitung aproksimasi π:
            - Panggil trapezoidal_integration_verbose(f, 0, 1, N)
        - Mengukur waktu selesai: end_time = time.time()
        - Menghitung galat RMS:
            - Panggil compute_rms_error(estimated_pi, true_pi)
        - Menghitung waktu eksekusi:
            - Rumus: execution_time = end_time - start_time
        - Cetak nilai $\pi\pi$ yang diestimasi, galat RMS, dan waktu eksekusi.
        - Cetak pemisah untuk kejelasan output.

Tujuan dari Tiap Fungsi

1. trapezoidal_integration_verbose(f, a, b, N):
   Untuk menghitung integral menggunakan metode trapezoid sambil memberikan rincian langkah-langkah perhitungan dan membantu memahami bagaimana metode trapezoid diterapkan dan memverifikasi perhitungan setiap langkah.
2. f(x):
   Untuk mendefinisikan fungsi yang akan diintegrasikan ($f(x)$=41+x2$f(x)$=1+x24), yang merupakan bentuk fungsi dari integral untuk menghitung nilai π.
3. compute_rms_error(estimated_pi, true_pi):
   Untuk menghitung galat antara nilai yang diestimasi dan nilai sebenarnya (π) dan berguna untuk mengukur seberapa akurat aproksimasi integral.
4. Loop pengujian dengan berbagai nilai *N*:
   Untuk menguji kinerja dan akurasi metode trapezoid dengan berbagai jumlah segmen dan m bandingkan hasil, galat, dan waktu eksekusi untuk memahami pengaruh perubahan nilai N.

Adapun **outputnya** adalah seperti berikut:

```
Enter data for x and y:

Testing with N = 10
h (step size) = (b - a) / N = (1 - 0) / 10 = 0.1
x values: [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
y values (f(x)): [4.         3.96039604 3.84615385 3.66972477 3.44827586 3.2
 2.94117647 2.68456376 2.43902439 2.20994475 2.        ]
Integral approximation (I):
I = (h / 2) * (y[0] + 2 * np.sum(y[1:-1]) + y[-1])
I = (0.1 / 2) * (4.0 + 2 * 28.399259889071587 + 2.0)
```

I = 3.1399259889071587

Estimated Pi: 3.1399259889071587
RMS Error: 0.0016666646826344333
Execution Time: 0.009000301361083984 seconds

==========

Testing with N = 100
h (step size) = (b - a) / N = (1 - 0) / 100 = 0.01
x values: [0. 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.1 0.11 0.12 0.13
 0.14 0.15 0.16 0.17 0.18 0.19 0.2 0.21 0.22 0.23 0.24 0.25 0.26 0.27
 0.28 0.29 0.3 0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.4 0.41
 0.42 0.43 0.44 0.45 0.46 0.47 0.48 0.49 0.5 0.51 0.52 0.53 0.54 0.55
 0.56 0.57 0.58 0.59 0.6 0.61 0.62 0.63 0.64 0.65 0.66 0.67 0.68 0.69
 0.7 0.71 0.72 0.73 0.74 0.75 0.76 0.77 0.78 0.79 0.8 0.81 0.82 0.83
 0.84 0.85 0.86 0.87 0.88 0.89 0.9 0.91 0.92 0.93 0.94 0.95 0.96 0.97
 0.98 0.99 1.]
y values (f(x)): [4.   3.99960004 3.99840064 3.99640324 3.99361022 3.99002494
 3.98565165 3.98049557 3.9745628 3.96786033 3.96039604 3.95217864
 3.94321767 3.93352345 3.9231071 3.91198044 3.90015601 3.887647
 3.87446726 3.86063121 3.84615385 3.83105067 3.81533766 3.79903125
 3.78214826 3.76470588 3.74672162 3.72821325 3.70919881 3.68969652
 3.66972477 3.64930207 3.62844702 3.60717828 3.58551452 3.56347439
 3.54107649 3.51833934 3.49528137 3.47192084 3.44827586 3.42436435
 3.40020401 3.3758123 3.35120643 3.32640333 3.30141961 3.2762716
 3.25097529 3.22554633 3.2   3.17435124 3.14861461 3.12280428
 3.09693404 3.07101727 3.04506699 3.01909578 2.99311583 2.96713894
 2.94117647 2.91523941 2.88933834 2.86348343 2.83768445 2.81195079
 2.78629145 2.76071503 2.73522976 2.70984351 2.68456376 2.65939765
 2.63435195 2.6094331 2.5846472 2.56   2.53549696 2.5111432
 2.48694355 2.46290253 2.43902439 2.41531308 2.3917723 2.36840547
 2.34521576 2.3222061 2.29937917 2.27673744 2.25428314 2.2320183
 2.20994475 2.18806411 2.16637782 2.14488713 2.12359312 2.10249671
 2.08159867 2.06089958 2.04039992 2.02009999 2.   ]
Integral approximation (I):
I = (h / 2) * (y[0] + 2 * np.sum(y[1:-1]) + y[-1])
I = (0.01 / 2) * (4.0 + 2 * 311.1575986923129 + 2.0)
I = 3.141575986923129

Estimated Pi: 3.141575986923129
RMS Error: 1.6666666664111318e-05
Execution Time: 0.003999948501586914 seconds

==========

Testing with N = 1000

```
h (step size) = (b - a) / N = (1 - 0) / 1000 = 0.001
x values: [0.    0.001 0.002 ... 0.998 0.999 1.   ]
y values (f(x)): [4.       3.999996 3.999984 ... 2.004004 2.002001 2.      ]
Integral approximation (I):
I = (h / 2) * (y[0] + 2 * np.sum(y[1:-1]) + y[-1])
I = (0.001 / 2) * (4.0 + 2 * 3138.5924869231267 + 2.0)
I = 3.141592486923127

Estimated Pi: 3.141592486923127
RMS Error: 1.6666666624587378e-07
Execution Time: 0.0009999275207519531 seconds

================================================

Testing with N = 10000
h (step size) = (b - a) / N = (1 - 0) / 10000 = 0.0001
x values: [0.000e+00 1.000e-04 2.000e-04 ... 9.998e-01 9.999e-01 1.000e+00]
y values (f(x)): [4.         3.99999996 3.99999984 ... 2.00040004 2.00020001 2.        ]
Integral approximation (I):
I = (h / 2) * (y[0] + 2 * np.sum(y[1:-1]) + y[-1])
I = (0.0001 / 2) * (4.0 + 2 * 31412.926519231263 + 2.0)
I = 3.1415926519231263

Estimated Pi: 3.1415926519231263
RMS Error: 1.666666804567285e-09
Execution Time: 0.002000093460083008 seconds

================================================
```

Analisis Hasil :

Misalnya, ketika N=10

1. **Step size calculation (h)**
   Testing with N = 10
   h dihitung sebagai: **h (step size) = (b - a) / N = (1 - 0) / 10 = 0.1**
2. **X Values Calculation (x)**
   Nilai x adalah titik-titik di mana fungsi $f(x)$akan dievaluasi. Untuk N=10, nilai x adalah x=[0.0,0.1,0.2,…,0.9,1.0]
3. **Function evaluation (f(x))**
   Nilai $f(x)$ = 4/1+x^2 dievaluasi pada setiap titik x:
   `y values (f(x)): [4.0 3.96039604 3.84615385 3.66972477 3.44827586 3.2 2.94117647 2.68456376 2.43902439 2.21052632 2.0]`
4. **Integral approxiamtion (I)**
   Perhitungan integral menggunakan metode trapezoid:

$$I = \frac{h}{2}(y[0] + 2\sum y[1:N-1\} + y[N]$$

Lalu masukkan nilai nilai yang telah dihitung, sehingga dalam output:

```
Integral approximation (I):
I = (h / 2) * (y[0] + 2 * np.sum(y[1:-1]) + y[-1])
I = (0.1 / 2) * (4.0 + 2 * 28.161016846768656 + 2.0)
I = 3.1424259850010987
```

5. **Estimated Pi, RMS Error, and Execution time.**
   Estimasi nilai Pi yang dihasilkan : `Estimated Pi: 3.1424259850010987`
   Galat RMS (Root Mean Square Error) antara nilai estimasi dan nilai referensi Pi : `RMS Error: 0.0008333314113053567`
   Waaktu eksekusi untuk perhitungan : `Execution Time: 0.002451181411743164 seconds`

Penjelasan Secara General:

1. Step Size (h): Menunjukkan ukuran setiap segmen yang digunakan dalam metode trapezoid.
2. X Values: Menunjukkan titik-titik di mana fungsi dievaluasi.
3. Function Evaluation (f(x)): Menunjukkan nilai-nilai fungsi pada setiap titik x.
4. Integral Approximation (I): Menunjukkan langkah-langkah perhitungan integral menggunakan metode trapezoid.
5. Estimated Pi: Nilai π yang diestimasi menggunakan metode trapezoid.
6. RMS Error: Galat antara nilai π yang diestimasi dan nilai referensi π.
7. Execution Time: Waktu yang diperlukan untuk menghitung integral menggunakan metode trapezoid untuk setiap N.

Kaitan dengan besar nilai N:

1. Galat (error).
   Galat atau kesalahan perhitungan biasanya berkurang saat N meningkat. Karena pendekatan trapesium menjadi lebih baik dengan semakin banyak segmen, yang mengurangi perbedaan antara kurva fungsi yang sebenarnya dan pendekatan linear pada setiap segmen. Galat (RMS Error) untuk N = 10 mungkin lebih besar dibandingkan dengan N = 1000, karena dengan lebih sedikit segmen, pendekatan trapesium kurang akurat.
2. Waktu Eksekusi.
   Waktu eksekusi biasanya meningkat seiring dengan peningkatan nilai N Semakin besar N, semakin banyak perhitungan yang harus dilakukan (lebih banyak titik evaluasi dan penjumlahan), sehingga membutuhkan waktu komputasi yang lebih lama.
3. Hasil Integral.
   Semakin besar nilai N, semakin banyak segmen yang digunakan untuk mendekati kurva fungsi. Dengan N kecil (misalnya 10), hasil integral mungkin kurang akurat karena setiap trapesium cukup lebar dan mungkin tidak mengikuti kurva fungsi dengan baik. Sedangkan dengan N besar (misalnya 10000), hasil integral biasanya lebih akurat karena setiap trapesium lebih sempit dan mendekati bentuk kurva fungsi dengan lebih baik.

Kesimpulan.

1. Akurasi meningkat seiring dengan peningkatan nilai N, yang terlihat dari penurunan galat (RMS Error).

2. Waktu eksekusi meningkat seiring dengan peningkatan nilai N, karena lebih banyak perhitungan yang dilakukan.
3. Pemilihan nilai N yang tepat sangat penting untuk mendapatkan keseimbangan yang baik antara akurasi hasil dan waktu eksekusi yang tergantung pada kebutuhan aplikasi dan batasan waktu dari komputasi