PROYEK METODE NUMERIS

INTEGRASI NUMERIS DENGAN METODE EXTRAPOLATION, ROMBERG INTEGRATION, ADAPTIVE INTEGRATION, DAN GAUSSIAN QUADRATURE

Disusun oleh:

1. Adnan Abdul Majid (24/544058/TK/60471)
2. Muhammad Ilkham Abdillah (24/537977/TK/59653)
3. Rafif Raihan Bahrul Alam (24/534432/TK/59237)
4. Rakan Hendian Ramadhan (24/540158/TK/59909)

Departemen Teknik Elektro dan Teknologi Informasi

Universitas Gadjah Mada

Daftar Isi

1 Pengantar

Integrasi numerik adalah proses fundamental dalam sains dan rekayasa untuk mengestimasi nilai integral tertentu $\int_a^b f(x)\,dx$ ketika solusi analitik (solusi eksak) sulit atau tidak mungkin ditemukan. Terdapat berbagai metode untuk melakukan estimasi ini, masing-masing dengan kelebihan, kekurangan, dan tingkat akurasi yang berbeda. Dalam laporan ini, kita akan menganalisis dan membandingkan empat metode integrasi numerik yang populer:

1. **Aturan Trapesium (Trapezoidal Rule)** sebagai metode dasar, dan **Ekstrapolasi Richardson** sebagai teknik untuk meningkatkan akurasinya.
2. **Integrasi Romberg**, yang merupakan aplikasi sistematis dari Ekstrapolasi Richardson pada Aturan Trapesium.
3. **Integrasi Adaptif (Adaptive Quadrature)**, sebuah metode cerdas yang menyesuaikan ukuran langkahnya secara dinamis untuk efisiensi.
4. **Kuadratur Gauss (Gaussian Quadrature)**, metode yang sangat akurat dengan memilih titik-titik evaluasi (nodes) secara optimal.

Setiap metode akan diuji menggunakan dua fungsi:

- $f(x) = \cos(x)$ pada interval $[0, \pi/2]$, dengan nilai sejati $\int_0^{\pi/2} \cos(x)\,dx = 1.0$.
- $f(x) = x^2$ pada interval $[0, 1]$, dengan nilai sejati $\int_0^1 x^2\,dx = 1/3 \approx 0.333333...$

Kita akan mengevaluasi setiap metode berdasarkan pengertian, rumus, implementasi kode (sesuai file yang dilampirkan), hasil perhitungan, dan analisis error.

2 Metode 1: Aturan Trapesium & Ekstrapolasi Richardson

2.1 Pengertian dan Rumus

Aturan Trapesium adalah metode integrasi paling dasar yang mengaproksimasi area di bawah kurva $f(x)$ dengan membaginya menjadi sejumlah trapesium.

- **Aturan Trapesium Komposit** untuk m interval dengan lebar $h = (b-a)/m$:

$$T(h) = \frac{h}{2}\left[f(a) + 2\sum_{i=1}^{m-1} f(x_i) + f(b)\right]$$

Ekstrapolasi Richardson adalah teknik umum untuk meningkatkan akurasi dari sebuah estimasi numerik. Jika kita memiliki dua estimasi, $I(h_1)$ dan $I(h_2)$ (misalnya $I(h)$ dan $I(h/2)$), kita dapat menggabungkannya untuk "membatalkan" suku error utama. Untuk Aturan Trapesium, yang memiliki error $O(h^2)$, rumusnya adalah:

$$I_{\text{estimasi}} \approx I(h/2) + \frac{I(h/2) - I(h)}{2^2 - 1} = \frac{4I(h/2) - I(h)}{3}$$

Hasil dari ekstrapolasi ini setara dengan Aturan Simpson.

2.2 Implementasi Kode

Kode berikut mengimplementasikan Aturan Trapesium Komposit untuk kedua fungsi.

```python
import math
PI = math.pi

def trapezoidal_1(n):
    h = (PI/2)/(n)
    sum = 0
    for i in range(0,n+1):
        if(i==0 or i==n):
            sum += math.cos(i*h)
        else:
            sum += math.cos(i*h) * 2
    return sum * h/2

def trapezoidal_2(n):
    h = 1/(n)
    sum = 0
    for i in range(0,n+1):
        if(i==0 or i==n):
            sum += (i*h)**2
        else:
            sum += (i*h)**2 * 2
    return sum * h/2
```

Listing 1: Implementasi metode Trapezoidal

Kode berikut mengimplementasikan Ekstrapolasi Richardson, yang menggunakan fungsi-fungsi dari. Parameter p adalah orde dari error (untuk Trapesium, $p = 2$) dan n adalah jumlah interval awal.

```python
from trapezoidal import *

def richardson_extrapolation_1(p, n):
    return ((2**p * trapezoidal_1(n*2) - trapezoidal_1(n))/(2**p - 1))
```

```
def richardson_extrapolation_2(p, n):
    return ((2**p * trapezoidal_2(n*2) - trapezoidal_2(n))/(2**p - 1))
```

Listing 2: Implementasi metode Ekstrapolasi Richardson

2.3 Hasil Perhitungan dan Analisis Error

Hasil dari kode di atas untuk $m = 1, 2, 4$ (yang setara dengan $R(0,0), R(1,0), R(2,0)$ dalam Romberg) adalah:

Tabel 1: Hasil dan Error Aturan Trapesium

	$\int_0^{\pi/2} \cos(x)\,dx$ (Sejati: 1.0)		$\int_0^1 x^2\,dx$ (Sejati: 1/3)	
m (Interval)	Hasil Trapesium	Error Absolut	Hasil Trapesium	Error Absolut
1 ($R(0,0)$)	0.785398	0.214602	0.500000	0.166667
2 ($R(1,0)$)	0.948059	0.051941	0.375000	0.041667
4 ($R(2,0)$)	0.987116	0.012884	0.343750	0.010417

Analisis (Aturan Trapesium): Error dari Aturan Trapesium cukup besar, tetapi berkurang sekitar faktor 4 setiap kali jumlah interval dikalikan dua (sesuai dengan $O(h^2)$).

Tabel 2: Hasil dan Error Ekstrapolasi Richardson ($p = 2$)

	$\int_0^{\pi/2} \cos(x)\,dx$ (Sejati: 1.0)		$\int_0^1 x^2\,dx$ (Sejati: 1/3)	
n (Basis)	Hasil Ekstrapolasi	Error Absolut	Hasil Ekstrapolasi	Error Absolut
$n = 1$ (T_1, T_2)	1.002279	2.28×10^{-3}	0.333333...	0.0
$n = 2$ (T_2, T_4)	1.000135	1.35×10^{-4}	0.333333...	0.0

Analisis (Ekstrapolasi Richardson): Dengan menerapkan Ekstrapolasi Richardson (setara dengan nilai $R(1,1)$ dan $R(2,1)$ dari tabel Romberg), akurasi meningkat secara drastis dibandingkan Trapesium murni.

- $\cos(x)$**:** Error berkurang dari 1.29×10^{-2} (Trapesium $m = 4$) menjadi 1.35×10^{-4} (Ekstrapolasi $n = 2$). Ini adalah peningkatan akurasi yang signifikan hanya dengan menggabungkan hasil yang sudah ada.
- x^2**:** Metode ini langsung memberikan hasil **eksak** (Error $= 0$). Ini karena ekstrapolasi orde pertama pada Aturan Trapesium ($p = 2$) secara matematis identik dengan Aturan Simpson, yang diketahui eksak untuk polinomial derajat ≤ 3.

3 Metode 2: Integrasi Romberg

3.1 Pengertian dan Rumus

Integrasi Romberg adalah aplikasi rekursif dan sistematis dari Ekstrapolasi Richardson pada Aturan Trapesium. Metode ini membangun tabel $R(i, j)$ di mana:

- **Kolom Pertama** $R(i, 0)$**:** Adalah hasil Aturan Trapesium dengan $m = 2^i$ interval.
$$R(i, 0) = \text{trapezoidal\_rule}(2^i)$$
- **Kolom Berikutnya** $R(i, j)$**:** Adalah hasil ekstrapolasi dari kolom sebelumnya.
$$R(i, j) = \frac{4^j R(i, j-1) - R(i-1, j-1)}{4^j - 1}$$

Nilai $R(i, 1)$ setara dengan Aturan Simpson, $R(i, 2)$ setara dengan Aturan Boole, dst. Perkiraan terbaik adalah nilai di diagonal, $R(n, n)$.

3.2 Implementasi Kode

Kode berikut mengimplementasikan fungsi Romberg secara rekursif dan membangun tabel 3×3 (hingga $R(2, 2)$).

```python
from trapezoidal import *

def romberg_1(m, n):
    if(n==0):
        return trapezoidal_1(2**m)
    return (4**n * romberg_1(m, n-1) - romberg_1(m-1, n-1)) / (4**n -
    1)

def romberg_2(m, n):
    if(n==0):
        return trapezoidal_2(2**m)
    return (4**n * romberg_2(m, n-1) - romberg_2(m-1, n-1)) / (4**n -
    1)
```

Listing 3: Implementasi metode Integrasi Romberg

3.3 Hasil Perhitungan dan Analisis Error

Output dari kode di atas menghasilkan tabel berikut (hingga $R(2, 2)$):

Tabel Romberg untuk $\int_0^{\pi/2} \cos(x)\, dx$ (Sejati: 1.0)

i	$j = 0$ (Trapesium)	$j = 1$ (Simpson)	$j = 2$ (Boole)
0	0.785398		
1	0.948059	1.002279	
2	0.987116	1.000135	0.999992

Tabel Romberg untuk $\int_0^1 x^2\, dx$ (Sejati: $1/3$)

i	$j = 0$ (Trapesium)	$j = 1$ (Simpson)	$j = 2$ (Boole)
0	0.500000		
1	0.375000	0.333333...	
2	0.343750	0.333333...	0.333333...

Analisis:

- $\cos(x)$**:** Perkiraan terbaik $R(2,2) = 0.999992$. Error absolutnya adalah $|1.0 - 0.999992| = 8.0 \times 10^{-6}$. Ini adalah peningkatan akurasi yang sangat signifikan dibandingkan Trapesium $R(2,0)$ (Error 0.0128).
- x^2**:** Metode ini memberikan hasil **eksak** $1/3$ (Error $= 0$) sudah pada $R(1,1)$. Ini karena $R(i,1)$ setara dengan Aturan Simpson, yang secara definisi eksak untuk polinomial derajat ≤ 3.

4 Metode 3: Integrasi Adaptif (Adaptive Quadrature)

4.1 Pengertian dan Rumus

Integrasi Adaptif adalah metode yang cerdas. Alih-alih menggunakan ukuran langkah h yang tetap di seluruh interval, metode ini "mengadaptasi" ukuran langkahnya.

- Ia menggunakan langkah besar (hemat komputasi) di area di mana fungsi $f(x)$ relatif datar.
- Ia menggunakan langkah kecil (perhitungan lebih intensif) di area di mana fungsi $f(x)$ berfluktuasi tajam.

Kode yang dilampirkan menggunakan Aturan Simpson sebagai basis. Ia membandingkan estimasi Simpson pada satu interval besar (I_1) dengan jumlah dua estimasi Simpson pada dua sub-interval (I_2). Jika $|I_2 - I_1|$ lebih besar dari toleransi (tol), interval dibagi dua dan proses diulang secara rekursif. Rumus estimasi errornya adalah $E \approx |I_2 - I_1|$. Perkiraan integral yang lebih baik (koreksi orde ke-4) adalah $I = I_2 + (I_2 - I_1)/15$.

4.2 Implementasi Kode

Kode berikut mengimplementasikan metode adaptif berbasis Simpson.

```python
import math
PI = math.pi

def quadpt_1(a, b):
    tol = 1e-10
    c = (a+b)/2
    fa = math.cos(a)
    fb = math.cos(b)
    fc = math.cos(c)
    I, E = qstep_1(a, b, tol, fa, fc, fb)
    return I, E

def qstep_1(a, b, tol, fa, fc, fb):
    h1 = (b - a)/2
    h2 = h1/2
    c = (a + b)/2
    fd = math.cos((a + c)/2)
    fe = math.cos((c + b)/2)
    I1 = h1/3 * (fa + 4*fc + fb)
    I2 = h2/3 * (fa + 4*fd + 2*fc + 4*fe + fb)

    E = abs(I2 - I1)  # estimated local error

    if E < tol:
        I = I2 + (I2 - I1) / 15
        return I, E
    else:
        Ia, Ea = qstep_1(a, c, tol / 2, fa, fd, fc)
        Ib, Eb = qstep_1(c, b, tol / 2, fc, fe, fb)
        return Ia + Ib, Ea + Eb

def quadpt_2(a, b):
    tol = 1e-10
    c = (a+b)/2
    fa = a**2
```

```python
    fb = b**2
    fc = c**2
    I, E = qstep_2(a, b, tol, fa, fc, fb)
    return I, E

def qstep_2(a, b, tol, fa, fc, fb):
    h1 = (b - a)/2
    h2 = h1/2
    c = (a + b)/2
    fd = ((a + c)/2)**2
    fe = ((c + b)/2)**2
    I1 = h1/3 * (fa + 4*fc + fb)
    I2 = h2/3 * (fa + 4*fd + 2*fc + 4*fe + fb)

    E = abs(I2 - I1)  # estimated local error

    if E < tol:
        I = I2 + (I2 - I1) / 15
        return I, E
    else:
        Ia, Ea = qstep_2(a, c, tol / 2, fa, fd, fc)
        Ib, Eb = qstep_2(c, b, tol / 2, fc, fe, fb)
        return Ia + Ib, Ea + Eb
```

Listing 4: Implementasi metode Adaptif Integral

4.3 Hasil Perhitungan dan Analisis Error

Output dari kode di atas (dengan tol $= 10^{-10}$):

Hasil untuk $\int_0^{\pi/2} \cos(x)\,dx$:

```
Aproximation first integral: 0.9999999999999999
Estimated local error: 2.669869663893265e-11
```

Hasil untuk $\int_0^1 x^2\,dx$:

```
Aproximation first integral: 0.3333333333333333
Estimated local error: 0.0
```

Analisis:

- **$\cos(x)$:** Metode ini mencapai nilai **1.0**, yang sangat akurat. Error aktualnya (terbatas oleh presisi float) jauh lebih kecil dari error Romberg $R(2,2)$, dan dicapai secara efisien dengan berhenti ketika toleransi terpenuhi.
- **x^2:** Metode ini memberikan hasil **eksak** $1/3$ dan estimasi error 0.0. Ini karena basisnya adalah Aturan Simpson, yang (seperti disebutkan) eksak untuk polinomial derajat ≤ 3.

5 Metode 4: Kuadratur Gauss (Gaussian Quadrature)

5.1 Pengertian dan Rumus

Kuadratur Gauss (secara spesifik Gauss-Legendre) adalah metode yang sangat berbeda dan kuat. Alih-alih menggunakan titik-titik yang berjarak sama (seperti Trapesium/Simpson), metode ini secara cerdas memilih n titik (disebut **nodes**, x_i) dan **bobot** (w_i) yang optimal.

Implementasi Kuadratur Gauss praktisnya menggunakan *library* seperti NumPy karena metode ini fundamentalnya bergantung pada sekumpulan "nodes" (titik x_i) dan "weights" (bobot w_i) yang optimal, yang merupakan akar-akar dari Polinomial Legendre. Menghitung nilai-nilai ini dari awal adalah tugas matematika yang sangat kompleks dan intensif secara komputasi, membutuhkan algoritma pencarian akar numerik yang canggih. Namun, dalam percobaan ini, Kuadratur Gauss hanya diimplementasikan hingga $n = 4$, sehingga kita dapat menggunakan nilai bobot dan titik yang mendasar saja. Berikut adalah nilai titik x_i dan bobot w_i untuk Kuadratur Gauss hingga $n = 4$:

Tabel 3: Titik dan Bobot pada Kuadratur Gauss

n	x_i	w_i
1	0.0	2.0
2	±0.5773502692	1.0
3	±0.7745966692	0.5555555556
	0.0	0.8888888889
4	±0.8611363116	0.3478548451
	±0.3399810436	0.6521451549

- Titik-titik ini adalah akar dari Polinomial Legendre.
- Keajaibannya adalah: n-titik Kuadratur Gauss dapat mengintegrasikan polinomial derajat $\leq 2n - 1$ secara **eksak**.

Rumus dasarnya adalah untuk interval $[-1, 1]$:

$$\int_{-1}^{1} g(x)\,dx \approx \sum_{i=1}^{n} w_i g(x_i)$$

Untuk mengubah interval $[a, b]$ ke $[-1, 1]$, kita gunakan transformasi:

$$t = \frac{x+1}{2}(b-a) + a \quad \Longrightarrow \quad \int_{a}^{b} f(t)\,dt \approx \frac{b-a}{2} \sum_{i=1}^{n} w_i f(t_i)$$

5.2 Implementasi Kode

Berikut adalah kode implementasi Kuadratur Gauss menggunakan data x_i dan w_i dari tabel 3 sebagai parameter.

```python
import math
PI = math.pi

f = lambda x: math.cos(x)
g = lambda x: pow(x, 2)

def gauss(f, a, b, n):
    data = {
        1: ([0.0], [2.0]),
        2: ([-0.5773502692, 0.5773502692], [1.0, 1.0]),
        3: ([0.0, -0.7745966692, 0.7745966692], [0.8888889, 0.5555556, 0.5555556]),
        4: ([-0.8611363116, -0.3399810436, 0.3399810436, 0.8611363116], [0.3478548451, 0.6521451549, 0.6521451549, 0.3478548451]),
    }
    x, w = data[n]
    total = 0
    for i in range(n):
        xi = 0.5 * (b - a) * x[i] + 0.5 * (a + b)
        total += w[i] * f(xi)
    return 0.5 * (b - a) * total
```

Listing 5: IImplementasi metode Kuadratur Gauss

Namun, memasukkan parameter secara manual bukanlah praktik yang standar dan efisien. Dalam skenario asli, akan lebih baik jika komputasi Kuadratur Gauss dilakukan menggunakan fungsi bawaan dari *library numpy* yang sudah termasuk parameter x_i dan w_i. Kode berikut mengimplementasikan Kuadratur Gauss menggunakan `numpy` untuk mendapatkan nilai x_i dan w_i.

```python
import numpy as np
import math
pi = math.pi

def gauss_1(a, b, n):
    x, w = np.polynomial.legendre.leggauss(n)
    t = (x+1)*(b-a)/2 + a
    return (b-a)*np.sum(w*np.cos(t))/2

def gauss_2(a, b, n):
    x, w = np.polynomial.legendre.leggauss(n)
    t = (x+1)*(b-a)/2 + a
    return (b-a)*np.sum(w*(t**2))/2
```

Listing 6: Implementasi metode Kuadratur Gauss dengan *library numpy*

5.3 Hasil Perhitungan dan Analisis Error

Hasil dari kode yang menggunakan *library numpy* lebih akurat. Hal ini mungkin terjadi karena nilai koefisien yang didapat dari algoritma di dalamnya lebih akurat daripada koefisien yang di atur manual. Oleh karena itu, analisis yang dilakukan adalah hasil output dari kode yang menggunakan *library numpy*. Berikut adalah output dari kode di tersebut dari $n = 1$ sampai $n = 5$ titik:

Tabel 4: Hasil dan Error Kuadratur Gauss

	$\int_0^{\pi/2} \cos(x)\, dx$ (Sejati: 1.0)		$\int_0^1 x^2\, dx$ (Sejati: 1/3)	
n (Titik)	Hasil Gauss	Error Absolut	Hasil Gauss	Error Absolut
1	0.998475	1.52×10^{-3}	0.250000	$\approx 8.33 \times 10^{-2}$
2	1.000003	3.00×10^{-6}	0.333333...	0.0
3	0.9999999996	3.31×10^{-10}	0.333333...	0.0
4	1.0000000000	≈ 0.0	0.333333...	0.0

Analisis:

- $\cos(x)$**:** Konvergensi sangat cepat. Hanya dengan $n = 3$ titik, error sudah $\approx 10^{-10}$. Dengan $n = 4$ titik, hasilnya sudah akurat hingga presisi mesin.
- x^2**:** Kuadratur Gauss $n = 1$ (setara Aturan Titik Tengah) memberikan hasil 0.25. Metode ini memberikan hasil **eksak** 1/3 (Error = 0) dimulai dari $n = 2$ titik, sesuai dengan teorinya ($2n - 1 = 3$).

6 Kesimpulan: Perbandingan Error

Mari kita rangkum perkiraan terbaik dari setiap metode untuk kedua fungsi dalam tabel perbandingan.

6.1 Perbandingan untuk $\int_0^{\pi/2} \cos(x)\, dx$ (Sejati: 1.0)

Metode	Perkiraan Terbaik	Error Absolut	Catatan
Trapesium	0.987116	$\approx 1.29 \times 10^{-2}$	$m = 4$ interval ($R(2, 0)$)
Ekstrapolasi	1.000135	$\approx 1.35 \times 10^{-4}$	Basis $m = 2$ & $m = 4$ ($R(2, 1)$)
Romberg	0.999992	$\approx 8.0 \times 10^{-6}$	$R(2, 2)$, (basis $m = 4$)
Adaptif	0.999999...	$\approx 2.67 \times 10^{-11}$	(Estimasi error oleh kode)
Gauss	1.000000...	$\approx 2.22 \times 10^{-16}$	$n = 4$ titik (Presisi mesin)

6.2 Perbandingan untuk $\int_0^1 x^2\, dx$ (Sejati: 1/3)

Metode	Perkiraan Terbaik	Error Absolut	Catatan
Trapesium	0.343750	$\approx 1.04 \times 10^{-2}$	$m = 4$ interval ($R(2, 0)$)
Ekstrapolasi	0.333333...	0.0	Eksak (setara Simpson)
Romberg	0.333333...	0.0	Eksak pada $R(1, 1)$
Adaptif	0.333333...	0.0	Eksak (basis Simpson)
Gauss	0.333333...	0.0	Eksak pada $n = 2$ titik

6.3 Metode dengan Error Terkecil

- **Untuk** $f(x) = x^2$ **(Polinomial Sederhana):** Metode **Ekstrapolasi Richardson**, **Romberg**, **Integrasi Adaptif**, dan **Kuadratur Gauss** ($n \geq 2$) semuanya menghasilkan error nol (0.0) dan memberikan jawaban eksak. Ini karena Ekstrapolasi Richardson (dan juga basis dari Romberg $R(i, 1)$ serta Integrasi Adaptif) secara matematis setara dengan Aturan Simpson, yang eksak untuk polinomial derajat ≤ 3. Kuadratur Gauss ($n = 2$) juga eksak untuk polinomial derajat ≤ 3.
- **Untuk** $f(x) = \cos(x)$ **(Fungsi Transendental):** Di sini kita melihat progresi akurasi yang jelas: Aturan Trapesium (10^{-2}), diikuti oleh Ekstrapolasi Richardson (10^{-4}), lalu Integrasi Romberg (10^{-6}). Namun, metode **Integrasi Adaptif** dan **Kuadratur Gauss** ($n \geq 4$) adalah pemenangnya, keduanya menghasilkan error terkecil yang pada dasarnya dibatasi oleh presisi floating-point komputer ($\approx 10^{-16}$). Integrasi Romberg jauh lebih baik daripada Trapesium dasar, tetapi akurasinya (untuk jumlah evaluasi fungsi yang sebanding) dikalahkan oleh efisiensi Integrasi Adaptif dan kekuatan Kuadratur Gauss.