

# Laporan Tugas Besar 1

IF2123 ALJABAR LINIER DAN GEOMETRI  
SISTEM PERSAMAAN LINIER, DETERMINAN, DAN PENERAPANNYA  
SEMESTER I TAHUN 2025/2026



Disusun oleh:  
13524030 Irvin Tandiarrang Sumual  
13524089 Aurelia Jennifer Gunawan  
13524122 Nathaniel Christian

Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung

## Daftar Isi

<b>1 Pendahuluan</b>	<b>1</b>
<b>2 Dasar Teori</b>	<b>2</b>
2.1 Sistem Persamaan Linear . . . . .	2
2.1.1 Metode eliminasi Gauss . . . . .	2
2.1.2 Metode eliminasi Gauss-Jordan . . . . .	2
2.1.3 Metode kaidah Cramer . . . . .	2
2.1.4 Metode matriks balikan . . . . .	3
2.2 Determinan Matriks . . . . .	3
2.2.1 Metode Ekspansi Kofaktor . . . . .	3
2.2.2 Metode Reduksi Baris (OBE) . . . . .	4
2.3 Invers Matriks . . . . .	4
2.3.1 Metode Augment . . . . .	4
2.3.2 Metode Adjoin . . . . .	4
2.4 Interpolasi . . . . .	5
2.4.1 Interpolasi Polinomial . . . . .	5
2.4.2 Interpolasi Splina Bezier Kubik . . . . .	5
2.5 Regresi Polinomial Berganda ( <i>Multivariate Polynomial Regression</i> ) . . . . .	6
<b>3 Implementasi</b>	<b>7</b>
3.1 Modules . . . . .	7
3.1.1 Matrix.java . . . . .	7
3.1.2 MatrixOperator.java . . . . .	9
3.1.3 SPL.java . . . . .	9
3.1.4 SPLResult.java . . . . .	10
3.1.5 Determinant.java . . . . .	10
3.1.6 DeterminantResult.java . . . . .	10
3.1.7 CofactorResult.java . . . . .	11
3.1.8 Inverse.java . . . . .	11
3.1.9 InverseResult.java . . . . .	11
3.1.10 Interpolation.java . . . . .	12
3.1.11 Regression.java . . . . .	12
3.2 GUI . . . . .	12
3.2.1 App.java . . . . .	13
3.2.2 MainMenuUI.java . . . . .	13
3.2.3 MainMenuController.java . . . . .	13
3.2.4 SubMenuController.java . . . . .	14
3.2.5 UIController.java . . . . .	14
3.2.6 FormatResult.java . . . . .	15
3.2.7 CalculationController.java . . . . .	16
3.2.8 InterpolationController.java . . . . .	17
3.2.9 RegressionController.java . . . . .	18
3.2.10 FileHandler.java . . . . .	18
3.2.11 MatrixParser.java . . . . .	19
<b>4 Eksperimen</b>	<b>20</b>
4.1 Sistem Persamaan Linear . . . . .	20
4.1.1 Kasus Uji 1a . . . . .	20
4.1.2 Kasus Uji 2a . . . . .	22
4.1.3 Kasus Uji 3a . . . . .	25
4.1.4 Kasus Uji 4a . . . . .	28
4.1.5 Kasus Uji 1b . . . . .	32
4.1.6 Kasus Uji 2b . . . . .	35
4.1.7 Kasus Uji 1c . . . . .	37
4.1.8 Kasus Uji 2c . . . . .	40
4.1.9 Kasus Uji 1d . . . . .	44
4.2 Determinan . . . . .	46

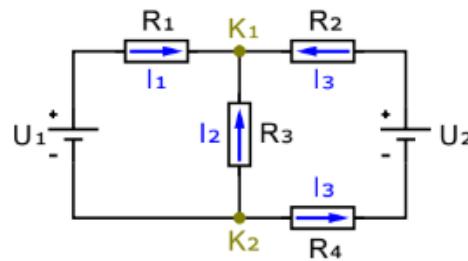
4.2.1	Kasus Uji 1 . . . . .	46
4.2.2	Kasus Uji 2 . . . . .	48
4.3	Inverse . . . . .	49
4.3.1	Kasus Uji 1 . . . . .	49
4.3.2	Kasus Uji 2 . . . . .	51
4.4	Interpolasi Polinomial . . . . .	52
4.4.1	Kasus Uji 1 . . . . .	52
4.4.2	Kasus Uji 2 . . . . .	56
4.5	Interpolasi Splina Bezier Kubik . . . . .	60
4.5.1	Kasus Uji 1 . . . . .	60
4.6	Regresi Polinomial Berganda . . . . .	62
4.6.1	Kasus Uji 1 . . . . .	62
<b>5</b>	<b>Penutup</b>	<b>64</b>
5.1	Kesimpulan . . . . .	64
5.2	Saran . . . . .	64
5.3	Komentar . . . . .	64

## 1 Pendahuluan

Aplikasi seperti Photomath telah menjadi solusi populer dalam membantu menyelesaikan berbagai permasalahan matematika, termasuk soal matriks dan sistem persamaan linier yang kompleks. Kemampuan aplikasi tersebut dalam memberikan langkah-langkah penyelesaian secara otomatis tidak terlepas dari penerapan algoritma dan konsep matematika yang kuat, khususnya aljabar linier.

Aljabar linier merupakan salah satu cabang matematika yang sangat penting dalam pengembangan ilmu komputer, teknik, fisika, ekonomi, dan berbagai bidang lainnya. Dalam kehidupan sehari-hari, konsep matriks, dan sistem persamaan linier sering kali diaplikasikan untuk menyelesaikan berbagai persoalan nyata, mulai dari pemodelan data, analisis statistik, hingga simulasi sistem dinamis.

Pada tugas besar ini, Anda akan mengimplementasikan berbagai metode penyelesaian sistem persamaan linier, perhitungan determinan, pencarian invers matriks, interpolasi polinomial, interpolasi kurva Bezier kubik, dan regresi polinomial menggunakan bahasa pemrograman Java dalam bentuk pustaka (library) yang dapat digunakan secara modular dan terdokumentasi dengan baik.



Gambar: Aplikasi Photomath (kiri) dan diagram rangkaian listrik (kanan). Sumber: <https://thejournal.com/articles/2016/09/23/photomath-app-update-solves-handwritten-math-problems.aspx>, [https://en.m.wikipedia.org/wiki/File:Kirchhoff\\_voltage\\_law.svg](https://en.m.wikipedia.org/wiki/File:Kirchhoff_voltage_law.svg)

Penyelesaian sistem persamaan linier, misalnya, sangat krusial dalam pemodelan sirkuit listrik, analisis struktur bangunan, hingga pemrosesan citra digital. Sementara itu, interpolasi dan regresi polinomial digunakan untuk memperkirakan nilai di antara data yang tersedia atau memprediksi tren data di masa depan.

Melalui tugas besar ini, diharapkan Anda tidak hanya memahami teori di balik metode-metode tersebut, tetapi juga mampu mengimplementasikannya secara nyata yang dapat digunakan untuk menyelesaikan berbagai kasus uji (test cases) yang telah disediakan.

Secara formal, tujuan dari tugas besar ini adalah:

- Mengimplementasikan berbagai metode penyelesaian sistem persamaan linier, perhitungan determinan, invers matriks, interpolasi, dan regresi polinomial secara mandiri dalam bahasa Java.
- Membuat pustaka (library) yang dapat digunakan secara modular dan terdokumentasi dengan baik.
- Mengintegrasikan pustaka yang dibuat ke dalam sebuah program yang dapat menerima masukan dari pengguna dan menampilkan hasilnya dengan format yang jelas.
- Menguji pustaka yang dibuat pada berbagai kasus uji dan menganalisis hasilnya.

Dengan demikian, tugas besar ini diharapkan dapat menjadi sarana pembelajaran yang efektif dalam memahami dan mengaplikasikan konsep-konsep aljabar linier secara komputasional, serta membekali Anda dengan keterampilan praktis yang sangat dibutuhkan di dunia profesional.

## 2 Dasar Teori

### 2.1 Sistem Persamaan Linear

Sistem persamaan linear (SPL) adalah himpunan berhingga dari kumpulan persamaan linear dengan variabel yang sama.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m. \end{aligned} \tag{1}$$

SPL ini dapat dinyatakan dalam bentuk perkalian matriks (kiri) dan matriks *augmented* (kanan):

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}}_A \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_x = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}}_b \rightarrow \left[ \begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{array} \right]$$

Solusi dari SPL dapat berupa solusi tunggal, banyak solusi, atau tidak ada solusi. Metode penyelesaian yang umum digunakan antara lain:

#### 2.1.1 Metode eliminasi Gauss

- Nyatakan SPL dalam bentuk matriks *augmented*
- Terapkan OBE pada matriks *augmented* sampai terbentuk matriks eselon baris

$$\left[ \begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{array} \right] \sim_{\text{OBE}} \left[ \begin{array}{cccc|c} 1 & * & * & \cdots & * & * \\ 0 & 1 & * & \cdots & * & * \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & * \end{array} \right]$$

- Selesaikan persamaan yang berkoresponden pada matriks eselon baris dengan teknik penyulihan mundur (*backward substitution*)

#### 2.1.2 Metode eliminasi Gauss-Jordan

- Merupakan pengembangan metode eliminasi Gauss b. Operasi baris elementer (OBE) diterapkan pada matriks *augmented* sehingga menghasilkan matriks eselon baris tereduksi

$$\left[ \begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{array} \right] \sim_{\text{OBE}} \left[ \begin{array}{cccc|c} 1 & 0 & 0 & \cdots & 0 & * \\ 0 & 1 & 0 & \cdots & 0 & * \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & * \end{array} \right]$$

- Tidak diperlukan lagi substitusi secara mundur untuk memperoleh nilai-nilai variabel. Nilai variabel langsung diperoleh dari matriks *augmented* akhir (jika solusinya unik).

#### 2.1.3 Metode kaidah Cramer

- Jika  $Ax = b$  adalah SPL yang terdiri dari n persamaan linier dengan n peubah (variable) sedemikian sehingga  $\det(A) \neq 0$ , maka SPL tersebut memiliki solusi yang unik yaitu:

$$x_1 = \frac{\det(A_1)}{\det(A)}, \quad x_2 = \frac{\det(A_2)}{\det(A)}, \quad \dots, \quad x_n = \frac{\det(A_n)}{\det(A)}$$

- Dalam hal ini,  $A_j$  adalah matriks yang diperoleh dengan mengganti entri pada kolom ke-j dari A dengan entri dari matriks b:

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

#### 2.1.4 Metode matriks balikan

- a. Tinjau SPL  $Ax = \mathbf{b}$   
 b. Kalikan kedua ruas persamaan dengan  $A^{-1}$

$$\begin{aligned} (A^{-1}A)\mathbf{x} &= (A^{-1})\mathbf{b} \\ I\mathbf{x} &= A^{-1}\mathbf{b} \quad (\text{karena } A^{-1}A = I) \\ \mathbf{x} &= A^{-1}\mathbf{b} \quad (\text{karena } I\mathbf{x} = \mathbf{x}) \end{aligned}$$

- c. Sehingga, solusi SPL  $A\mathbf{x} = \mathbf{b}$  adalah  $\mathbf{x} = A^{-1}\mathbf{b}$

## 2.2 Determinan Matriks

Determinan adalah sebuah nilai skalar khusus yang dapat dihitung dari elemen-elemen sebuah matriks persegi. Determinan dari matriks  $A$  dinotasikan sebagai  $\det(A)$  atau  $|A|$ . Nilai determinan dapat mengindikasikan apakah matriks tersebut memiliki invers. Metode yang umum digunakan untuk menghitung determinan matriks antara lain adalah:

### 2.2.1 Metode Ekspansi Kofaktor

- a. Misalkan  $A$  adalah matriks berukuran  $n \times n$ :

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

Maka, didefinisikan:

$M_{ij}$  = minor entri  $a_{ij}$   
 = determinan upa-matriks (*submatrix*) yang elemennya tidak berada pada baris  $i$  dan kolom  $j$

$$C_{ij} = (-1)^{i+j} M_{ij} = \text{kofaktor entri } a_{ij}$$

- b. Dengan menggunakan kofaktor, maka determinan matriks

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

dapat dihitung dengan salah satu dari persamaan berikut:

### 2.2.2 Metode Reduksi Baris (OBE)

a. Determinan matriks  $A$  dapat diperoleh dengan melakukan OBE pada matriks  $A$  hingga diperoleh matriks segitiga (atas atau bawah):

$$A \xrightarrow{\text{OBE}} \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \xrightarrow{\text{OBE}} \begin{bmatrix} a'_{11} & a'_{12} & \cdots & a'_{1n} \\ 0 & a'_{22} & \cdots & a'_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a'_{nn} \end{bmatrix}$$

Maka:

$$\det(A) = (-1)^p a'_{11} a'_{22} \cdots a'_{nn}$$

dengan  $p$  menyatakan banyaknya operasi pertukaran baris dalam OBE.

*Catatan: diasumsikan tidak ada operasi perkalian baris dengan konstanta  $k$ .* b. Jika selama reduksi baris terdapat OBE berupa perkalian baris matriks dengan konstanta  $k_1, k_2, \dots, k_m$ , maka:

$$\det(A) = \frac{(-1)^p a'_{11} a'_{22} \cdots a'_{nn}}{k_1 k_2 \cdots k_m}$$

dengan:

- $p$  = banyaknya operasi pertukaran baris,
- $k_1, k_2, \dots, k_m$  = konstanta pengali pada baris matriks selama OBE,
- $a'_{11}, a'_{22}, \dots, a'_{nn}$  = elemen diagonal utama matriks segitiga hasil reduksi.

## 2.3 Invers Matriks

Invers dari sebuah matriks persegi  $A$  adalah matriks  $A^{-1}$  yang memenuhi properti  $AA^{-1} = A^{-1}A = I$ , di mana  $I$  adalah matriks identitas. Sebuah matriks hanya memiliki invers jika dan hanya jika determinannya tidak nol ( $\det(A) \neq 0$ ). Matriks yang memiliki invers disebut matriks invertible atau non-singular. Metode perhitungan invers dari suatu matriks meliputi:

### 2.3.1 Metode Augment

a. Misalkan  $A$  adalah matriks persegi berukuran  $n \times n$ . Balikan (inverse) matriks  $A$  adalah  $A^{-1}$  sehingga berlaku

$$AA^{-1} = A^{-1}A = I,$$

dengan  $I$  matriks identitas.

b. Langkah Metode Gauss–Jordan:

1. Bentuk matriks augmented:

$$[A | I]$$

2. Lakukan operasi baris elementer (OBE) hingga sisi kiri menjadi matriks identitas.

3. Hasil akhirnya adalah

$$[A | I] \xrightarrow{G-J} [I | A^{-1}]$$

### 2.3.2 Metode Adjoin

Balikan matriks  $A$  dapat dihitung dengan menggunakan rumus:

$$A^{-1} = \frac{1}{\det(A)} \text{adj}(A)$$

- $\det(A)$  = determinan matriks  $A$ ,
- $\text{adj}(A)$  = adjoin (atau adjugat) matriks  $A$ , yaitu transpose dari matriks kofaktor,
- Syarat:  $\det(A) \neq 0$  agar  $A^{-1}$  ada.

Adjoin  $A$  adalah transpose dari matriks kofaktor  $A$ .

## 2.4 Interpolasi

Interpolasi merupakan suatu metode dalam analisis numerik yang digunakan untuk membentuk suatu kurva halus yang melalui sekumpulan titik data tertentu.

### 2.4.1 Interpolasi Polinomial

Interpolasi polinomial adalah metode untuk menentukan suatu polinomial  $p_n(x)$  berderajat  $n$  yang melalui  $n + 1$  titik data  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$ , dengan  $x_i$  diurutkan sehingga  $x_0$  terkecil dan  $x_n$  terbesar.

Secara umum, polinomial tersebut dapat dituliskan sebagai:

$$p_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

dengan koefisien  $a_0, a_1, \dots, a_n$  yang ditentukan sedemikian rupa sehingga  $p_n(x_{\text{perkiraan}}) = y_{\text{perkiraan}}$  untuk semua nilai  $x_0 < x_{\text{perkiraan}} < x_n$ . Derajat persamaan polinom ditentukan oleh jumlah titik data dikurangi satu.

Koefisien polinomial dapat dicari dengan menyulihkan titik-titik data ke dalam persamaan polinomial dan menyelesaikan sistem persamaan linear yang dihasilkan dengan metode eliminasi Gauss yang dijelaskan sebelumnya. Kita dapat menyusun sistem persamaan linear sebagai berikut:

$$\begin{cases} a_0 + a_1x_0 + a_2x_0^2 + \cdots + a_nx_0^n = y_0 \\ a_0 + a_1x_1 + a_2x_1^2 + \cdots + a_nx_1^n = y_1 \\ \vdots \\ a_0 + a_1x_n + a_2x_n^2 + \cdots + a_nx_n^n = y_n \end{cases}$$

Persamaan-persamaan di atas dapat dipetakan menjadi sebuah permasalahan sistem persamaan linear, sehingga dapat diterapkan metode eliminasi Gauss untuk menyelesaikan sistem tersebut dan menemukan koefisien polinomial  $a_0, a_1, \dots, a_n$ . Dari polinom  $P_n(x)$  yang dibentuk dengan koefisien tersebut, kita dapat memperkirakan nilai  $y$  untuk nilai  $x$  yang berada di antara titik-titik data yang diberikan (interpolasi) maupun di luar titik-titik data tersebut (ekstrapolasi).

### 2.4.2 Interpolasi Splina Bezier Kubik

Bentuk interpolasi yang banyak digunakan adalah interpolasi splina, yaitu interpolasi yang memanfaatkan potongan-potongan fungsi polinomial dengan derajat rendah namun tersambung halus pada titik-titik simpulnya (knots).

Salah satu varian dari interpolasi splina yang populer adalah interpolasi splina Bézier kubik. Interpolasi splina Bézier kubik didasarkan pada polinomial Bézier derajat tiga. Kurva Bézier kubik banyak digunakan dalam grafik komputer, desain grafis, serta pemodelan bentuk karena memiliki sifat fleksibel, mudah dikendalikan, dan menghasilkan kurva halus. Kurva Bézier kubik didefinisikan secara parametrik oleh empat titik kontrol: titik awal ( $P_0$ ), titik akhir ( $P_3$ ), dan dua titik kontrol perantara ( $P_1$  dan  $P_2$ ) yang menentukan bentuk dan kelengkungan kurva. Persamaannya adalah:

$$B(t) = (1 - t)^3 P_0 + 3t(1 - t)^2 P_1 + 3t^2(1 - t)P_2 + t^3 P_3, \quad \text{untuk } 0 \leq t \leq 1$$

Ketika kita ingin membuat kurva halus yang melewati serangkaian titik interpolasi  $(S_0, S_1, \dots, S_n)$ , kita membangunnya dari segmen-segmen kurva Bézier kubik yang saling bersambungan. Masalahnya adalah kita perlu menemukan titik-titik kontrol perantara  $(B_1, B_2, \dots, B_{n-1})$  yang memastikan kurva tetap mulus di setiap titik sambungan.

Hubungan antara titik interpolasi  $S_k$  dan titik kontrol  $B_k$  dapat dirumuskan menjadi sebuah sistem persamaan linear. Untuk  $n$  titik interpolasi, sistem persamaan untuk menemukan  $n - 1$  titik kontrol perantara adalah sebagai berikut:

$$\begin{cases} 4\mathbf{b}_1 + \mathbf{b}_2 &= 6\mathbf{s}_1 - \mathbf{s}_0 \\ \mathbf{b}_1 + 4\mathbf{b}_2 + \mathbf{b}_3 &= 6\mathbf{s}_2 \\ &\vdots \\ \mathbf{b}_{n-2} + 4\mathbf{b}_{n-1} &= 6\mathbf{s}_{n-1} - \mathbf{s}_n \end{cases}$$

Sistem ini dapat ditulis dalam bentuk matriks:

$$\begin{bmatrix} 4 & 1 & & \\ 1 & 4 & 1 & \\ \ddots & \ddots & \ddots & \\ & 1 & 4 & 1 \\ & & 1 & 4 \end{bmatrix} \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_{n-2} \\ \mathbf{b}_{n-1} \end{bmatrix} = \begin{bmatrix} 6\mathbf{s}_1 - \mathbf{s}_0 \\ 6\mathbf{s}_2 \\ \vdots \\ 6\mathbf{s}_{n-2} \\ 6\mathbf{s}_{n-1} - \mathbf{s}_n \end{bmatrix}$$

Karena setiap titik  $\mathbf{b}_k$  dan  $\mathbf{s}_k$  memiliki komponen  $(x, y)$ , sistem persamaan linear ini harus diselesaikan dua kali: sekali untuk semua komponen  $x$  dan sekali lagi untuk semua komponen  $y$ .

## 2.5 Regresi Polinomial Berganda (*Multivariate Polynomial Regression*)

Analisis regresi merupakan salah satu metode statistika yang digunakan untuk memodelkan hubungan antara variabel dependen ( $y$ ) dengan satu atau lebih variabel independen ( $x$ ). Pada kasus yang paling sederhana, yaitu regresi linear, hubungan ini dimodelkan sebagai sebuah garis lurus dengan persamaan  $y = \beta_0 + \beta_1 x$ . Namun, dalam banyak situasi di dunia nyata, hubungan antarvariabel tidaklah sesederhana itu dan tidak dapat dimodelkan secara akurat dengan sebuah garis lurus.

Untuk mengatasi keterbatasan tersebut, kita dapat menggunakan regresi polinomial berganda (*multivariate polynomial regression*). Metode ini merupakan pengembangan dari regresi linear yang memungkinkan kita untuk memodelkan hubungan yang lebih kompleks dan non-linear. Ide dasarnya adalah dengan menambahkan suku-suku polinomial, seperti  $x_1^2, x_1x_2, x_2^2, \dots, x_n^2$ , ke dalam model regresi.

Sebagai contoh, jika kita memiliki dua variabel independen,  $x_1$  dan  $x_2$ , dan ingin membuat model polinomial hingga derajat dua, maka fungsi regresinya akan berbentuk:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_1 x_2 + \beta_5 x_2^2 + \epsilon$$

Secara umum, untuk  $k$  variabel independen  $(x_1, x_2, \dots, x_k)$  dengan derajat  $d$ , jumlah total suku dalam model polinomialnya ( $p$ ) dapat dihitung dengan rumus kombinasi:

$$p = \binom{d+k}{k} = \frac{(d+k)!}{d!k!}$$

Untuk menyelesaikan model regresi ini, kita dapat menuliskannya dalam bentuk matriks. Misalkan kita memiliki  $n$  set data pengamatan, maka model regresi polinomial berganda dapat diringkas menjadi persamaan:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \epsilon$$

Di mana:

- $\mathbf{y}$  adalah vektor variabel dependen berukuran  $n \times 1$ .
- $\mathbf{X}$  adalah matriks desain (matriks variabel independen beserta suku-suku polinomialnya) berukuran  $n \times p$ .
- $\boldsymbol{\beta}$  adalah vektor koefisien regresi yang ingin kita cari, berukuran  $p \times 1$ .
- $\epsilon$  adalah vektor galat (error) berukuran  $n \times 1$ .

Tujuan utama kita adalah menemukan vektor koefisien  $\boldsymbol{\beta}$  yang dapat meminimalkan jumlah kuadrat dari galat ( $\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2$ ). Dengan menggunakan metode *least squares*, kita dapat menurunkan persamaan normal berikut:

$$\mathbf{X}^T \mathbf{X} \boldsymbol{\beta} = \mathbf{X}^T \mathbf{y}$$

Dari persamaan normal tersebut, kita dapat memperoleh estimasi untuk vektor koefisien  $\boldsymbol{\beta}$ , yang biasa dinotasikan sebagai  $\hat{\boldsymbol{\beta}}$ , dengan rumus:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (2)$$

### 3 Implementasi

Program kami dibangun dalam folder src yang berisi subfolder modules. Pada subfolder tersebut, terdapat tujuh file utama yang terdiri atas lima file untuk fitur utama dan dua file untuk ADT Matrix beserta operator-operatornya. Pemisahan file dilakukan agar setiap modul memiliki tanggung jawab spesifik, sehingga memudahkan proses pengembangan, pengujian, dan pemeliharaan kode.

Berikut adalah penjelasan struktur folder dan file, peran masing-masing modul, serta uraian implementasi fungsi-fungsi utama yang mendukung keseluruhan sistem:

#### 3.1 Modules

Modules berisi implementasi ADT Matrix, beserta algoritma-algoritma yang akan dilakukan pada matriks. Package ini berisi 7 *class*, yaitu:

- Matrix.java
- MatrixOperator.java
- SPL.java
- SPLResult.java
- Determinant.java
- DeterminantResult.java
- CofactorResult.java
- Inverse.java
- InverseResult.java
- Regression.java
- Interpolation.java

##### 3.1.1 Matrix.java

###### Attribute

```
// Jumlah kolom dan baris matriks  
private int rows, cols  
  
// Matriks bernama data yang terdiri atas tipe data double  
private double[ ][ ] data
```

###### Constructor

```
// Menginisialisasi nilai rows dan cols, dalam tipe data integer  
public Matrix(int rows, int cols)  
  
// Membuat matriks bernama val, yang terdiri atas tipe data double  
public Matrix(double[ ][ ] val)
```

###### Selector

```
// Mengembalikan jumlah baris pada matriks  
public int getRowsCount()  
  
// Mengembalikan jumlah kolom pada matriks  
public int getColsCount()  
  
// Mengembalikan elemen matriks pada baris ke-i dan kolom ke-j  
public double getElmt(int i, int j)  
  
// Mengembalikan baris ke-r pada matriks dalam bentuk array
```

```
public double[ ] getRow(int r)
// Mengembalikan kolom ke-c pada matriks dalam bentuk array
public double[ ] getCol(int c)
```

#### Setter

```
// Mengubah elemen matriks pada baris ke-i dan kolom ke-j dengan nilai val
public void setElmt(int i, int j, double val)
```

```
// Mengubah seluruh elemen pada baris ke-r dengan elemen pada array values sesuai urutan
public void setRow(int r, double[] values)
```

```
// Mengubah seluruh elemen pada kolom ke-c dengan elemen pada array values sesuai urutan
public void setCol(int c, double[] values)
```

#### Methods

```
// Menampilkan matriks
public void displayMatrix()
```

```
// Menyalin matriks ke dalam matriks lain
public Matrix copyMatrix()
```

```
// Menukar baris ke-i dengan baris ke-j pada matriks
public void swapRow(int i, int j)
```

```
// Mengalikan seluruh elemen pada baris ke-row pada matriks dengan konstanta constant
public void scaleRow(int row, double constant)
```

```
// Menambahkan seluruh elemen pada baris ke-targetRow pada matriks dengan seluruh elemen pada baris ke-rowHelper pada matriks yang telah dikalikan dengan konstanta constant
public void addRowMultiple(int targetRow, int rowHelper, double constant)
```

Mengganti kolom ke-targetCol pada matriks dengan nilai pada matriks newValue yang berukuran 1 kolom

```
public Matrix replaceCol(int targetCol, Matrix newValue)
```

```
// Mengembalikan true jika matriks memiliki jumlah baris dan kolom yang sama, false jika tidak
public boolean isSquare()
```

```
// Mengembalikan hasil operasi transpose pada matriks
public Matrix transpose()
```

```
// Mengembalikan matriks identitas
public static Matrix identity
```

```
// Menghapus baris ke-row dan kolom ke-col pada matriks
public Matrix removeRowColMatrix(int row, int col)
```

```
// Menghapus kolom terakhir pada matriks
public Matrix removeLastCol()
```

```
// Menampilkan seluruh elemen matriks dalam format string
public String toString()
```

```
// Menggabungkan matriks A dan B menjadi sebuah matriks augmented
public static Matrix augment(Matrix A, Matrix B)
```

```
// Mengembalikan submatrix dengan baris dari startRow hingga endRow, dan kolom dari startCol hingga endCol
public Matrix subMatrix(int startRow, int endRow, int startCol, int endCol)
```

### 3.1.2 MatrixOperator.java

<b>Attribute</b>
-
<b>Methods</b>
// Mengalikan seluruh elemen pada matriks a dengan skalar scalar public static Matrix scalarMultiplication(Matrix a, double scalar)
// Membagi seluruh elemen pada matriks a dengan skalar scalar public static Matrix scalarDivision(Matrix a, double scalar)
// Mengalikan matriks a dengan matriks b public static Matrix matrixMultiplication(Matrix a, Matrix b)

### 3.1.3 SPL.java

<b>Attribute</b>
// Atribut yang mewakili batasan dimensi matriks, berupa konstanta bernilai 11 private static final int DIMENSION_THRESHOLD = 11
<b>Methods</b>
// Menyelesaikan SPL dengan metode eliminasi Gauss public static SPLResult gauss(Matrix augmented)
// Menyelesaikan SPL dengan metode eliminasi Gauss-Jordan public static SPLResult gaussJordan(Matrix augmented)
// Menyelesaikan SPL dengan kaidah Cramer public static SPLResult cramer(Matrix coeffMatrix, Matrix constMatrix)
// Menyelesaikan SPL dengan metode matriks balikan public static SPLResult inverseMethod(Matrix coef, Matrix constantM)
// Membentuk matriks eselon baris dan menyimpan langkah-langkahnya dalam array public static SPLResult echelonFormWithSteps(Matrix augmented)
// Membentuk matriks eselon baris terseduksi dan menyimpan langkah-langkahnya dalam array public static SPLResult reducedEchelonFormWithSteps(Matrix m)
// Melakukan eliminasi gauss pada matriks dan tidak menyimpan langkah-langkahnya dalam array public static Matrix gaussWithoutSteps(Matrix augmented)
// Melakukan eliminasi gauss jordan tanpa menyimpan langkah-langkahnya dalam array private static Matrix gaussJordanWithoutSteps(Matrix augmented)
// Menghasilkan solusi parametrik dan menyimpannya dalam matriks public static Matrix getParametricSolution(Matrix a, double[] b)
// Membuat semua elemen di bawah sebuah elemen pivot menjadi nol dalam sebuah kolom matriks public static Matrix zeroBelowPivot(Matrix m, int pivotRow, int pivotCol)
// Membentuk matriks eselon baris tereduksi tanpa menyimpan langkah-langkahnya dalam matriks public static Matrix reducedEchelonFormWithoutSteps(Matrix m)
// Mengembalikan indeks dari elemen tak nol pertama dalam baris sebuah matriks private static int idxNotZero(double[] row)

```

// Menghasilkan nilai true jika SPL tidak memiliki solusi, yaitu ketika terdapat baris nol pada matrix a tetapi
// nilai pada indeks yang sama di array b tidak nol, sehingga menimbulkan kontradiksi
public static boolean checkNoSolution(Matrix a, double[] b)

// Menghasilkan nilai true jika SPL memiliki tak hingga solusi, yaitu ketika terdapat baris nol pada matriks a
// dan nilai pada indeks yang sama di array b juga nol
public static boolean checkManySolution(Matrix a, double[] b)

// Menghasilkan nilai true jika SPL memiliki solusi yang tunggal
public static boolean checkUniqueSolution(Matrix a, double[] b)

// Melakukan penyulihan mundur dari matriks A, yang mewakili koefisien, dan array B, yang mewakili
// hasil persamaan, lalu menyimpannya dalam sebuah matriks
private static Matrix backSubstitute(Matrix A, double[] B)

```

### 3.1.4 SPLResult.java

#### Attribute

```

// Atribut yang menyimpan matriks solusi SPL
public final Matrix solution

// Atribut yang mewakili langkah-langkah penyelesaian SPL
public final String steps

```

#### Methods

```

// Metode ini membuat objek SPLResult yang menyimpan hasil penyelesaian SPL
public SPLResult(Matrix solution, String steps)

```

### 3.1.5 Determinant.java

#### Attributes

```

// Atribut yang mewakili batasan dimensi matriks, berupa konstanta bernilai 11
private static final int DIMENSION_THRESHOLD = 11

```

#### Methods

```

// Menghasilkan determinan dengan metode ekspansi kofaktor untuk matriks a
public static DeterminantResult detCofactor (Matrix a)

```

```

// Metode pembantu untuk menghitung determinan dengan metode ekspansi kofaktor, dan secara opsional
// mencatat langkah-langkah.
private static double detCofactorHelper(Matrix a, int depth, StringBuilder steps)

```

```

// Menghasilkan determinan dengan metode reduksi baris untuk matriks a
public static DeterminantResult detReduksiBaris(Matrix a)

```

```

// Menghasilkan determinan dengan metode reduksi baris untuk matriks a, tanpa menyimpan
// langkah-langkah penyelesaian
private static double detReduksiBarisWithoutSteps(Matrix a)

```

### 3.1.6 DeterminantResult.java

#### Attribute

```

// Atribut yang menyimpan nilai determinan matriks yang dihitung
public final double value

```

```
// Atribut yang mewakili langkah-langkah perhitungan nilai determinan
public final String steps
```

**Methods**

```
// Metode ini membuat objek DeterminantResult yang menyimpan hasil perhitungan nilai determinan
public DeterminantResult(double value, String steps)
```

### 3.1.7 CofactorResult.java

**Attribute**

```
// Atribut yang menyimpan matriks kofaktor
public final Matrix matrix
```

```
// Atribut yang mewakili langkah-langkah pembuatan matriks kofaktor
public final String steps
```

**Methods**

```
// Metode ini membuat objek CofactorResult yang menyimpan matriks kofaktor dan langkah-langkah
untuk mendapatkannya
public CofactorResult(Matrix matrix, String steps)
```

### 3.1.8 Inverse.java

**Attribute**

```
// Atribut yang mewakili batasan dimensi matriks, berupa konstanta bernilai 11
private static final int DIMENSION_THRESHOLD = 11
```

**Methods**

```
// Menghitung inverse dengan metode augment, yaitu dengan menyelesaikan [a | I] dimana I adalah
matriks identitas
```

```
public static InverseResult inverseAugment(Matrix a)
```

```
// Menghitung inverse dengan metode augment, tanpa menyimpan langkah-langkah penyelesaian
private static Matrix inverseAugmentWithoutSteps(Matrix a)
```

```
// Menghitung inverse dengan metode adjoin
```

```
public static InverseResult inverseAdjoin(Matrix a)
```

```
// Membentuk matriks kofaktor untuk matriks a
```

```
public static CofactorResult cofactorMatrix(Matrix a)
```

### 3.1.9 InverseResult.java

**Attribute**

```
// Atribut yang menyimpan matriks inverse
public final Matrix matrix
```

```
// Atribut yang mewakili langkah-langkah pembuatan matriks inverse
public final String steps
```

**Methods**

```
// Metode ini membuat objek InverseResult yang menyimpan matriks inverse dan langkah-langkahnya
public InverseResult(Matrix matrix, String steps)
```

### 3.1.10 Interpolation.java

#### Attribute

-

#### Methods

```
// Metode ini menghasilkan matriks yang menyimpan koefisien fungsi interpolasi polinomial dari masukan titik-titik interpolasi
public static Matrix polynomialInterpolation(Matrix points)

// Metode ini menghasilkan matriks yang menyimpan titik kontrol interpolasi splina bezier kubik dari masukan titik-titik interpolasi
public static Matrix[] interpolasiSplinaBezierKubik(Matrix points)

// Metode ini membuat tridiagonal matrix sesuai ukuran size
private static Matrix tridiagonalMatrix(int size)

// Metode pembantu untuk membuat vektor konstanta (sisi kanan dari SPL) dari titik-titik kontrol interpolasi splina bezier, berdasarkan formula spline kubik.
private static Matrix createVectorFromPoints(Matrix points, int col)
```

### 3.1.11 Regression.java

#### Attribute

-

#### Methods

```
// Metode ini menghitung koefisien regresi polinomial berganda, yang disimpan dalam bentuk matriks, menggunakan persamaan normal (Normal Equation)
public static Matrix multiRegression(Matrix X, Matrix y)
```

## 3.2 GUI

GUI berisi implementasi semua kelas yang dibutuhkan untuk membuat graphical user interface. Package ini berisi 11 *class*, yaitu:

- App.java
- MainMenuUI.java
- MainMenuController.java
- SubMenuController.java
- UIController.java
- FormatResult.java
- CalculationController.java
- InterpolationController.java
- RegressionController.java
- FileHandler.java
- MatrixParser.java

### 3.2.1 App.java

#### Attribute

-

#### Methods

```
// Titik masuk program saat dijalankan, metode ini memanggil launch(args) untuk memulai siklus hidup aplikasi
public static void main(String[] args)

// Metode yang dijalankan setelah JavaFX menyiapkan aplikasi, fungsinya meliputi pengaturan judul main window
app, pembuatan instance UIController dengan primaryStage agar UI dapat dikontrol, penampilan main menu app,
dan penampilan stage
public void start(Stage primaryStage)
```

### 3.2.2 MainMenuUI.java

#### Attribute

```
// Atribut ini merupakan objek pengendali (controller) yang mengatur navigasi antar scene
private final UIController controller
```

#### Constructor

```
// Membuat objek MainMenuUI baru dengan controller yang terhubung
public MainMenuUI(UIController controller)
```

#### Methods

```
//Metode ini membuat scene untuk menu utama app, yang berisi tombol "Sistem Persamaan Linear",
"Determinan Matriks", "Invers Matriks", "Regresi Polinomial Berganda", "Interpolasi", dan "Keluar"
public Scene createMainMenuScene()
```

```
// Metode ini membuat scene untuk menu SPL, yang meliputi tombol metode penyelesaiannya (Gauss, Gauss-Jordan, Matriks Balikan, Kaidah Cramer), dan tombol kembali
public Scene createSplMenuScene()
```

```
// Metode ini membuat scene untuk menu determinan matriks, yang meliputi tombol metode (Ekspansi kofaktor, dan reduksi baris), dan tombol kembali
public Scene createDeterminantMenuScene()
```

```
// Metode ini membuat scene untuk menu Inverse, yang meliputi tombol metode (Augment, dan adjoint), dan tombol kembali
public Scene createInverseMenuScene()
```

```
// Metode ini membuat layout menu dengan judul dan subjudul, serta mengatur padding, alignment, dan styling label.
private VBox createMenuLayout(String title, String subtitle)
```

```
// Metode ini membuat tombol menu standar
private Button createMenuButton(String text)
```

```
// Metode ini membuat tombol "Kembali ke Menu Utama"
private Button createBackButton()
```

```
// Metode ini membuat tombol "Keluar"
private Button createExitButton()
```

### 3.2.3 MainMenuController.java

**Attribute**

```
// Atribut yang meneruskan semua aksi dari menu ke UIController agar navigasi scene bisa dijalankan
private UIController uiController
```

**Methods**

```
// Memberikan referensi UIController ke controller ini
public void setUiController(UIController uiController)
```

```
// Navigasi ke menu Sistem Persamaan Linear
@FXML void showSplMenu(ActionEvent event)
```

```
// Navigasi ke menu Determinan Matriks
@FXML void showDeterminantMenu(ActionEvent event)
```

```
// Navigasi ke menu Invers Matriks
@FXML void showInverseMenu(ActionEvent event)
```

```
// Navigasi ke menu Interpolasi
@FXML void showInterpolationUI(ActionEvent event)
```

```
// Navigasi ke menu Regresi Polinomial Berganda
@FXML void showRegressionUI(ActionEvent event)
```

```
// Menutup aplikasi
@FXML void handleExit(ActionEvent event)
```

### 3.2.4 SubMenuController.java

**Attribute**

```
// Atribut ini mewakili label judul submenu yang ditampilkan di bagian atas scene
@FXML private Label titleLabel
```

```
// Atribut ini mewakili kontainer tempat tombol submenu, yang akan ditambahkan
@FXML private VBox buttonContainer
```

```
// Atribut ini mewakili referensi ke controller utama aplikasi, yang dipakai untuk navigasi
private UIController uiController
```

**Methods**

```
// Metode ini memberikan referensi UIController agar SubMenuController bisa mengakses logika navigasi
public void setUiController(UIController uiController)
```

```
// Metode ini mengubah label judul submenu sesuai title, menghapus semua isi dan menambahkan daftar tombol
baru ke dalam buttonContainer
```

```
public void configureMenu(String title, List<Button> buttons)
```

```
// Metode ini menangani aksi tombol "Kembali" untuk kembali ke menu utama app
@FXML void handleBackToMain(ActionEvent event)
```

### 3.2.5 UIController.java

**Attribute**

```
// Attribut ini mewakili stage utama aplikasi (main window). Semua scene ditampilkan melalui stage ini
private final Stage primaryStage
```

```
// Attribut ini digunakan untuk memilih file (buka/simpan)
private final FileChooser fileChooser = new FileChooser()
```

**Constructor**

```
// Menerima stage utama aplikasi, dan menginisialisasi fileChooser dengan filter .txt
public UIController(Stage primaryStage)
```

**Methods**

```
// Menampilkan menu utama
public void showMainMenu()
```

```
// Menampilkan submenu Sistem Persamaan Linear
public void showSplMenu()
```

```
// Menampilkan submenu Determinan Matriks
public void showDeterminantMenu()
```

```
// Menampilkan submenu Invers Matriks
public void showInverseMenu()
```

```
// Keluar dari aplikasi
public void exit()
```

```
// Menampilkan scene perhitungan generik
public void showCalculationScene(String type, String method)
```

```
// Menampilkan scene untuk interpolasi
public void showInterpolationUI()
```

```
// Menampilkan scene untuk regresi
public void showRegressionUI()
```

```
// Membuka submenu, memberikan referensi UIController ke SubMenuController, dan memanggil configureMenu
untuk mengisi subMenu
```

```
private void showSubMenu(String title, List<Button> menuButtons)
```

```
// Metode generik untuk load FXML, set controller, dan menampilkan scene
private <T> void loadScene(String fxmlPath, Class<T> controllerClass, Consumer<T> controllerInitializer)
```

```
// Membuat tombol dengan ukuran standar, yang diberi teks dan event handler
private Button createMenuItem(String text, javafx.event.EventHandler<javafx.event.ActionEvent> handler)
```

```
// Metode ini adalah metode file I/O yang membuka dialog file .txt, lalu membaca file matriks, dan menuliskan hasilnya ke TextArea
```

```
public void loadFileToTextArea(TextArea textView)
```

```
// Metode ini adalah metode file I/O yang membuka dialog simpan file, lalu menulis content ke file .txt
public void saveTextToFile(String content)
```

```
// Membuat alert bertipe error, dan menampilkan pesan kesalahan
public void showErrorDialog(String title, String message)
```

### 3.2.6 FormatResult.java

**Attribute**

-
<b>Methods</b>
// Metode ini mengubah objek Matrix menjadi string public static String matrixToString(Matrix m)
// Metode ini memformat hasil solusi SPL menjadi string public static String formatSolutionResult(Matrix solutionMatrix)
// Metode ini membentuk representasi string dari polinomial berdasarkan koefisien public static String buildPolynomialString(Matrix coeffs)

### 3.2.7 CalculationController.java

#### Attribute

```
// Atribut yang mewakili label judul yang menampilkan jenis perhitungan
@FXML private Label headerLabel

// Atribut yang mewakili tempat pengguna memasukkan matriks input
@FXML private TextArea inputArea

// Atribut yang mewakili tempat menampilkan hasil akhir perhitungan
@FXML private TextArea outputArea

// Atribut yang mewakili tempat menampilkan langkah-langkah perhitungan
@FXML private TextArea stepsArea

// Referensi ke controller utama aplikasi, untuk navigasi dan utilitas (load/save file, error dialog)
private UIController uiController

// Stage utama aplikasi, digunakan untuk mengatur scene
private Stage primaryStage

// Atribut yang mewakili jenis perhitungan (SPL,Determinan, atau Inverse)
private String type

// Atribut yang mewakili metode spesifik yang dipilih
private String method

// Aksi yang dijalankan ketika pengguna menekan tombol "Back"
private Runnable backAction
```

#### Methods

```
// Inisialisasi data controller sebelum ditampilkan
public void initData(String type, String method, UIController uiController, Stage stage, Runnable backAction)

// Membaca input matriks, melakukan parsing, lalu memanggil metode perhitungan yang sesuai berdasarkan
type, dan menampilkan hasil dan langkah-langkah
@FXML void handleCalculate(ActionEvent event)

// Membuka dialog pilih file dan memuat matriks ke area input
@FXML void handleLoadFile(ActionEvent event)

// Menyimpan input, metode, dan output ke file .txt
@FXML void handleSaveFile(ActionEvent event)

// Menjalankan aksi kembali
```

```

@FXML void handleBack(ActionEvent event)

// Menyelesaikan SPL berdasarkan method
private SPLResult solveSPL(Matrix augmentedMatrix)

// Menyelesaikan Determinan berdasarkan method
private DeterminantResult solveDeterminant(Matrix matrix)

// Menyelesaikan Inverse berdasarkan method
private InverseResult solveInverse(Matrix matrix)

// Memisahkan matriks augmented [A—b] menjadi 2 matriks: A (koefisien variabel), dan b (kolom konstanta).
private Matrix[] separateAugmentedMatrix(Matrix augmented)
    
```

### 3.2.8 InterpolationController.java

#### Attribute

```

// Grup radio button untuk memilih metode interpolasi (Polinomial atau Splina Bezier Kubik)
@FXML private ToggleGroup methodGroup

// Area teks input titik-titik interpolasi (x, y)
@FXML private TextArea inputArea

// Field input nilai x yang akan diprediksi hasil y-nya (hanya untuk metode polinomial)
@FXML private TextField estimateField

// Area teks untuk menampilkan hasil interpolasi, baik persamaan maupun titik kontrol
@FXML private TextArea outputArea

// Kontainer field estimasi
@FXML private HBox estimateBox

// Objek controller utama, yang berperan untuk navigasi layar, load file, save file, dan dialog error
private UIController uiController
    
```

#### Methods

```

// Metode ini mengatur agar estimateBox hanya terlihat saat metode Polinomial dipilih
@FXML public void initialize()

// Setter untuk menghubungkan controller ini dengan UIController
public void setUiController(UIController uiController)

// Membuka file dan memuat isinya ke area input
@FXML void handleLoadFile(ActionEvent event)

// Menyimpan hasil interpolasi (input, metode, output) ke filet
@FXML void handleSaveFile(ActionEvent event)

// Melakukan proses utama interpolasi, dari parsing titik dari area input ke matriks, menghitung sesuai
// metode, hingga menampilkan hasil ke area output
@FXML void handleCalculate(ActionEvent event)

// Menangani proses kembali
@FXML void handleBack(ActionEvent event)

// Menghitung nilai y dari polinomial dengan koefisien coeffs pada titik x
private double predictPolynomial(Matrix coeffs, double x)
    
```

```
// Membuat string hasil berupa daftar titik kontrol dari kurva Splina Bezier Kubik
private String formatBezierResult(Matrix[] segments)

// Menentukan domain valid interpolasi polinomial dengan mencari nilai minimum dan maksimum dari
x pada input
private static double[] findMinMaxX(Matrix M)
```

### 3.2.9 RegressionController.java

Attribute
// Area teks untuk input data regresi @FXML private TextArea infoArea
// @FXML private TextField estimateField
//Area teks untuk menampilkan hasil persamaan regresi @FXML private TextArea outputArea
// Objek controller utama, dipakai untuk navigasi menu, file handling, dan error dialog private UIController uiController
// Stage utama aplikasi, dipakai saat membuka file chooser private Stage primaryStage
// Menyimpan koefisien hasil regresi setelah perhitungan private Matrix coefficients
Methods
// Menghubungkan controller ini dengan UIController dan Stage utama aplikasi public void setUiController(UIController uiController, Stage primaryStage)
// Membuka file chooser, membaca file .txt, dan mengisi info area @FXML void handleLoadFile(ActionEvent event)
// @FXML void handleEstimate(ActionEvent event)
// Parsing teks input dan menghitung koefisien persamaan regresi @FXML void handleProcessRegression(ActionEvent event)
// private double predictRegression(double[] xValues)
// Membuat string persamaan regresi private String formatRegressionString(Matrix coeffs)
// Navigasi kembali @FXML void handleBack(ActionEvent event)
// Menyimpan hasil regresi ke file (input, metode, dan hasil persamaan) @FXML void handleSave(ActionEvent event)

### 3.2.10 FileHandler.java

<b>Attribute</b>
------------------

-

<b>Methods</b>
----------------

```
// Metode ini melakukan parsing teks input yang berisi data regresi (X, y, dan derajat polinomial)
public static RegressionInput parseRegresi(String inputText)
```

```
// Metode ini mengubah representasi angka dalam string ke double
private static double parseNumber(String token)
```

```
// Metode ini membaca file berisi data matriks, lalu memanggil metode untuk parsing
public static Matrix readMatrix(String filename) throws FileNotFoundException
```

```
// Metode ini melakukan parsing 1 baris string, yang dipisah spasi, menjadi array double
private static double[] parseRow(String line)
```

<b>Inner Class</b>
--------------------

```
// Kelas ini menyimpan hasil parsing regresi
public static class RegressionInput
```

<b>Inner Class' Attribute</b>
-------------------------------

```
// Atribut ini mewakili matriks fitur (input variables, dan kolom bias)
public Matrix X
```

```
// Atribut ini mewakili matriks target (output values)
public Matrix y
```

```
// Atribut ini mewakili derajat polinom regresi
public int derajatPolim
```

<b>Inner Class' Constructor</b>
---------------------------------

```
// Metode ini membuat objeks RegressionInput baru
public RegressionInput(Matrix X, Matrix y, int derajatPolim)
```

<b>Inner Class' Methods</b>
-----------------------------

-

### 3.2.11 MatrixParser.java

<b>Attribute</b>
------------------

-

<b>Methods</b>
----------------

```
// Melakukan parsing input menjadi objek Matrix
public static Matrix parseMatrix(String text) throws IllegalArgumentException
```

```
// Metode helper untuk mem-parsing sebuah string yang bisa berupa desimal atau pecahan
private static double parseNumber(String numberStr)
```

## 4 Eksperimen

### 4.1 Sistem Persamaan Linear

#### 4.1.1 Kasus Uji 1a

$$A = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 2 & 5 & -7 & -5 \\ 2 & -1 & 1 & 3 \\ 5 & 2 & -4 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ -2 \\ 4 \\ 6 \end{bmatrix}$$

##### a. Metode Eliminasi Gauss

Lintasjava Calculator

**SPL - Eliminasi Gauss**

Input:

1.000	1.000	-1.000	-1.000	1.000
2.000	5.000	-7.000	-5.000	-2.000
2.000	-1.000	1.000	3.000	4.000
5.000	2.000	-4.000	2.000	6.000

Output:

Sistem Persamaan Linier tidak memiliki solusi.

Langkah-langkah:

Menyelesaikan SPL dengan Eliminasi Gauss:

LANGKAH 1: Mengubah matriks ke bentuk Eselon Baris (Row Echelon Form).  
-> B2 = B2 - (2.000 \* B1)  
-> B3 = B3 - (2.000 \* B1)  
-> B4 = B4 - (5.000 \* B1)

Matriks setelah proses di kolom pivot 1:

1.000	1.000	-1.000	-1.000	1.000
0.000	3.000	-5.000	-3.000	-4.000
0.000	-3.000	3.000	5.000	2.000
0.000	-3.000	1.000	7.000	1.000

##### b. Metode Eliminasi Gauss-Jordan

**SPL - Eliminasi Gauss-Jordan**

Input:

1.000	1.000	-1.000	-1.000	1.000
2.000	5.000	-7.000	-5.000	-2.000
2.000	-1.000	1.000	3.000	4.000
5.000	2.000	-4.000	2.000	6.000

Output:

Sistem Persamaan Linier tidak memiliki solusi.

Langkah-langkah:

```
Menyelesaikan SPL dengan Eliminasi Gauss-Jordan:  
Mengubah ke Bentuk Eselon Baris Tereduksi (RREF):  
Matriks Awal:  
1.000 1.000 -1.000 -1.000 1.000  
2.000 5.000 -7.000 -5.000 -2.000  
2.000 -1.000 1.000 3.000 4.000  
5.000 2.000 -4.000 2.000 6.000  
-> B2 = B2 - (2.000 * B1)  
-> B3 = B3 - (2.000 * B1)  
-> B4 = B4 - (5.000 * B1)
```

### c. Metode Matriks Balikan

**SPL - Metode Matriks Balikan**

Input:

1.000	1.000	-1.000	-1.000	1.000
2.000	5.000	-7.000	-5.000	-2.000
2.000	-1.000	1.000	3.000	4.000
5.000	2.000	-4.000	2.000	6.000

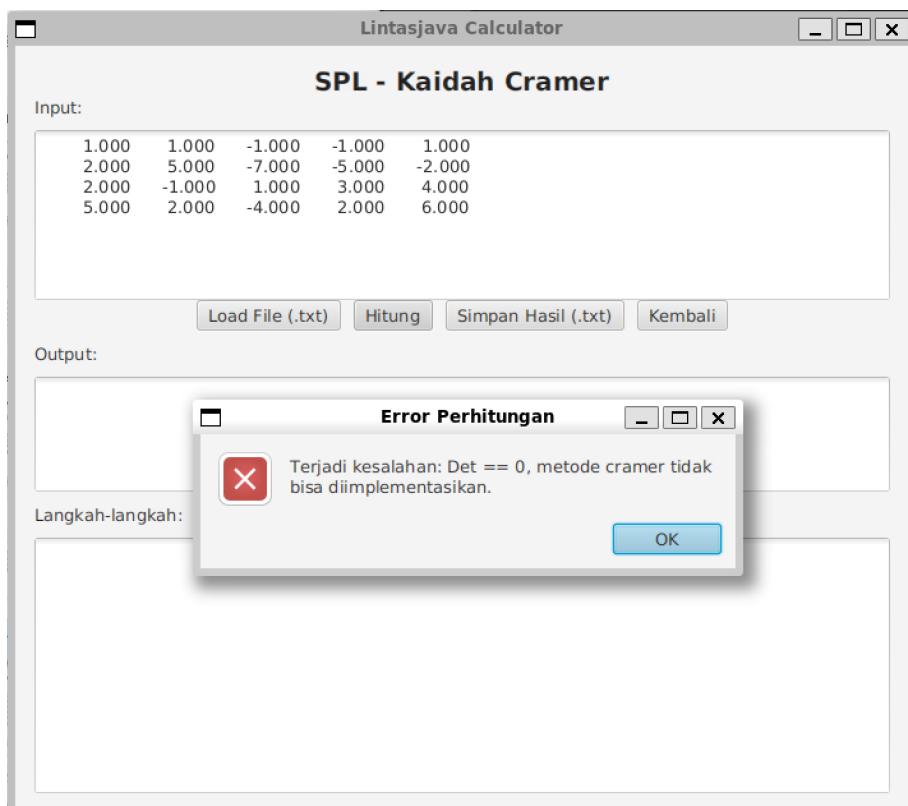
Output:

Sistem Persamaan Linier tidak memiliki solusi.

Langkah-langkah:

Matrix Singular, Invers tidak terdefinisi

### d. Kaidah Cramer



#### 4.1.2 Kasus Uji 2a

##### a. Metode Eliminasi Gauss

**Lintasjava Calculator**

### SPL - Eliminasi Gauss

Input:

```
1 -1 0 0 1 3
1 1 0 -3 0 6
2 -1 0 1 -1 5
-1 2 0 -2 -1 -1
```

Load File (.txt) Hitung Simpan Hasil (.txt) Kembali

Output:

```
terdapat banyak solusi (solusi parametrik).
x1 = 3.000 + s
x2 = 2.000s
x3 = t
x4 = -1.000 + s
x5 = s
```

Langkah-langkah:

Menyelesaikan SPL dengan Eliminasi Gauss:

```
LANGKAH 1: Mengubah matriks ke bentuk Eselon Baris (Row Echelon Form).
-> B2 = B2 - (1.000 * B1)
-> B3 = B3 - (2.000 * B1)
-> B4 = B4 - (-1.000 * B1)
Matriks setelah proses di kolom pivot 1:
 1.000   -1.000    0.000    0.000    1.000    3.000
 0.000    2.000    0.000   -3.000   -1.000    3.000
 0.000    1.000    0.000    1.000   -3.000   -1.000
 0.000    1.000    0.000   -2.000    0.000    2.000
```

#### b. Metode Eliminasi Gauss-Jordan

**SPL - Eliminasi Gauss-Jordan**

Input:

1.000	-1.000	0.000	0.000	1.000	3.000
1.000	1.000	0.000	-3.000	0.000	6.000
2.000	-1.000	0.000	1.000	-1.000	5.000
-1.000	2.000	0.000	-2.000	-1.000	-1.000

Output:

```
Terdapat banyak solusi (solusi parametrik):
x1 = 3.000 + s
x2 = 2.000s
x3 = t
x4 = -1.000 + s
x5 = s
```

Langkah-langkah:

```
Menyelesaikan SPL dengan Eliminasi Gauss-Jordan:
Mengubah ke Bentuk Eselon Baris Tereduksi (RREF):
Matriks Awal:
1.000 -1.000 0.000 0.000 1.000 3.000
1.000 1.000 0.000 -3.000 0.000 6.000
2.000 -1.000 0.000 1.000 -1.000 5.000
-1.000 2.000 0.000 -2.000 -1.000 -1.000
-> B2 = B2 - (1.000 * B1)
-> B3 = B3 - (2.000 * B1)
```

### c. Metode Matriks Balikan

**SPL - Metode Matriks Balikan**

Input:

1.000	-1.000	0.000	0.000	1.000	3.000
1.000	1.000	0.000	-3.000	0.000	6.000
2.000	-1.000	0.000	1.000	-1.000	5.000
-1.000	2.000	0.000	-2.000	-1.000	-1.000

Output:

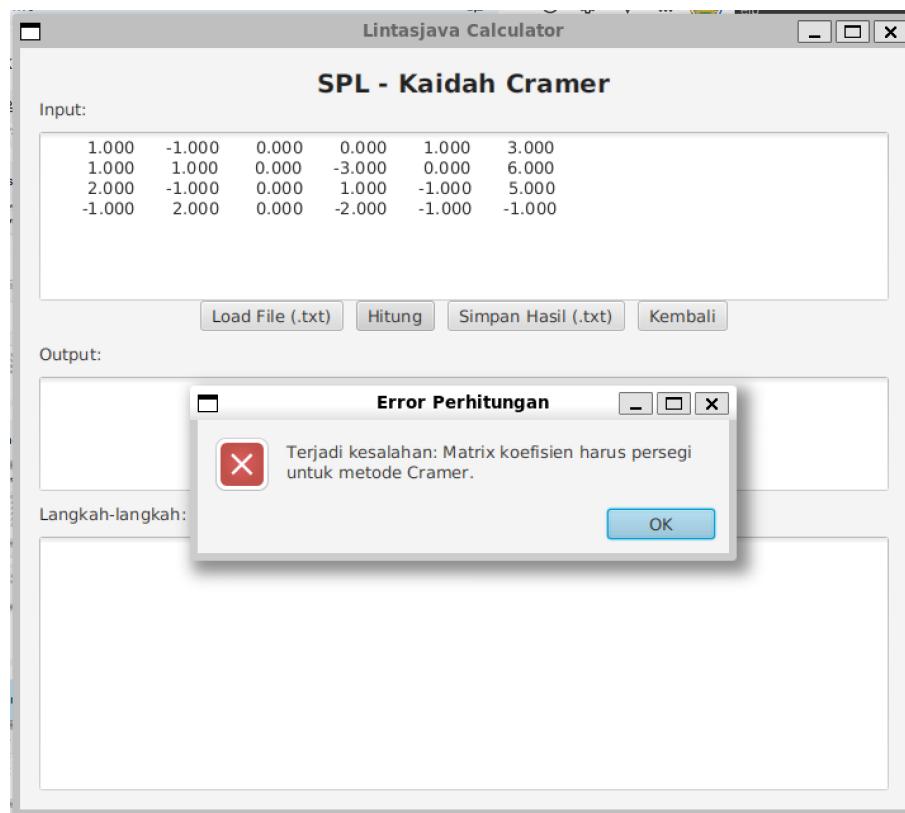
Langkah-langkah:

**Error Perhitungan**

Terjadi kesalahan: Matrix A harus persegi

**OK**

### d. Kaidah Cramer



#### 4.1.3 Kasus Uji 3a

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$$

a. Metode Eliminasi Gauss

**Lintasjava Calculator**

### SPL - Eliminasi Gauss

**Input:**

```
0.000 1.000 0.000 0.000 1.000 0.000 2.000
0.000 0.000 0.000 1.000 1.000 1.000 -1.000
0.000 1.000 0.000 0.000 0.000 1.000 1.000
```

**Output:**

```
x1 = t
x2 = 1.000 - r
x3 = s
x4 = -2.000 - 2.000r
x5 = 1.000 + r
x6 = r
```

**Langkah-langkah:**

Menyelesaikan SPL dengan Eliminasi Gauss:

LANGKAH 1: Mengubah matriks ke bentuk Eselon Baris (Row Echelon Form).  
 $\rightarrow B3 = B3 - (1.000 * B1)$

Matriks setelah proses di kolom pivot 2:

0.000	1.000	0.000	0.000	1.000	0.000	2.000
0.000	0.000	0.000	1.000	1.000	1.000	-1.000
0.000	0.000	0.000	0.000	-1.000	1.000	-1.000

Matriks setelah proses di kolom pivot 4:

0.000	1.000	0.000	0.000	1.000	0.000	2.000
0.000	0.000	0.000	1.000	1.000	1.000	-1.000
0.000	0.000	0.000	0.000	-1.000	1.000	-1.000

### b. Metode Eliminasi Gauss Jordan

**Lintasjava Calculator**

### SPL - Eliminasi Gauss-Jordan

**Input:**

```
0.000 1.000 0.000 0.000 1.000 0.000 2.000
0.000 0.000 0.000 1.000 1.000 1.000 -1.000
0.000 1.000 0.000 0.000 0.000 1.000 1.000
```

**Output:**

```
x1 = t
x2 = 1.000 - r
x3 = s
x4 = -2.000 - 2.000r
x5 = 1.000 + r
x6 = r
```

**Langkah-langkah:**

Menyelesaikan SPL dengan Eliminasi Gauss-Jordan:

Mengubah ke Bentuk Eselon Baris Tereduksi (RREF):

Matriks Awal:

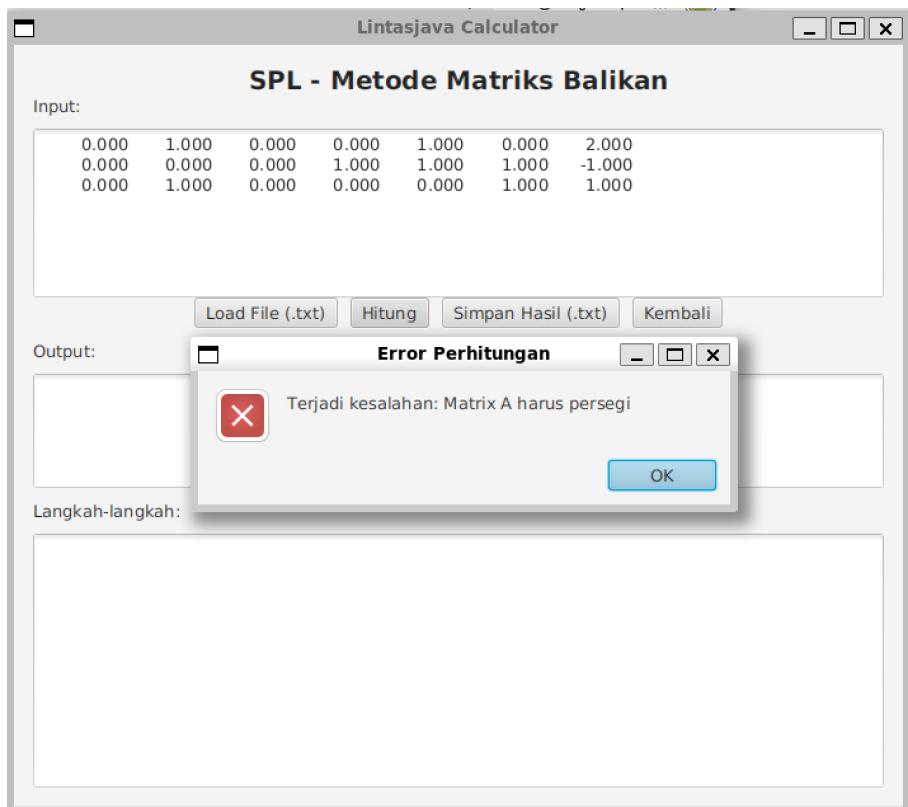
0.000	1.000	0.000	0.000	1.000	0.000	2.000
0.000	0.000	0.000	1.000	1.000	1.000	-1.000
0.000	1.000	0.000	0.000	0.000	1.000	1.000

$\rightarrow B3 = B3 - (1.000 * B1)$

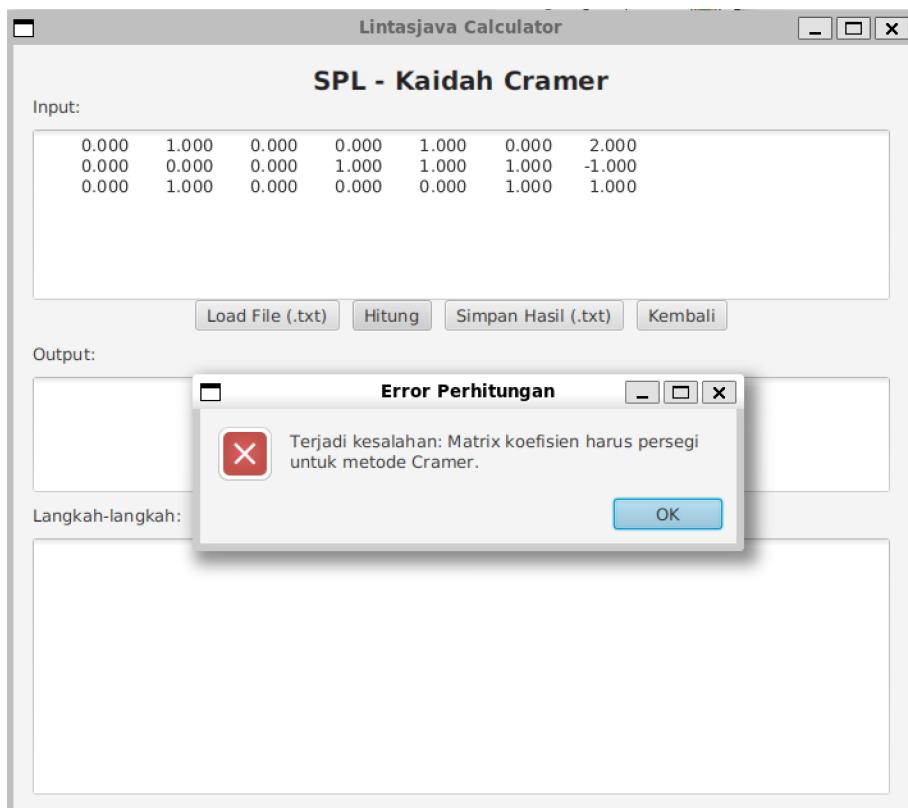
Matriks setelah proses di kolom pivot 2:

0.000	1.000	0.000	0.000	1.000	0.000	2.000
0.000	0.000	0.000	1.000	1.000	1.000	-1.000
0.000	0.000	0.000	0.000	0.000	1.000	1.000

### c. Metode Matriks Balikan



#### d. Kaidah Cramer



#### 4.1.4 Kasus Uji 4a

$$A = H = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots & \frac{1}{n+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n-1} \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$H$  adalah matriks Hilbert. Coba uji untuk nilai  $n = 6$  dan  $n = 10$ .

Untuk nilai  $n = 6$

a. Metode Eliminasi Gauss

**Lintasjava Calculator**

**SPL - Eliminasi Gauss**

**Input:**

```
1 1/2 1/3 1/4 1/5 1/6 1
1/2 1/3 1/4 1/5 1/6 1/7 0
1/3 1/4 1/5 1/6 1/7 1/8 0
1/4 1/5 1/6 1/7 1/8 1/9 0
1/5 1/6 1/7 1/8 1/9 1/10 0
1/6 1/7 1/8 1/9 1/10 1/11 0
```

**Output:**

```
x1 = 36.000
x2 = -630.000
x3 = 3360.000
x4 = -7560.000
x5 = 7560.000
x6 = -2772.000
```

**Langkah-langkah:**

```
Bentuk Eselon Baris tercapai.

LANGKAH 2: Menganalisis hasil matriks eselon baris dan melakukan substitusi balik.
-> Setiap variabel adalah variabel pivot. SPL memiliki solusi unik.

Solusi unik yang ditemukan:
36.000
-630.000
3360.000
-7560.000
7560.000
-2772.000
```

b. Metode Eliminasi Gauss-Jordan

**Lintasjava Calculator**

### SPL - Eliminasi Gauss-Jordan

**Input:**

```
1 1/2 1/3 1/4 1/5 1/6 1
1/2 1/3 1/4 1/5 1/6 1/7 0
1/3 1/4 1/5 1/6 1/7 1/8 0
1/4 1/5 1/6 1/7 1/8 1/9 0
1/5 1/6 1/7 1/8 1/9 1/10 0
1/6 1/7 1/8 1/9 1/10 1/11 0
```

**Output:**

```
x1 = 36.000
x2 = -630.000
x3 = 3360.000
x4 = -7560.000
x5 = 7560.000
x6 = -2772.000
```

**Langkah-langkah:**

```
Bentuk Eselon Baris Tereduksi (RREF) akhir tercapai.

Analisis Solusi dari matriks RREF:
-> Setiap variabel adalah variabel pivot. SPL memiliki solusi unik.

Solusi:
36.000
-630.000
3360.000
-7560.000
7560.000
-2772.000
```

### c. Metode Matriks Balikan

**Lintasjava Calculator**

### SPL - Metode Matriks Balikan

**Input:**

```
1 1/2 1/3 1/4 1/5 1/6 1
1/2 1/3 1/4 1/5 1/6 1/7 0
1/3 1/4 1/5 1/6 1/7 1/8 0
1/4 1/5 1/6 1/7 1/8 1/9 0
1/5 1/6 1/7 1/8 1/9 1/10 0
1/6 1/7 1/8 1/9 1/10 1/11 0
```

**Output:**

```
x1 = 36.000
x2 = -630.000
x3 = 3360.000
x4 = -7560.000
x5 = 7560.000
x6 = -2772.000
```

**Langkah-langkah:**

```
0.000
0.000
0.000
0.000
=
36.000
-630.000
3360.000
-7560.000
7560.000
-2772.000
```

### d. Kaidah Cramer

**Lintasjava Calculator**

### SPL - Kaidah Cramer

Input:

```
1 1/2 1/3 1/4 1/5 1/6 1
1/2 1/3 1/4 1/5 1/6 1/7 0
1/3 1/4 1/5 1/6 1/7 1/8 0
1/4 1/5 1/6 1/7 1/8 1/9 0
1/5 1/6 1/7 1/8 1/9 1/10 0
1/6 1/7 1/8 1/9 1/10 1/11 0
```

Output:

```
x1 = 36.000
x2 = -630.000
x3 = 3360.000
x4 = -7560.000
x5 = 7560.000
x6 = -2772.000
```

Langkah-langkah:

0.250	0.200	0.167	0.143	0.125	0.000
0.200	0.167	0.143	0.125	0.111	0.000
0.167	0.143	0.125	0.111	0.100	0.000

Hasil:  $\det(A_6) = -0.000$

Solusi akhir:

```
36.000
-630.000
3360.000
-7560.000
7560.000
-2772.000
```

Untuk nilai n = 10

#### a. Metode Eliminasi Gauss

**Lintasjava Calculator**

### SPL - Eliminasi Gauss

Input:

```
1/3 1/4 1/5 1/6 1/7 1/8 1/9 1/10 1/11 1/12 0
1/4 1/5 1/6 1/7 1/8 1/9 1/10 1/11 1/12 1/13 0
1/5 1/6 1/7 1/8 1/9 1/10 1/11 1/12 1/13 1/14 0
1/6 1/7 1/8 1/9 1/10 1/11 1/12 1/13 1/14 1/15 0
1/7 1/8 1/9 1/10 1/11 1/12 1/13 1/14 1/15 1/16 0
1/8 1/9 1/10 1/11 1/12 1/13 1/14 1/15 1/16 1/17 0
1/9 1/10 1/11 1/12 1/13 1/14 1/15 1/16 1/17 1/18 0
1/10 1/11 1/12 1/13 1/14 1/15 1/16 1/17 1/18 1/19 0
```

Output:

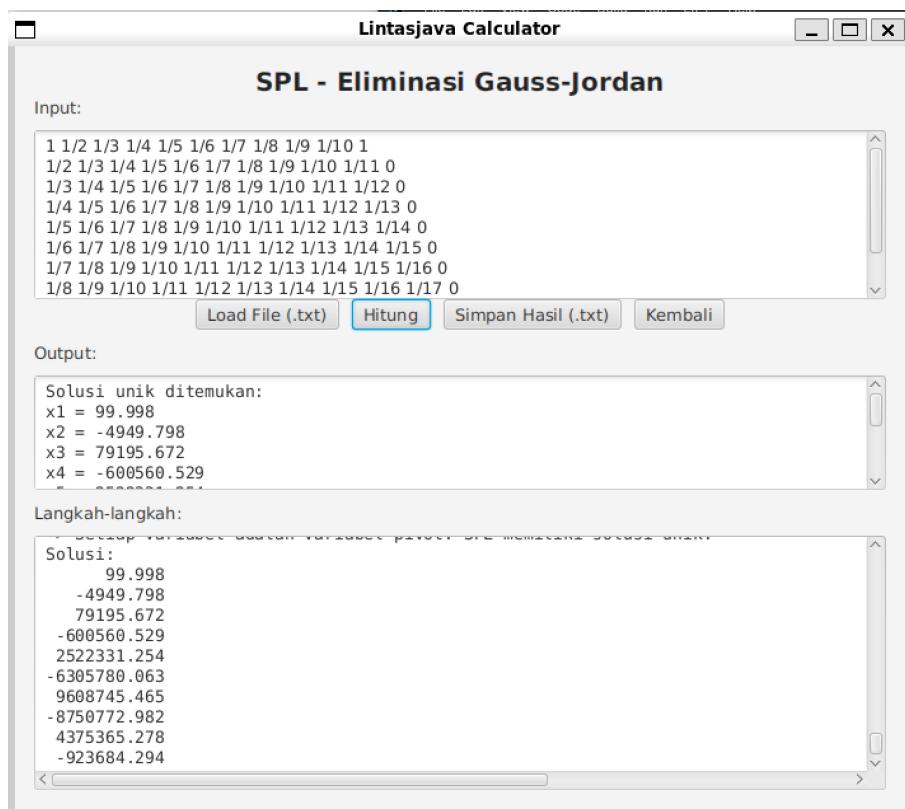
```
Solusi unik ditemukan:
x1 = 80.000
x2 = -3167.998
x3 = 40319.968
x4 = -240239.767
```

Langkah-langkah:

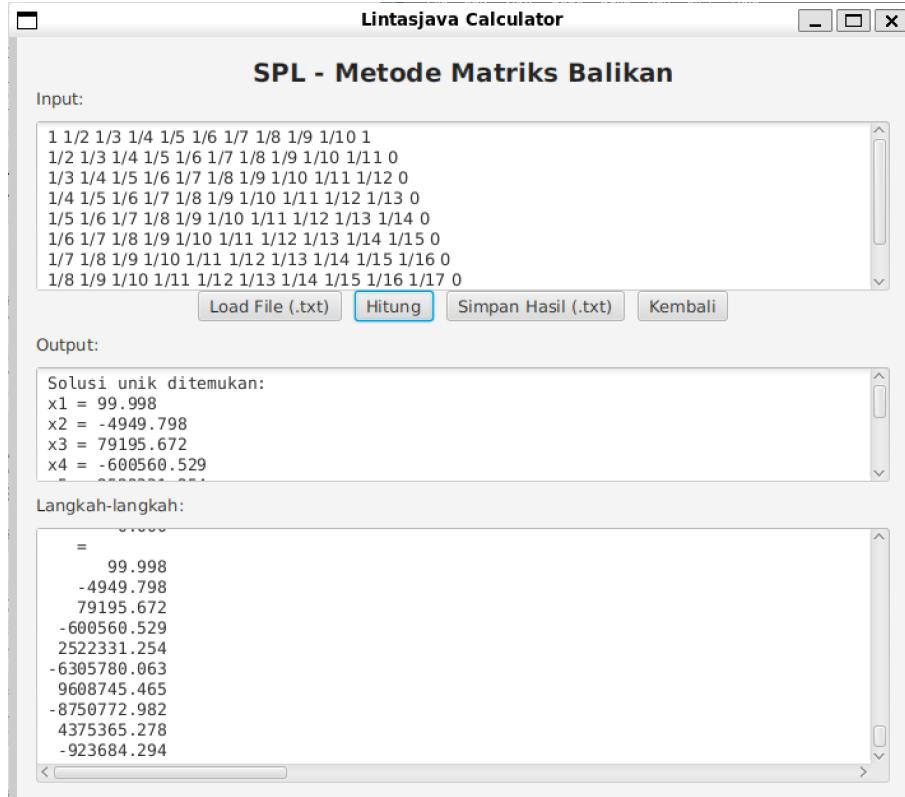
Solusi unik yang ditemukan:

```
80.000
-3167.998
40319.968
-240239.767
776159.137
-1441438.225
1537533.950
-875158.756
205919.692
0.000
```

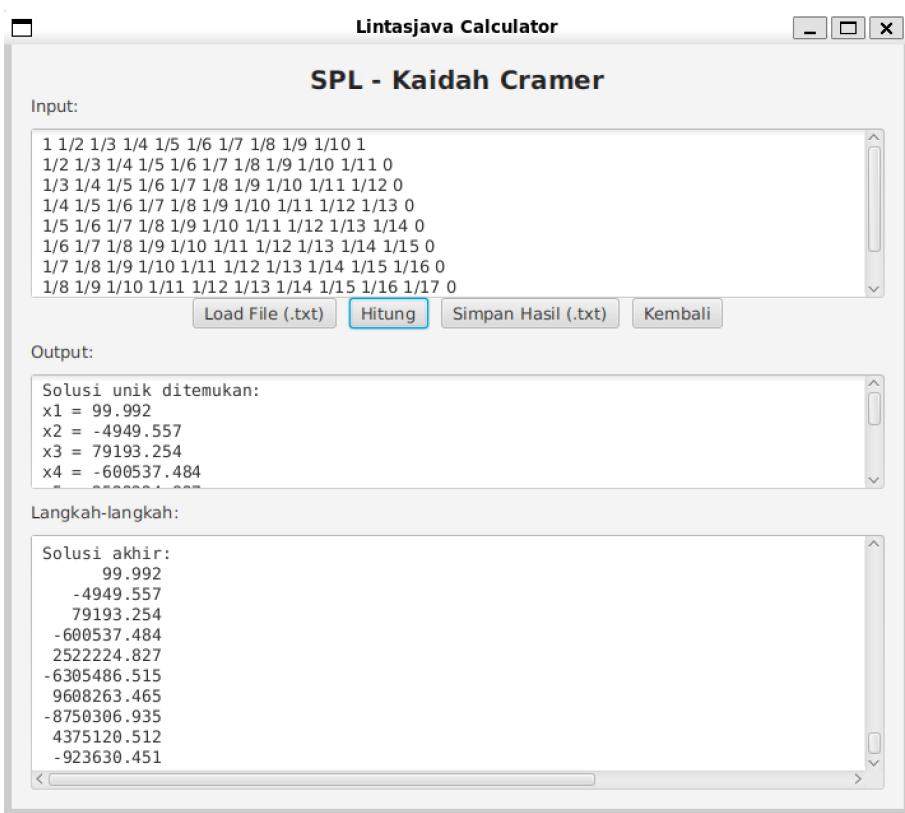
#### b. Metode Eliminasi Gauss-Jordan



### c. Metode Matriks Balikan



### d. Kaidah Cramer



#### 4.1.5 Kasus Uji 1b

$$\begin{bmatrix} 1 & -1 & 2 & -1 & -1 \\ 2 & 1 & -2 & -2 & -2 \\ -1 & 2 & -4 & 1 & 1 \\ 3 & 0 & 0 & -3 & -3 \end{bmatrix}$$

##### a. Metode Eliminasi Gauss

**SPL - Eliminasi Gauss**

**Input:**

```
1 -1 2 -1 -1
2 1 -2 -2 -2
-1 2 -4 1 1
3 0 0 -3 -3
```

**Output:**

```
Terdapat banyak solusi (solusi parametrik):
x1 = -1.000 + s
x2 = 2.000t
x3 = t
x4 = s
```

**Langkah-langkah:**

```
Menyelesaikan SPL dengan Eliminasi Gauss:
LANGKAH 1: Mengubah matriks ke bentuk Eselon Baris (Row Echelon Form).
-> B2 = B2 - (2.000 * B1)
-> B3 = B3 - (-1.000 * B1)
-> B4 = B4 - (3.000 * B1)
Matriks setelah proses di kolom pivot 1:
 1.000      -1.000      2.000      -1.000      -1.000
 0.000       3.000      -6.000      0.000      0.000
 0.000       1.000      -2.000      0.000      0.000
 0.000       3.000      -6.000      0.000      0.000
```

### b. Metode Eliminasi Gauss-Jordan

**SPL - Eliminasi Gauss-Jordan**

**Input:**

```
1.000  -1.000  2.000  -1.000  -1.000
2.000   1.000  -2.000  -2.000  -2.000
-1.000   2.000  -4.000   1.000   1.000
3.000   0.000   0.000  -3.000  -3.000
```

**Output:**

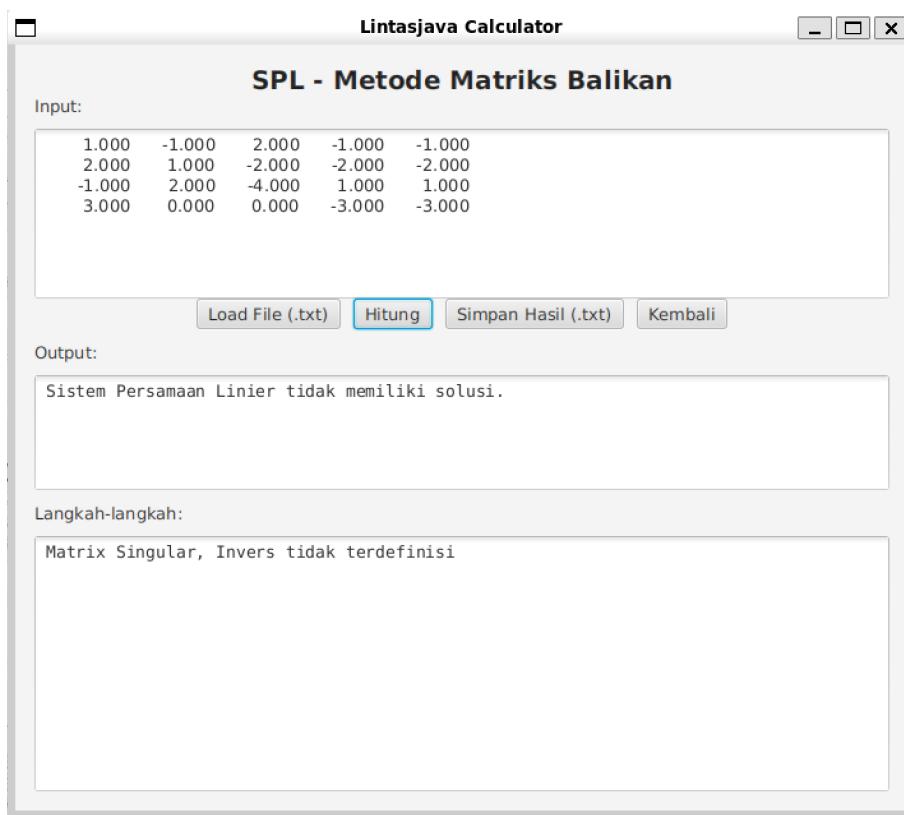
```
Terdapat banyak solusi (solusi parametrik):
x1 = -1.000 + s
x2 = 2.000t
x3 = t
x4 = s
```

**Langkah-langkah:**

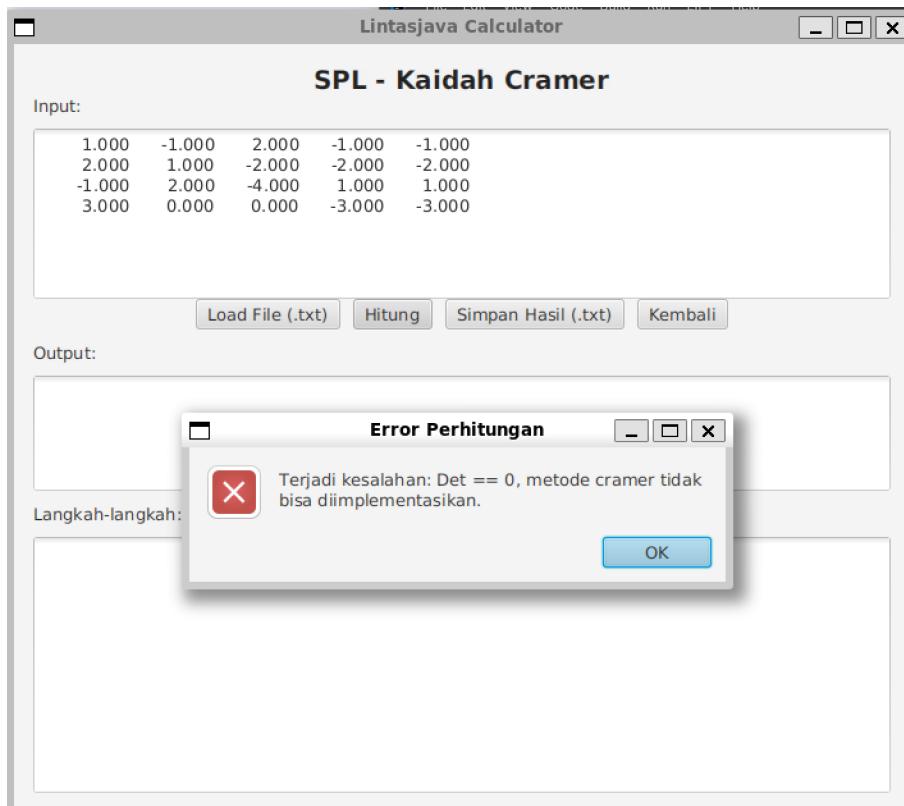
```
0.000      0.000      0.000      0.000      0.000
Bentuk Eselon Baris Tereduksi (RREF) akhir tercapai.

Analisis Solusi dari matriks RREF:
-> Terdapat variabel bebas. SPL memiliki banyak solusi (parametrik).
Solusi Parametrik (kolom pertama adalah konstanta, kolom berikutnya adalah parameter)
 -1.000      -0.000      1.000
  0.000       2.000      -0.000
  0.000       1.000      0.000
  0.000       0.000      1.000
```

### c. Metode Matriks Balikan



#### d. Kaidah Cramer



#### 4.1.6 Kasus Uji 2b

$$\left[ \begin{array}{ccccc} 2 & 0 & 8 & 0 & 8 \\ 0 & 1 & 0 & 4 & 4 \\ -4 & 0 & 6 & 0 & 6 \\ 0 & -2 & 0 & 3 & -1 \\ 2 & 0 & -4 & 0 & -4 \\ 0 & 1 & 0 & -2 & 0 \end{array} \right]$$

##### a. Metode Eliminasi Gauss

**SPL - Eliminasi Gauss**

Input:

```
2.000 0.000 8.000 0.000 8.000
0.000 1.000 0.000 4.000 4.000
-4.000 0.000 6.000 0.000 6.000
0.000 -2.000 0.000 3.000 -1.000
2.000 0.000 -4.000 0.000 -4.000
0.000 1.000 0.000 -2.000 0.000
```

Output:

Sistem Persamaan Linier tidak memiliki solusi.

Langkah-langkah:

Menyelesaikan SPL dengan Eliminasi Gauss:

LANGKAH 1: Mengubah matriks ke bentuk Eselon Baris (Row Echelon Form).  
 -> B1 = B1 / 2.000  
 -> B3 = B3 - (-4.000 \* B1)  
 -> B5 = B5 - (2.000 \* B1)

Matriks setelah proses di kolom pivot 1:

1.000	0.000	4.000	0.000	4.000
0.000	1.000	0.000	4.000	4.000
0.000	0.000	22.000	0.000	22.000
0.000	-2.000	0.000	3.000	-1.000

##### b. Metode Eliminasi Gauss-Jordan

**SPL - Eliminasi Gauss-Jordan**

Input:

2.000	0.000	8.000	0.000	8.000
0.000	1.000	0.000	4.000	4.000
-4.000	0.000	6.000	0.000	6.000
0.000	-2.000	0.000	3.000	-1.000
2.000	0.000	-4.000	0.000	-4.000
0.000	1.000	0.000	-2.000	0.000

Load File (.txt) Hitung Simpan Hasil (.txt) Kembali

Output:

Sistem Persamaan Linier tidak memiliki solusi.

Langkah-langkah:

Menyelesaikan SPL dengan Eliminasi Gauss-Jordan:  
Mengubah ke Bentuk Eselon Baris Tereduksi (RREF):  
Matriks Awal:

2.000	0.000	8.000	0.000	8.000
0.000	1.000	0.000	4.000	4.000
-4.000	0.000	6.000	0.000	6.000
0.000	-2.000	0.000	3.000	-1.000
2.000	0.000	-4.000	0.000	-4.000
0.000	1.000	0.000	-2.000	0.000

-> B1 = B1 / 2.000

### c. Metode Matriks Balikan

**SPL - Metode Matriks Balikan**

Input:

2.000	0.000	8.000	0.000	8.000
0.000	1.000	0.000	4.000	4.000
-4.000	0.000	6.000	0.000	6.000
0.000	-2.000	0.000	3.000	-1.000
2.000	0.000	-4.000	0.000	-4.000
0.000	1.000	0.000	-2.000	0.000

Load File (.txt) Hitung Simpan Hasil (.txt) Kembali

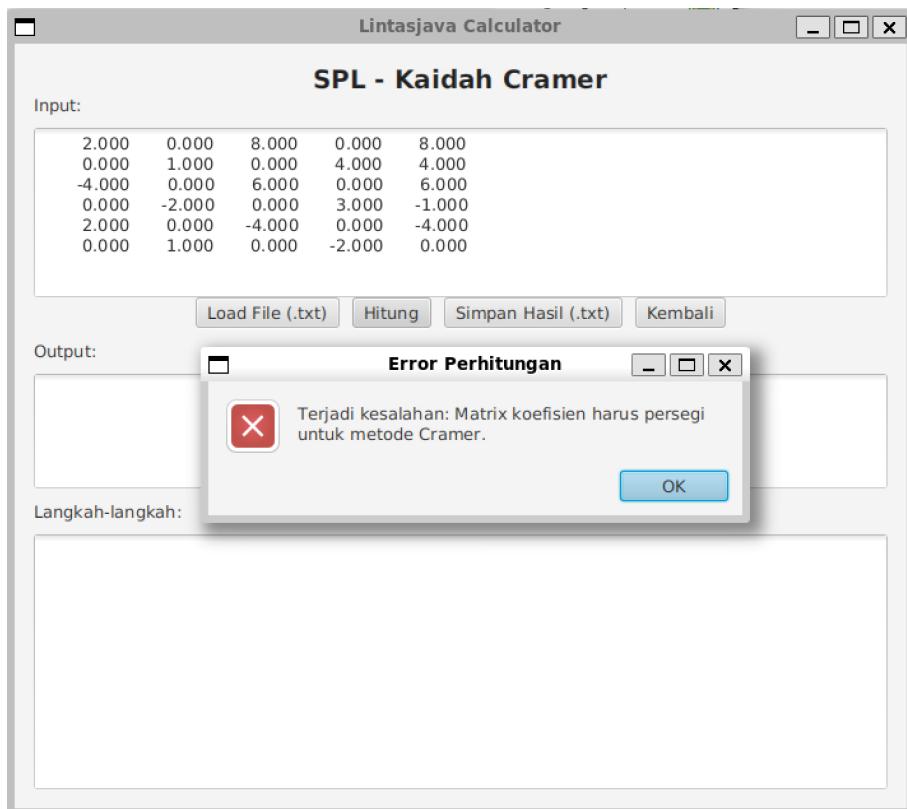
Output:

Langkah-langkah:

Terjadi kesalahan: Matrix A harus persegi

OK

### d. Kaidah Cramer



#### 4.1.7 Kasus Uji 1c

$$\begin{aligned} 8x_1 + x_2 + 3x_3 + 2x_4 &= 0 \\ 2x_1 + 9x_2 - x_3 - 2x_4 &= 1 \\ x_1 + 3x_2 + 2x_3 - x_4 &= 2 \\ x_1 + 6x_3 &+ 4x_4 = 3 \end{aligned}$$

##### a. Metode Eliminasi Gauss

**Lintasjava Calculator**

### SPL - Eliminasi Gauss

**Input:**

8.000	1.000	3.000	2.000	0.000
2.000	9.000	-1.000	-2.000	1.000
1.000	3.000	2.000	-1.000	2.000
1.000	0.000	6.000	4.000	3.000

**Output:**

```
Solusi unik ditemukan:
x1 = -0.224
x2 = 0.182
x3 = 0.709
x4 = -0.258
```

**Langkah-langkah:**

```
Menyelesaikan SPL dengan Eliminasi Gauss:
LANGKAH 1: Mengubah matriks ke bentuk Eselon Baris (Row Echelon Form).
-> B1 = B1 / 8.000
-> B2 = B2 - (2.000 * B1)
-> B3 = B3 - (1.000 * B1)
-> B4 = B4 - (1.000 * B1)
Matriks setelah proses di kolom pivot 1:
 1.000      0.125      0.375      0.250      0.000
 0.000      8.750     -1.750     -2.500      1.000
 0.000      2.875      1.625     -1.250      2.000
```

### b. Metode Eliminasi Gauss-Jordan

**Lintasjava Calculator**

### SPL - Eliminasi Gauss-Jordan

**Input:**

8.000	1.000	3.000	2.000	0.000
2.000	9.000	-1.000	-2.000	1.000
1.000	3.000	2.000	-1.000	2.000
1.000	0.000	6.000	4.000	3.000

**Output:**

```
Solusi unik ditemukan:
x1 = -0.224
x2 = 0.182
x3 = 0.709
x4 = -0.258
```

**Langkah-langkah:**

```
Menyelesaikan SPL dengan Eliminasi Gauss-Jordan:
Mengubah ke Bentuk Eselon Baris Tereduksi (RREF):
Matriks Awal:
 8.000      1.000      3.000      2.000      0.000
 2.000      9.000     -1.000     -2.000      1.000
 1.000      3.000      2.000     -1.000      2.000
 1.000      0.000      6.000      4.000      3.000

-> B1 = B1 / 8.000
-> B2 = B2 - (2.000 * B1)
-> B3 = B3 - (1.000 * B1)
```

### c. Metode Matriks Balikan

**SPL - Metode Matriks Balikan**

**Input:**

8.000	1.000	3.000	2.000	0.000
2.000	9.000	-1.000	-2.000	1.000
1.000	3.000	2.000	-1.000	2.000
1.000	0.000	6.000	4.000	3.000

**Output:**

```
Solusi unik ditemukan:
x1 = -0.224
x2 = 0.182
x3 = 0.709
x4 = -0.258
```

**Langkah-langkah:**

```
Menyelesaikan SPL dengan Metode Matriks Balikan: X = A-1 * B

LANGKAH 1: Cari matriks balikan (A-1).
Menghitung invers menggunakan metode Augment.

Langkah 1: Bentuk matriks augmented [A | I]:
8.000 1.000 3.000 2.000 1.000 0.000 0.000
2.000 9.000 -1.000 -2.000 0.000 1.000 0.000
1.000 3.000 2.000 -1.000 0.000 0.000 1.000
1.000 0.000 6.000 4.000 0.000 0.000 0.000
```

#### d. Kaidah Cramer

**SPL - Kaidah Cramer**

**Input:**

8.000	1.000	3.000	2.000	0.000
2.000	9.000	-1.000	-2.000	1.000
1.000	3.000	2.000	-1.000	2.000
1.000	0.000	6.000	4.000	3.000

**Output:**

```
Solusi unik ditemukan:
x1 = -0.224
x2 = 0.182
x3 = 0.709
x4 = -0.258
```

**Langkah-langkah:**

```
Menyelesaikan SPL dengan Kaidah Cramer: xi = det(Ai) / det(A)

LANGKAH 1: Menghitung determinan matriks koefisien (A)...
Hasil: det(A) = 740.000

LANGKAH 2: Hitung det(A1).
Matriks A1 (kolom 1 diganti):
0.000 1.000 3.000 2.000
1.000 9.000 -1.000 -2.000
2.000 3.000 2.000 -1.000
3.000 0.000 6.000 4.000
Hasil: det(A1) = -166.000
```

#### 4.1.8 Kasus Uji 2c

$$x_7 + x_8 + x_9 = 13.00$$

$$x_4 + x_5 + x_6 = 15.00$$

$$x_1 + x_2 + x_3 = 8.00$$

$$0.04289(x_3 + x_5 + x_7) + 0.75(x_6 + x_8) + 0.61396x_9 = 14.79$$

$$0.91421(x_3 + x_5 + x_7) + 0.25(x_2 + x_4 + x_6 + x_8) = 14.31$$

$$0.04289(x_3 + x_5 + x_7) + 0.75(x_2 + x_4) + 0.61396x_1 = 3.81$$

$$x_3 + x_6 + x_9 = 18.00$$

$$x_2 + x_5 + x_8 = 12.00$$

$$x_1 + x_4 + x_7 = 6.00$$

$$0.04289(x_1 + x_5 + x_9) + 0.75(x_2 + x_6) + 0.61396x_3 = 10.51$$

$$0.91421(x_1 + x_5 + x_9) + 0.25(x_2 + x_4 + x_6 + x_8) = 16.13$$

$$0.04289(x_1 + x_5 + x_9) + 0.75(x_4 + x_8) + 0.61396x_7 = 7.04$$

##### a. Metode Eliminasi Gauss

**Lintasjava Calculator**

**SPL - Eliminasi Gauss**

Input:

```
0 0 1 0 0 0 1 1 1 13.00
0 0 0 1 1 1 0 0 0 15.00
1 1 1 0 0 0 0 0 0 8.00
0 0 0 0 0.04289 0.75 0.75 0.04289 0.75 0.61396 14.79
0 0.25 0.91421 0.25 0.25 0.25 0.91421 0.25 0 14.31
0.61396 0.75 0.04289 0.75 0 0.75 0.04289 0 0 3.81
0 0 1 0 0 1 0 0 1 18.00
0 1 0 0 1 0 0 1 0 12.00
```

Load File (.txt) Hitung Simpan Hasil (.txt) Kembali

Output:

```
Solusi unik ditemukan:
x1 = 5.299
x2 = -5.299
x3 = 8.000
x4 = -4.703
```

Langkah-langkah:

```
Langkah-langkah tidak ditampilkan pada matriks besar.
```

**Lintasjava Calculator**

**SPL - Eliminasi Gauss**

Input:

```
0 0 1 0 0 0 1 1 1 13.00
0 0 0 1 1 1 0 0 0 15.00
1 1 1 0 0 0 0 0 0 8.00
0 0 0 0 0.04289 0.75 0.75 0.04289 0.75 0.61396 14.79
0 0.25 0.91421 0.25 0.25 0.25 0.91421 0.25 0 14.31
0.61396 0.75 0.04289 0.75 0 0.75 0.04289 0 0 3.81
0 0 1 0 0 1 0 0 1 18.00
0 1 0 0 1 0 0 1 0 12.00
```

Load File (.txt) Hitung Simpan Hasil (.txt) Kembali

Output:

```
x5 = 9.703
x6 = 10.000
x7 = 5.000
x8 = 0.000
x9 = 0.000
```

Langkah-langkah:

```
Langkah-langkah tidak ditampilkan pada matriks besar.
```

### b. Metode Eliminasi Gauss-Jordan

**Lintasjava Calculator**

### SPL - Eliminasi Gauss-Jordan

**Input:**

```
0 0 1 0 0 0 1 1 1 13.00
0 0 0 1 1 1 0 0 0 15.00
1 1 1 0 0 0 0 0 8.00
0 0 0 0 0.04289 0.75 0.75 0.04289 0.75 0.61396 14.79
0.25 0.91421 0.25 0.25 0.25 0.91421 0.25 0 14.31
0.61396 0.75 0.04289 0.75 0 0.75 0.04289 0 0 3.81
0 0 1 0 0 1 0 1 18.00
0 1 0 0 1 0 0 1 0 12.00
```

**Output:**

```
Solusi unik ditemukan:
x1 = -0.025
x2 = 8.025
x3 = 0.000
x4 = -7.356
```

**Langkah-langkah:**

Langkah-langkah tidak ditampilkan pada matriks besar.

**Lintasjava Calculator**

### SPL - Eliminasi Gauss-Jordan

**Input:**

```
0 0 1 0 0 0 1 1 1 13.00
0 0 0 1 1 1 0 0 0 15.00
1 1 1 0 0 0 0 0 8.00
0 0 0 0 0.04289 0.75 0.75 0.04289 0.75 0.61396 14.79
0.25 0.91421 0.25 0.25 0.25 0.91421 0.25 0 14.31
0.61396 0.75 0.04289 0.75 0 0.75 0.04289 0 0 3.81
0 0 1 0 0 1 0 1 18.00
0 1 0 0 1 0 0 1 0 12.00
```

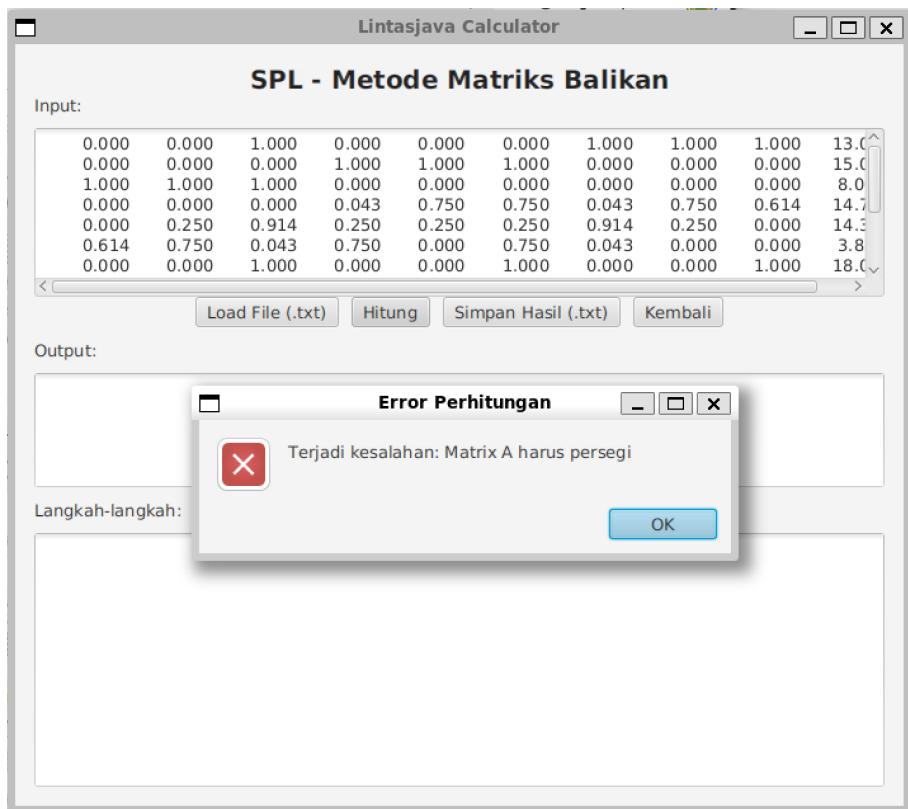
**Output:**

```
x5 = 18.690
x6 = 3.666
x7 = 13.380
x8 = -14.714
x9 = 14.334
```

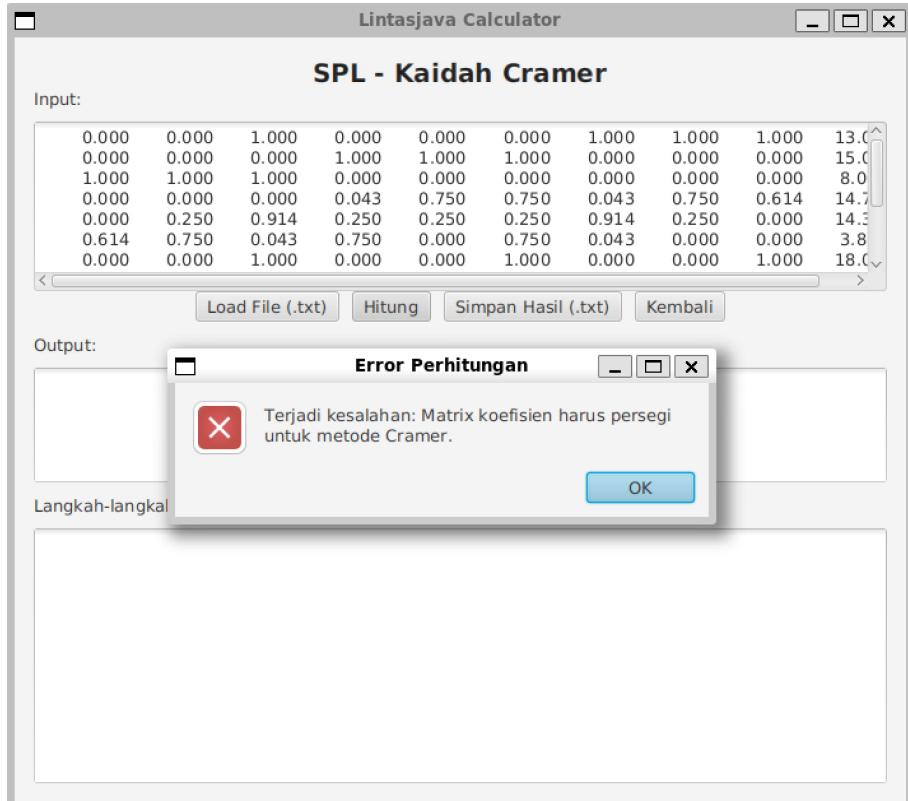
**Langkah-langkah:**

Langkah-langkah tidak ditampilkan pada matriks besar.

### c. Metode Matriks Balikan

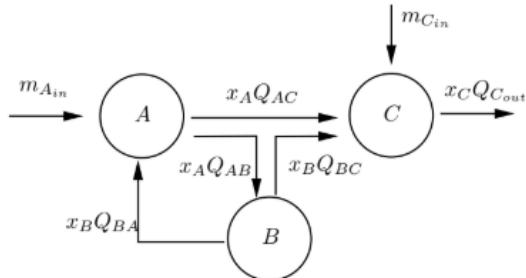


#### d. Kaidah Cramer



#### 4.1.9 Kasus Uji 1d

Lihatlah sistem reaktor pada gambar berikut:



Gambar 4: Model rangkaian reaktor

Dengan laju volume  $Q$  dalam  $\text{m}^3/\text{s}$  dan input massa min dalam  $\text{mg/s}$ . Konservasi massa pada tiap inti reaktor adalah sebagai berikut:

$$\begin{aligned}
 \text{A: } & m_{Ain} + Q_{BA}x_B - Q_{AB}x_A - Q_{AC}x_A = 0 \\
 \text{B: } & Q_{AB}x_A - Q_{BA}x_B - Q_{BC}x_B = 0 \\
 \text{C: } & m_{Cin} + Q_{AC}x_A + Q_{BC}x_B - Q_{Cout}x_C = 0
 \end{aligned}$$

Tentukan solusi  $x_A, x_B, x_C$  dengan menggunakan parameter berikut:

$Q_{AB} = 40, Q_{AC} = 80, Q_{BA} = 60, Q_{BC} = 20, Q_{Cout} = 150 \text{ m}^3/\text{s}, m_{Ain} = 1300$  dan  $m_{Cin} = 200 \text{ mg/s}$ .

##### a. Metode Eliminasi Gauss

**SPL - Eliminasi Gauss**

**Input:**

-120.000	60.000	0.000	-1300.000
40.000	-80.000	0.000	0.000
80.000	20.000	-150.000	-200.000

**Output:**

Solusi unik ditemukan:  
 $x_1 = 14.444$   
 $x_2 = 7.222$   
 $x_3 = 10.000$

**Langkah-langkah:**

Menyelesaikan SPL dengan Eliminasi Gauss:

LANGKAH 1: Mengubah matriks ke bentuk Eselon Baris (Row Echelon Form).  
 $\rightarrow B1 = B1 / -120.000$   
 $\rightarrow B2 = B2 - (40.000 * B1)$   
 $\rightarrow B3 = B3 - (80.000 * B1)$

Matriks setelah proses di kolom pivot 1:

1.000	-0.500	-0.000	10.833
0.000	-60.000	0.000	-433.333
0.000	60.000	-150.000	-1066.667

##### b. Metode Eliminasi Gauss-Jordan

**Lintasjava Calculator**

### SPL - Eliminasi Gauss-Jordan

Input:

```
-120.000 60.000 0.000 -1300.000
40.000 -80.000 0.000 0.000
80.000 20.000 -150.000 -200.000
```

Output:

Solusi unik ditemukan:  
 $x_1 = 14.444$   
 $x_2 = 7.222$   
 $x_3 = 10.000$

Langkah-langkah:

```
Menyelesaikan SPL dengan Eliminasi Gauss-Jordan:
Mengubah ke Bentuk Eselon Baris Tereduksi (RREF):
Matriks Awal:
-120.000 60.000 0.000 -1300.000
40.000 -80.000 0.000 0.000
80.000 20.000 -150.000 -200.000
-> B1 = B1 / -120.000
-> B2 = B2 - (40.000 * B1)
-> B3 = B3 - (80.000 * B1)
Matriks setelah proses di kolom pivot 1:
```

### c. Metode Matriks Balikan

**Lintasjava Calculator**

### SPL - Metode Matriks Balikan

Input:

```
-120.000 60.000 0.000 -1300.000
40.000 -80.000 0.000 0.000
80.000 20.000 -150.000 -200.000
```

Output:

Solusi unik ditemukan:  
 $x_1 = 14.444$   
 $x_2 = 7.222$   
 $x_3 = 10.000$

Langkah-langkah:

```
Menyelesaikan SPL dengan Metode Matriks Balikan:  $X = A^{-1} * B$ 
LANGKAH 1: Cari matriks balikan ( $A^{-1}$ ).
Menghitung invers menggunakan metode Augment.
Langkah 1: Bentuk matriks augmented [A | I]:
-120.000 60.000 0.000 1.000 0.000 0.000
40.000 -80.000 0.000 0.000 1.000 0.000
80.000 20.000 -150.000 0.000 0.000 1.000
Langkah 2: Lakukan OBE untuk mengubah matriks menjadi [I |  $A^{-1}$ ]:
Mengubah ke Bentuk Eselon Baris Tereduksi (RREF):
```

### d. Kaidah Cramer

**Lintasjava Calculator**

**SPL - Kaidah Cramer**

Input:

```
-120.000  60.000   0.000 -1300.000
 40.000  -80.000   0.000   0.000
 80.000   20.000  -150.000 -200.000
```

Load File (.txt) Hitung Simpan Hasil (.txt) Kembali

Output:

```
Solusi unik ditemukan:
x1 = 14.444
x2 = 7.222
x3 = 10.000
```

Langkah-langkah:

```
Menyelesaikan SPL dengan Kaidah Cramer: xi = det(Ai) / det(A)

LANGKAH 1: Menghitung determinan matriks koefisien (A)...
Hasil: det(A) = -1080000.000

LANGKAH 2: Hitung det(A1).
Matriks A1 (kolom 1 diganti):
 -1300.000    60.000    0.000
  0.000   -80.000    0.000
 -200.000   20.000  -150.000
Hasil: det(A1) = -15600000.000
```

## 4.2 Determinan

### 4.2.1 Kasus Uji 1

$$\begin{bmatrix} 5 & 7 & 9 & 11 \\ 10 & 14 & 18 & 22 \\ 15 & 21 & 27 & 33 \\ 20 & 28 & 36 & 44 \end{bmatrix}$$

#### a. Metode Ekspansi Kofaktor

**Lintasjava Calculator**

### Determinan - Ekspansi Kofaktor

**Input:**

5.000	7.000	9.000	11.000
10.000	14.000	18.000	22.000
15.000	21.000	27.000	33.000
20.000	28.000	36.000	44.000

**Output:**

Nilai Determinan: 0.000

**Langkah-langkah:**

```
Menghitung determinan dengan Metode Ekspansi Kofaktor:

Matriks Awal:
5.000    7.000    9.000    11.000
10.000   14.000   18.000   22.000
15.000   21.000   27.000   33.000
20.000   28.000   36.000   44.000

-> Menghitung determinan matriks 4x4
det = (5.000 * C11) + (7.000 * C12) + (9.000 * C13) + (11.000 * C14)
-> Menghitung determinan matriks 3x3
det = (14.000 * C11) + (18.000 * C12) + (22.000 * C13)
-> Menghitung determinan matriks 2x2
```

### b. Metode Reduksi Baris

**Lintasjava Calculator**

### Determinan - Reduksi Baris

**Input:**

5.000	7.000	9.000	11.000
10.000	14.000	18.000	22.000
15.000	21.000	27.000	33.000
20.000	28.000	36.000	44.000

**Output:**

Nilai Determinan: 0.000

**Langkah-langkah:**

```
Matriks Awal:
5.000    7.000    9.000    11.000
10.000   14.000   18.000   22.000
15.000   21.000   27.000   33.000
20.000   28.000   36.000   44.000

-> B2 = B2 - (2.000 * B1)
-> B3 = B3 - (3.000 * B1)
-> B4 = B4 - (4.000 * B1)
Matriks setelah eliminasi kolom 1:
5.000    7.000    9.000    11.000
0.000    0.000    0.000    0.000
```

#### 4.2.2 Kasus Uji 2

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

##### a. Metode Ekspansi Kofaktor

The screenshot shows a window titled "Determinan - Ekspansi Kofaktor". The input field contains the matrix elements: 1 1/2 1/3 1/4; 1/2 1/3 1/4 1/5; 1/3 1/4 1/5 1/6; 1/4 1/5 1/6 1/7. The output field displays "Nilai Determinan: 0.000". The steps section shows the calculation process:

```

Menghitung determinan dengan Metode Ekspansi Kofaktor:

Matriks Awal:
1.000 0.500 0.333 0.250
0.500 0.333 0.250 0.200
0.333 0.250 0.200 0.167
0.250 0.200 0.167 0.143

-> Menghitung determinan matriks 4x4
det = (1.000 * C11) + (0.500 * C12) + (0.333 * C13) + (0.250 * C14)
-> Menghitung determinan matriks 3x3
det = (0.333 * C11) + (0.250 * C12) + (0.200 * C13)
-> Menghitung determinan matriks 2x2

```

##### b. Metode Reduksi Baris

**Determinan - Reduksi Baris**

**Input:**

```
1 1/2 1/3 1/4
1/2 1/3 1/4 1/5
1/3 1/4 1/5 1/6
1/4 1/5 1/6 1/7
```

**Output:**

```
Nilai Determinan: 0.000
```

**Langkah-langkah:**

```
Matriks Awal:
1.000 0.500 0.333 0.250
0.500 0.333 0.250 0.200
0.333 0.250 0.200 0.167
0.250 0.200 0.167 0.143

-> B2 = B2 - (0.500 * B1)
-> B3 = B3 - (0.333 * B1)
-> B4 = B4 - (0.250 * B1)
Matriks setelah eliminasi kolom 1:
1.000 0.500 0.333 0.250
0.000 0.083 0.083 0.075
```

## 4.3 Inverse

### 4.3.1 Kasus Uji 1

$$\begin{bmatrix} 5 & 2 & -3 & 4 \\ 7 & 7 & 8 & -9 \\ 12 & 13 & 2 & 14 \\ 17 & 18 & 19 & 4 \end{bmatrix}$$

#### a. Metode Augment

**Lintasjava Calculator**

### Invers - Metode Augment

Input:

5.000	2.000	-3.000	4.000
7.000	7.000	8.000	-9.000
12.000	13.000	2.000	14.000
17.000	18.000	19.000	4.000

Output:

Matriks Invers:
0.314      -0.041      -0.139      0.081
-0.303      0.162      0.231      -0.140
0.004      -0.095      -0.092      0.103
0.012      -0.102      -0.010      0.046

Langkah-langkah:

```
Menghitung invers menggunakan metode Augment.

Langkah 1: Bentuk matriks augmented [A | I]:
5.000      2.000      -3.000      4.000      1.000      0.000      0.000
7.000      7.000       8.000      -9.000      0.000      1.000      0.000
12.000     13.000      2.000      14.000      0.000      0.000      1.000
17.000     18.000      19.000      4.000      0.000      0.000      0.000

Langkah 2: Lakukan OBE untuk mengubah matriks menjadi [I | A⁻¹]:
Mengubah ke Bentuk Eselon Baris Tereduksi (RREF):
Matriks Awal:
<table border="1">
<tr><td>5.000</td><td>2.000</td><td>-3.000</td><td>4.000</td><td>1.000</td><td>0.000</td><td>0.000</td></tr>
<tr><td>7.000</td><td>7.000</td><td>8.000</td><td>-9.000</td><td>0.000</td><td>1.000</td><td>0.000</td></tr>
<tr><td>12.000</td><td>13.000</td><td>2.000</td><td>14.000</td><td>0.000</td><td>0.000</td><td>1.000</td></tr>
<tr><td>17.000</td><td>18.000</td><td>19.000</td><td>4.000</td><td>0.000</td><td>0.000</td><td>0.000</td></tr>
</table>
```

## b. Metode Adjoint

**Lintasjava Calculator**

### Invers - Metode Adjoint

Input:

5.000	2.000	-3.000	4.000
7.000	7.000	8.000	-9.000
12.000	13.000	2.000	14.000
17.000	18.000	19.000	4.000

Output:

Matriks Invers:
0.314      -0.041      -0.139      0.081
-0.303      0.162      0.231      -0.140
0.004      -0.095      -0.092      0.103
0.012      -0.102      -0.010      0.046

Langkah-langkah:

```
Menghitung invers metode Adjoin: A⁻¹ = (1/det(A)) * Adj(A)

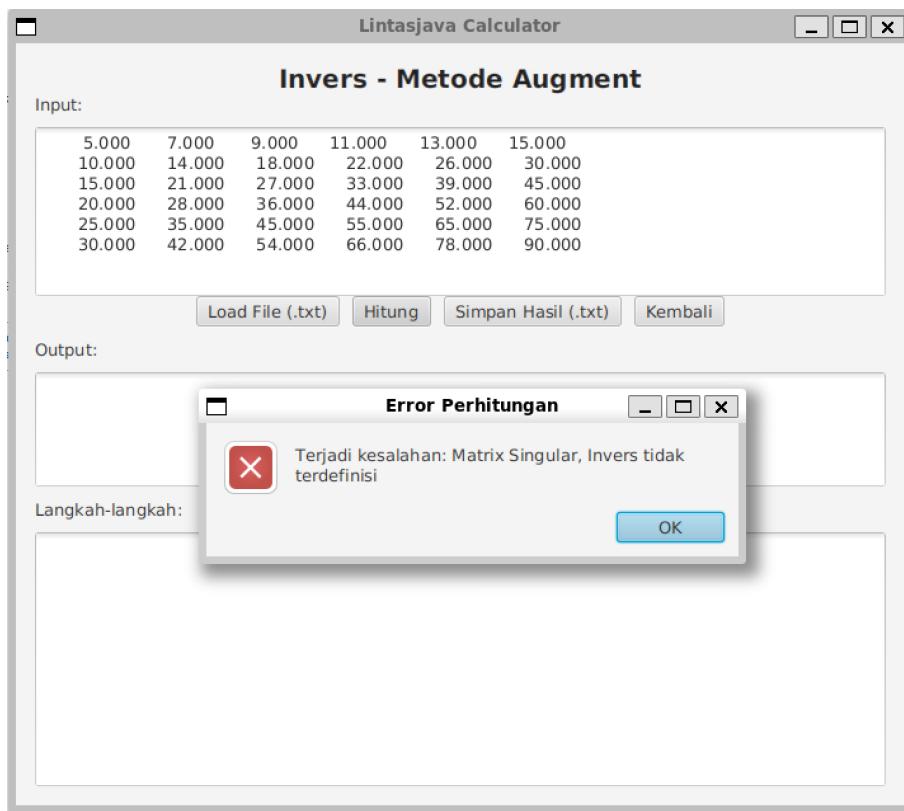
LANGKAH 1: Menghitung determinan matriks A...
Hasil: det(A) = -6702.000

LANGKAH 2: Membentuk matriks Adjoin (Adj(A) = Cᵀ).
Membentuk Matriks Kofaktor:
- Menghitung Kofaktor C(1,1)
-> det(Minor) = -2105.000 -> Kofaktor = -2105.000
- Menghitung Kofaktor C(1,2)
-> det(Minor) = -2032.000 -> Kofaktor = 2032.000
- Menghitung Kofaktor C(1,3)
```

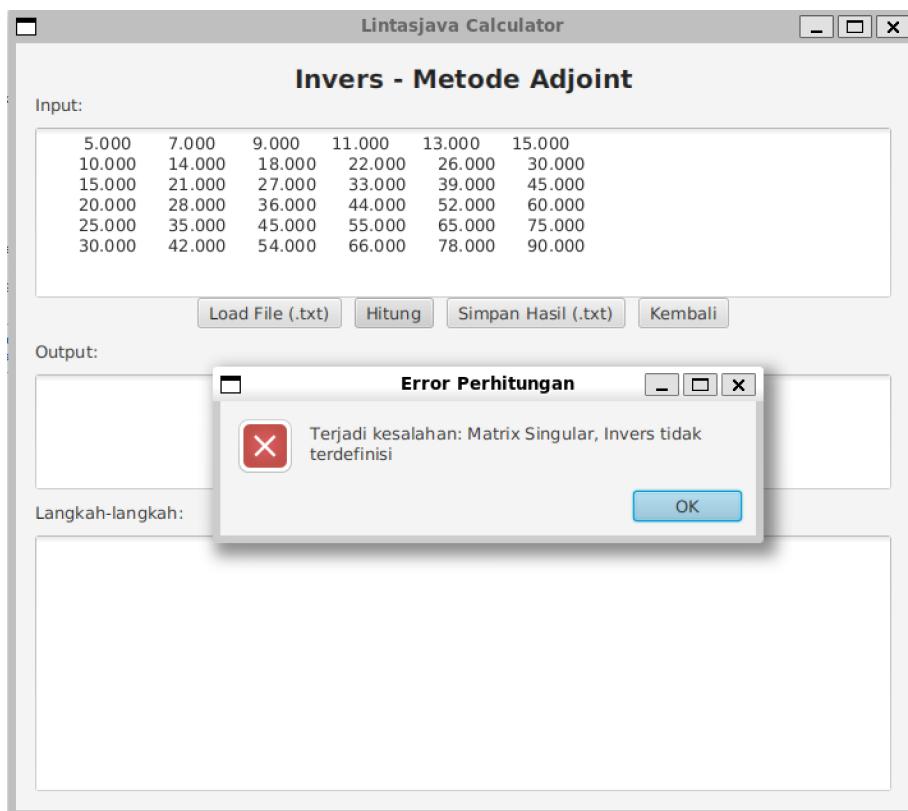
#### 4.3.2 Kasus Uji 2

$$\begin{bmatrix} 5 & 7 & 9 & 11 & 13 & 15 \\ 10 & 14 & 18 & 22 & 26 & 30 \\ 15 & 21 & 27 & 33 & 39 & 45 \\ 20 & 28 & 36 & 44 & 52 & 60 \\ 25 & 35 & 45 & 55 & 65 & 75 \\ 30 & 42 & 54 & 66 & 78 & 90 \end{bmatrix}$$

##### a. Metode Augment



##### b. Metode Adjoint



## 4.4 Interpolasi Polinomial

### 4.4.1 Kasus Uji 1

Gunakan tabel di bawah ini untuk mencari polinom interpolasi dari pasangan titik-titik yang terdapat dalam tabel. Program menerima masukan nilai  $x$  yang akan dicari nilai fungsi  $f(x)$ .

$x$	0.1	0.3	0.5	0.7	0.9	1.1	1.3
$f(x)$	0.003	0.067	0.148	0.248	0.370	0.518	0.697

Lakukan pengujian pada nilai-nilai berikut:

$$x = 0.2 \quad f(x) = ?$$

$$x = 0.55 \quad f(x) = ?$$

$$x = 0.85 \quad f(x) = ?$$

$$x = 1.28 \quad f(x) = ?$$

Untuk  $x = 0.2$ :

### Interpolasi

Pilih Metode:  Polinomial  Spline Bezier Kubik

Input Titik:

0.100	0.003
0.300	0.067
0.500	0.148
0.700	0.248
0.900	0.370
1.100	0.518
1.300	0.697

Taksir  $f(x)$  untuk  $x$ :

Output:

```
Persamaan Polinomial:  
y(x) = 0.026x^4 + 0.197x^2 + 0.240x - 0.023  
  
Domain prediksi interpolasi yang valid: [0.100, 1.300]  
  
Hasil Taksiran:  
f(0.200) = 0.033
```

Untuk  $x = 0.55$ :

### Interpolasi

Pilih Metode:  Polinomial  Spline Bezier Kubik

Input Titik:

0.100	0.003
0.300	0.067
0.500	0.148
0.700	0.248
0.900	0.370
1.100	0.518
1.300	0.697

Taksir  $f(x)$  untuk  $x$ :

Output:

```
Persamaan Polinomial:  
y(x) = 0.026x^4 + 0.197x^2 + 0.240x - 0.023  
  
Domain prediksi interpolasi yang valid: [0.100, 1.300]  
  
Hasil Taksiran:  
f(0.550) = 0.171
```

Untuk  $x = 0.85$ :

### Interpolasi

Pilih Metode:  Polinomial  Spline Bezier Kubik

Input Titik:

0.100	0.003
0.300	0.067
0.500	0.148
0.700	0.248
0.900	0.370
1.100	0.518
1.300	0.697

Taksir  $f(x)$  untuk  $x$ :

Output:

Persamaan Polinomial:  
 $y(x) = 0.026x^4 + 0.197x^2 + 0.240x - 0.023$

Domain prediksi interpolasi yang valid: [0.100, 1.300]

Hasil Taksiran:  
 $f(0.850) = 0.337$

untuk  $x = 1.28$ :

### Interpolasi

Pilih Metode:  Polinomial  Spline Bezier Kubik

Input Titik:

0.100	0.003
0.300	0.067
0.500	0.148
0.700	0.248
0.900	0.370
1.100	0.518
1.300	0.697

Taksir  $f(x)$  untuk  $x$ :

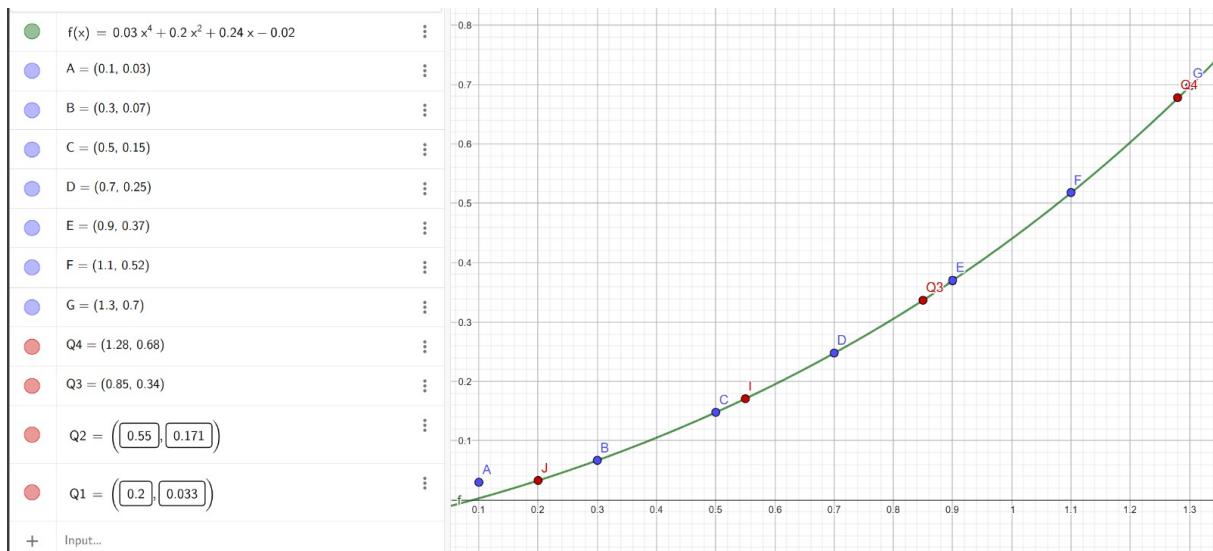
Output:

Persamaan Polinomial:  
 $y(x) = 0.026x^4 + 0.197x^2 + 0.240x - 0.023$

Domain prediksi interpolasi yang valid: [0.100, 1.300]

Hasil Taksiran:  
 $f(1.280) = 0.678$

Analisis kasus:



Keterangan: titik ungu adalah titik yang diberitahu soal, dan titik merah adalah titik yang diujikan pada jawaban

Pada studi kasus 1 ini, kami telah mengimplementasikan metode interpolasi dengan memanfaatkan data titik sampel yang diberikan. Program yang kami buat mampu membentuk persamaan polinomial yang sesuai dengan titik-titik masukan, kemudian digunakan untuk memperkirakan nilai pada titik baru yang tidak terdapat pada data awal.

Sebagai contoh, dari data titik  $(x, y(x))$  yang dimasukkan, program menghasilkan persamaan polinomial:

$$y(x) = 0.026x^4 + 0.197x^2 + 0.240x - 0.023$$

Nilai tersebut konsisten dengan kecenderungan data awal, di mana fungsi  $y(x)$  memang bersifat meningkat seiring bertambahnya nilai  $x$ . Dari implementasi ini dapat dilihat bahwa metode interpolasi polinomial berhasil memberikan pendekatan yang cukup akurat terhadap nilai fungsi pada titik-titik di antara data sampel.

#### 4.4.2 Kasus Uji 2

Jumlah kasus positif baru Covid-19 di Indonesia semakin fluktuatif dari hari ke hari. Di bawah ini diperlihatkan jumlah kasus baru Covid-19 di Indonesia mulai dari tanggal 17 Juni 2022 hingga 31 Agustus 2022:

Tanggal	Tanggal (desimal)	Jumlah Kasus Baru
17/06/2022	6.567	12.624
30/06/2022	6.967	21.807
08/07/2022	7.258	38.391
14/07/2022	7.451	19.541
21/07/2022	7.548	19.582
01/08/2022	8.032	28.935
08/08/2022	8.258	25.854
15/08/2022	8.484	20.935
22/08/2022	8.709	12.408
31/08/2022	8.997	10.534

Tanggal (desimal) adalah tanggal yang sudah diolah ke dalam bentuk desimal 3 angka di belakang koma dengan memanfaatkan perhitungan sebagai berikut:

$$\text{Tanggal (desimal)} = \text{bulan} + (\text{tanggal} / \text{jumlah hari pada bulan tersebut})$$

Sebagai contoh, untuk tanggal 17/06/2022 (dibaca: 17 Juni 2022) diperoleh tanggal (desimal) sebagai berikut:

$$\text{Tanggal (desimal)} = 6 + \frac{17}{30} = 6.567$$

Gunakan data di atas dengan memanfaatkan **interpolasi polinomial** untuk melakukan prediksi jumlah kasus baru Covid-19 pada tanggal-tanggal berikut:

- a. 16/07/2022
- b. 10/08/2022
- c. 05/09/2022
- d. Masukan user lainnya berupa *tanggal (desimal)* yang sudah diolah dengan asumsi prediksi selalu dilakukan untuk tahun 2022.

Untuk 16/07/2022:

**Lintasjava Calculator**

### Interpolasi

Pilih Metode:  Polinomial  Spline Bezier Kubik

Input Titik:

```
6.567 12.624
6.967 21.807
7.258 38.391
7.451 19.541
7.548 19.582
8.032 28.935
8.258 25.854
8.484 20.935
8.709 12.408
8.997 10.534
```

Taksir  $f(x)$  untuk  $x$ :

Output:

```
Persamaan Polinomial:
y(x) = -500.373x^9 + 35652.932x^8 - 1127698.109x^7 + 20781618.394x^6 - 245892682.258x^5 + 20781618.394x^4 - 1127698.109x^3 + 35652.932x^2 - 500.373x + 1.0000000000000002e-10

Domain prediksi interpolasi yang valid: [6.567, 8.997]

Hasil Taksiran:
f(7.533) = 19.244
```

Untuk 10/08/2022:

**Interpolasi**

Pilih Metode:  Polinomial  Spline Bezier Kubik

Input Titik:

```
6.567 12.624
6.967 21.807
7.258 38.391
7.451 19.541
7.548 19.582
8.032 28.935
8.258 25.854
8.484 20.935
8.709 12.408
8.997 10.534
```

Taksir  $f(x)$  untuk  $x$ :

Output:

```
Persamaan Polinomial:
y(x) = -500.373x^9 + 35652.932x^8 - 1127698.109x^7 + 20781618.394x^6 - 245892682.258x^5 + 20781618.394x^4 - 1127698.109x^3 + 35652.932x^2 - 500.373x + 1.0000000000000002e-10

Domain prediksi interpolasi yang valid: [6.567, 8.997]

Hasil Taksiran:
f(8.323) = 25.126
```

Untuk 05/09/2022:

**Interpolasi**

Pilih Metode:  Polinomial  Spline Bezier Kubik

Input Titik:

6.567	12.624
6.967	21.807
7.258	38.391
7.451	19.541
7.548	19.582
8.032	28.935
8.258	25.854
8.484	20.935
8.709	12.408
8.997	10.534

Taksir  $f(x)$  untuk  $x:$

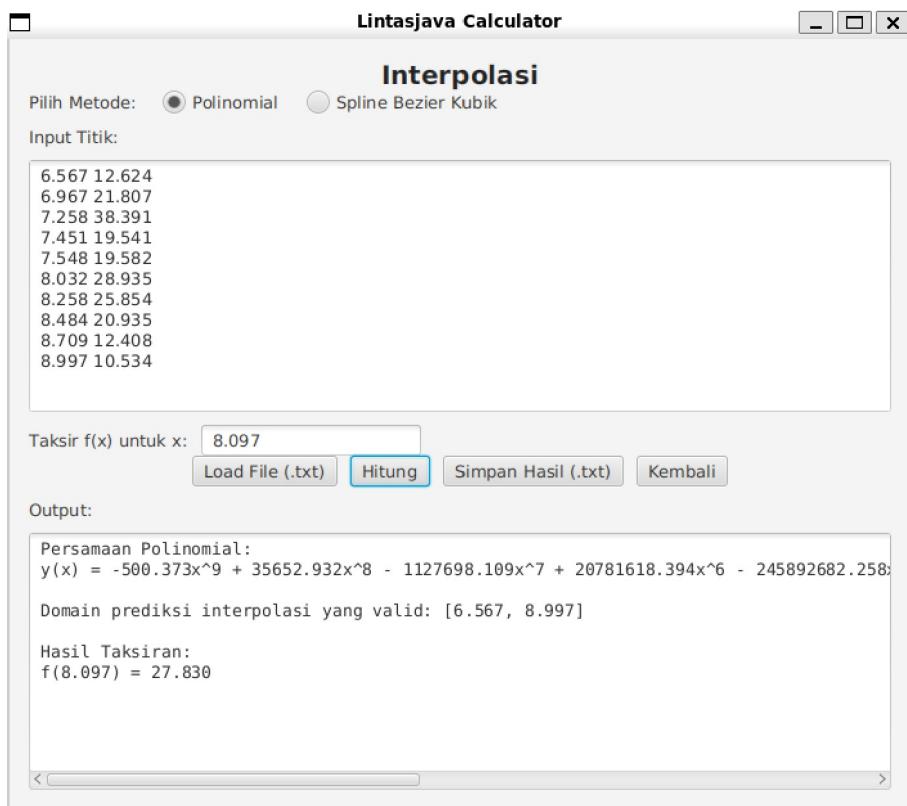
Output:

Persamaan Polinomial:  
 $y(x) = -500.373x^9 + 35652.932x^8 - 1127698.109x^7 + 20781618.394x^6 - 2.$

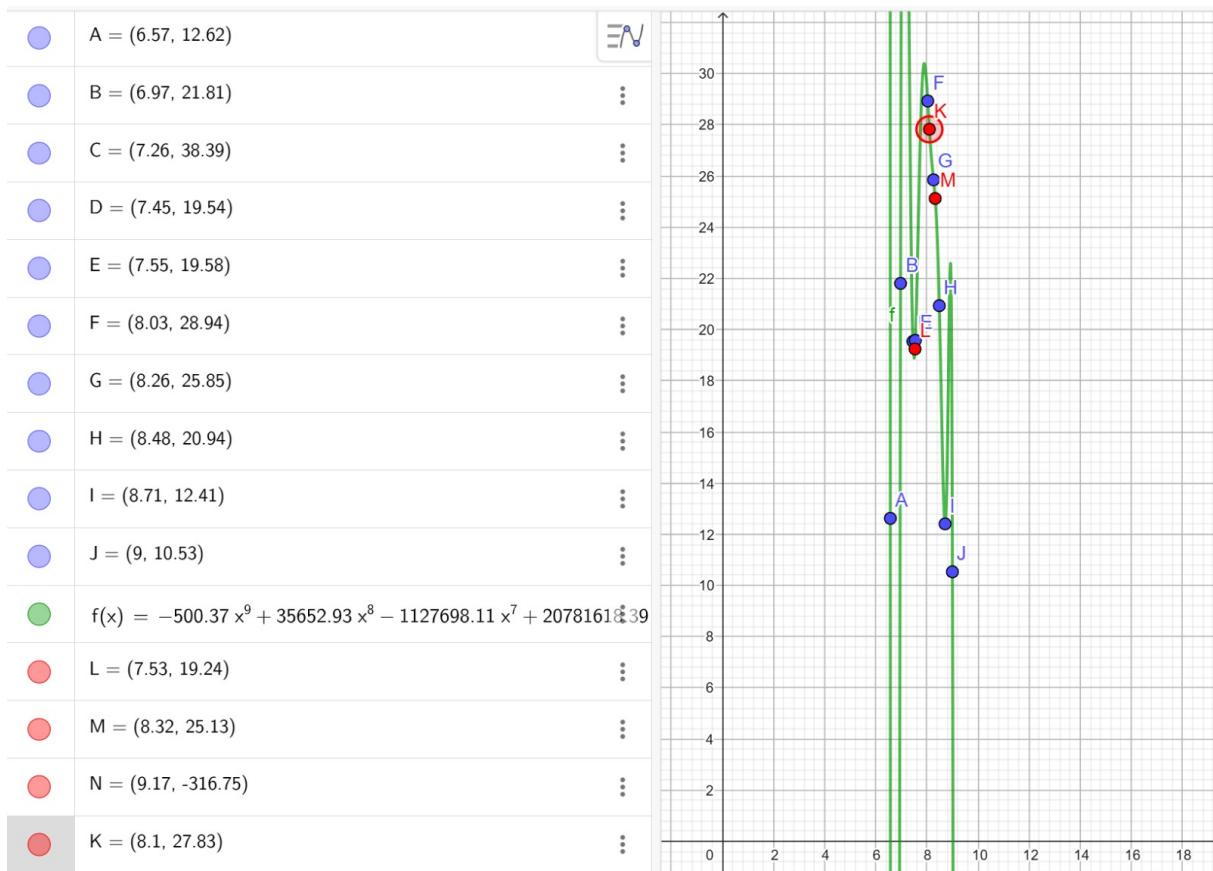
Domain prediksi interpolasi yang valid: [6.567, 8.997]

Hasil Taksiran:  
 $f(9.167) = -316.752$

Untuk masukan User lainnya, yaitu 03/08/2022:



#### Analisis kasus:



Keterangan: titik ungu adalah titik yang diberitahu soal, dan titik merah adalah titik yang diujikan pada jawaban

Pada studi kasus 2 ini, kami telah mengimplementasikan metode interpolasi dengan memanfaatkan data titik sampel yang diberikan. Program yang kami buat mampu membentuk persamaan polinomial yang sesuai dengan titik-titik masukan, kemudian digunakan untuk memperkirakan nilai pada titik baru yang tidak terdapat pada data awal.

Sebagai contoh, dari data titik  $(x, y(x))$  yang dimasukkan, program menghasilkan persamaan polinomial:

$$\begin{aligned}
 y = & -500.373134509231000x^9 + 35652.931905427660000x^8 - 1127698.109038503800000x^7 \\
 & + 20781618.39421901000000x^6 - 245892682.258270030000000x^5 + 1937219567.257768600000000x^4 \\
 & - 10161831524.202194000000000x^3 + 34223431749.618958000000000x^2 - 67147300223.451810000000000x \\
 & + 58476353071.733550000000000
 \end{aligned}$$

**Keterangan:** jumlah angka di belakang desimal pada koefisien  $y(x)$  yang digunakan lebih dari tiga angka untuk meningkatkan akurasi pada grafik.

Nilai tersebut konsisten dengan kecenderungan data awal, di mana fungsi  $y(x)$  memang bersifat meningkat seiring bertambahnya nilai  $x$ . Dari implementasi ini dapat dilihat bahwa metode interpolasi polinomial berhasil memberikan pendekatan yang cukup akurat terhadap nilai fungsi pada titik-titik di antara data sampel. Namun, perlu dicatat bahwa derajat polinomial yang tinggi berpotensi menimbulkan osilasi, yang disebut sebagai fenomena Runge, terutama jika jumlah titik data semakin besar. Oleh karena itu, dalam kasus nyata dengan data berukuran besar, metode interpolasi lain dapat memberikan hasil yang lebih stabil.

## 4.5 Interpolasi Splina Bezier Kubik

### 4.5.1 Kasus Uji 1

Tentukan titik kontrol dari tiap segmen kurva splina Bézier kubik yang melalui titik-titik berikut ini:

$$(0, 0), (3, 11), (6, -4), (8, 0), (11, -10), (17, 0).$$

**Interpolasi**

Pilih Metode:  Polinomial  Spline Bezier Kubik

Input Titik:

0.000	0.000
3.000	11.000
6.000	-4.000
8.000	0.000
11.000	-10.000
17.000	0.000

Output:

```
Titik-titik Kontrol Kurva Splina Bezier Kubik:
-----
Titik Kontrol 1: (2.923, 19.694)
Titik Kontrol 2: (6.306, -12.775)
Titik Kontrol 3: (7.852, 7.407)
Titik Kontrol 4: (10.287, -16.852)
-----
```

**Analisis kasus:** Dalam upaya menyelesaikan permasalahan ini, kami telah menyusun algoritma yang dimulai dengan mempersiapkan komponen-komponen yang akan diperlukan sehingga terbentuk suatu sistem persamaan linear yang terdiri dari  $A * B = S$ . Di mana A merupakan matrix tridiagonal,

$$\begin{bmatrix} 4.000 & 1.000 & 0.000 & 0.000 \\ 1.000 & 4.000 & 1.000 & 0.000 \\ 0.000 & 1.000 & 4.000 & 1.000 \\ 0.000 & 0.000 & 1.000 & 4.000 \end{bmatrix}$$

B merupakan titik-titik kontrol yang dicari, dan S merupakan titik-titik input yang nantinya dalam pengerjaan akan dilakukan terpisah untuk masing-masing komponen x dan y. Setelah komponen sistem persamaan linear telah tersedia, sistem persamaan linear akan disusun untuk masing-masing komponen S sehingga akan dilakukan dua penyelesaian SPL, satu untuk komponen x dan satu untuk komponen y. Penyelesaian SPL yang ada dilakukan dengan menggunakan metode gauss. Setelah masing-masing SPL untuk x maupun y telah diselesaikan, dihasilkan masing-masing Bx dan By yang setelahnya digabungkan kembali menjadi titik yang di mana titik-titik ini merupakan titik kontrol.

#### Interpretasi:

- Pada splina Bézier komposit, titik kontrol internal menentukan bentuk segmen-segmen Bézier kubik yang melewati titik sampel pada ujung-ujung segmen ( $S_0, S_1, \dots$ ). Hasil di atas adalah titik kontrol internal yang diselesaikan dari sistem linear tridiagonal standar untuk menghasilkan kurva dengan kontinuitas  $C^2$ .
- Koordinat  $x$  titik kontrol berada di sekitar koordinat  $x$  titik sampel. Misalnya,  $x$  kontrol 1, yang bernilai sekitar 2.92, berada di antara titik  $x = 0$  dan  $x = 3$ , sedangkan kontrol 2–4 juga berada pada rentang yang sesuai.
- Nilai  $y$  pada titik kontrol cukup besar (positif maupun negatif), misalnya Titik Kontrol 1 dengan  $y \approx 19.7$  lebih tinggi dari titik sampel (3, 11). Hal ini menyebabkan segmen pertama membentuk tonjolan ke atas. Titik Kontrol 2 dan 4 bernilai negatif besar sehingga menghasilkan lekukan ke bawah untuk menyesuaikan titik sampel negatif (6, -4) dan (11, -10).
- Sistem linear yang digunakan menjamin kontinuitas posisi dan turunan hingga orde kedua ( $C^2$ ) di sambungan segmen, sehingga kurva tetap halus walaupun ada overshoot.

## 4.6 Regresi Polinomial Berganda

### 4.6.1 Kasus Uji 1

Wak Rusdi sedang meneliti pertumbuhan alga berdasarkan faktor cahaya, suhu, dan pH. Ia meneliti 50 sampel yang hasilnya dapat dilihat [di sini](#).

Silakan proses data tersebut menjadi file **.txt** sesuai dengan format input yang telah dijelaskan sebelumnya lalu tentukan persamaan regresi polinomial berganda yang sesuai dengan data tersebut menggunakan polinomial kubik (derajat 3).

Light	Temperature	pH	Population				
1011	23.39	7.27	4736.51				
1206.5	14.74	7.94	4677.34				
1450.63	28.58	7.35	3388.85				
1158.4	27.18	7.43	4899.17				
1159.92	22.52	7.92	4974.25				
1635.29	12.56	7.38	3204.32				
565.81	20.81	7.91	3985.53				
982.2	15.17	7.26	4254.6				
1457.01	17.91	7.43	3472.52				
1976.44	18.39	7.48	0	1871.19	23.19	7.33	1149.73
195.56	21.48	7.08	1422.44	164.58	25.01	7.8	592.29
203.04	27.25	7.73	1042.87	1994.72	17.05	7.72	0
1844.44	17.92	7.92	1483.83	319.26	13.33	7.47	2663
53.67	19.25	7.37	0	1855.75	15.95	7.01	1396.9
1576.58	28.37	7.98	2974.64	751.04	22.56	7.27	4068.05
444.43	20.2	7.81	2670.12	921.3	26.77	7.15	4799.43
1139.49	22.82	7.75	4205.04	1101.71	23.41	7.57	4711.99
245.66	19.69	7.73	1779.42	1179.9	16.77	7.76	4250.72
275.37	19.59	7.12	1643.84	925.88	21.39	7.41	4672.41
315.16	27.52	7.37	2261.36	153.53	28.27	7.63	926.16
1187.49	25.19	7.58	4234.07	2006.38	24	7.49	370.86
532.71	20.84	7.43	3920.46	1540.87	12.26	7.96	3831.27
747.33	19.98	7.78	4798.94	687.66	17.16	7.92	3783.48
482.07	14.25	7.79	3754.74	1054.31	16.84	7.13	4360.49
1531.83	18.39	7.03	3824.25	1302.26	19.94	7.6	3891.71
581.63	17.21	7.56	3962.81	945.35	11.01	7.7	4524.58
796.03	14.93	7.64	4025.55	755.42	10.02	7.73	4286.65
788.42	13.65	7.22	4460.01	537.17	13.14	7.66	3620.27
1190.39	23.48	7.43	4516.85	1897.53	20.52	7.94	517.21
75.62	13.58	7.53	526.22				

Lintasjava Calculator

### Regresi Polinomial Berganda

Info Data & Persamaan:

```
1011 23.39 7.27 4736.51
1206.5 14.74 7.94 4677.34
1450.63 28.58 7.35 3388.85
1158.4 27.18 7.43 4899.17
1159.92 22.52 7.92 4974.25
1635.29 12.56 7.38 3204.32
565.81 20.81 7.91 3985.53
982.2 15.17 7.26 4254.6
1457.01 17.91 7.43 3472.52
1976.44 18.39 7.48 0
195.56 21.48 7.08 1422.44
```

Load File Proses Regresi Simpan Hasil Kembali

Input nilai  $x_1, x_2, \dots$  untuk prediksi:

Output:  
 Persamaan Regresi Polinomial Berganda:  

$$y = -438043.299 + 1741747.203x_3 - 230953.009x_3^2 + 10226.006x_3^3 + 6176.308x_2 - 1460.557x_2x_3 + 81.249x_2x_3^2 + 4.473x_2^2 - 18.647x_2^2x_3 - 2.366x_2^3 - 0.138x_1 - 2.613x_1x_3 - 1057.01$$
 Derajat polinom: 3  
 Prediksi untuk input (1800 18 0):  
 $y = -89499.710$

**Analisis kasus:**

Pada studi kasus ini, Wak Rusdi meneliti pertumbuhan alga berdasarkan tiga faktor: intensitas cahaya ( $x_1$ ), suhu ( $x_2$ ), dan pH ( $x_3$ ). Dengan memanfaatkan 50 sampel data yang diberikan, program regresi polinomial berganda derajat 3 digunakan untuk membentuk persamaan prediksi. Program mampu menemukan hubungan kompleks antar variabel input terhadap hasil pengukuran pertumbuhan alga ( $y$ ).

Berdasarkan perhitungan regresi polinomial berganda, diperoleh persamaan:

$$\begin{aligned}y = & -4380843.299 + 1741747.203x_3 - 230953.009x_3^2 + 10226.006x_3^3 \\& + 6176.308x_2 - 1460.557x_2x_3 + 81.249x_2x_3^2 + 4.473x_2^2 \\& - 18.647x_2^2x_3 - 2.366x_2^3 - 0.138x_1 - 2.613x_1x_3 - 1.317x_1x_3^2 \\& + 0.098x_1x_2 + 0.659x_1x_2x_3 + 0.028x_1x_2^2 - 0.005x_1^2 \\& - 0.001x_1^2x_3 - 0.000x_1^2x_2 + 0.000x_1^3\end{aligned}$$

Derajat polinom: 3

Prediksi hasil pertumbuhan alga untuk input contoh  $(x_1, x_2, x_3) = (1800, 18, 8)$  adalah:

$$y \approx -89499.710$$

**Interpretasi:**

- Nilai koefisien menunjukkan pengaruh masing-masing variabel terhadap pertumbuhan alga.
- Interaksi antar variabel ( $x_1x_2$ ,  $x_2x_3$ , dsb.) juga dimodelkan dalam polinom derajat 3.
- Meskipun prediksi negatif untuk input tertentu muncul (misal (1800, 18, 8)), ini menandakan model polinomial belum tentu akurat di luar rentang data sampel yang diberikan (fenomena *extrapolation*).
- Model ini tetap berguna untuk memahami tren dan hubungan antar faktor di rentang data pengamatan.

## 5 Penutup

### 5.1 Kesimpulan

Melalui penggerjaan Tugas Besar 1 ini, kami berhasil mengimplementasikan berbagai konsep aljabar linier dan geometri yang diajarkan di kelas ke dalam bentuk pustaka Java yang modular dan terdokumentasi. Fitur-fitur utama yang kami kembangkan mencakup penyelesaian sistem persamaan linier dengan berbagai metode, yaitu Eliminasi Gauss, Eliminasi Gauss-Jordan, Kaidah Cramer, dan Matriks Balikan. Selain itu, kami mengembangkan fitur perhitungan determinan dengan metode ekspansi kofaktor dan reduksi baris, pencarian matriks invers dengan metode adjoin dan augmentasi, interpolasi polinomial, interpolasi splina Bézier kubik, serta regresi polinomial berganda.

Dari pengujian yang dilakukan terhadap seluruh kasus uji yang disediakan, program mampu memberikan hasil yang sesuai dengan teori, termasuk dalam menangani kasus dengan solusi unik, banyak solusi, maupun tanpa solusi. Selain itu, implementasi interpolasi dan regresi terbukti dapat digunakan untuk memperkirakan nilai maupun memodelkan data dengan cukup baik.

Secara keseluruhan, tugas besar ini tidak hanya memperdalam pemahaman kami mengenai teori aljabar linier, tetapi juga melatih kemampuan praktis dalam mengimplementasikan teori tersebut secara komputasional menggunakan Java. Tantangan yang kami hadapi, seperti penanganan error input, optimasi perhitungan pada matriks berukuran besar, dan penyusunan struktur program yang modular, memberikan pengalaman berharga dalam pengembangan perangkat lunak ilmiah.

Dengan demikian, kami menyimpulkan bahwa tujuan dari Tugas Besar 1 telah tercapai, yaitu memahami, mengimplementasikan, serta mengaplikasikan konsep aljabar linier ke dalam solusi komputasional yang nyata.

### 5.2 Saran

Saran untuk asisten, spesifikasi yang diberikan sudah cukup jelas dan terstruktur, tetapi pada spesifikasi terkait laporan dan langkah-langkah masih kurang jelas, seperti apakah di laporan harus ditunjukkan langkah-langkah yang lengkap atau tidak perlu. Selain itu, akan lebih baik jika format penulisan langkah-langkah diperjelas dan tidak sepenuhnya dibebaskan ke setiap kelompok, karena langkah-langkah merupakan fitur wajib.

Saran untuk kelompok dan penggerjaan tugas di masa mendatang, kelompok kami menyadari pentingnya perencanaan waktu dan pembagian tugas yang lebih terstruktur agar setiap modul dapat selesai lebih cepat dan menyisakan cukup waktu untuk pengujian, perbaikan bug, dan penyusunan laporan. Selain itu, pemahaman teori terkait object oriented programming juga perlu diperdalam sejak awal agar implementasi berjalan lebih lancar dan minim kesalahan.

### 5.3 Komentar

Irvin: "kata nathan bebas kata ojen no comment"

Nathan: "bebaslah"

Ojen: "no comment"

## References

- [1] LibreTexts, “5.1: Polynomial Interpolation,” *Mathematics LibreTexts*, May 31, 2022. [Online]. Available: [https://math.libretexts.org/Bookshelves/Applied\\_Mathematics/Numerical\\_Methods\\_\(Chasnov\)/05%3A\\_Interpolation/5.01%3A\\_Polynomial\\_Interpolation](https://math.libretexts.org/Bookshelves/Applied_Mathematics/Numerical_Methods_(Chasnov)/05%3A_Interpolation/5.01%3A_Polynomial_Interpolation)
- [2] Laboratorium Ilmu dan Rekayasa Komputasi, *Spesifikasi Tugas Besar 1 – IF2123 Aljabar Linier dan Geometri: Sistem Persamaan Linier, Determinan, dan Penerapannya (v1.1)*, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2025. [Online]. Available: <https://drive.google.com/file/d/10TB372xhcXu6TjY008Z7uYHM4Uf-qnnX/view>
- [3] R. Munir, *Aljabar Geometri: Tiga Kemungkinan Solusi SPL*, 2025–2026, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-04-Tiga-Kemungkinan-Solusi-SPL-2025.pdf>
- [4] R. Munir, *Algeo-03: Sistem Persamaan Linier (SPL)*, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2025. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-03-Sistem-Persamaan-Linier-2025.pdf>
- [5] R. Munir, *Algeo-05: Sistem Persamaan Linier 2 – Metode Gauss-Jordan*, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2025. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-05-Sistem-Persamaan-Linier-2-2025.pdf>
- [6] R. Munir, *Algeo-08: Determinan Bagian 1*, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2025. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-08-Determinan-bagian1-2025.pdf>
- [7] R. Munir, *Algeo-09: Determinan Bagian 2*, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2025. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-09-Determinan-bagian2-2025.pdf>

## Lampiran

Berikut ini adalah tautan menuju repository github kelompok kami:

<https://github.com/IRK-23/algeo1-lintasjava>