# Implementasi Interpolasi Metode Lagrange dan Newton

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

## Metode

### 1. Ringkasan

Dokumen ini berisi tentang implementasi interpolasi polinomial dengan metode Lagrange dan Newton dengan bahasa pemrograman Python. Terdapat sebuah pengukuran fiska telah dilakukan untuk menentukan hubungan antara tegangan yang diberikan kepada baja tahan-karat dan waktu yang diperlukan hingga baja tersebut patah. Delapan nilai tegangan yang berbeda dicobakan, dan data yang dihasilkan adalah sebagai berikut.

| Tegangan, $x\left(\frac{kg}{mm^2}\right)$ | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 |
|---|---|---|---|---|---|---|---|---|
| Waktu patah, y (Jam) | 40 | 30 | 25 | 40 | 18 | 20 | 22 | 15 |

Dari data di atas, buat kode program yang dapat mengimplementasikan konsep interpolasi metode Lagrange dan metode Newton dengan bentuk penyelesaian dalam bentuk plot grafik hasil interpolasi dari tiap-tiap metode.

### 2. Implementasi Kode

- Metode Polinomial Lagrange

Implementasi kode integrasi Metode Polinomial Newton dilakukan dengan bahasa pemrograman Python. Python dipilih karena memiliki sintaks yang relatif lebih sederhana dan singkat dibanding bahasa pemrograman lain. Selain itu, Python memiliki beragam *library* dan modul yang menyediakan fungsi dan algoritma numerik yang lengkap, sehingga sangat cocok untuk implementasi metode numerik ini. Untuk kode sumber (*source code*) secara lengkapnya adalah sebagai berikut:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Data Tegangan (x) dan Waktu patah (y)
x = np.array([5, 10, 15, 20, 25, 30, 35, 40])
y = np.array([40, 30, 25, 40, 18, 20, 22, 15])

# Fungsi untuk interpolasi Newton
def newton_interpolation(x_points, y_points, x):
    n = len(x_points)
    divided_diff = np.zeros((n, n))
    divided_diff[:,0] = y_points

    for j in range(1, n):
        for i in range(n - j):
            divided_diff[i, j] = (divided_diff[i + 1, j - 1] -
divided_diff[i, j - 1]) / (x_points[i + j] - x_points[i])

    def newton_basis(j, x):
        terms = [(x - x_points[i]) for i in range(j)]
        return np.prod(terms, axis=0)

    return sum(divided_diff[0, j] * newton_basis(j, x) for j in range(n))

# Range untuk x dari 5 sampai 40
x_range = np.linspace(5, 40, 500)

# Menghitung hasil interpolasi
y_newton = newton_interpolation(x, y, x_range)

# Plot hasil interpolasi
plt.figure(figsize=(14, 8))
plt.plot(x_range, y_newton, label='Hasil Interpolasi', color='green')
plt.scatter(x, y, color='red', label='Titik Data')
plt.title('Interpolasi Polinom dengan Metode Newton')
plt.xlabel('Tegangan (kg/mm^2)')
plt.ylabel('Waktu patah (jam)')
plt.legend()
plt.grid(True)
plt.show()
```

Untuk penjelasan detail dari kode di atas adalah sebagai berikut:

a. Import Library

```
import numpy as np
import matplotlib.pyplot as plt
```

Pertama-tama, kode ini dimulai dengan mengimpor modul `numpy` dengan alias `np` operasi numerik dan array, dan mengimpor `matplotlib.pyplot` dengan alias `plt` untuk *plotting* data.

b. Memasukkan data x dan y:

```
x_points = np.array([5, 10, 15, 20, 25, 30, 35, 40])
y_points = np.array([40, 30, 25, 40, 18, 20, 22, 15])
```

Mendefinisikan dua array `x_points` dan `y_points` yang masing-masing berisi data tegangan dan waktu patah yang sudah diberikan sebelumnya.

c. Definisikan Fungsi `newton_interpolation`.

```
def newton_interpolation(x_points, y_points, x):
```

Ini adalah definisi fungsi newton_interpolation yang menerima tiga argumen: x_points (titik-titik data x), y_points (titik-titik data y), dan x (nilai atau array nilai di mana kita ingin mengevaluasi polinom interpolasi).

d. Definisikan variabel n

```
    n = len(x_points)
    divided_diff = np.zeros((n, n))
    divided_diff[:,0] = y_points
```

Variabel `n` menyimpan jumlah titik data. `divided_diff` adalah array 2D yang akan menyimpan tabel selisih terbagi Newton. Kolom pertama `divided_diff` diisi dengan nilai `y_points`.

e. Buat *loop* Untuk Menghitung Selisih Terbagi Newton

```
    for j in range(1, n):
        for i in range(n - j):
            divided_diff[i, j] = (divided_diff[i + 1, j - 1] -
divided_diff[i, j - 1]) / (x_points[i + j] - x_points[i])
```

Dua loop bersarang ini menghitung tabel selisih terbagi Newton. Loop luar dengan indeks `j` berjalan melalui kolom, dan loop dalam dengan indeks `i` berjalan melalui baris dari setiap kolom.

f. Mengembalikan Nilai Interpolasi pada Titik x

```
    return sum(divided_diff[0, j] * newton_basis(j, x) for j in range(n))
```

Baris ini mengembalikan nilai interpolasi pada titik x dengan menjumlahkan hasil kali dari elemen pertama di setiap kolom tabel selisih terbagi dengan basis polinomial Newton yang sesuai. Hasil dari newton_basis(j, x) dikalikan dengan divided_diff[0, j] dan dijumlahkan untuk semua j dari 0 hingga n-1.

g. Menghitung Nilai Interpolasi

```
x_range = np.linspace(5, 40, 500)
```

Menghitung nilai interpolasi untuk setiap titik dalam `x_range` dengan memanggil fungsi `lagrange_interpolation`. Hasilnya disimpan dalam `y_lagrange`.

h. Menghitung nilai interpolasi

```
y_newton = newton_interpolation(x, y, x_range)
```

Baris ini menghitung nilai interpolasi untuk setiap titik dalam `x_range` menggunakan fungsi `newton_interpolation` yang telah didefinisikan sebelumnya. Hasilnya disimpan dalam `y_newton`.

i. *Print* Hasil

```
plt.figure(figsize=(14, 8))
plt.plot(x_range, y_newton, label='Hasil Interpolasi', color='green')
plt.scatter(x, y, color='red', label='Titik Data')
plt.title('Interpolasi Polinom dengan Metode Newton')
plt.xlabel('Tegangan (kg/mm^2)')
plt.ylabel('Waktu patah (jam)')
plt.legend()
plt.grid(True)
plt.show()
```

Setelah semua selesai dihitung, plot hasil tersebut dalam bentuk grafik. Pertama, Baris-baris ini membuat plot dari hasil interpolasi. `plt.figure(figsize=(14, 8))` membuat

kanvas plot dengan ukuran 14x8 inci. `plt.plot(x_range, y_newton, label='Hasil Interpolasi', color='green')` membuat plot garis dari hasil interpolasi. `plt.scatter(x, y, color='red', label='Titik Data')` menambahkan titik data asli sebagai titik-titik merah. plt.title, plt.xlabel, dan plt.ylabel menetapkan judul dan label sumbu. `plt.legend()` menambahkan legenda ke plot, `plt.grid(True)` menambahkan grid ke plot, dan `plt.show()` menampilkan plot.

- Metode Polinomial Lagrange

Implementasi kode integrasi Metode Polinomial Lagrange dilakukan dengan bahasa pemrograman Python. Python dipilih karena memiliki sintaks yang relatif lebih sederhana dan singkat dibanding bahasa pemrograman lain. Selain itu, Python memiliki beragam *library* dan modul yang menyediakan fungsi dan algoritma numerik yang lengkap, sehingga sangat cocok untuk implementasi metode numerik ini. Untuk kode sumber (*source code*) secara lengkapnya adalah sebagai berikut:

```
import numpy as np
import matplotlib.pyplot as plt

# Data Tegangan (x) dan Waktu patah (y)

x_points = np.array([5, 10, 15, 20, 25, 30, 35, 40])
y_points = np.array([40, 30, 25, 40, 18, 20, 22, 15])

# Fungsi untuk interpolasi Lagrange
def lagrange_interpolation(x_points, y_points, x):
    def L(k, x):
        terms = [(x - x_points[i]) / (x_points[k] - x_points[i]) for i in
range(len(x_points)) if i != k]
        return np.prod(terms, axis=0)

    return sum(y_points[k] * L(k, x) for k in range(len(x_points)))

# Range untuk x dari 5 sampai 40
x_range = np.linspace(5, 40, 1000)

# Menghitung hasil interpolasi
y_lagrange = lagrange_interpolation(x_points, y_points, x_range)
```

```
# Plot hasil interpolasi
plt.figure(figsize=(14, 8))
plt.plot(x_range, y_lagrange, label='Hasil Interpolasi', color='blue')
plt.scatter(x_points, y_points, color='red', label='Titik Data')
plt.title('Interpolasi Polinom dengan Metode Lagrange')
plt.xlabel('Tegangan (kg/mm^2)')
plt.ylabel('Waktu patah (jam)')
plt.legend()
plt.grid(True)
plt.show()
```

Untuk penjelasan detail dari kode di atas adalah sebagai berikut:

a. Import Library

```
import numpy as np
import matplotlib.pyplot as plt
```

Pertama-tama, kode ini dimulai dengan mengimpor modul `numpy` dengan alias `np` operasi numerik dan array, dan mengimpor `matplotlib.pyplot` dengan alias `plt` untuk *plotting* data.

b. Memasukkan data x dan y:

```
x_points = np.array([5, 10, 15, 20, 25, 30, 35, 40])
y_points = np.array([40, 30, 25, 40, 18, 20, 22, 15])
```

Mendefinisikan dua array `x_points` dan `y_points` yang masing-masing berisi data tegangan dan waktu patah yang sudah diberikan sebelumnya.

c. Definisikan Fungsi `lagrange_interpolation`.

```
def lagrange_interpolation(x_points, y_points, x):
    def L(k, x):
        terms = [(x - x_points[i]) / (x_points[k] - x_points[i]) for i in
range(len(x_points)) if i != k]
        return np.prod(terms, axis=0)
```

```
    return sum(y_points[k] * L(k, x) for k in range(len(x_points)))# Normalisasi integrasi dengan h/3
    return integration
```

Ini adalah definisi fungsi `lagrange_interpolation` yang menerima tiga parameter: `x_points` (titik data pada sumbu x), `y_points` (titik data pada sumbu y), dan `x` (titik atau array titik di mana nilai interpolasi akan dihitung). Untuk penjelasan lebih detail tentang fungsi **`simpson13`** adalah sebagai berikut:

d. Definisikan Fungsi `L`.

```
def L(k, x):
    terms = [(x - x_points[i]) / (x_points[k] - x_points[i]) for i in range(len(x_points)) if i != k]
    return np.prod(terms, axis=0)
```

Mendefinisikan fungsi nested `L(k, x)` untuk menghitung basis polinomial Lagrange. `terms` adalah daftar yang menyimpan hasil perhitungan. `np.prod(terms, axis=0)` menghitung hasil kali semua elemen dalam `terms`.

e. Mengembalikan nilai interpolasi

```
    return sum(y_points[k] * L(k, x) for k in range(len(x_points)))
```

baris ini menghitung lebar setiap sub-interval dengan membagi selisih antara `upper_limit` dan `lower_limit` dengan jumlah sub-interval (`sub_interval`).

f. Mendefinisikan `x_range`

```
x_range = np.linspace(5, 40, 1000)
```

Mendefinisikan `x_range`, sebuah array yang terdiri dari 1000 titik yang merata dari 5 hingga 40. Ini akan digunakan untuk menggambarkan kurva interpolasi.

g. Menghitung nilai interpolasi

```
y_lagrange = lagrange_interpolation(x_points, y_points, x_range)
```

Menghitung nilai interpolasi untuk setiap titik dalam `x_range` dengan memanggil fungsi `lagrange_interpolation`. Hasilnya disimpan dalam `y_lagrange`.

h. *Print* Hasil

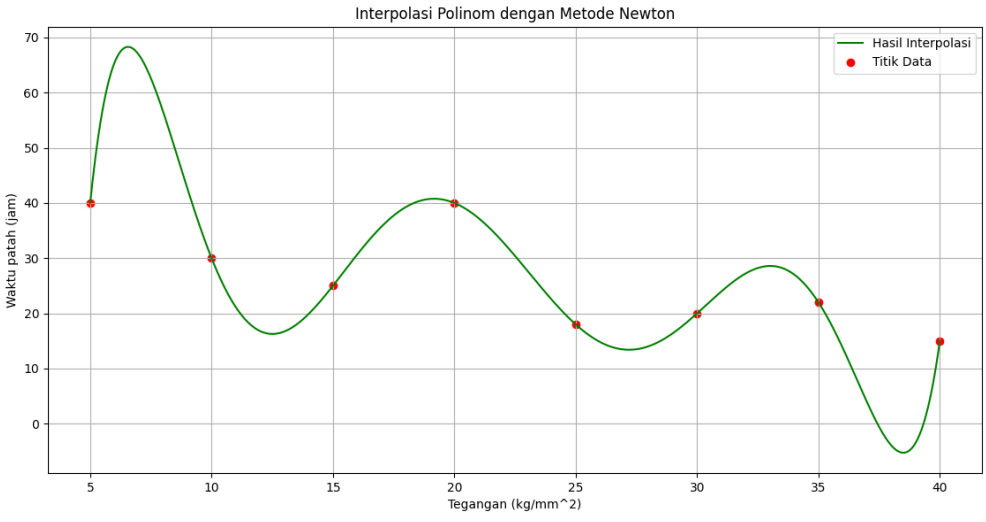
```
plt.figure(figsize=(14, 8))
plt.plot(x_range, y_lagrange, label='Hasil Interpolasi', color='blue')
```

```
plt.scatter(x_points, y_points, color='red', label='Titik Data')
plt.title('Interpolasi Polinom dengan Metode Lagrange')
plt.xlabel('Tegangan (kg/mm^2)')
plt.ylabel('Waktu patah (jam)')
plt.legend()
plt.grid(True)
plt.show()
```

Setelah semua selesai dihitung, plot hasil tersebut dalam bentuk grafik. Pertama, membuat plot menggunakan `matplotlib`. `plt.figure(figsize=(14, 8))` untuk mengatur ukuran gambar plot. `plt.plot(x_range, y_lagrange, label='Hasil Interpolasi', color='blue')` untuk menggambar kurva interpolasi Lagrange. `plt.scatter(x_points, y_points, color='red', label='Titik Data')` untuk menambahkan titik data asli ke plot sebagai scatter plot. `plt.title('Interpolasi Polinom dengan Metode Lagrange')` untuk menambahkan judul plot. `plt.xlabel('Tegangan (kg/mm^2)')` dan `plt.ylabel('Waktu patah (jam)')` untuk menambahkan label sumbu x dan y. `plt.legend()` untuk menampilkan legenda untuk plot. `plt.grid(True)` menenambahkan grid ke plot. `plt.show()` untukenampilkan plot.

## 3. Hasil Pengujian dan Analisis

a. Hasil Interpolasi dengan Metode Newton

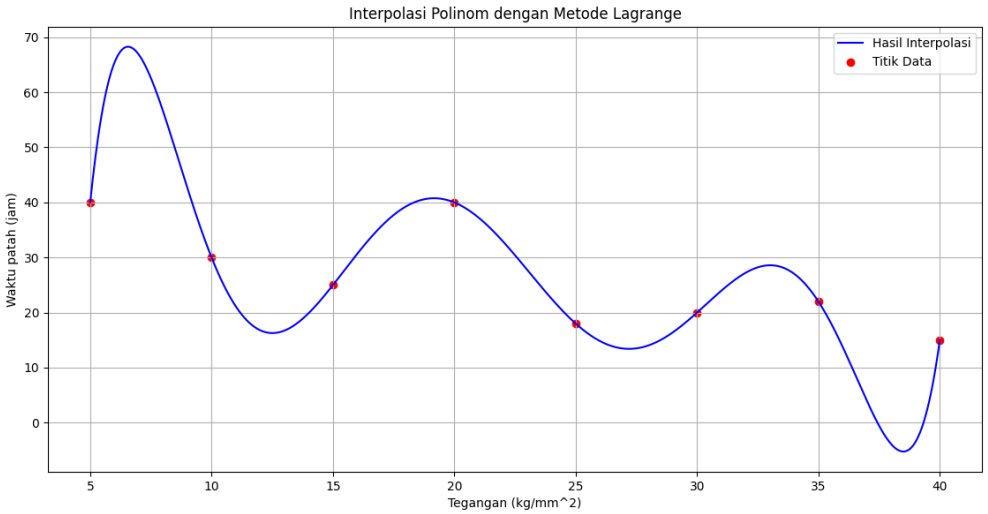



Kode ini pertama-tama menghitung tabel selisih terbagi, yang merupakan dasar dari interpolasi Newton. Tabel ini dibangun secara rekursif, di mana setiap entri dihitung berdasarkan entri sebelumnya. Fungsi `` `newton_basis` `` menghitung produk dari faktor-

faktor, dan fungsi utama `newton_interpolation` menggunakan hasil dari tabel selisih terbagi untuk menggabungkan nilai-nilai basis ini menjadi polinomial interpolasi. Plot hasil menunjukkan bagaimana polinomial ini mendekati titik-titik data asli, dengan warna hijau untuk kurva interpolasi dan merah untuk titik-titik data.

Kode ini pertama-tama menghitung tabel selisih terbagi, yang merupakan dasar dari interpolasi Newton. Tabel ini dibangun secara rekursif, di mana setiap entri dihitung berdasarkan entri sebelumnya. Fungsi `newton_basis` menghitung produk dari faktor-faktor (`x - x_points[i])`, dan fungsi utama `newton_interpolation` menggunakan hasil dari tabel selisih terbagi untuk menggabungkan nilai-nilai basis ini menjadi polinomial interpolasi. Plot hasil menunjukkan bagaimana polinomial ini mendekati titik-titik data asli, dengan warna hijau untuk kurva interpolasi dan merah untuk titik-titik data.

b. Hasil Interpolasi dengan Metode Polinomial Lagrange




Pada kode kedua, digunakan metode interpolasi Lagrange. Di sini, setiap basis Lagrange dihitung dengan fungsi `L(k, x)`, yang mengalikan beberapa faktor untuk setiap titik data kecuali titik k itu sendiri. Fungsi `lagrange_interpolation` kemudian menjumlahkan produk dari nilai basis ini dengan nilai y yang sesuai untuk setiap titik

data. Hasil interpolasi ini diplot dengan warna biru, sementara titik data asli juga ditampilkan dengan warna merah.

Analisis hasil dari kedua metode interpolasi ini menunjukkan bahwa kedua polinomial interpolasi dapat mendekati titik data dengan baik, namun dengan pendekatan yang berbeda. Interpolasi Newton menggunakan selisih terbagi untuk membangun polinomial secara efisien, sementara interpolasi Lagrange membangun polinomial secara langsung dengan menggunakan basis Lagrange. Meskipun hasil akhirnya tampak serupa, performa dan stabilitas dari kedua metode ini bisa berbeda, terutama untuk jumlah titik data yang lebih besar atau distribusi titik data yang tidak merata. Metode Newton umumnya lebih efisien dalam hal perhitungan, sementara metode Lagrange lebih mudah dipahami dan diimplementasikan dalam bentuk langsung.