UNIVERSITAS GADJAH MADA
FAKULTAS TEKNIK
DEPARTEMEN TEKNIK ELEKTRO DAN TEKNOLOGI INFORMASI

TUGAS METODE NUMERIS

INTEGRASI NUMERIK DENGAN METODE TRAPEZOIDAL, RICHARDSON, ROMBERG, ADAPTIVE, DAN GAUSSIAN QUADRATURE

Ditulis oleh:

Kelompok:

1. Nathanael Satya Saputra (NIM NAEL)
2. Muhammad Nafal Zakin Rustanto (24/535255/TK/59364)
3. Yohanes Anthony Saputra (NIM ANTHONY)
4. Johannes De Deo Dimas Aryobimo (24/540351/TK/59948)

BAGIAN 1: Dasar Teori

1.1. Integrasi Analitik (Metode Eksak)

Integrasi analitik adalah metode perhitungan integral menggunakan rumus-rumus kalkulus secara langsung. Metode ini memberikan nilai eksak (tepat) dari suatu integral jika fungsi yang diintegralkan memiliki antiturunan yang dapat ditentukan.

Definisi Integral Tentu:

Integral tentu dari fungsi $f(x)$ pada interval $[a, b]$ didefinisikan sebagai:

$$I = \int_a^b f(x)\,dx = F(b) - F(a)$$

di mana $F(x)$ adalah antiturunan dari $f(x)$, yaitu $F'(x) = f(x)$.

Teorema Fundamental Kalkulus:

Jika $f(x)$ kontinu pada interval $[a, b]$ dan $F(x)$ adalah antiturunan dari $f(x)$, maka:

$$\int_a^b f(x)\,dx = F(x)\Big|_a^b = F(b) - F(a)$$

Kegunaan:

Hasil dari integrasi analitik digunakan sebagai nilai pembanding (nilai eksak) untuk mengevaluasi akurasi metode-metode integrasi numerik. Error dari metode numerik dihitung sebagai selisih absolut antara hasil numerik dengan nilai eksak ini.

Algoritma:

```
1. Tentukan fungsi f(x) dan batas integrasi [a, b]
2. Cari antiturunan F(x) dari f(x)
3. Hitung F(b) - F(a)
4. Hasil adalah nilai eksak integral
```

Implementasi Python:

```python
import numpy as np

# Contoh untuk f(x) = cos(x) pada [0, pi/2]
a = 0
b = np.pi / 2
# Antiturunan cos(x) adalah sin(x)
exact = np.sin(b) - np.sin(a)   # = 1.0

# Contoh untuk f(x) = x^2 pada [0, 1]
a = 0
b = 1
# Antiturunan x^2 adalah x^3/3
exact = (b**3 / 3) - (a**3 / 3)   # = 1/3
```

1.2. Metode Trapezoidal Rule

Metode *Trapezoidal Rule* merupakan salah satu metode numerik untuk menghitung pendekatan integral tentu dari suatu fungsi yang sulit atau tidak dapat diintegralkan secara analitik. Ide dari metode ini adalah dengan membagi daerah di bawah kurva fungsi $f(x)$ pada interval $[a, b]$ menjadi sejumlah bagian kecil yang berbentuk trapesium, kemudian menjumlahkan luas seluruh trapesium tersebut untuk memperoleh nilai pendekatan dari integral.

Secara matematis, integral tentu dari $f(x)$ pada interval $[a, b]$ dapat ditulis sebagai:

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

Jika interval $[a, b]$ dibagi menjadi n subinterval yang sama panjang dengan lebar $h = \frac{b-a}{n}$, maka dengan pendekatan *Trapezoidal Rule* nilai integral yakni

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

dengan $x_0 = a$, $x_n = b$, dan $x_i = a + i \cdot h$ untuk $i = 1, 2, \ldots, n-1$.

Algoritma:

```
Input: fungsi f(x), batas a dan b, jumlah subinterval n
1. Hitung h = (b - a) / n
```

```
2. Buat array x dengan n+1 titik: x[i] = a + i*h
3. Hitung nilai fungsi y[i] = f(x[i]) untuk semua i
4. result = h * (0.5*y[0] + sum(y[1] sampai y[n-1]) + 0.5*y[n])
Output: result
```

Implementasi Python:

```python
import numpy as np

def trapezoidal_rule(f, a, b, n):
    """
    Metode Trapezoidal Rule
    f: fungsi yang akan diintegralkan
    a: batas bawah
    b: batas atas
    n: jumlah subinterval
    """
    h = (b - a) / n
    x = np.linspace(a, b, n + 1)
    y = f(x)

    result = h * (0.5 * y[0] + np.sum(y[1:-1]) + 0.5 * y[-1])
    return result

# Contoh penggunaan
def f(x):
    return np.cos(x)

result = trapezoidal_rule(f, 0, np.pi/2, 4)
```

1.3. Metode Richardson Extrapolation

Richardson Extrapolation adalah teknik numerik untuk meningkatkan akurasi hasil pendekatan dengan mengkombinasikan beberapa hasil pendekatan yang memiliki ukuran langkah berbeda. Metode ini sangat efektif untuk mempercepat konvergensi hasil numerik.

Prinsip Dasar:

Misalkan $N(h)$ adalah pendekatan numerik dari nilai eksak N dengan ukuran langkah h. Jika error dapat dinyatakan sebagai deret pangkat dari h:

$$N(h) = N + a_1 h^p + a_2 h^{2p} + a_3 h^{3p} + \cdots$$

Maka dengan menggunakan dua pendekatan dengan ukuran langkah berbeda (h dan $\frac{h}{2}$), kita dapat mengeliminasi suku error orde terendah.

Rumus Richardson Extrapolation:

Untuk integrasi numerik menggunakan Trapezoidal Rule, formula Richardson Extrapolation diberikan oleh:

$$R_{i,j} = R_{i,j-1} + \frac{R_{i,j-1} - R_{i-1,j-1}}{4^j - 1}$$

di mana:

- $R_{i,0}$ adalah hasil Trapezoidal Rule dengan $n = 2^i$ subinterval
- $R_{i,j}$ adalah hasil ekstrapolasi pada level i dan kolom j
- Setiap kolom j mengeliminasi error orde $O(h^{2j+2})$

Tabel Richardson:

Tabel Richardson memiliki struktur:

i	$j=0$	$j=1$	$j=2$	$j=3$
0	$R_{0,0}$			
1	$R_{1,0}$	$R_{1,1}$		
2	$R_{2,0}$	$R_{2,1}$	$R_{2,2}$	
3	$R_{3,0}$	$R_{3,1}$	$R_{3,2}$	$R_{3,3}$

Nilai pada diagonal utama ($R_{0,0}, R_{1,1}, R_{2,2}, \ldots$) menunjukkan peningkatan akurasi yang signifikan.

Algoritma:

```
Input: fungsi f(x), batas a dan b, max_level
1. Inisialisasi matriks R berukuran max_level x max_level
2. Untuk i = 0 sampai max_level-1:
   a. n = 2^i
   b. R[i][0] = trapezoidal_rule(f, a, b, n)
3. Untuk j = 1 sampai max_level-1:
   a. Untuk i = j sampai max_level-1:
      R[i][j] = R[i][j-1] + (R[i][j-1] - R[i-1][j-1])/(4^j - 1)
4. Hasil terbaik adalah R[max_level-1][max_level-1]
Output: matriks R
```

Implementasi Python:

```python
import numpy as np

def richardson_extrapolation(f, a, b, max_level=5):
    """
    Metode Richardson Extrapolation
    Menggunakan trapezoidal rule sebagai basis dan melakukan
    ekstrapolasi
    """
    R = np.zeros((max_level, max_level))

    # Kolom pertama: hasil trapezoidal rule dengan n = 2^i
    for i in range(max_level):
        n = 2**i
        R[i][0] = trapezoidal_rule(f, a, b, n)

    # Richardson extrapolation untuk kolom-kolom berikutnya
    for j in range(1, max_level):
        for i in range(j, max_level):
            R[i][j] = R[i][j-1] + (R[i][j-1] - R[i-1][j-1]) / (4**j
```

```
                 - 1)

    return R

# Contoh penggunaan
richardson = richardson_extrapolation(f, 0, np.pi/2, max_level=5)
result = richardson[4][4]  # Hasil terbaik
```

1.4. Metode Romberg Integration

Romberg Integration adalah metode integrasi numerik yang menggabungkan Trapezoidal Rule dengan Richardson Extrapolation. Perbedaan utama dengan Richardson standar adalah cara menghitung kolom pertama menggunakan teknik rekursif yang lebih efisien.

Rumus Romberg:

Kolom pertama dihitung dengan:

$$R_{0,0} = \frac{b-a}{2}[f(a) + f(b)]$$

$$R_{i,0} = \frac{1}{2}R_{i-1,0} + h_i \sum_{k=1}^{2^{i-1}} f(a + (2k-1)h_i)$$

di mana $h_i = \frac{b-a}{2^i}$.

Untuk kolom berikutnya, menggunakan Richardson Extrapolation:

$$R_{i,j} = R_{i,j-1} + \frac{R_{i,j-1} - R_{i-1,j-1}}{4^j - 1}$$

Algoritma:

```
Input: fungsi f(x), batas a dan b, max_level
1. Inisialisasi matriks R berukuran max_level x max_level
2. R[0][0] = 0.5 * (b - a) * (f(a) + f(b))
3. Untuk i = 1 sampai max_level-1:
   a. h = (b - a) / 2^i
   b. sum_val = 0
   c. Untuk k = 1 sampai 2^(i-1):
      sum_val += f(a + (2*k - 1) * h)
   d. R[i][0] = 0.5 * R[i-1][0] + h * sum_val
4. Untuk i = 1 sampai max_level-1:
   a. Untuk j = 1 sampai i:
      R[i][j] = R[i][j-1] + (R[i][j-1] - R[i-1][j-1])/(4^j - 1)
Output: matriks R
```

Implementasi Python:

```
import numpy as np

def romberg_integration(f, a, b, max_level=5):
```

```
    """
    Metode Romberg Integration
    Menghasilkan tabel Romberg R[i][j]
    """
    R = np.zeros((max_level, max_level))

    # R[0][0] menggunakan trapezoidal rule dengan 1 interval
    R[0][0] = 0.5 * (b - a) * (f(a) + f(b))

    for i in range(1, max_level):
        h = (b - a) / (2**i)

        # Hitung sum untuk titik-titik baru
        sum_val = 0
        for k in range(1, 2**(i-1) + 1):
            sum_val += f(a + (2*k - 1) * h)

        R[i][0] = 0.5 * R[i-1][0] + h * sum_val

        # Richardson extrapolation
        for j in range(1, i + 1):
            R[i][j] = R[i][j-1] + (R[i][j-1] - R[i-1][j-1]) / (4**j
                - 1)

    return R

# Contoh penggunaan
romberg = romberg_integration(f, 0, np.pi/2, max_level=5)
result = romberg[4][4]  # Hasil terbaik
```

1.5. Metode Adaptive Integration

Adaptive Integration adalah metode integrasi numerik yang secara otomatis menyesuaikan ukuran interval berdasarkan perilaku fungsi. Metode ini menggunakan Simpson's Rule sebagai basis dan membagi interval menjadi subinterval yang lebih kecil jika diperlukan untuk mencapai toleransi error yang ditentukan.

Simpson's Rule:

Untuk interval $[a, b]$:

$$S = \frac{h}{6}[f(a) + 4f(c) + f(b)]$$

di mana $c = \frac{a+b}{2}$ dan $h = b - a$.

Kriteria Adaptif:

Interval dibagi dua jika:

$$|S_2 - S| > 15 \times \text{tol}$$

di mana S adalah hasil Simpson untuk keseluruhan interval dan $S_2 = S_{\text{left}} + S_{\text{right}}$ adalah hasil dari dua subinterval.

Algoritma:

```
Input: fungsi f(x), batas a dan b, toleransi tol, max_depth
1. Fungsi simpson_rule(a, b):
   c = (a + b) / 2
   h = b - a
   return (h/6) * (f(a) + 4*f(c) + f(b))

2. Fungsi adaptive_aux(a, b, tol, S, fa, fb, fc, depth):
   a. Hitung titik tengah: d = (a+c)/2, e = (c+b)/2
   b. Hitung nilai fungsi: fd = f(d), fe = f(e)
   c. Sleft = (c-a)/6 * (fa + 4*fd + fc)
   d. Sright = (b-c)/6 * (fc + 4*fe + fb)
   e. S2 = Sleft + Sright
   f. Jika depth <= 0 atau |S2 - S| <= 15*tol:
      return S2 + (S2 - S)/15
   g. Jika tidak:
      return adaptive_aux(a, c, ...) + adaptive_aux(c, b, ...)

3. c = (a + b) / 2
4. Hitung S = simpson_rule(a, b)
5. return adaptive_aux(a, b, tol, S, f(a), f(b), f(c), max_depth)
Output: hasil integral
```

Implementasi Python:

```python
import numpy as np

def adaptive_simpson(f, a, b, tol=1e-10, max_depth=50):
    """
    Metode Adaptive Integration menggunakan Simpson's Rule
    """
    def simpson_rule(f, a, b):
        """Simpson's rule untuk interval [a, b]"""
        c = (a + b) / 2
        h = b - a
        return (h / 6) * (f(a) + 4 * f(c) + f(b))

    def adaptive_aux(f, a, b, tol, S, fa, fb, fc, depth):
        """Fungsi rekursif untuk adaptive integration"""
        c = (a + b) / 2
        d = (a + c) / 2
        e = (c + b) / 2

        fd = f(d)
        fe = f(e)

        Sleft = (c - a) / 6 * (fa + 4 * fd + fc)
        Sright = (b - c) / 6 * (fc + 4 * fe + fb)
        S2 = Sleft + Sright

```

```python
        if depth <= 0 or abs(S2 - S) <= 15 * tol:
            return S2 + (S2 - S) / 15

        return (adaptive_aux(f, a, c, tol/2, Sleft, fa, fc, fd,
           depth-1) +
                adaptive_aux(f, c, b, tol/2, Sright, fc, fb, fe,
                   depth-1))

    c = (a + b) / 2
    fa = f(a)
    fb = f(b)
    fc = f(c)
    S = simpson_rule(f, a, b)

    return adaptive_aux(f, a, b, tol, S, fa, fb, fc, max_depth)

# Contoh penggunaan
result = adaptive_simpson(f, 0, np.pi/2)
```

1.6. Metode Gaussian Quadrature

Gaussian Quadrature adalah metode integrasi numerik yang menggunakan titik-titik sampling optimal (nodes) dan bobot (weights) yang telah ditentukan secara matematis untuk memberikan akurasi maksimal dengan jumlah evaluasi fungsi minimal.

Rumus Umum:

Untuk interval $[-1, 1]$:

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

di mana x_i adalah nodes (Gauss-Legendre) dan w_i adalah weights.

Transformasi ke Interval $[a, b]$:

Untuk interval $[a, b]$, gunakan transformasi:

$$x = \frac{1}{2}[(b-a)t + (a+b)]$$

sehingga:

$$\begin{aligned}\int_a^b f(x)\,dx &= \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{1}{2}[(b-a)t + (a+b)]\right) dt \\ &\approx \frac{b-a}{2}\sum_{i=1}^{n} w_i f\left(\frac{1}{2}[(b-a)x_i + (a+b)]\right)\end{aligned}$$

Nodes dan Weights untuk 5-point Gauss-Legendre:

x_i (nodes)	w_i (weights)
−0.9061798459	0.2369268851
−0.5384693101	0.4786286705
0.0	0.5688888889
0.5384693101	0.4786286705
0.9061798459	0.2369268851

Algoritma:

```
Input: fungsi f(x), batas a dan b, jumlah titik n
1. Tentukan nodes dan weights untuk n-point:
   - Jika n=5, gunakan tabel di atas
   - Jika tidak, hitung dari polinomial Legendre
2. result = 0
3. Untuk i = 1 sampai n:
   a. x = 0.5 * ((b - a) * nodes[i] + (a + b))
   b. result += weights[i] * f(x)
4. result *= 0.5 * (b - a)
Output: result
```

Implementasi Python:

```python
import numpy as np

def gaussian_quadrature(f, a, b, n=5):
    """
    Metode Gaussian Quadrature
    Menggunakan n-point Gauss-Legendre quadrature
    """
    if n == 5:
        # 5-point Gauss-Legendre nodes dan weights
        nodes = np.array([
            -0.9061798459386640,
            -0.5384693101056831,
            0.0,
            0.5384693101056831,
            0.9061798459386640
        ])

        weights = np.array([
            0.2369268850561891,
            0.4786286704993665,
            0.5688888888888889,
            0.4786286704993665,
            0.2369268850561891
        ])
    else:
        # Untuk n lain, gunakan numpy (opsional)
        nodes, weights = np.polynomial.legendre.leggauss(n)

    # Transformasi dari [-1, 1] ke [a, b]
    result = 0
```

```python
    for i in range(len(nodes)):
        x = 0.5 * ((b - a) * nodes[i] + (a + b))
        result += weights[i] * f(x)

    result *= 0.5 * (b - a)
    return result

# Contoh penggunaan
result = gaussian_quadrature(f, 0, np.pi/2, n=5)
```

BAGIAN 2: Langkah Perhitungan

2.1. Perhitungan Integrasi Analitik (Nilai Eksak)

Fungsi 1: $f(x) = \cos(x)$ pada interval $[0, \frac{\pi}{2}]$

Langkah perhitungan:

$$\begin{aligned} I_1 &= \int_0^{\pi/2} \cos(x)\,dx \\ &= \sin(x)\Big|_0^{\pi/2} \\ &= \sin\left(\frac{\pi}{2}\right) - \sin(0) \\ &= 1 - 0 \\ &= 1 \end{aligned}$$

Nilai Eksak: $I_1 = 1.0$

Fungsi 2: $f(x) = x^2$ pada interval $[0, 1]$

Langkah perhitungan:

$$\begin{aligned} I_2 &= \int_0^1 x^2\,dx \\ &= \frac{x^3}{3}\Big|_0^1 \\ &= \frac{1^3}{3} - \frac{0^3}{3} \\ &= \frac{1}{3} - 0 \\ &= \frac{1}{3} \end{aligned}$$

Nilai Eksak: $I_2 = 0.333333...$ atau $\frac{1}{3}$

2.2. Perhitungan Metode Trapezoidal Rule

[Isi langkah-langkah perhitungan menggunakan metode Trapezoidal Rule]

2.3. Perhitungan Metode Richardson Extrapolation

[Isi langkah-langkah perhitungan menggunakan metode Richardson Extrapolation]

2.4. Perhitungan Metode Romberg Integration

[Isi langkah-langkah perhitungan menggunakan metode Romberg Integration]

2.5. Perhitungan Metode Adaptive Integration

[Isi langkah-langkah perhitungan menggunakan metode Adaptive Integration]

2.6. Perhitungan Metode Gaussian Quadrature

[Isi langkah-langkah perhitungan menggunakan metode Gaussian Quadrature]

BAGIAN 3: Hasil dan Perbandingan Error

3.1. Hasil Perhitungan

[Isi tabel hasil perhitungan dari semua metode]

3.2. Perbandingan Error

[Isi analisis perbandingan error dari semua metode]

3.3. Kesimpulan

[Isi kesimpulan dari hasil dan perbandingan error]