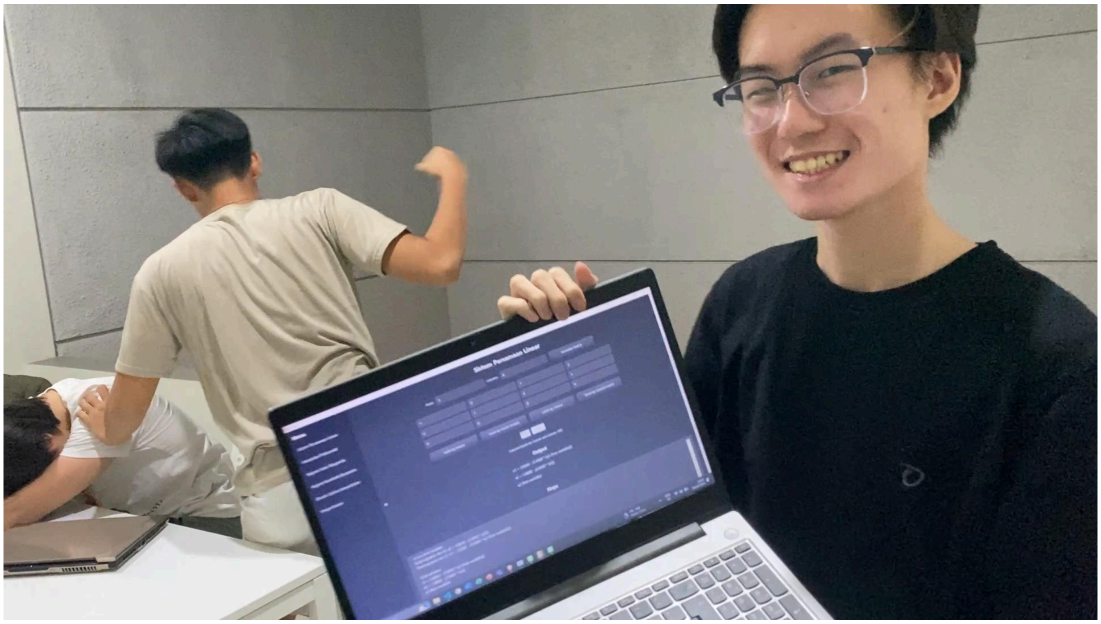
CHISLI



Anggota:

- 13523003 Dave Daniell Yanni
- 13523033 Alvin Christopher Santausa
- 13523036 Yonatan Edward Njoto

Daftar Isi

BAB 1. Deskripsi Masalah

I. Sistem Persamaan Linier (SPL)

Sistem persamaan linier (SPL) banyak ditemukan di dalam bidang sains dan rekayasa. Andstrea sudah mempelajari berbagai metode untuk menyelesaikan SPL, termasuk menghitung determinan matriks. Sembarang SPL dapat diselesaikan dengan beberapa metode, yaitu metode eliminasi Gauss, metode eliminasi Gauss-Jordan, metode matriks balikan ($x = A^{-1}b$), dan kaidah *Cramer* (khusus untuk SPL dengan n peubah dan n persamaan). Solusi sebuah SPL mungkin tidak ada, banyak (tidak berhingga), atau hanya satu (unik/tunggal).

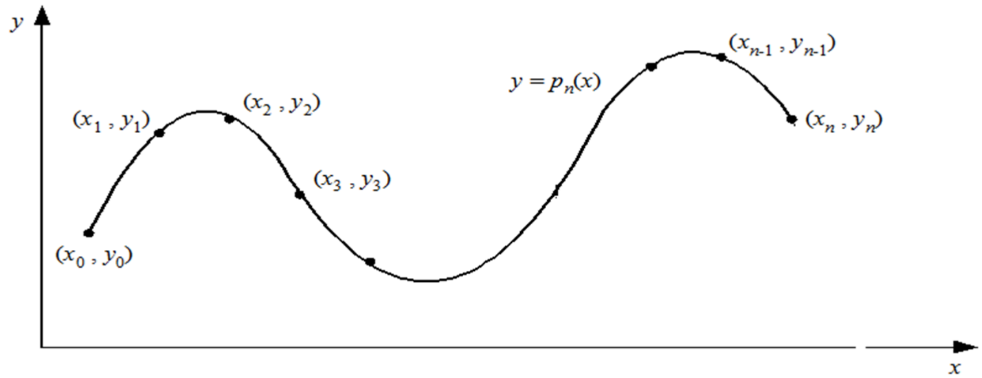
$$\begin{bmatrix} 0 & \mathbf{2} & 1 & -1 \\ 0 & 0 & \mathbf{3} & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \quad \begin{bmatrix} 0 & \mathbf{1} & 0 & -\frac{2}{3} \\ 0 & 0 & \mathbf{1} & \frac{1}{3} \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Gambar 1. Eliminasi Gauss dilakukan dengan matriks eselon baris dan eliminasi Gauss-Jordan dengan matriks eselon baris tereduksi.

Di dalam Tugas Besar 1 ini, Anda diminta membuat satu atau lebih *library* aljabar linier dalam Bahasa Java. Library tersebut berisi fungsi-fungsi seperti eliminasi Gauss, eliminasi Gauss-Jordan, menentukan balikan matriks, menghitung determinan, kaidah Cramer (kaidah Cramer khusus untuk SPL dengan n peubah dan n persamaan). Selanjutnya, gunakan *library* tersebut di dalam program Java untuk menyelesaikan berbagai persoalan yang dimodelkan dalam bentuk SPL, menyelesaikan persoalan interpolasi, dan persoalan regresi. Penjelasan tentang interpolasi dan regresi adalah seperti di bawah ini.

II. Interpolasi Polinomial

Persoalan interpolasi polinom adalah sebagai berikut: Diberikan n+1 buah titik berbeda, (x_0, y_0), (x_1, y_1), ..., (x_n, y_n). Tentukan polinom $p_n(x)$ yang menginterpolasi (melewati) semua titik-titik tersebut sedemikian rupa sehingga $y_i = p_n(x_i)$ untuk i = 0, 1, 2, ..., n.



Gambar 2. Ilustrasi beberapa titik yang diinterpolasi secara polinomial.

Setelah polinom interpolasi $p_n(x)$ ditemukan, $p_n(x)$ dapat digunakan untuk menghitung perkiraan nilai y di sembarang titik di dalam selang $[x_0, x_n]$.

Polinom interpolasi derajat n yang menginterpolasi titik-titik $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$. adalah berbentuk $p_n(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$. Jika hanya ada dua titik, (x_0, y_0) dan (x_1, y_1), maka polinom yang menginterpolasi kedua titik tersebut adalah $p_1(x) = a_0 + a_1x$ yaitu berupa persamaan garis lurus. Jika tersedia tiga titik, (x_0, y_0), (x_1, y_1), dan (x_2, y_2), maka polinom yang menginterpolasi ketiga titik tersebut adalah $p_2(x) = a_0 + a_1x + a_2x^2$ atau persaman kuadrat dan kurvanya berupa parabola. Jika tersedia empat titik, (x_0, y_0), (x_1, y_1), (x_2, y_2), dan (x_3, y_3), polinom yang menginterpolasi keempat titik tersebut adalah $p_3(x) = a_0 + a_1x + a_2x^2 + a_3x^3$, demikian seterusnya. Dengan cara yang sama kita dapat membuat polinom interpolasi berderajat n untuk n yang lebih tinggi asalkan tersedia $(n+1)$ buah titik data. Dengan menyulihkan (x_i, y_i) ke dalam persamaan polinom $p_n(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$ untuk $i = 0, 1, 2, ..., n$, akan diperoleh n buah sistem persamaan lanjar dalam $a_0, a_1, a_2, ..., a_n$,

$$
\begin{gathered}
a_0 + a_1x_0 + a_2x_0^2 + ... + a_n x_0^n = y_0 \\
a_0 + a_1x_1 + a_2x_1^2 + ... + a_n x_1^n = y_1 \\
... \qquad\qquad ... \\
a_0 + a_1x_n + a_2x_n^2 + ... + a_n x_n^n = y_n
\end{gathered}
$$

Solusi sistem persamaan lanjar ini, yaitu nilai $a_0, a_1, ..., a_n$, diperoleh dengan menggunakan metode eliminasi Gauss yang sudah anda pelajari. Sebagai contoh, misalkan diberikan tiga buah titik yaitu (8.0, 2.0794), (9.0, 2.1972), dan (9.5, 2.2513). Tentukan polinom interpolasi kuadratik lalu estimasi nilai fungsi pada $x = 9.2$. Polinom kuadratik berbentuk $p_2(x) = a_0 + a_1x + a_2x^2$. Dengan menyulihkan ketiga buah titik data ke dalam polinom tersebut, diperoleh sistem persamaan lanjar yang terbentuk adalah

$$a_0 + 8.0a_1 + 64.00a_2 = 2.0794$$

$$a_0 + 9.0a_1 + 81.00a_2 = 2.1972$$

$$a_0 + 9.5a_1 + 90.25a_2 = 2.2513$$

Penyelesaian sistem persamaan dengan metode eliminasi Gauss menghasilkan $a_0 = 0.6762$, $a_1 = 0.2266$, dan $a_2 = -0.0064$. Polinom interpolasi yang melalui ketiga buah titik tersebut adalah $p_2(x) = 0.6762 + 0.2266x - 0.0064x^2$. Dengan menggunakan polinom ini, maka nilai fungsi pada $x = 9.2$ dapat ditaksir sebagai berikut: $p_2(9.2) = 0.6762 + 0.2266(9.2) - 0.0064(9.2)^2 = 2.2192$.

III. Regresi Berganda

Regresi (akan dipelajari lebih lanjut di Probabilitas dan Statistika) merupakan salah satu metode untuk memprediksi nilai selain menggunakan Interpolasi Polinom. Pada tugas besar ini, anda diminta untuk membuat 2 jenis regresi yaitu Regresi Linier Berganda dan Regresi Kuadratik Berganda.

1. Regresi Linier Berganda

Meskipun sudah ada persamaan jadi untuk menghitung regresi linear sederhana, terdapat persamaan umum dari regresi linear yang bisa digunakan untuk regresi linear berganda, yaitu.

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_k x_{ki} + \epsilon_i$$

Untuk mendapatkan nilai dari setiap β_i dapat digunakan *Normal Estimation Equation for Multiple Linear Regression* sebagai berikut:

$$
\begin{aligned}
nb_0 + b_1\sum_{i=1}^{n} x_{1i} \quad &+ b_2\sum_{i=1}^{n} x_{2i} \quad + \ \cdots + b_k\sum_{i=1}^{n} x_{ki} \quad = \sum_{i=1}^{n} y_i \\
b_0\sum_{i=1}^{n} x_{1i} + b_1\sum_{i=1}^{n} x_{1i}^2 \quad &+ b_2\sum_{i=1}^{n} x_{1i}x_{2i} + \ \cdots + b_k\sum_{i=1}^{n} x_{1i}x_{ki} = \sum_{i=1}^{n} x_{1i}y_i \\
\vdots \qquad\quad \vdots \qquad &\qquad\quad \vdots \qquad\qquad\qquad \vdots \qquad\qquad \vdots \\
b_0\sum_{i=1}^{n} x_{ki} + b_1\sum_{i=1}^{n} x_{ki}x_{1i} &+ b_2\sum_{i=1}^{n} x_{ki}x_{2i} + \ \cdots + b_k\sum_{i=1}^{n} x_{ki}^2 \quad = \sum_{i=1}^{n} x_{ki}y_i
\end{aligned}
$$

2. Regresi Kuadratik Berganda

Dalam kasus ini, proses mengubah data-data dalam regresi kuadratik berganda cukup berbeda dengan Regresi Linier Berganda. Bentuk persamaan dari regresi kuadratik ada 3, yaitu:

a. Variabel Linier: Variabel dengan derajat satu seperti X, Y, dan Z
b. Variabel Kuadrat: Variabel dengan derajat dua seperti X^2
c. Variabel Interaksi: 2 Variabel dengan derajat satu yang dikalikan dengan satu sama lain seperti XY, YZ, dan XZ

Setiap n-peubah, jumlah variabel linier, kuadrat, dan interaksi akan berbeda-beda. Perhatikan contoh regresi kuadratik 2 variabel peubah sebagai berikut!

$$
\begin{pmatrix}
N & \sum u_i & \sum v_i & \sum u_i^2 & \sum u_iv_i & \sum v_i^2 \\
\sum u_i & \sum u_i^2 & \sum u_iv_i & \sum u_i^3 & \sum u_i^2v_i & \sum u_iv_i^2 \\
\sum v_i & \sum u_iv_i & \sum v_i^2 & \sum u_i^2v_i & \sum u_iv_i^2 & \sum v_i^3 \\
\sum u_i^2 & \sum u_i^3 & \sum u_i^2v_i & \sum u_i^4 & \sum u_i^3v_i & \sum u_i^2v_i^2 \\
\sum u_iv_i & \sum u_i^2v_i & \sum u_iv_i^2 & \sum u_i^3v_i & \sum u_i^2v_i^2 & \sum u_iv_i^3 \\
\sum v_i^2 & \sum u_iv_i^2 & \sum v_i^3 & \sum u_i^2v_i^2 & \sum u_iv_i^3 & \sum v_i^4
\end{pmatrix}
\begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix}
=
\begin{pmatrix} \sum y_i \\ \sum y_iu_i \\ \sum y_iv_i \\ \sum y_iu_i^2 \\ \sum y_iu_iv_i \\ \sum y_iv_i^2 \end{pmatrix}
$$

N menandakan jumlah peubah, terdapat 2 variabel linier yaitu u_i dan v_i, 2 variabel kuadrat yaitu u_i^2 dan v_i^2, dan 1 variabel interaksi yaitu uv. Untuk setiap n-peubah, akan terdapat 1 konstan N (Terlihat di bagian atas kiri gambar), n variabel linier, n variabel kuadrat, dan C_2^n variabel linier (dengan syarat n > 1). Tentu dengan bertambahnya peubah n, ukuran matriks akan bertumbuh lebih besar dibandingkan regresi linier berganda tetapi solusi tetap bisa didapat dengan menggunakan SPL.

Kedua model regresi yang dijadikan sistem persamaan linier tersebut diselesaikan dengan menggunakan metode eliminasi Gauss.

IV. Bicubic Spline Interpolation

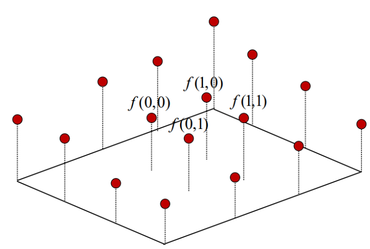
Bicubic spline interpolation adalah metode interpolasi yang digunakan untuk mengaproksimasi fungsi di antara titik-titik data yang diketahui. *Bicubic spline interpolation* melibatkan konsep *spline* dan konstruksi serangkaian polinomial kubik di dalam setiap sel segi empat dari data yang diberikan. Pendekatan ini menciptakan permukaan yang halus dan kontinu, memungkinkan untuk perluasan data secara visual yang lebih akurat daripada metode interpolasi linear.

Dalam pemrosesan menggunakan interpolasi *bicubic spline* digunakan 16 buah titik, 4 titik referensi utama di bagian pusat, dan 12 titik di sekitarnya sebagai aproksimasi turunan dari keempat titik referensi untuk membagun permukaan bikubik. Bentuk pemodelannya adalah sebagai berikut.

Normalization: $f(0,0), f(1,0)$
$f(0,1), f(1,1)$

Model: $f(x,y)=\sum_{j=0}^{3}\sum_{i=0}^{3} a_{ij} x^i y^j$

Solve: a_{ij}



Gambar 3. Pemodelan interpolasi *bicubic spline*.

Selain melibatkan model dasar, juga digunakan model turunan berarah dari kedua sumbu, baik terhadap sumbu *x*, sumbu *y*, mapun keduanya. Persamaan polinomial yang digunakan adalah sebagai berikut.

$$f(x,y)=\sum_{i=0}^{3}\sum_{j=0}^{3}a_{ij}x^{i}y^{j}$$

$$f_x(x,y)=\sum_{j=0}^{3}\sum_{i=1}^{3}a_{ij}ix^{i-1}y^{j}$$

$$f_y(x,y)=\sum_{j=1}^{3}\sum_{i=0}^{3}a_{ij}jx^{i}y^{j-1}$$

$$f_{xy}(x,y)=\sum_{j=0}^{3}\sum_{i=0}^{3}a_{ij}ijx^{i-1}y^{j-1}$$

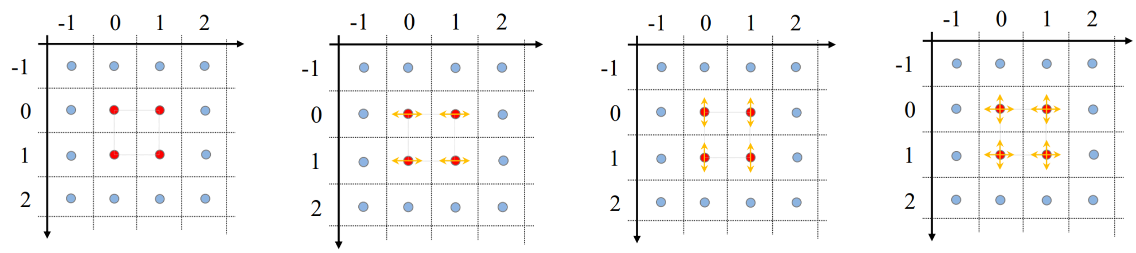
Dengan menggunakan nilai fungsi dan turunan berarah tersebut, dapat terbentuk sebuah matriks solusi *X* yang membentuk persamaan penyelesaian sebagai berikut.

$$y = Xa$$

$$\begin{bmatrix} f(0,0)\\ f(1,0)\\ f(0,1)\\ f(1,1)\\ f_x(0,0)\\ f_x(1,0)\\ f_x(0,1)\\ f_x(1,1)\\ f_y(0,0)\\ f_y(1,0)\\ f_y(0,1)\\ f_y(1,1)\\ f_{xy}(0,0)\\ f_{xy}(1,0)\\ f_{xy}(0,1)\\ f_{xy}(1,1) \end{bmatrix} = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\ 1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\ 0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&2&3&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0\\ 0&1&2&3&0&1&2&3&0&1&2&3&0&1&2&3\\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&2&0&0&0&3&0&0&0\\ 0&0&0&0&1&1&1&1&2&2&2&2&3&3&3&3\\ 0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&2&3&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&2&0&0&0&3&0&0\\ 0&0&0&0&0&1&2&3&0&2&4&6&0&3&6&9 \end{bmatrix} \begin{bmatrix} a_{00}\\ a_{10}\\ a_{20}\\ a_{30}\\ a_{01}\\ a_{11}\\ a_{21}\\ a_{31}\\ a_{02}\\ a_{12}\\ a_{22}\\ a_{32}\\ a_{03}\\ a_{13}\\ a_{23}\\ a_{33} \end{bmatrix}$$

Perlu diketahui bahwa elemen pada matriks *X* adalah nilai dari setiap komponen koefisien a_{ij} yang diperoleh dari persamaan fungsi maupun persamaan turunan yang telah dijelaskan sebelumnya. Sebagai contoh, elemen matriks *X* pada baris 8 kolom ke 2 adalah koefisien dari a_{10} pada ekspansi sigma untuk $f_x(1, 1)$ sehingga diperoleh nilai konstanta $1 \times 1^{1-1} \times 1^{0} = 1$, sesuai dengan isi matriks *X*.

Nilai dari vektor *a* dapat dicari dari persamaan $y = Xa$, lalu vektor *a* tersebut digunakan sebagai nilai variabel dalam $f(x, y)$, sehingga terbentuk fungsi interpolasi bicubic sesuai model. Tugas Anda pada studi kasus ini adalah membangun persamaan *f*(*x*, *y*) yang akan digunakan untuk melakukan interpolasi berdasarkan nilai *f*(*a*, *b*) dari masukan matriks 4 x 4. Nilai masukan *a* dan *b* berada dalam rentang [0, 1]. Nilai yang akan diinterpolasi dan turunan berarah disekitarnya dapat diilustrasikan pada titik berwarna merah pada gambar di bawah.



Gambar 4. Nilai fungsi yang akan di interpolasi pada titik merah, turunan berarah terhadap sumbu *x*, terhadap sumbu *y*, dan keduanya (kiri ke kanan).

Untuk studi kasus ini, buatlah matriks *X* menggunakan persamaan yang ada (tidak *hardcode*) serta carilah invers matriks *X* dengan *library* yang telah kalian buat dalam penyelesaian masalah. Berikut adalah sebuah tautan yang dapat dijadikan referensi.

Bonus 1: Video Penjelasan Program (Nilai 4)

Terdapat banyak cara untuk menjelaskan cara kerja sebuah program. Namun penjelasan secara visual dan kreatif adalah teknik penjelasan yang sangat menarik perhatian orang-orang. Oleh karena itu, anda diperbolehkan membuat sebuah video penjelasan tentang program yang dibuat. Buatlah video ini sekreatif mungkin (Jangan hanya menjelaskan programmu saja, tunjukkan *skill* actingmu :D). Untuk referensi, silakan cek tautan ini.

Bonus 2: GUI (Nilai 4)

Silahkan gunakan GUI jika ingin membuat aplikasi yang lebih interaktif. Perlu diingat bahwa tugas besar ini menggunakan Java, sehingga kakas yang digunakan dalam membuat GUI haruslah berasal dari bahasa Java. Anda juga hanya diperbolehkan untuk membuat *Desktop App* dari tugas besar ini dan tidak diperbolehkan untuk membuat web. Kakas GUI yang boleh digunakan pada tugas besar ini adalah JavaFX atau Swing. IDE yang digunakan dibebaskan (Contoh: IntelliJ IDEA, NetBeans, dll). Semua fitur wajib harus dapat dijalankan dengan GUI jika anda membuat bonus ini.

Bonus 3: Image Resizing and Stretching (Nilai 7)

Seperti yang telah dijelaskan sebelumnya bahwa interpolasi *bicubic spline* dapat digunakan untuk menciptakan permukaan yang halus pada gambar. Oleh karena itu, selain persamaan dasar $y = Xa$ yang telah dijabarkan, persamaan ini juga dapat menggunakan data sebuah citra untuk menciptakan kualitas gambar yang lebih baik. Misalkan $I(x, y)$ merupakan nilai dari suatu citra gambar pada posisi (x, y), maka dapat digunakan persamaan nilai dan persamaan turunan berarah sebagai berikut.

$$f(x,y) = I(x,y)$$

$$f_x(x,y) = [I(x+1,y) - I(x-1,y)]/2$$

$$f_y(x,y) = [I(x,y+1) - I(x,y-1)]/2$$

$$f_{xy}(x,y) = [I(x+1,y+1) - I(x-1,y) - I(x,y-1) - I(x,y)]/4$$

Sistem persamaan tersebut dapat dipetakan menjadi sebuah matriks (dalam hal ini matriks D) dengan gambaran lengkap seperti yang tertera di bawah.

$$y = DI$$

$$\begin{bmatrix} f(0,0) \\ f(1,0) \\ f(0,1) \\ f(1,1) \\ f_x(0,0) \\ f_x(1,0) \\ f_x(0,1) \\ f_x(1,1) \\ f_y(0,0) \\ f_y(1,0) \\ f_y(0,1) \\ f_y(1,1) \\ f_{xy}(0,0) \\ f_{xy}(1,0) \\ f_{xy}(0,1) \\ f_{xy}(1,1) \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I(-1,-1) \\ I(0,-1) \\ I(1,-1) \\ I(2,-1) \\ I(-1,0) \\ I(0,0) \\ I(1,0) \\ I(2,0) \\ I(-1,1) \\ I(0,1) \\ I(1,1) \\ I(2,1) \\ I(-1,2) \\ I(0,2) \\ I(1,2) \\ I(2,2) \end{bmatrix}$$

Dengan menggunakan kedua persamaan nilai y yang telah disebutkan dan dibahas sebelumnya, dapatkan nilai a yang lebih baik dan akurat dalam pemrosesan citra gambar, kemudian gunakan nilai dan persamaan $f(x, y)$ yang terbentuk untuk memperbaiki kualitas citra gambar monokrom pasca perbesaran dengan skala tertentu dengan melakukan interpolasi *bicubic spline*. Berikut adalah contohnya.

Gambar 5. Sebuah citra gambar asal (kiri) dan hasil pemrosesan gambar dengan skala 1.5 pada *width* dan skala 2 pada *height* (kanan).

Untuk bonus ini, buatlah matriks D menggunakan persamaan citra gambar yang ada (tidak *hardcode*) serta gunakan kembali persamaan y yang sebelumnya ($y = Xa$) dan korelasikan dengan persamaan $y = DI$ untuk mendapatkan nilai a yang lebih tepat untuk membangun persamaan $f(x, y)$. Tambahkan pula masukan berupa skala perbesaran untuk *width* dan *height* pada gambar sesuai keinginan pengguna.

BAB 2. Teori singkat

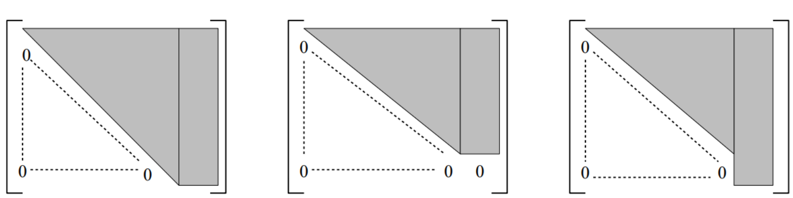
I. Metode Eliminasi Gauss

Metode Eliminasi Gauss adalah algoritma untuk menyelesaikan sistem persamaan linear. Tujuan utamanya adalah mengubah matriks koefisien menjadi bentuk segitiga atas dengan menggunakan operasi baris elementer, seperti menukar baris, mengalikan baris dengan konstanta, serta menjumlahkan atau mengurangkan baris. Setelah penerapan operasi baris elementer, matriks dapat terbentuk dalam tiga jenis, yaitu:

- Matriks Segitiga Atas: Matriks ini menghasilkan solusi yang unik. Setiap variabel dapat ditemukan melalui substitusi mundur karena setiap persamaan memiliki koefisien yang tidak nol.
- Baris dengan Semua Koefisien Variabel Nol: Jika terdapat satu baris dengan semua koefisien variabel bernilai nol namun tetap menghasilkan persamaan yang valid (misalnya, $0 = 0$), sistem ini memiliki solusi tak hingga *(**banyak solusi jika ditemukan sebelum baris = kolom**)*. Kondisi ini menunjukkan bahwa terdapat variabel bebas.
- Baris dengan Semua Koefisien Variabel Nol tetapi Konstanta Tidak Nol: Jika terdapat satu baris dengan semua koefisien variabel nol, namun menghasilkan konstanta yang tidak nol (misalnya, $0 = b$ dengan $b \neq 0$), sistem ini tidak memiliki solusi (inkonsisten).

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} & b_n \end{bmatrix} \sim\text{OBE}\sim \begin{bmatrix} 1 & * & * & \dots & * & * \\ 0 & 1 & * & \dots & * & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \vdots & 1 & * \end{bmatrix}$$

Operasi Baris Elementer



Solusi unik | Solusi banyak | Tidak ada solusi

jumlah baris = (kolom -1)

II. Metode Eliminasi Gauss-Jordan

Metode Gauss-Jordan adalah penambahan dari eliminasi Gauss, karena matriks diubah menjadi bentuk reduced row echelon form (RREF). Dengan bentuk ini, solusi diperoleh langsung tanpa perlu substitusi mundur. Setiap baris berisi nol kecuali pada elemen diagonal yang semuanya bernilai 1.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} & b_m \end{bmatrix} \sim \text{OBE} \sim \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & * \\ 0 & 1 & 0 & \dots & 0 & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \vdots & 1 & * \end{bmatrix}$$

III. Determinan

Determinan adalah nilai skalar yang dihitung dari suatu matriks persegi dan memiliki berbagai aplikasi, seperti menentukan apakah suatu matriks memiliki balikan atau tidak. Determinan juga digunakan dalam penghitungan volume dan dalam Kaidah Cramer untuk menyelesaikan sistem persamaan linear.

IV. Matriks Balikan

Matriks balikan (atau invers) dari suatu matriks A adalah matriks A^{-1} yang memenuhi $A^{-1} \times A = I$, di mana I adalah matriks identitas. Hanya matriks persegi yang memiliki balikan, dan hanya jika determinan matriks tersebut tidak nol.

V. Matriks Kofaktor

Kofaktor adalah elemen-elemen matriks yang diperoleh dari minor matriks dengan mengalikan minor dengan tanda tergantung pada posisi elemen. Matriks kofaktor digunakan dalam proses mencari matriks balikan menggunakan metode adjoint.

VI. Matriks Adjoin

Matriks adjoin adalah matriks yang diperoleh dari matriks kofaktor dengan mentransposisikannya. Matriks adjoin digunakan bersama determinan untuk menghitung balikan matriks.

VII. Kaidah Cramer

Kaidah Cramer adalah metode untuk menyelesaikan sistem persamaan linear dengan menggunakan determinan. Jika sistem $AX = B$ memiliki solusi unik, maka solusi untuk setiap variabel diperoleh dengan mengganti kolom koefisien dari variabel yang ingin dicari dengan kolom hasil (B) dan menghitung determinan dari matriks yang diubah.

VIII. Interpolasi Polinom

Interpolasi polinom adalah metode yang digunakan untuk menemukan polinom yang melewati serangkaian titik data. Polinom interpolasi biasanya digunakan untuk aproksimasi fungsi, dan salah satu metode populer untuk menghitungnya adalah menggunakan formula Lagrange atau Newton.

IX. Interpolasi Bicubic Spline

Interpolasi bicubic spline adalah teknik interpolasi yang digunakan untuk memperhalus gambar atau peta 2D dengan memanfaatkan kubus spline pada dua dimensi (x dan y). Ini memberikan hasil yang lebih halus dibandingkan interpolasi bilinear.

X. Regresi Linier Berganda

Regresi Linier Berganda adalah metode statistik yang digunakan untuk memodelkan hubungan antara satu variabel dependen (target) dengan beberapa variabel independen (prediktor). Persamaan umum regresi linier berganda dapat ditulis sebagai:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ ... \ + \beta_n x_n$$

y adalah variabel dependen, x_1, x_2, ... , x_n adalah variabel independen, β_0 adalah konstanta intersep, dan $\beta_1 + \beta_2 + ... + \beta_n$ adalah koefisien regresi. Untuk menghitung nilai β, digunakan Normal Estimation Equation untuk regresi linier berganda, yang merupakan turunan dari metode kuadrat terkecil (least squares method). Regresi linier berganda memungkinkan kita untuk menganalisis pengaruh beberapa variabel independen terhadap variabel dependen.

XI. Regresi kuadratik Berganda

Regresi Kuadratik Berganda adalah pengembangan dari regresi linier berganda yang memungkinkan adanya hubungan non-linear antara variabel dependen dan independen. Dalam regresi kuadratik, persamaan melibatkan variabel linier yaitu variabel dengan satu derajat, variabel kuadrat yaitu variabel dengan dua derajat, dan variabel interaksi yaitu produk dari dua variabel independen. Persamaan regresi kuadratik berganda untuk dua variabel independen U dan V adalah:

$$y = \beta_0 + \beta_1 U + \beta_2 V + \beta_3 U^2 + \beta_4 V^2 + \beta_5 UV$$

Bab 3. Implementasi pustaka

Program kami dibuat menggunakan:

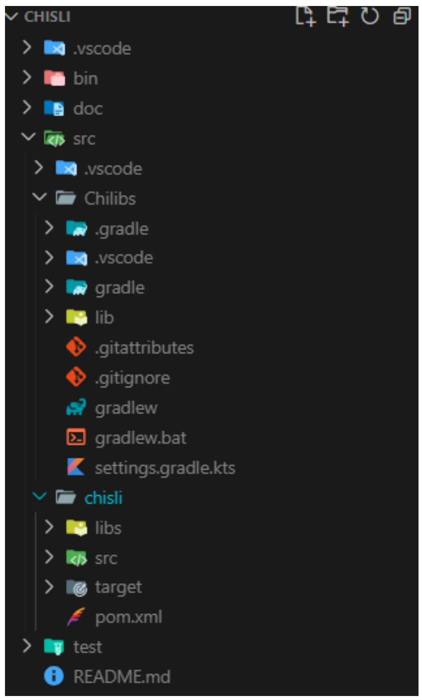
- Library: *gradle*

  Gradle digunakan untuk membentuk library beserta manifest dan module-info untuk meletakkan .jar nya pada aplikasi kami yang berjalan dalam JavaFX

- GUI: *JavaFX*

  JavaFX diperlukan untuk membuat interaksi dengan tampilan (menggunakan fxml) supaya lebih indah dilihat

Oleh karena itu, bentuk folder yang kami gunakan mengikuti standarisasi keduanya, terlihat seperti:



Berikut beberapa penjelasan Implementasi terkait fitur:

I. Floats

Tujuan Kelas SmallFloat

Kelas ini dirancang untuk mendeteksi dan memodifikasi nilai-nilai yang mendekati nol, khususnya -0.0, yang dapat muncul ketika melakukan perhitungan aritmatika dalam masalah SPL. Masalah ini timbul akibat representasi angka floating-point dalam sistem komputer, di mana angka negatif nol (-0.0) dianggap berbeda dengan nol positif (0.0).

Penjelasan Kode:

1. Metode: handleMinus0(double val)
   - Metode ini bertujuan untuk memastikan bahwa jika nilai val sama persis dengan -0.0 atau berada dalam kisaran negatif yang sangat kecil mendekati nol (antara -0.0001 dan 0), metode ini akan mengembalikan nilai 0.0.
   - Dengan menggunakan perbandingan bit-level melalui Double.doubleToRawLongBits(val), metode ini dapat membandingkan nilai val dengan akurat terhadap -0.0. Perbandingan langsung terhadap nilai floating-point membutuhkan penanganan khusus karena dalam spesifikasi IEEE 754, -0.0 dianggap berbeda dari 0.0.
2. Pentingnya Penanganan ini:
   - Dalam SPL, kesalahan pembulatan yang sangat kecil dapat menyebabkan hasil berupa -0.0, yang secara numerik seharusnya dianggap sebagai nol. Penanganan ini memastikan bahwa hasil perhitungan yang berada di sekitar nol tetap konsisten, sehingga tidak terjadi kebingungan atau hasil yang tidak diharapkan dalam algoritma yang sensitif terhadap tanda dari angka kecil seperti ini.

Kesimpulan:

Dengan adanya kelas SmallFloat, kejadian seperti -0.0 dapat diatasi sehingga perhitungan dalam SPL menjadi lebih stabil dan hasilnya lebih akurat. Ini sangat penting dalam konteks komputasi numerik dan pemodelan matematika yang mengandalkan presisi tinggi.

II. Matrix Steps

Tujuan Kelas MatrixSteps

Kelas ini dirancang untuk melacak dan menyimpan setiap langkah dari penyelesaian matriks. Dengan menggunakan kelas ini, setiap perubahan yang terjadi pada matriks selama proses pengerjaan dapat dicatat dan ditampilkan kembali. Hal ini membantu memberikan pemahaman yang lebih baik tentang bagaimana suatu solusi diperoleh, terutama dalam SPL atau metode numerik lainnya.

Penjelasan Kode:

1. Atribut: steps
   - Atribut ini adalah list yang menyimpan langkah-langkah yang telah dilakukan selama proses pengerjaan matriks. Setiap kali terjadi perubahan pada matriks, langkah tersebut akan ditambahkan ke dalam daftar ini.
2. Konstruktor: MatrixSteps()
   - Konstruktor ini menginisialisasi objek MatrixSteps dengan membuat list kosong untuk menyimpan langkah-langkah pengerjaan matriks.
3. Metode: addStep(String step)
   - Metode ini digunakan untuk menambahkan deskripsi langkah tertentu ke dalam daftar steps. Langkah ini bisa berupa penjelasan tertulis dari operasi yang dilakukan, seperti "Baris 1 ditukar dengan Baris 2" atau "Baris 3 dikalikan dengan konstanta 2".
4. Metode: addMatrixState(String matrixState)
   - Metode ini bertujuan untuk menambahkan representasi dari kondisi matriks saat ini ke dalam daftar steps. Biasanya digunakan untuk mencatat keadaan matriks setelah operasi tertentu dilakukan.
5. Metode: getSteps()
   - Metode ini mengembalikan seluruh langkah yang telah disimpan dalam daftar steps. Hal ini memungkinkan pengguna untuk melihat seluruh riwayat dari proses pengerjaan matriks.
6. Metode: clearSteps()

- Metode ini berfungsi untuk menghapus seluruh langkah yang tersimpan di dalam daftar, memulai ulang proses pencatatan langkah untuk pengerjaan matriks baru.

Kesimpulan:

Kelas MatrixSteps memungkinkan pelacakan setiap perubahan yang terjadi pada matriks selama proses penyelesaian, membuat proses pemecahan masalah lebih transparan. Ini sangat membantu dalam pembelajaran dan debugging, karena setiap langkah yang diambil tercatat secara sistematis.

III. Determinan Matrix

Tujuan Kelas MatrixDeterminant

Kelas ini menyediakan cara untuk menghitung determinan matriks menggunakan dua pendekatan berbeda:

1. determinantByElementaryRowOperation: Menggunakan Operasi Baris Elementer (OBE), yang melibatkan modifikasi matriks hingga menjadi matriks segitiga atas, kemudian mengalikan elemen-elemen diagonalnya.
2. determinantByCofactorExpansion: Menggunakan metode perluasan kofaktor (atau metode adjoin), yang memecah matriks menjadi submatriks yang lebih kecil untuk menghitung determinannya secara rekursif.

Kedua metode ini memiliki keunggulan dan kekurangannya masing-masing. Metode OBE umumnya lebih efisien secara komputasi untuk matriks berukuran besar, sementara metode kofaktor lebih intuitif tetapi lebih lambat untuk matriks besar.

Penjelasan Kode

1. Metode determinantByElementaryRowOperation
   Metode ini menghitung determinan dengan mengubah matriks menjadi matriks segitiga atas melalui operasi baris elementer. Setelah matriks berbentuk segitiga atas, determinan diperoleh dengan mengalikan elemen-elemen diagonal. Jika terdapat pertukaran baris, maka nilai determinan harus disesuaikan dengan mengganti tanda (+/-).

- Parameter:

- Matrix matrix: Matriks input yang akan dihitung determinannya.
- boolean captureSteps: Jika di set menjadi true, setiap langkah dalam proses akan dicatat menggunakan kelas MatrixSteps.

- Langkah Utama:
  1. Mengecek Matriks Persegi: Matriks harus persegi (jumlah baris = jumlah kolom) agar determinan dapat dihitung.
  2. Salinan Matriks: Matriks asli tidak diubah. Sebagai gantinya, dibuat salinan dari data matriks agar tidak memodifikasi matriks input.
  3. Pencarian Pivot: Pada setiap baris, elemen diagonal (pivot) digunakan. Jika elemen ini mendekati nol, baris-baris ditukar agar pivot tidak nol. Jika pivot tidak ditemukan (matriks singular), determinan = 0.
  4. Eliminasi Baris Bawah Pivot: Elemen-elemen di bawah pivot di-nol-kan dengan mengurangi kelipatan pivot dari baris lainnya.
  5. Perhitungan Determinan: Setelah matriks berbentuk segitiga atas, determinan diperoleh dengan mengalikan elemen-elemen diagonal. Jika ada pertukaran baris ganjil, nilai determinan dibalik.

2. Metode determinantByCofactorExpansion

   Metode ini menghitung determinan menggunakan perluasan kofaktor, yang melibatkan pemecahan matriks menjadi submatriks yang lebih kecil. Proses ini dilakukan secara rekursif hingga mencapai matriks ukuran 1x1 dan 2x2, di mana determinan dapat dihitung secara langsung.

- Langkah Utama:
  1. Matriks 1x1 atau 2x2: Jika ukuran matriks adalah 1x1, determinan adalah elemen itu sendiri. Jika ukuran 2x2, determinan dihitung menggunakan formula sederhana.
  2. Perluasan Kofaktor: Untuk matriks berukuran lebih dari 2x2, kofaktor dari setiap elemen di baris pertama dihitung dengan menghapus baris dan kolom yang terkait dan kemudian menghitung determinan submatriks yang dihasilkan. Hasil determinan ini dikalikan dengan elemen awal dan dijumlahkan untuk mendapatkan determinan akhir.
  3. Kofaktor: Kofaktor dihitung dengan membuat submatriks dari matriks asli, dan determinan dari submatriks ini dihitung secara rekursif.

3. Penanganan Langkah
   Setiap metode memiliki opsi untuk mencatat langkah-langkah perhitungannya menggunakan objek MatrixSteps. Jika opsi captureSteps diaktifkan, setiap operasi yang dilakukan (misalnya, pertukaran baris, eliminasi, atau perhitungan kofaktor) akan dicatat ke dalam daftar langkah. Hal ini berguna untuk analisis lebih lanjut atau debugging, terutama dalam kasus operasi yang kompleks.
4. Kelas SmallFloat
   Kelas ini menangani kasus-kasus khusus seperti mengubah nilai "-0.00" menjadi "0.00" untuk menjaga kestabilan numerik dan menghindari kesalahan perhitungan yang disebabkan oleh batasan representasi angka desimal.

Kesimpulan

Kelas MatrixDeterminant memberikan dua pendekatan berbeda untuk menghitung determinan matriks dalam Java: menggunakan Operasi Baris Elementer (OBE) dan metode Adjoin (kofaktor). Kedua metode ini sangat bermanfaat tergantung pada ukuran matriks dan konteks perhitungannya.

IV. Matrix

Tujuan Kelas Matrix

Kelas Matrix dirancang untuk merepresentasikan dan memanipulasi matriks, dengan berbagai operasi seperti mendapatkan dan mengatur elemen, mencetak matriks, menukar baris, invers matriks, perkalian matriks, dan perhitungan determinan. Kelas ini juga memiliki metode bantu untuk membersihkan matriks serta utilitas seperti memeriksa baris yang identik atau berisi nol.

Metode Utama dan Tujuan:

1. **Constructor**: Menginisialisasi matriks dengan array 2D yang diberikan.
2. **get/set**: Mengakses dan memodifikasi elemen pada posisi tertentu.
3. **print**: Mencetak matriks dalam format yang dapat dibaca.
4. **swapRows**: Menukar dua baris, berguna untuk eliminasi Gauss.

5. **inverse**: Mencari invers matriks persegi menggunakan OBE (Elementary Row Operation) atau Ekspansi Kofaktor (Cofactor Expansion).
6. **multiply**: Mengalikan dua matriks.
7. **determinant**: Menghitung determinan menggunakan perluasan kofaktor atau operasi baris elementer.
8. **getString**: Mengembalikan matriks sebagai string terformat.
9. **getCleanedMatrix**: Membersihkan matriks dengan menghapus baris identik dan menempatkan baris nol di bagian bawah.

V. Penyelesaian SPL dengan Gauss

Penyelesaian dengan Gauss dalam garis besar meliputi dua langkah utama: **Pembentukan Row Echelon Form (REF)** dan **Back Substitution** untuk mendapatkan solusi. Berikut adalah penjelasan lebih lanjut:

Langkah-langkah:

- **Pembentukan REF (Row Echelon Form)**
  i. Pada langkah ini, kita menggunakan **forward elimination** untuk mengubah matriks augmented menjadi bentuk segitiga atas (row echelon form). Ini dilakukan dengan memilih elemen pivot di setiap baris dan melakukan eliminasi Gauss untuk mengurangi elemen di bawah pivot menjadi nol.
  ii. Jika elemen pivot adalah nol, baris ditukar dengan baris di bawahnya yang memiliki elemen terbesar untuk mencegah pembagian dengan nol.
- **Back Substitution**
  i. Setelah matriks berada dalam bentuk echelon, **back substitution** digunakan untuk menghitung solusi setiap variabel dari bawah ke atas. Untuk variabel bebas, solusi disesuaikan dengan variabel dependen lainnya.
  ii. Algoritma ini juga menangani kasus **tidak konsisten** di mana tidak ada solusi dan **variabel bebas** yang menyebabkan banyak solusi.

Penjelasan Kode:

A. **getEchelon(Matrix augmentedMatrix)**:
    1. Melakukan eliminasi maju dengan mengubah matriks augmented menjadi bentuk echelon. Baris-baris yang tidak berada dalam posisi pivot akan dihilangkan dengan menggunakan baris pivot.

B. **getResultFromEchelon(Matrix echelonMatrix)**:
    1. Setelah bentuk echelon diperoleh, metode ini melakukan substitusi mundur untuk mendapatkan solusi. Ini menangani kasus di mana persamaan memiliki variabel bebas, menghasilkan solusi yang mungkin tidak unik atau memiliki banyak solusi.

C. **solve(Matrix augmentedMatrix)**:
    1. Metode utama yang menggabungkan langkah-langkah eliminasi Gauss dan substitusi mundur untuk menyelesaikan SPL.

VI. Penyelesaian SPL dengan Gauss-Jordan

Metode **Gauss-Jordan** adalah teknik aljabar linear yang digunakan untuk menyelesaikan sistem persamaan linear (SPL) dengan mengubah matriks yang merepresentasikan sistem tersebut ke dalam bentuk **Reduced Row Echelon Form (RREF)**. Dalam bentuk ini, solusi dari sistem lebih mudah diidentifikasi melalui proses eliminasi dan substitusi balik.

Langkah-langkah penyelesaian SPL dengan metode Gauss-Jordan melibatkan:

1. Pembentukan Reduced Row Echelon Form (RREF)

1. Proses **eliminasi baris** dilakukan untuk mengubah matriks yang diperluas (augmented matrix) menjadi bentuk RREF. RREF adalah matriks di mana:
    - Elemen utama pada tiap baris (pivot) adalah 1.
    - Semua elemen di atas dan di bawah pivot adalah 0.
2. Langkah-langkah eliminasi yang dilakukan mencakup:

- **Pemilihan baris pivot**: Jika pivot pada diagonal adalah nol atau sangat kecil, dilakukan pertukaran baris dengan baris yang memiliki nilai terbesar di kolom tersebut.
- **Normalisasi baris pivot**: Pivot (elemen pada diagonal) dibagi dengan dirinya sendiri untuk menjadi 1.
- **Eliminasi baris lain**: Setelah baris pivot dinormalisasi, nilai-nilai pada kolom di bawah dan di atas pivot dijadikan 0 dengan menggunakan operasi baris elementer.

3. Proses ini diulang hingga semua baris memenuhi syarat bentuk RREF.

2. Mendapatkan Solusi dari Matriks RREF

Setelah matriks berada dalam bentuk RREF, solusi dari sistem dapat ditentukan melalui **substitusi balik**. Terdapat dua jenis solusi yang bisa didapatkan:

- **Solusi unik**: Jika setiap variabel memiliki satu nilai solusi yang pasti.
- **Solusi tak terbatas**: Jika terdapat variabel bebas, yaitu variabel yang tidak memiliki nilai solusi pasti dan bisa diisi dengan sembarang nilai.

3. Menangani Inkonistensi dan Variabel Bebas

- Jika terdapat **baris dengan semua koefisien 0 namun konstanta bukan nol**, maka sistem tidak konsisten dan tidak memiliki solusi.
- Jika suatu variabel tidak dapat dihitung secara langsung (variabel bebas), variabel tersebut ditandai dan diikutsertakan dalam solusi sebagai variabel bebas, dan sistem tetap memiliki solusi.

Komponen Kode yang Penting:

- **reduce**: Fungsi ini menjalankan eliminasi Gauss-Jordan dan menangkap langkah-langkah yang diambil jika opsi pencatatan langkah diaktifkan.
- **getResultFromReducedRowEchelon**: Fungsi ini mengekstraksi solusi dari matriks yang telah berada dalam bentuk RREF, baik secara simbolik (misalnya, `x1`, `x2`, dll.) maupun numerik (hasil angka langsung).

- **solve**: Fungsi ini menyelesaikan sistem SPL dengan menangkap langkah-langkah eliminasi yang dilakukan.
- **solveWithoutSteps**: Fungsi ini menyelesaikan SPL tanpa mencatat langkah-langkah yang dilakukan.

VII. Penyelesaian SPL dengan Cramer

Penyelesaian Sistem Persamaan Linear (SPL) menggunakan **Aturan Cramer** secara umum mencakup langkah-langkah berikut:

1. Menghitung Determinan dari Matriks Koefisien

- Pertama, determinan dari matriks koefisien SPL dihitung. Jika determinannya nol, sistem tidak memiliki solusi unik.
- Jika matriks memiliki lebih banyak persamaan daripada variabel (sistem overdetermined), matriks koefisien dipotong agar menjadi persegi dan kemudian dihitung determinannya.

2. Modifikasi Matriks untuk Setiap Variabel

- Untuk setiap variabel xix_ixi, kolom ke-iii pada matriks koefisien diganti dengan kolom dari matriks konstanta.
- Setelah kolom diganti, determinan dari matriks yang dimodifikasi dihitung.

3. Menghitung Nilai Variabel

- Setelah determinan matriks yang dimodifikasi ditemukan, nilai setiap variabel dihitung dengan menggunakan formula: xi=Determinant Matriks ModifikasiDeterminant Matriks Koefisienx_i = \frac{\text{Determinant Matriks Modifikasi}}{\text{Determinant Matriks Koefisien}}xi=Determinant Matriks KoefisienDeterminant Matriks Modifikasi
- Hasil dihitung untuk semua variabel.

4. Memeriksa Konsistensi untuk Persamaan Tambahan (Jika Ada)

- Jika ada lebih banyak persamaan daripada variabel, hasil yang didapat diverifikasi dengan substitusi ke persamaan tambahan. Jika hasil substitusi tidak konsisten, maka sistem dianggap tidak konsisten.

Alur Langkah dalam Kode:

1. **Inisialisasi Matriks Langkah** untuk menyimpan setiap langkah perhitungan.
2. **Verifikasi Determinan Matriks Koefisien**, jika nol, langsung lempar pengecualian (tidak ada solusi unik).
3. **Perhitungan Setiap Variabel** dilakukan dengan menghitung determinan matriks yang dimodifikasi dan dibagi dengan determinan matriks koefisien.
4. **Verifikasi Persamaan Tambahan** (jika SPL memiliki lebih banyak persamaan daripada variabel), memastikan konsistensi.

VIII. Penyelesaian SPL dengan invers

Penyelesaian Sistem Persamaan Linear (SPL) menggunakan **Invers Matriks** melibatkan langkah-langkah sebagai berikut:

1. Menghitung Invers dari Matriks Koefisien

- Langkah pertama adalah menghitung **determinan** dari matriks koefisien untuk memastikan bahwa matriks tersebut dapat di-invers. Jika determinannya 0, matriks tidak memiliki invers dan sistem tidak memiliki solusi unik.
- Jika SPL memiliki lebih banyak persamaan daripada variabel (overdetermined system), baris ekstra dipotong untuk membuat matriks persegi sehingga bisa dihitung inversnya.

2. Mengalikan Invers Matriks dengan Matriks Konstanta

- Setelah invers matriks koefisien dihitung, matriks tersebut dikalikan dengan matriks konstanta untuk menemukan solusi.
- Operasi ini dilakukan dengan: X=A−1⋅BX = A^{-1} \cdot BX=A−1⋅B
- X adalah solusi yang dicari,
- A−1A^{-1}A−1 adalah invers dari matriks koefisien,
- B adalah matriks konstanta.

3. Verifikasi Persamaan Tambahan

- Jika terdapat persamaan tambahan (lebih banyak persamaan daripada variabel), hasil solusi diverifikasi dengan mensubstitusi ke dalam persamaan tambahan tersebut. Jika tidak konsisten, maka sistem dianggap tidak konsisten.

Alur dalam Kode:

a. **Memotong Matriks jika Diperlukan**: Jika SPL memiliki lebih banyak persamaan daripada variabel, matriks koefisien dan konstanta dipotong agar memiliki dimensi yang cocok untuk perhitungan invers.
b. **Menghitung Invers Matriks Koefisien**: Jika determinan matriks koefisien tidak nol, invers dihitung.
c. **Mengalikan Invers dengan Matriks Konstanta**: Solusi dihitung dengan mengalikan invers matriks koefisien dengan matriks konstanta.
d. **Verifikasi Persamaan Tambahan**: Jika ada persamaan tambahan, hasil solusi diverifikasi untuk memastikan konsistensi.

IX. Interpolasi Polinomial

Kelas Interpolasi Polinomial di atas menggunakan metode eliminasi Gauss untuk melakukan interpolasi polinomial pada sekumpulan titik data yang diberikan.

1. Konstruktor dan Metode Utama

- Metode **`solve(double[] xValues, double[] yValues, double xToEvaluate)`** digunakan untuk menentukan persamaan polinomial yang melewati semua titik yang diberikan oleh `xValues` dan `yValues`. Metode ini juga menghitung nilai polinomial pada titik `xToEvaluate`.
- Proses utama dari metode ini terdiri dari beberapa langkah:
    1. **Validasi Input**:
        - Jika jumlah elemen dalam `xValues` tidak sama dengan `yValues`, maka program melempar `IllegalArgumentException`.
    2. **Membangun Matriks Augmentasi**:
        - Matriks augmentasi dibangun untuk menyusun sistem persamaan linear yang sesuai dengan interpolasi polinomial. Baris-baris dari matriks ini merepresentasikan persamaan polinomial dengan derajat 0 hingga n-1 (misalnya, 1, x, x^2, dst.). Kolom terakhir diisi dengan nilai `yValues`.
    3. **Eliminasi Gauss**:
        - Matriks augmentasi kemudian diubah menjadi objek `Matrix` dan diselesaikan menggunakan metode eliminasi Gauss.
        - Kode juga melakukan pengecekan apakah hanya terdapat satu titik unik di dalam matriks (dengan menghitung jumlah baris yang memiliki semua elemen nol), dan jika demikian, program akan menghentikan proses interpolasi.
    4. **Solusi dari Gauss**:
        - Setelah penyelesaian matriks menggunakan Gauss, hasil yang diperoleh adalah solusi berupa koefisien dari polinomial interpolasi. Koefisien ini kemudian dikonversi dari string (solusi dari eliminasi Gauss) ke array `double`.
        - Jika semua koefisien adalah nol, ini menunjukkan adanya variabel bebas, sehingga polinomial tidak dapat dihitung dengan tepat, dan pesan kesalahan akan dikembalikan.
    5. **Membangun Polinomial**:

        - Koefisien digunakan untuk membangun bentuk persamaan polinomial. Misalnya, jika koefisiennya adalah `a0`, `a1`, `a2`, maka persamaan akan berbentuk seperti `f(x) = a0 + a1x + a2x²`.
    6. **Evaluasi Titik**:
        - Setelah polinomial dibangun, metode ini juga menghitung nilai polinomial di titik `xToEvaluate` menggunakan metode `evaluatePolynomial`.

2. Metode Bantu (Helper)

- **`evaluatePolynomial(double[] coefficients, double x)`**:
    - Metode ini menghitung nilai polinomial di titik `x`, dengan menggunakan koefisien yang dihitung sebelumnya. Setiap koefisien dikalikan dengan pangkat dari `x` yang sesuai.
- **`convertSolution(String[] solution)`**:
    - Metode ini mengonversi solusi dalam bentuk string (misalnya hasil dari eliminasi Gauss) menjadi array `double`. Jika string tersebut mengandung "free variable" (variabel bebas), maka koefisien yang sesuai diatur menjadi nol, karena tidak ada nilai tetap yang dapat digunakan untuk variabel bebas.

3. Contoh Keluaran

- Jika proses berhasil, metode ini akan mengembalikan daftar yang berisi dua string:
    1. **Polinomial yang dibangun** – misalnya, `f(x) = 1.0000 + 2.0000x + 3.0000x²`.
    2. **Nilai dari polinomial pada titik yang dievaluasi** – misalnya, `f(2.0000) = 17.0000`.

  Namun, jika terdapat variabel bebas atau input tidak valid, pesan kesalahan yang sesuai akan dikembalikan.

Kesimpulan

Kelas ini memungkinkan Anda untuk:

- **Menentukan polinomial** yang melewati sekumpulan titik.
- **Memastikan titik** berada dalam jangkauan.
- **Menghitung** nilai polinomial di titik tertentu.

Kombinasi metode interpolasi polinomial dan eliminasi Gauss disini berguna untuk menghasilkan solusi yang akurat pada data yang sesuai dengan model polinomial, dengan penanganan yang baik terhadap kasus-kasus tepi seperti variabel bebas.

X. Regresi Linier Berganda

Kelas `RegresiLinier` di atas digunakan untuk menyelesaikan masalah **Regresi Linier Berganda** menggunakan **Normal Estimation Equation**. Kelas ini memungkinkan prediksi nilai pada data baru berdasarkan sejumlah fitur (variabel input) dengan menggunakan koefisien regresi yang dihitung dari data pelatihan.

1. Konstruktor dan Variabel

- **`private double[] b`**: Array ini menyimpan koefisien regresi yang dihitung. Koefisien ini akan digunakan untuk prediksi nilai di kemudian hari.

2. Metode solve(double[][] xValues, double[] yValues)

Metode ini digunakan untuk menghitung koefisien regresi linier berganda. Berikut adalah langkah-langkah yang terjadi dalam metode ini:

1. **Inisialisasi Variabel**:
    - `n` adalah jumlah titik data (baris dalam dataset).
    - `m` adalah jumlah fitur atau variabel bebas (kolom dalam dataset).
2. **Membangun Matriks X**:
    - Matriks **X** dibuat dari `xValues`, namun dengan tambahan satu kolom di paling kiri yang seluruh elemennya diisi dengan nilai 1.0. Ini untuk menangani **intercept** atau **b0**, yaitu konstanta dalam model regresi linier.

    - **Intercept** diperlukan agar model bisa menggeser hasil prediksi secara vertikal tanpa bergantung sepenuhnya pada variabel input.
3. **Matriks Y**:
    - Matriks **Y** adalah representasi dari nilai target `yValues` yang diubah menjadi matriks kolom dengan ukuran yang sesuai.
4. **Transposisi Matriks X**:
    - Metode ini menghitung transposisi dari **X** (X^T), yang nantinya digunakan untuk menghitung normal equation.
5. **Perkalian Matriks**:
    - Matriks **X^T * X** dan **X^T * Y** dihitung. Kedua matriks ini diperlukan untuk membentuk sistem persamaan linear yang dapat diselesaikan menggunakan metode eliminasi Gauss.
6. **Matriks Augmentasi**:
    - Setelah menghitung **X^T * X** dan **X^T * Y**, kita membuat **matriks augmentasi** yang berisi hasil tersebut. Matriks augmentasi ini berukuran `(m+1) x (m+2)` karena mencakup koefisien (b0, b1, ..., bm) serta kolom tambahan untuk nilai hasil (Y).
7. **Pengecekan Unik**:
    - Sebelum melanjutkan ke eliminasi Gauss, metode ini memeriksa apakah hanya ada satu titik data unik dalam matriks augmentasi. Jika hanya ada satu titik unik, maka model regresi linier tidak dapat dihitung dan program akan mengembalikan kesalahan (exception).
8. **Eliminasi Gauss**:
    - Matriks augmentasi kemudian diselesaikan menggunakan metode **eliminasi Gauss** (memanggil kelas `Gauss`), yang menghasilkan solusi berupa string yang menggambarkan nilai koefisien (b0, b1, ..., bm).
9. **Konversi Solusi**:
    - Solusi dari eliminasi Gauss yang berupa string diubah menjadi array `double[]` menggunakan metode `convertSolution()`. Setiap koefisien regresi dikonversi menjadi angka.
10. **Pengembalian Koefisien**:
    - Setelah semua langkah selesai, array `b[]` berisi koefisien regresi yang ditemukan, dan array ini dikembalikan oleh metode.

3. Metode predict(double[] xk)

Metode ini digunakan untuk **memprediksi** nilai `y` pada titik baru `xk` (misalnya data uji). Berikut adalah langkah-langkahnya:

1. **Validasi Model**:
   - Metode ini pertama-tama memeriksa apakah model telah dilatih (apakah `b` sudah diinisialisasi) dan apakah panjang dari `xk` sesuai dengan jumlah fitur yang diharapkan (jumlah koefisien regresi - 1).
   - Jika tidak, program akan melemparkan `IllegalArgumentException`.
2. **Kalkulasi Prediksi**:
   - Prediksi dimulai dengan nilai intercept `b[0]`, yang merupakan konstanta dalam regresi linier.
   - Kemudian, setiap fitur dalam `xk` dikalikan dengan koefisien regresi yang sesuai (`b[i+1]`) dan hasilnya dijumlahkan untuk mendapatkan prediksi akhir.

4. Metode convertSolution(String[] solution)

Metode ini mengonversi solusi dari eliminasi Gauss, yang awalnya berupa string, menjadi array `double`. Berikut adalah langkah-langkah dalam metode ini:

1. **Inisialisasi Array**:
   - `numVars` adalah jumlah koefisien yang harus dikonversi (berdasarkan panjang array solusi).
   - `result[]` adalah array yang akan menyimpan hasil dalam bentuk `double`.
2. **Pengolahan String Solusi**:
   - Setiap string dalam array solusi diolah untuk memeriksa apakah terdapat kata "free variable" yang mengindikasikan adanya variabel bebas.
   - Jika string mengandung kata tersebut, nilai koefisien diatur menjadi 0.0.
   - Jika tidak, program mencoba mengekstrak nilai numerik setelah tanda = dan mengonversinya menjadi angka `double`. Jika terjadi kesalahan dalam parsing, nilai tersebut juga diatur menjadi 0.0.

Kesimpulan

Kelas **RegresiLinier** ini mengimplementasikan regresi linier berganda menggunakan pendekatan **Normal Estimation Equation** dengan langkah-langkah berikut:

- Membentuk sistem persamaan berdasarkan data input.
- Menyelesaikan sistem tersebut menggunakan eliminasi Gauss.
- Menghasilkan prediksi berdasarkan koefisien regresi yang dihitung.

Ini memungkinkan untuk memprediksi nilai keluaran berdasarkan beberapa variabel input, dengan syarat model telah dilatih pada data yang representatif.

XI. Regresi Kuadratik Berganda

Kelas `RegresiKuadratik` digunakan untuk menyelesaikan masalah Regresi Kuadratik Berganda menggunakan Normal Estimation Equation. Kelas ini memperluas konsep regresi linier berganda dengan menambahkan elemen kuadratik dan interaksi antar fitur, memungkinkan model untuk menangkap hubungan non-linear antara variabel independen (x) dan variabel dependen (y).

1. Variabel dan Konstruktor

- `private double[] coef`: Array ini menyimpan koefisien regresi yang dihasilkan setelah menyelesaikan persamaan regresi. Koefisien ini mencakup intercept, istilah linier, kuadratik, dan interaksi antar fitur.

2. Metode solve(double[][] xValues, double[] yValues)

Metode ini digunakan untuk menghitung koefisien regresi kuadratik berganda. Berikut adalah langkah-langkah yang terjadi dalam metode ini:

Inisialisasi Variabel:

- `n`: Jumlah data atau titik pengamatan (baris pada dataset).
- `m`: Jumlah fitur (kolom dalam dataset).

- `XCols`: Menghitung jumlah kolom dalam matriks desain X, termasuk intercept, istilah linier, kuadratik, dan interaksi antar fitur.

Membangun Matriks X:

- Matriks X dibentuk dari `xValues` dengan beberapa penambahan:
    1. **Intercept**: Kolom pertama diisi dengan nilai 1.0 untuk menangkap intercept.
    2. **Istilah Linier**: Termasuk nilai asli dari fitur (x1, x2, ..., xm).
    3. **Istilah Kuadratik**: Setiap fitur dikuadratkan ($x1^2$, $x2^2$, ..., xm^2).
    4. **Istilah Interaksi**: Semua kombinasi produk dua fitur yang berbeda (misalnya x1 * x2, x1 * x3, dll.).

Matriks Y:

- Matriks Y dibentuk dari `yValues` dan diubah menjadi matriks kolom untuk persamaan normal.

Perkalian Matriks:

- `X^T`: Transpos dari matriks X.
- `X^T * X`: Produk matriks transpos dari X dengan X itu sendiri, membentuk sistem persamaan normal.
- `X^T * Y`: Produk matriks transpos dari X dengan Y, digunakan untuk membentuk matriks augmentasi.

Matriks Augmentasi:

- Matriks augmentasi digabungkan dari `X^T * X` dan `X^T * Y`, yang kemudian akan diselesaikan menggunakan eliminasi Gauss.

Pengecekan Unik:

- Sebelum melanjutkan eliminasi Gauss, dilakukan pengecekan apakah hanya ada satu titik data unik. Jika hanya ada satu titik unik, regresi kuadratik tidak dapat dihitung, dan program akan mengembalikan kesalahan.

Eliminasi Gauss:

- Metode eliminasi Gauss digunakan untuk menyelesaikan matriks augmentasi dan menghasilkan solusi berupa string yang menggambarkan nilai koefisien regresi (termasuk intercept, istilah linier, kuadratik, dan interaksi).

Konversi Solusi:

- Solusi yang berupa string diubah menjadi array `double[]` menggunakan metode `convertSolution()`. Setiap koefisien dikonversi dari string menjadi angka.

Pengembalian Koefisien:

- Array `coef[]` berisi koefisien regresi yang ditemukan dikembalikan oleh metode.

3. Metode predict(double[] xk)

Metode ini digunakan untuk memprediksi nilai `y` berdasarkan nilai baru `xk` (misalnya data uji). Berikut adalah langkah-langkahnya:

Validasi Model:

- Metode ini memeriksa apakah model telah dilatih (yaitu, apakah `coef` telah diinisialisasi) dan apakah panjang `xk` sesuai dengan jumlah fitur yang diharapkan. Jika tidak, program akan melemparkan `IllegalArgumentException`.

Kalkulasi Prediksi:

- **Istilah Intercept**: Prediksi dimulai dengan nilai intercept dari `coef[0]`.
- **Istilah Linier**: Nilai setiap fitur dalam `xk` dikalikan dengan koefisien linier yang sesuai dan dijumlahkan.
- **Istilah Kuadratik**: Setiap fitur dalam `xk` dikuadratkan, kemudian dikalikan dengan koefisien kuadratik dan dijumlahkan.

- **Istilah Interaksi**: Kombinasi dua fitur yang berbeda dalam `xk` dikalikan satu sama lain, kemudian dikalikan dengan koefisien interaksi yang sesuai dan dijumlahkan.

Hasil Akhir:

- Hasil prediksi `y` dikembalikan setelah semua komponen linier, kuadratik, dan interaksi dijumlahkan.

4. Metode convertSolution(String[] solution)

Metode ini mengonversi solusi yang diberikan oleh eliminasi Gauss (berupa string) menjadi array `double`. Berikut adalah langkah-langkahnya:

Inisialisasi Array:

- `numVars`: Jumlah variabel (koefisien yang harus dikonversi).
- `result[]`: Array untuk menyimpan nilai koefisien dalam bentuk double.

Pengolahan String Solusi:

- Jika string mengandung kata "free variable", nilai koefisien diatur menjadi 0.0.
- Jika string berisi nilai numerik, program mencoba mengekstrak nilai setelah tanda "=" dan mengonversinya menjadi angka double.
- Jika terjadi kesalahan dalam parsing, nilai diatur menjadi 0.0.

Kesimpulan

Kelas `RegresiKuadratik` ini memungkinkan model untuk memprediksi nilai keluaran berdasarkan fitur linier, kuadratik, dan interaksi antar fitur. Hal ini berguna dalam memodelkan hubungan yang lebih kompleks antara variabel independen dan variabel dependen dibandingkan dengan regresi linier sederhana.

XII. Bicubic Spline Interpolation

Bicubic Spline Interpolation adalah sebuah metode yang digunakan untuk melakukan interpolasi dua dimensi pada sebuah grid atau matriks data. Metode ini merupakan pengembangan dari interpolasi bilinear, yang memberikan hasil yang lebih halus dan akurat karena mempertimbangkan perubahan turunan parsial (derivatif) di sepanjang sumbu x dan y. Pada kode yang diberikan, proses interpolasi dilakukan dengan menyusun sistem persamaan linier yang diselesaikan menggunakan metode Gauss-Jordan.

1. Konstruktor Kelas

Kelas `BicubicSplineInterpolation` memiliki dua konstruktor:

- Satu konstruktor menerima input matriks 4x4 dan otomatis mengaktifkan pelacakan langkah (`trackSteps`).
- Konstruktor lainnya memungkinkan kita untuk menentukan apakah pelacakan langkah ingin diaktifkan atau tidak. Jika diaktifkan, langkah-langkah penyelesaian sistem persamaan akan direkam di objek `MatrixSteps`.

Kedua konstruktor memverifikasi bahwa ukuran matriks input harus 4x4, sesuai dengan kebutuhan bicubic spline interpolation.

2. Metode `populatePowers`

Metode ini digunakan untuk mengisi nilai pangkat dari variabel x dan y untuk digunakan dalam interpolasi. Misalnya, untuk setiap kombinasi (i, j), array `xPowers` dan `yPowers` diisi dengan pangkat dari i dan j hingga pangkat ke-3.

3. Metode `interpolate`

Ini adalah metode utama untuk melakukan interpolasi bicubic di titik (x, y). Beberapa hal penting dalam metode ini:

- **Validasi input**: `x` dan `y` harus berada dalam rentang 0 hingga 1.
- **Membuat matriks persamaan**: Sistem persamaan linier dibuat dengan memanfaatkan turunan parsial dari fungsi interpolasi di titik-titik grid.

- **Solusi sistem persamaan**: Persamaan linier diselesaikan menggunakan metode Gauss-Jordan, dan solusi yang diperoleh adalah koefisien dari polinomial kubik.
- **Interpolasi nilai akhir**: Setelah mendapatkan koefisien, interpolasi dilakukan dengan menghitung nilai akhir berdasarkan polinomial yang terbentuk.

Jika pelacakan langkah diaktifkan, semua langkah perhitungan (seperti matriks X, matriks solusi, dll.) akan disimpan di dalam `matrixSteps` untuk ditinjau lebih lanjut.

4. Metode `eq`

Metode ini mengisi baris-baris dari matriks persamaan yang mewakili syarat-syarat interpolasi bicubic, termasuk turunan pertama dan kedua terhadap x dan y.

5. Metode `getY`

Metode ini menghitung nilai turunan yang dibutuhkan untuk interpolasi, seperti turunan pertama terhadap x, turunan pertama terhadap y, dan turunan silang.

6. Metode `getSubMatrix` dan `getColumn`

Kedua metode ini adalah utilitas untuk memanipulasi matriks. `getSubMatrix` mengekstraksi sub-matriks dari matriks utama, dan `getColumn` mengekstrak sebuah kolom dari matriks.

7. Pelacakan Langkah dengan `MatrixSteps`

Jika pelacakan langkah diaktifkan, setiap perubahan pada matriks atau solusi akan direkam dalam objek `MatrixSteps`. Ini berguna untuk debugging atau melacak bagaimana solusi diperoleh.

Kesimpulan:

Metode Bicubic Spline Interpolation yang diimplementasikan di sini adalah cara efektif untuk mendapatkan interpolasi yang halus dan akurat pada data dua dimensi. Kode ini memanfaatkan turunan parsial untuk memperbaiki interpolasi dan menggunakan metode Gauss-Jordan untuk menyelesaikan sistem persamaan. Pelacakan langkah-langkah penyelesaian memungkinkan kita untuk memahami proses perhitungan lebih mendalam.

XIII. Image Resizer

Proses perubahan ukuran gambar (*Image Resize)* menggunakan metode Bicubic Spline Interpolation yang telah dijelaskan sebelumnya. Image Resize melibatkan beberapa langkah, yaitu:

1. **Konversi Gambar ke BufferedImage**
   Sebelum gambar dapat diproses untuk diubah ukurannya, gambar diubah dari objek `Image` JavaFX ke `BufferedImage` yang digunakan oleh AWT. Jika tipe gambar asli tidak sesuai dengan format standar seperti ARGB, gambar akan dikonversi ke tipe `TYPE_INT_ARGB`.
2. **Menghitung Faktor Skala**
   Dua slider digunakan untuk menentukan faktor skala lebar dan tinggi gambar. Faktor ini digunakan untuk menghitung dimensi baru dari gambar hasil resize.
3. **Menggunakan Bicubic Spline Interpolation**
   Bicubic spline interpolation diterapkan untuk memperhalus proses resize. Setiap piksel dalam gambar hasil akan diinterpolasi menggunakan metode bicubic berdasarkan piksel tetangga dalam gambar asli. Proses interpolasi ini dilakukan pada setiap channel warna (merah, hijau, biru, dan alpha) sehingga mendukung gambar dengan transparansi (RGBA).
4. **Penggunaan Multithreading untuk Menghindari UI Block**
   Proses resize dilakukan di dalam thread terpisah agar tidak menghalangi thread utama JavaFX, yang bertanggung jawab atas antarmuka pengguna. Dengan demikian, UI tetap responsif selama gambar sedang diproses.
5. **Canvas Update dan Rendering Gambar**
   Setelah gambar selesai diproses, ukuran kanvas diperbarui sesuai dengan dimensi gambar baru. Gambar kemudian digambar ulang di kanvas dengan mempertahankan rasio aspek. Proses ini juga mencakup sentralisasi gambar agar sesuai dengan canvas.

6. **Menyimpan Gambar Hasil**

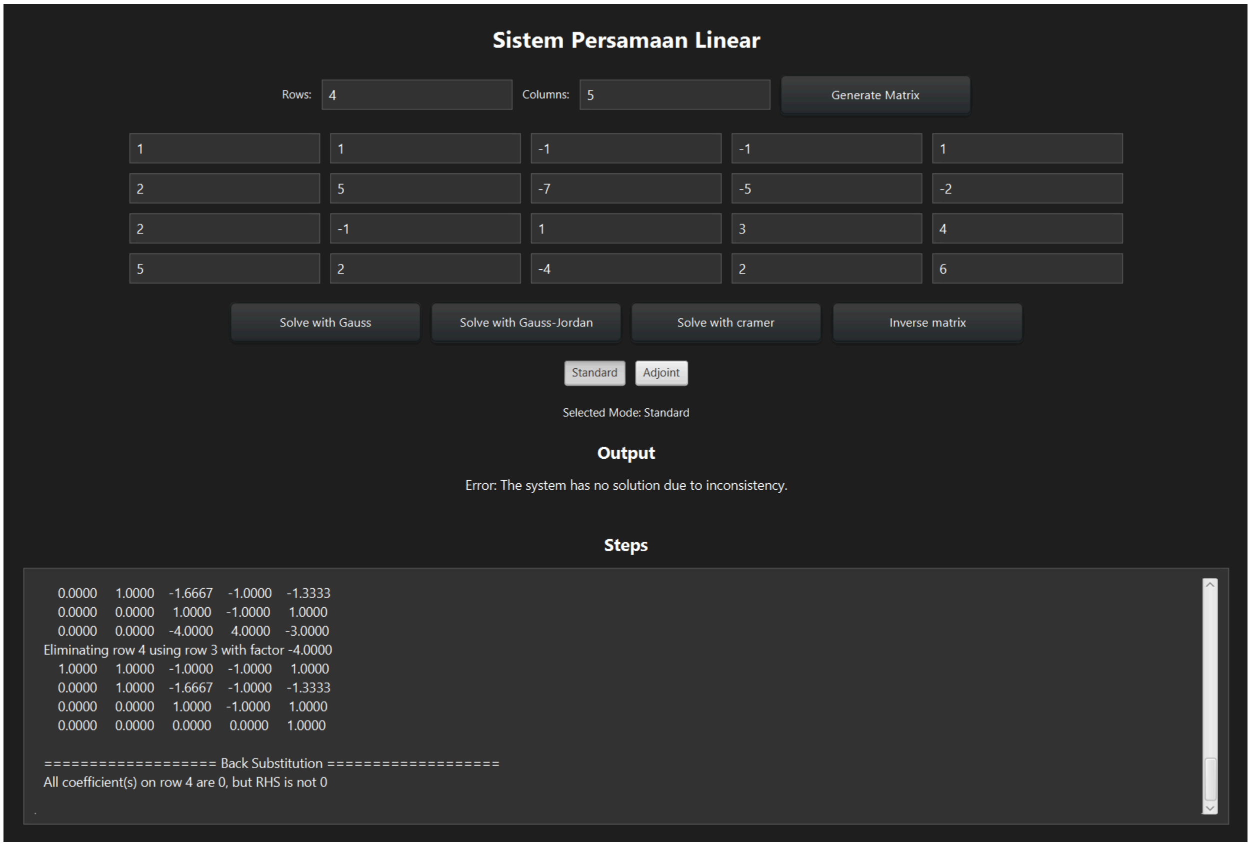
   Setelah gambar selesai diubah ukurannya, *user* dapat memilih untuk menyimpannya dalam format PNG. Proses penyimpanan juga dilakukan dalam thread terpisah untuk menjaga responsivitas UI.

7. **Indikator Loading**

   Indikator loading ditampilkan selama proses resize atau penyimpanan berlangsung, dan akan disembunyikan secara otomatis setelah proses selesai, menambah pengalaman pengguna yang lebih halus dan informatif.

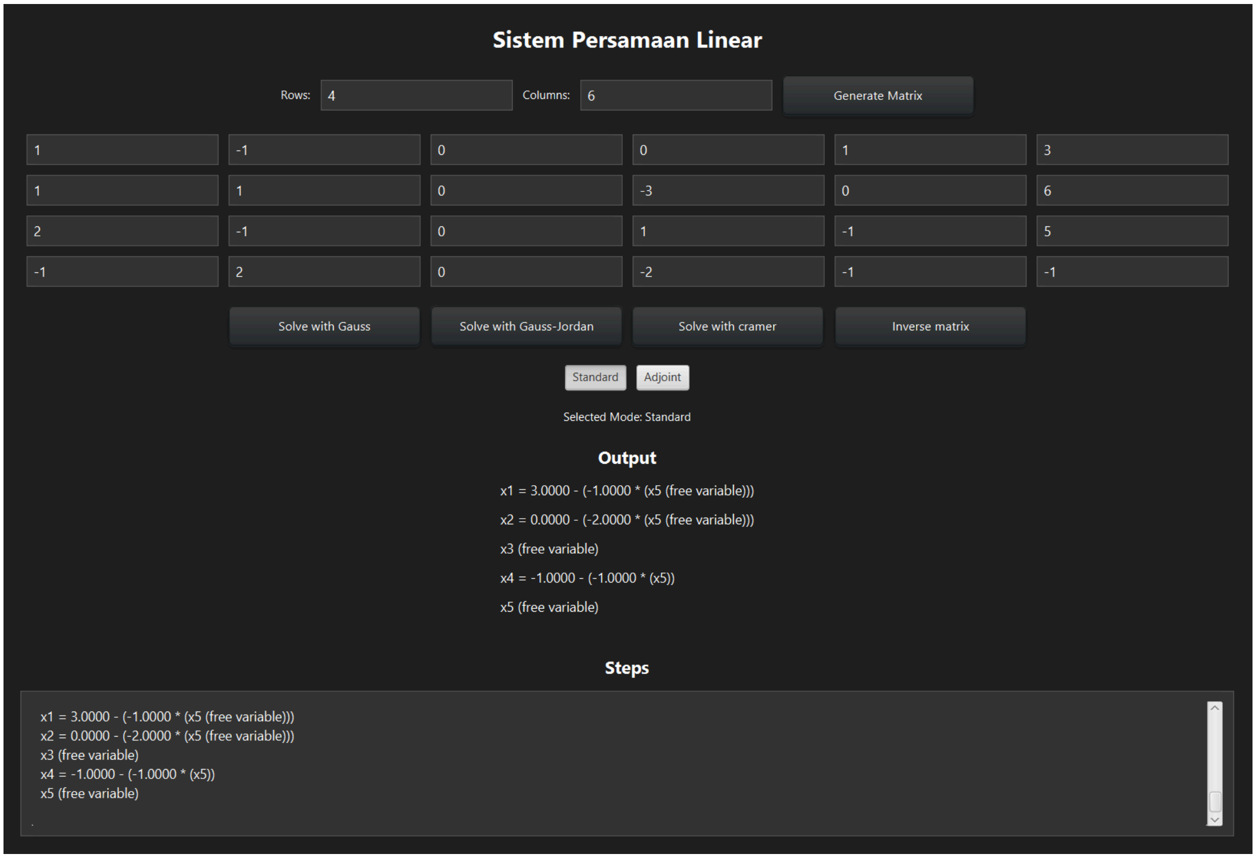
Bab 4. Eksperimen

4.1.1



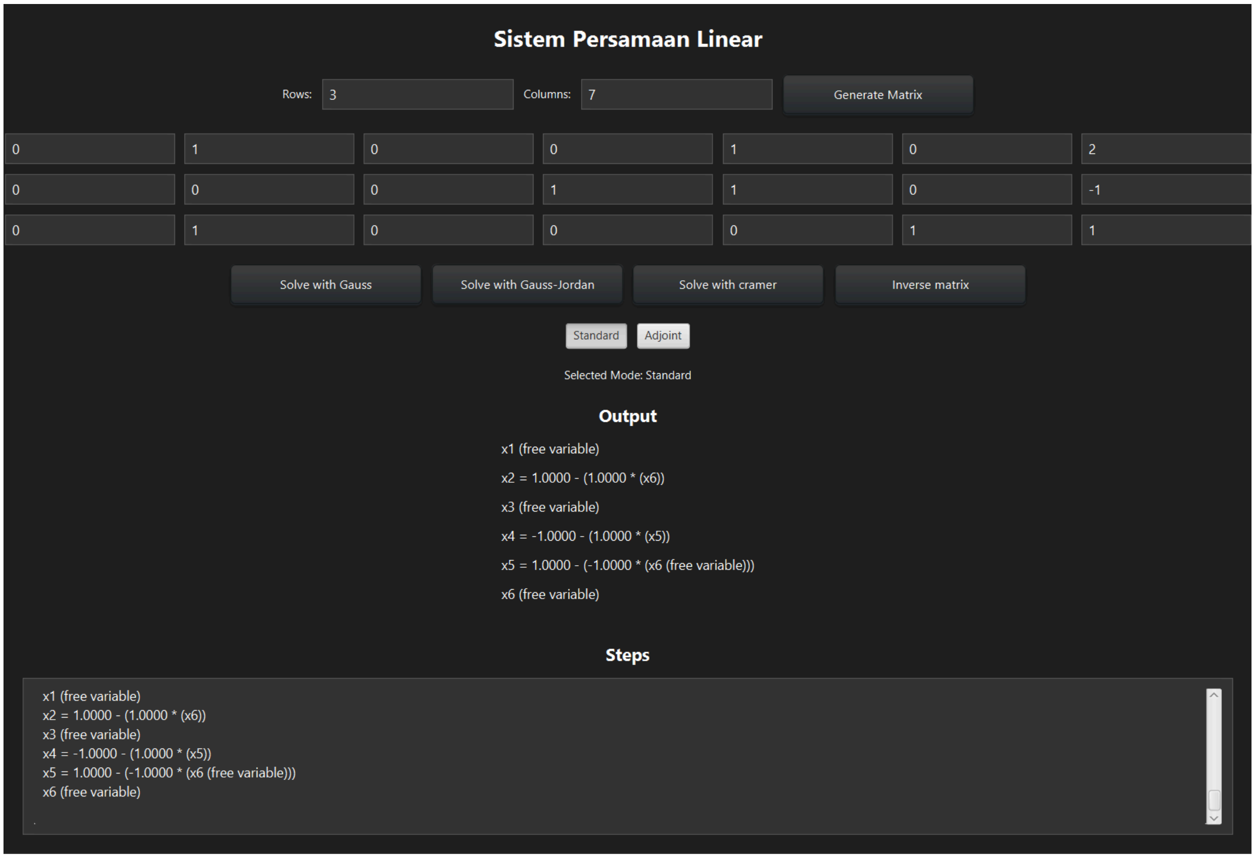
Diselesaikan dengan Gauss, tidak menghasilkan solusi karena ada 1 baris yang koefisien variabelnya 0 tapi hasilnya atau RHS tidak 0.

4.1.2



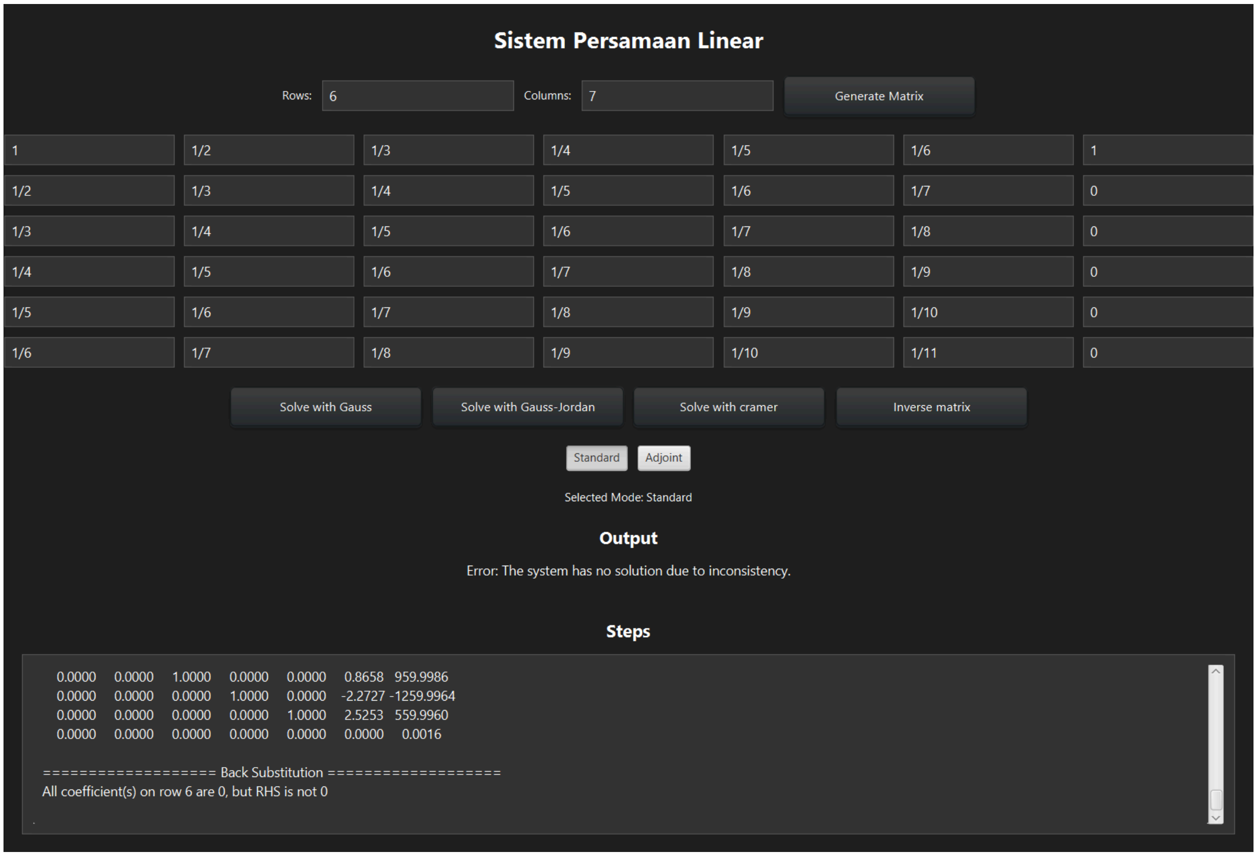
Diselesaikan dengan Gauss-Jordan, menghasilkan solusi parametrik.

4.1.3



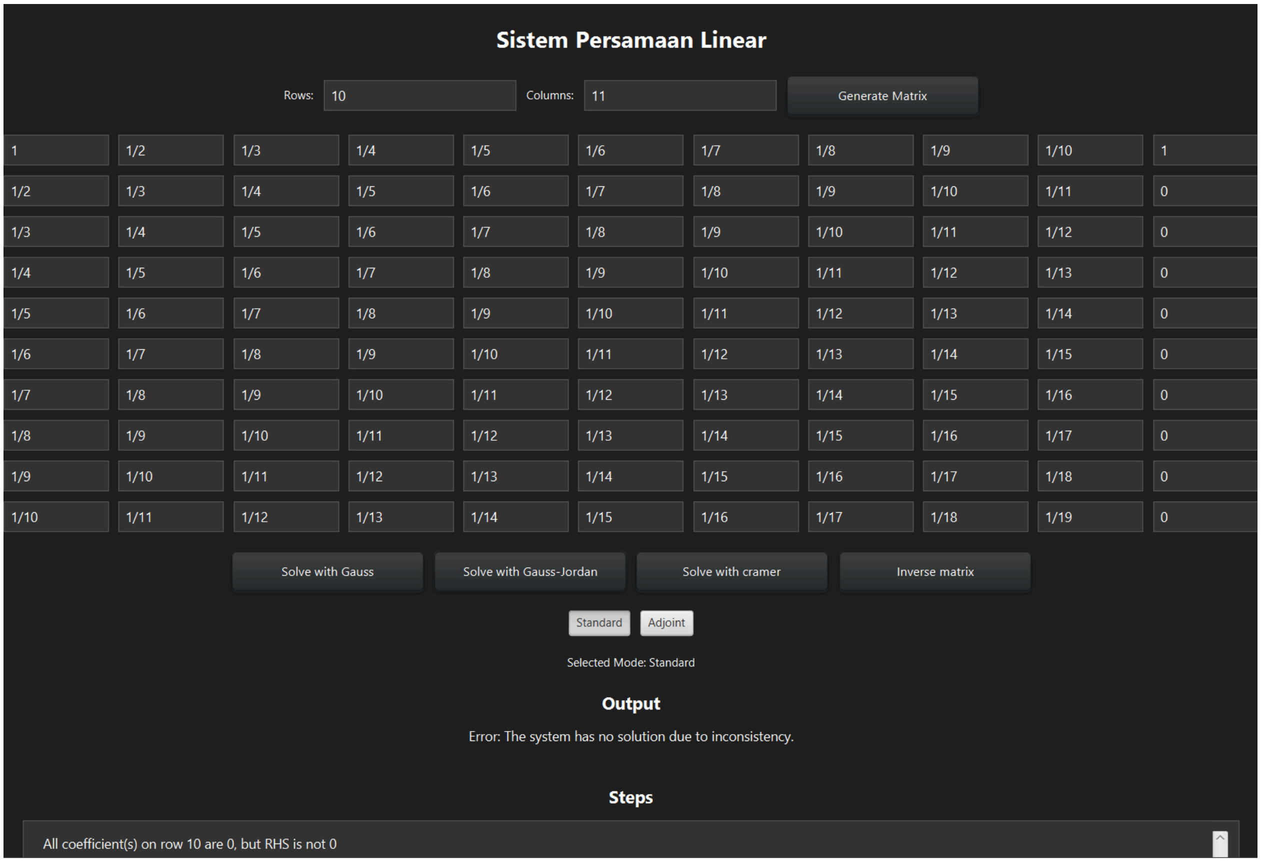
Diselesaikan dengan Gauss-Jordan, menghasilkan solusi parametrik.

4.1.4



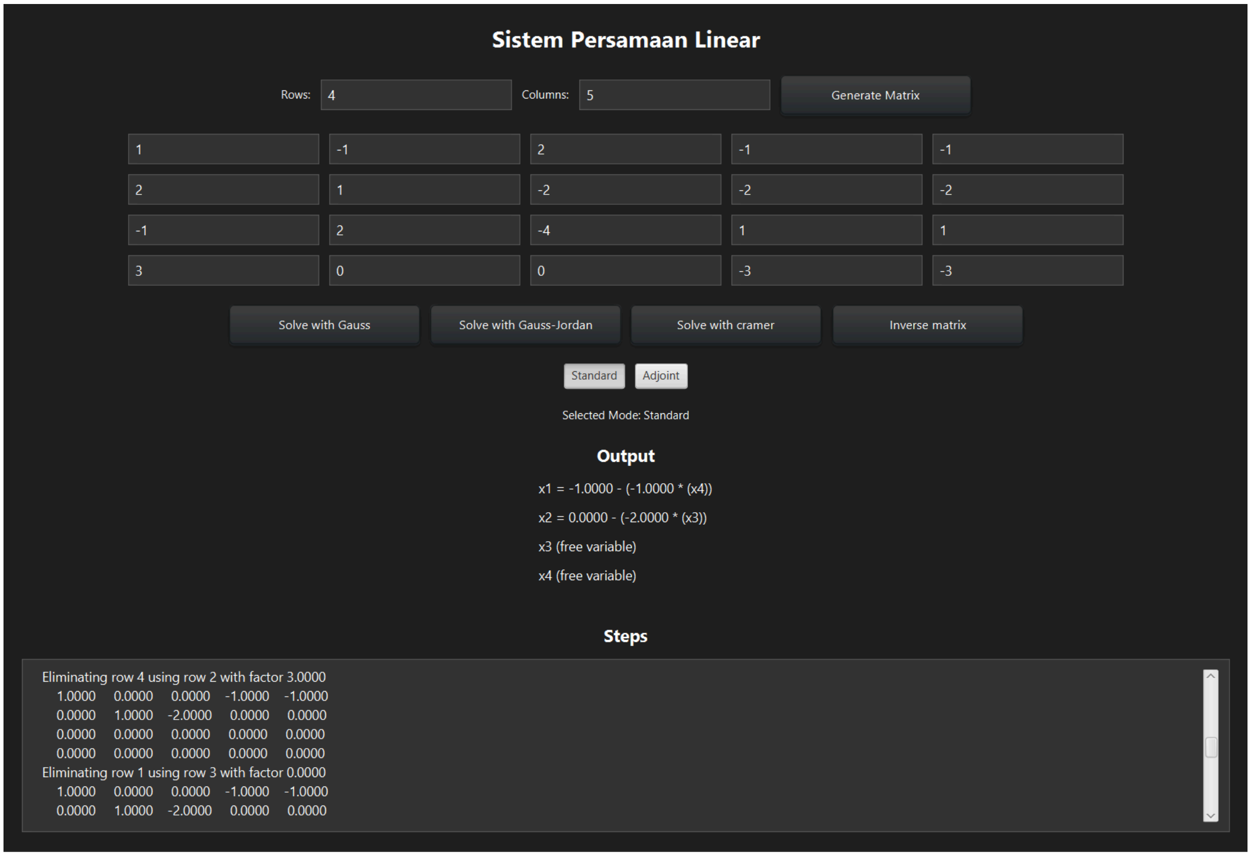
Diselesaikan dengan Gauss-Jordan, tidak menghasilkan solusi karena ada 1 baris yang koefisien variabelnya 0 tapi hasilnya tidak 0.

4.1.5



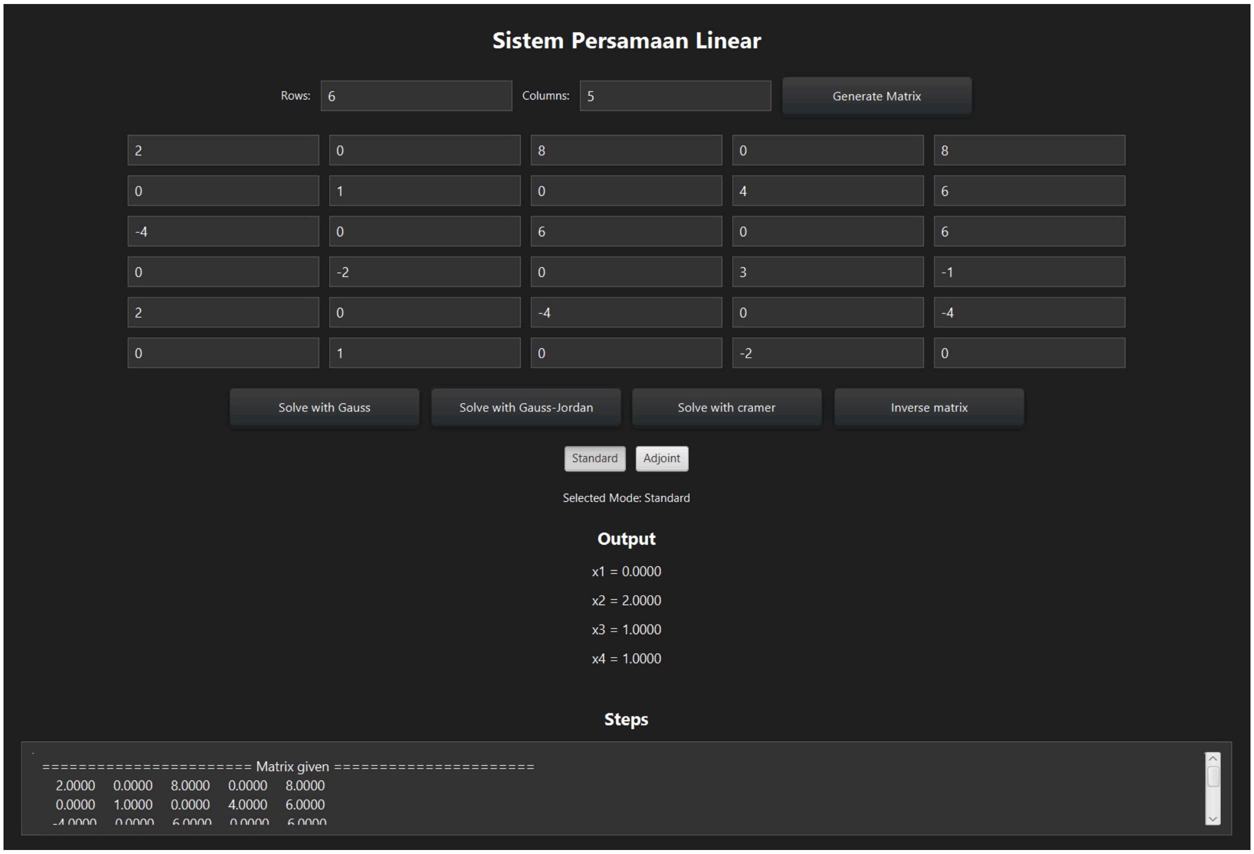
Diselesaikan dengan Gauss-Jordan, tidak menghasilkan solusi karena ada 1 baris yang koefisien variabelnya 0 tapi hasilnya tidak 0.

4.2.1



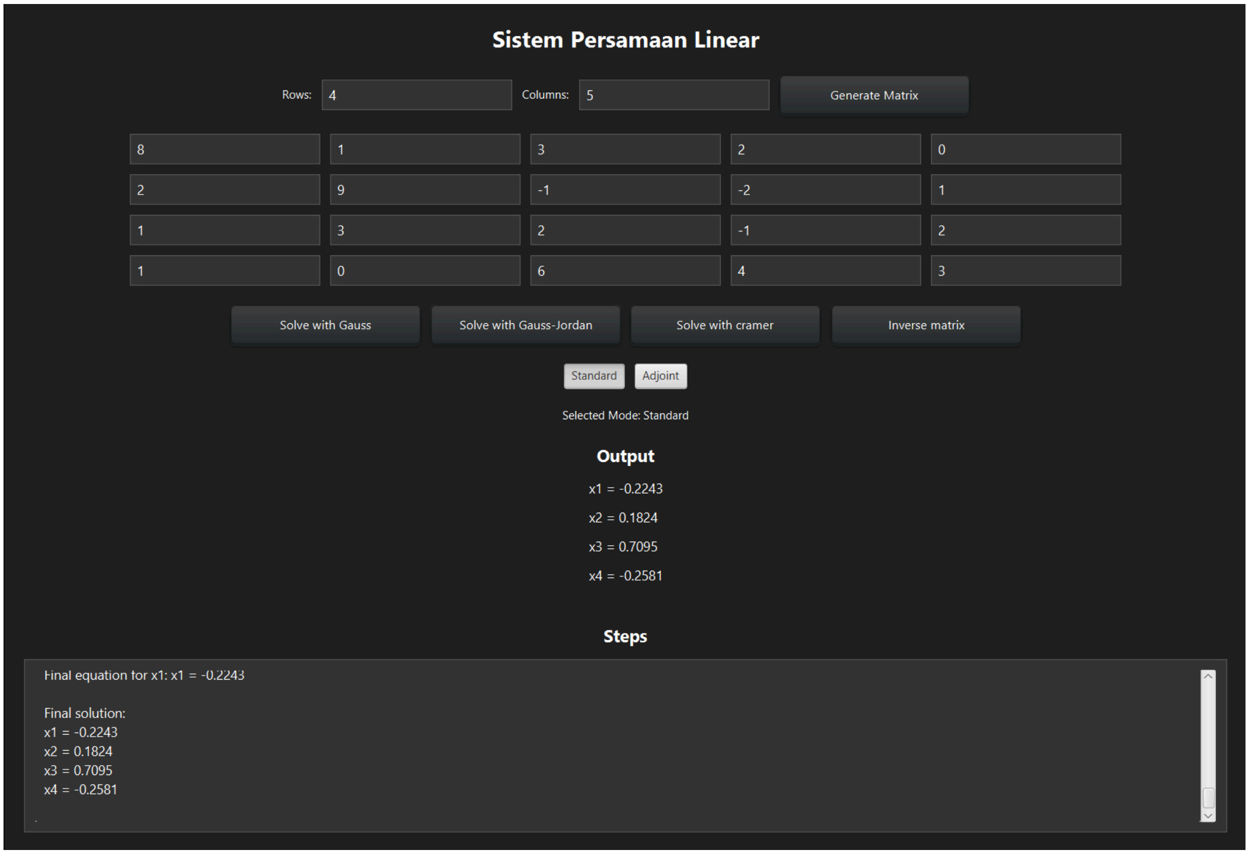
Diselesaikan dengan Gauss-Jordan, menghasilkan solusi parametrik.

4.2.2



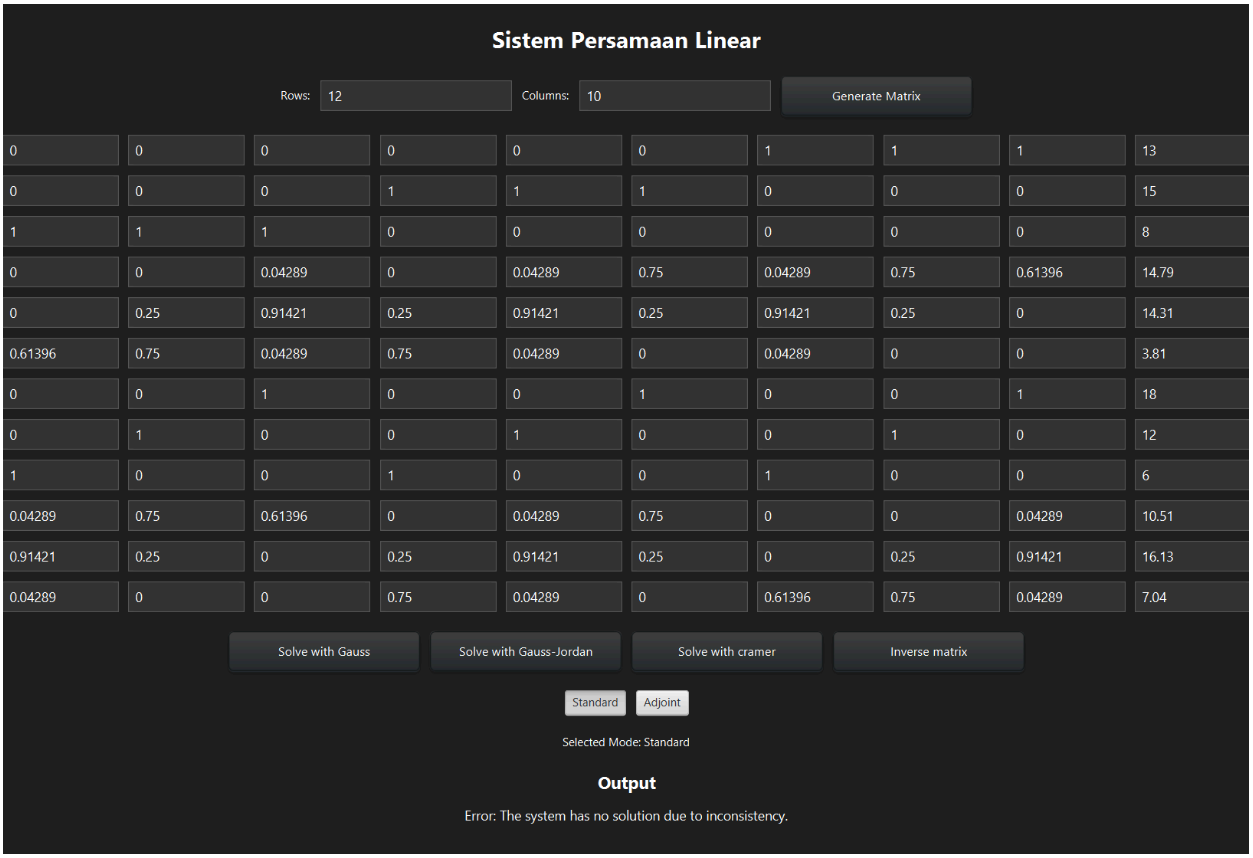
Diselesaikan dengan Gauss-Jordan.

4.3.1



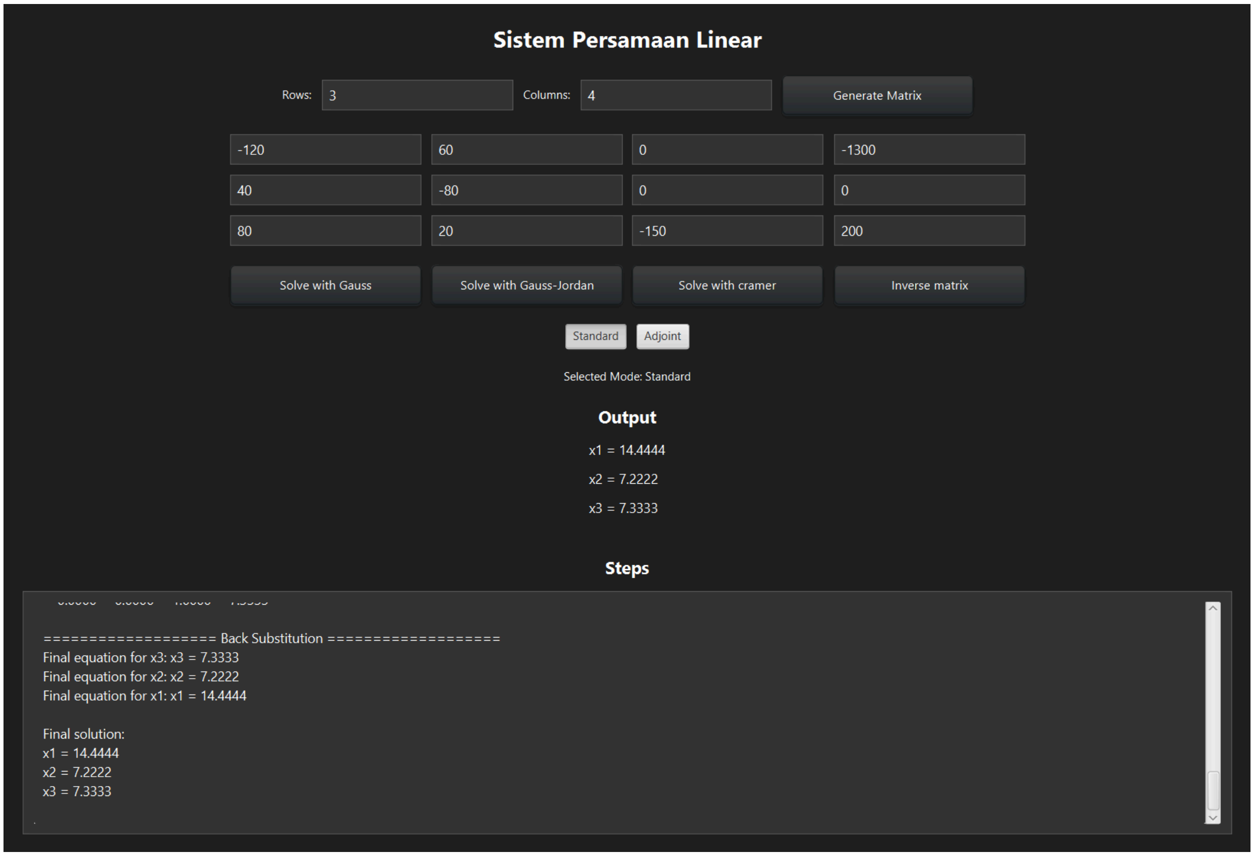
Diselesaikan dengan Gauss

4.3.2



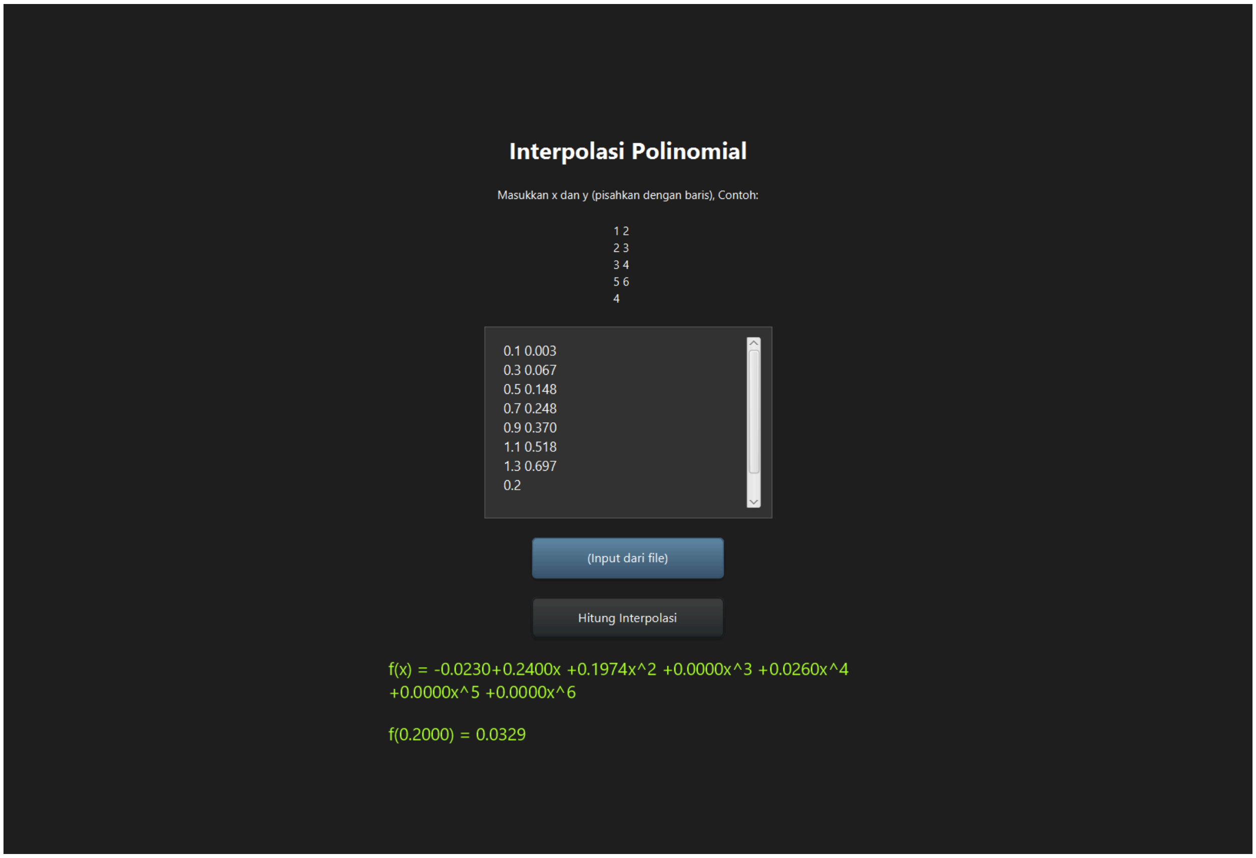
Disiselesaikan dengan Gauss-Jordan

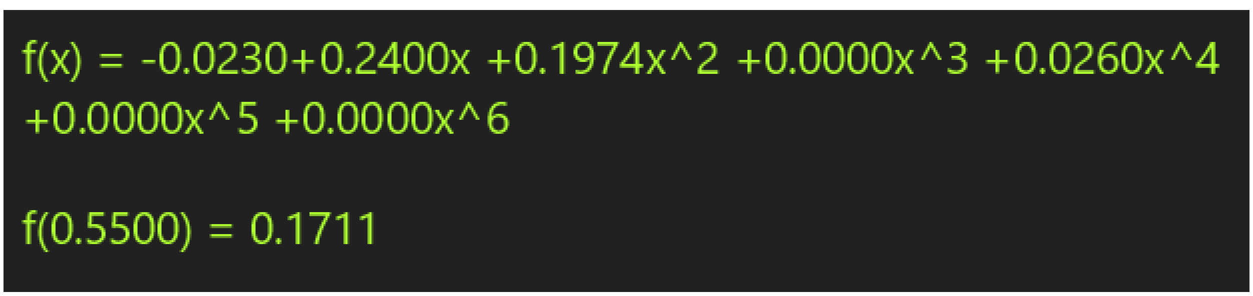
4.4

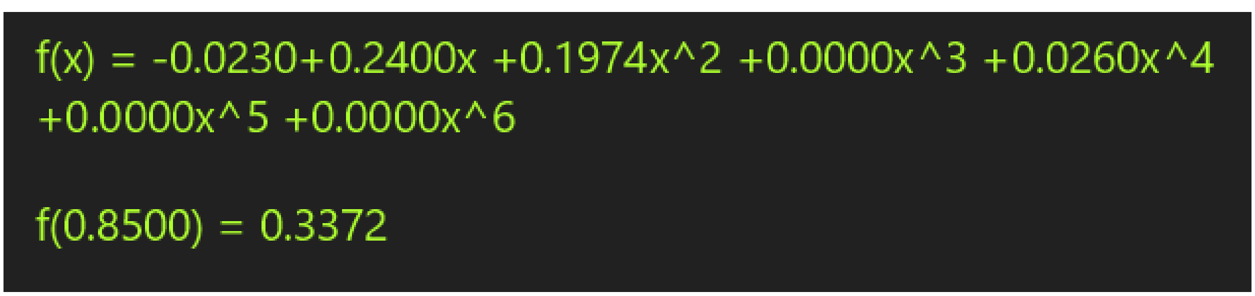


Jawaban dalam mg/m^3

4.5.1

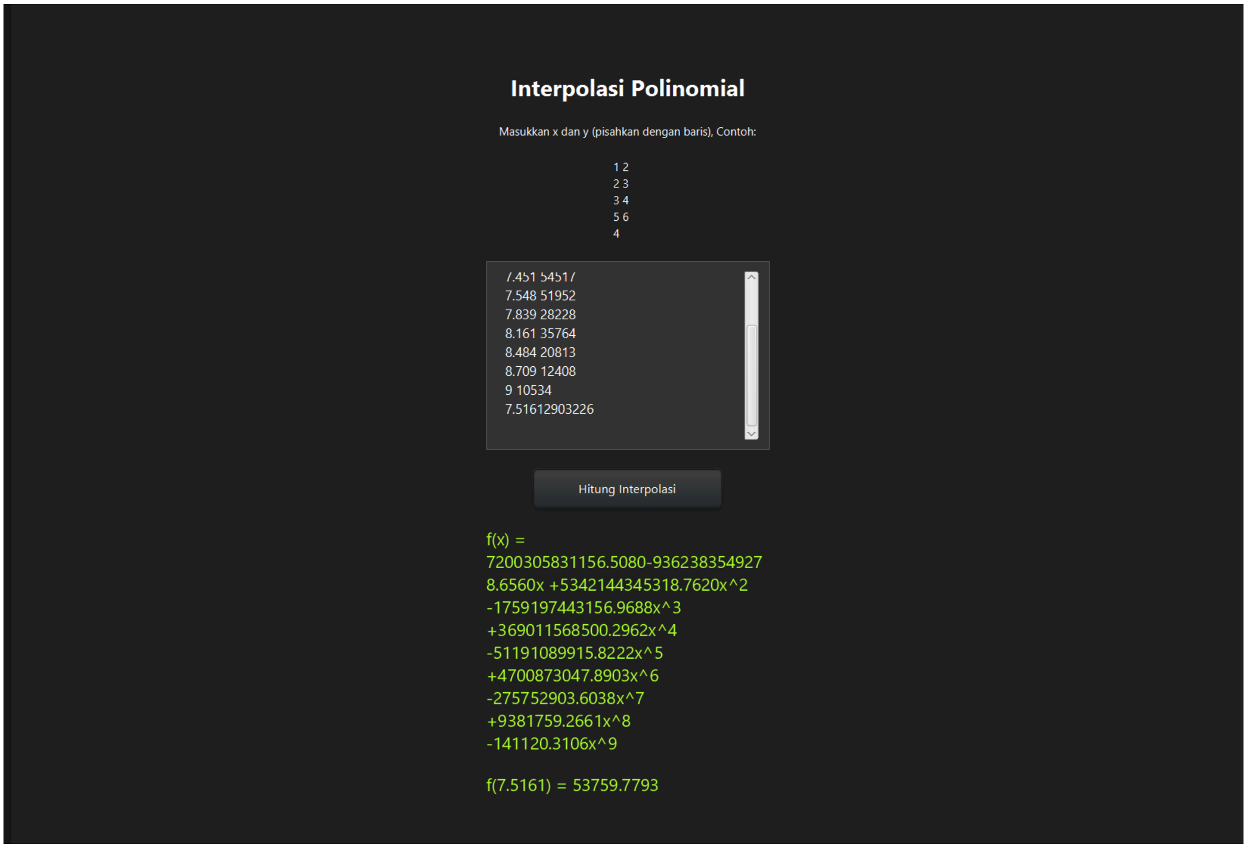

Interpolasi Polinomial
Masukkan x dan y (pisahkan dengan baris), Contoh:
1 2
2 3
3 4
5 6
4
0.1 0.003
0.3 0.067
0.5 0.148
0.7 0.248
0.9 0.370
1.1 0.518
1.3 0.697
0.2
(Input dari file)
Hitung Interpolasi
f(x) = -0.0230+0.2400x +0.1974x^2 +0.0000x^3 +0.0260x^4 +0.0000x^5 +0.0000x^6
f(0.2000) = 0.0329

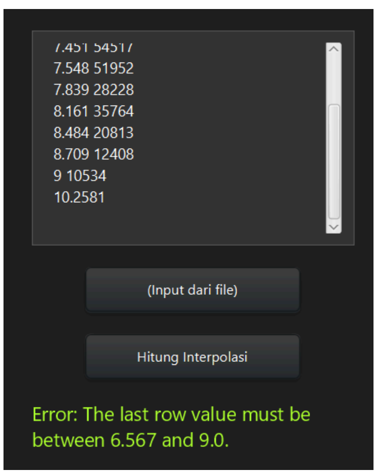

f(x) = -0.0230+0.2400x +0.1974x^2 +0.0000x^3 +0.0260x^4 +0.0000x^5 +0.0000x^6
f(0.5500) = 0.1711

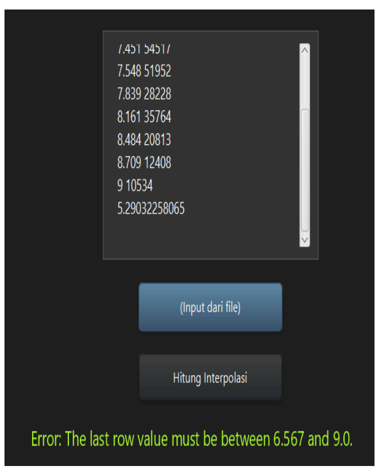

f(x) = -0.0230+0.2400x +0.1974x^2 +0.0000x^3 +0.0260x^4 +0.0000x^5 +0.0000x^6
f(0.8500) = 0.3372

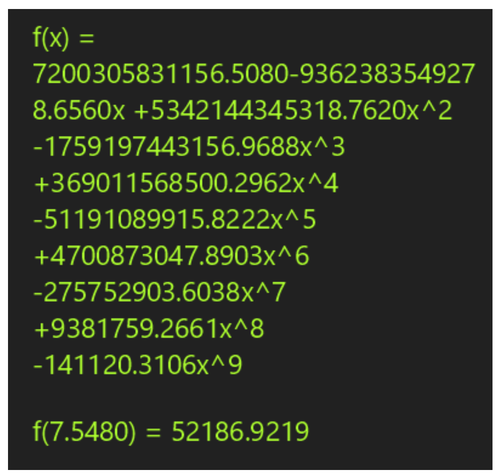
```
f(x) = -0.0230+0.2400x +0.1974x^2 +0.0000x^3 +0.0260x^4
+0.0000x^5 +0.0000x^6

f(1.2800) = 0.6774
```

4.5.2



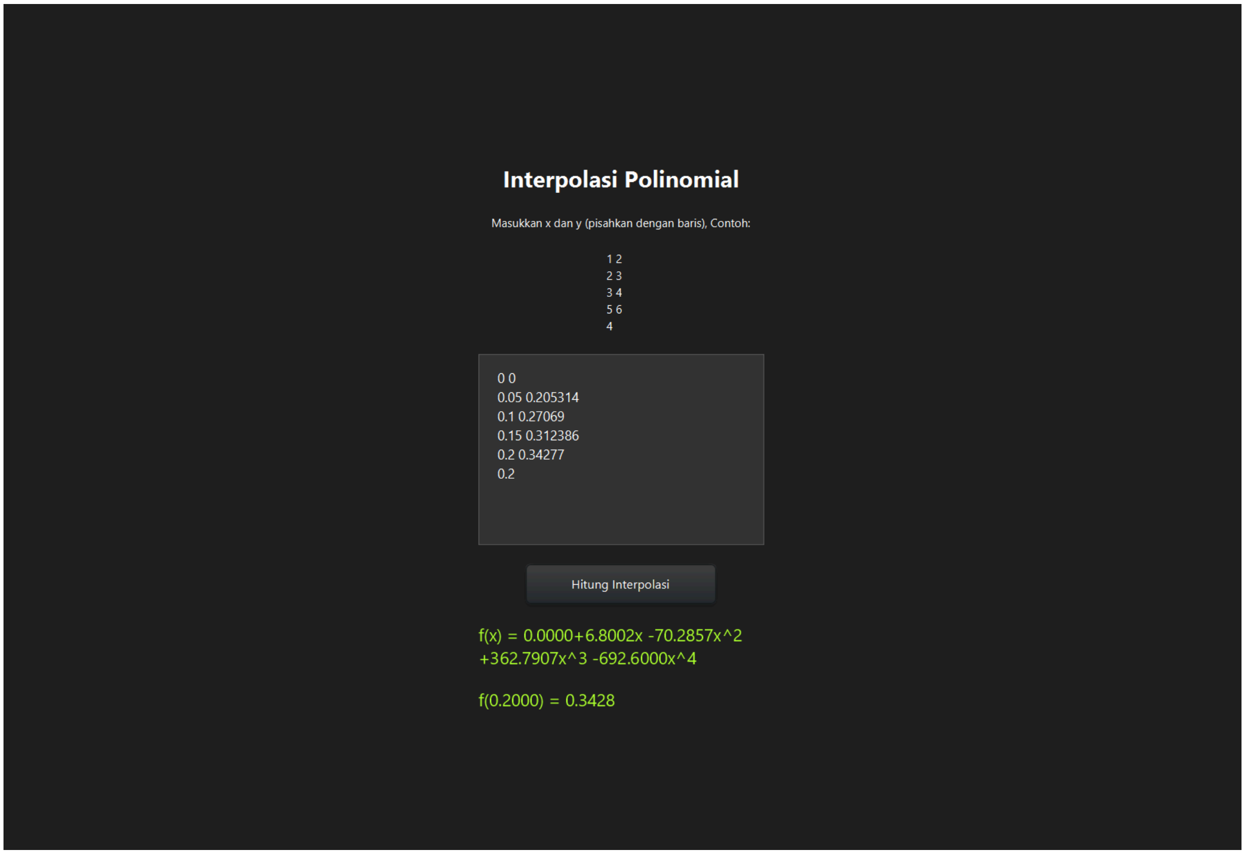






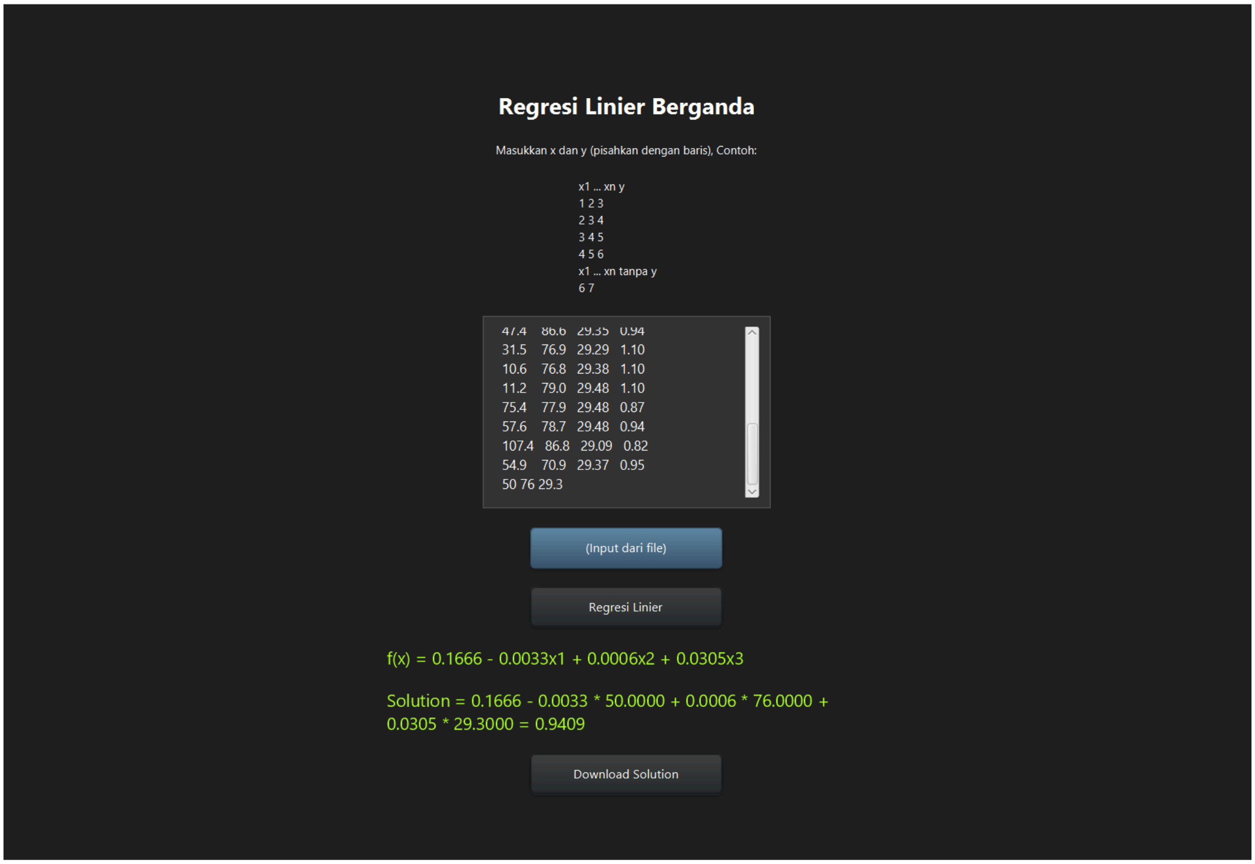
Test case 1 dan 2 tidak menghasilkan value karena input berada diluar minimum input x atau maximum input x. Test case 3 menghasilkan prediksi 52185.9219, nilai aktual nya 51952, error $\left|\frac{51952-52185.9219}{51952}\right| = 0.450\%$.

4.5.3



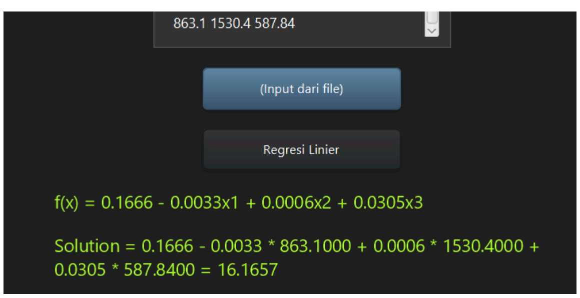
N=4, mengambil titik setiap $\frac{0.2}{4} = 0.05$, menghasilkan rumus $6.8002x - 70.2857x^2 + 362.7907x^3 - 692.6x^4$ untuk $0 \leq x \leq 2$.

4.6

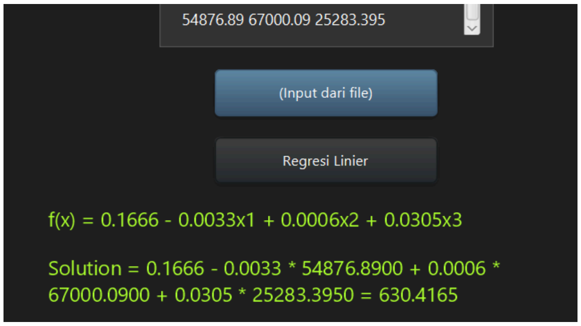


Menggunakan regresi linier berganda menghasilkan nilai $\beta_0 = 0.1666,\ \beta_1 = -0.0033,\ \beta_2 = 0.0006,\ \beta_3 = 0.0305$

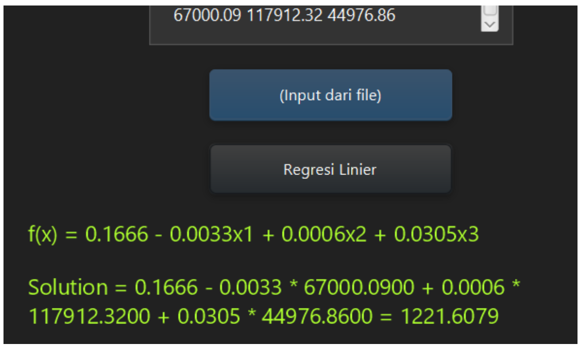
$f(50, 76, 29.3) = 0.9409$



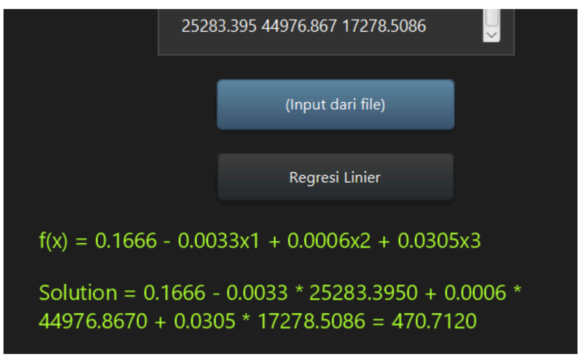
$16.1657 + 19\beta_0 = 19.3311$, nilai aktual 19.42 maka $error = \frac{(19.42-19.3311)}{19.42} x100\% = 0.4583\%$



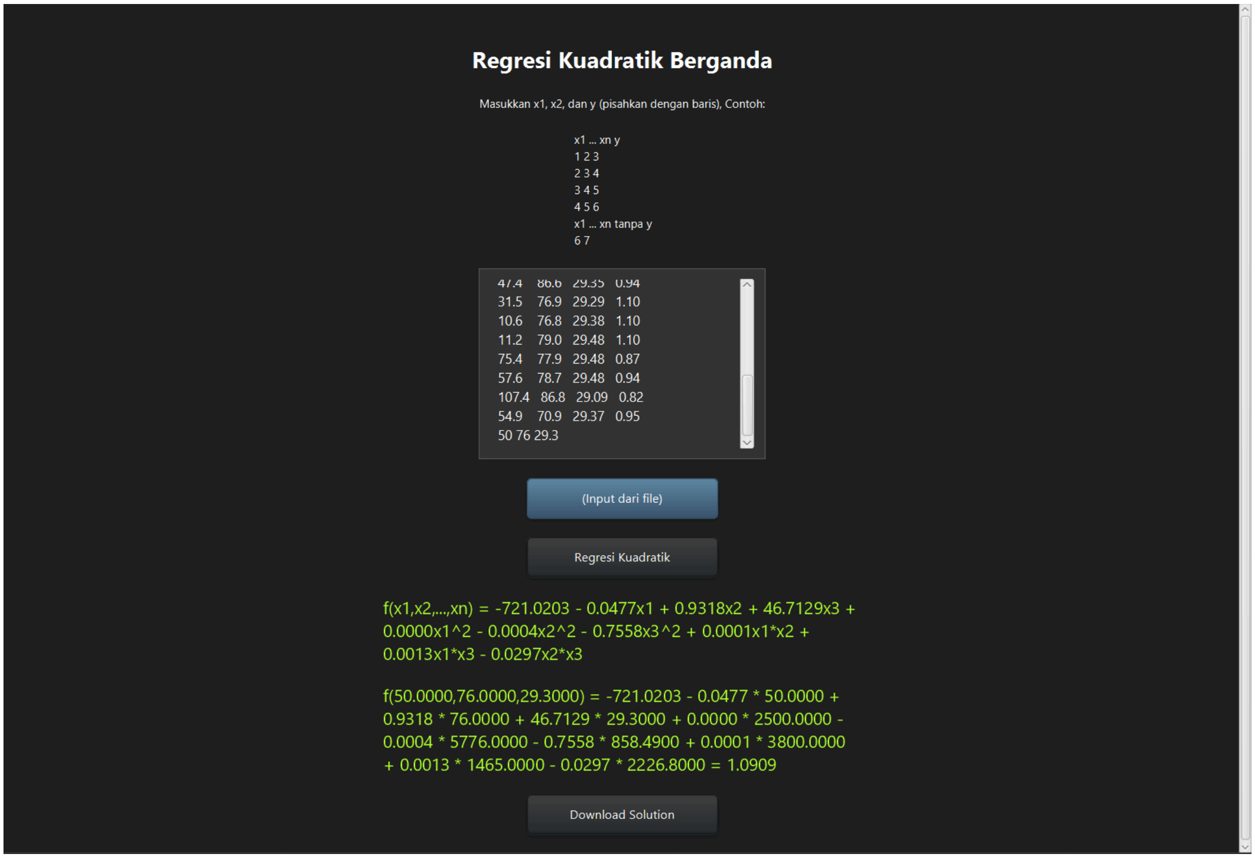
$630.4165 + 862.1\beta_0 = 774.0424,$ nilai aktual 779.477 maka $error = \frac{(779.477-774.0424)}{779.477} x100\% = 0.6972\%$



$1221.6079 + 1529.4\beta_0 = 1476.4060,$ nilai aktual 1483.437 maka $error = \frac{(1483.437-1476.4060)}{1483.437} x100\% = 0.47400$

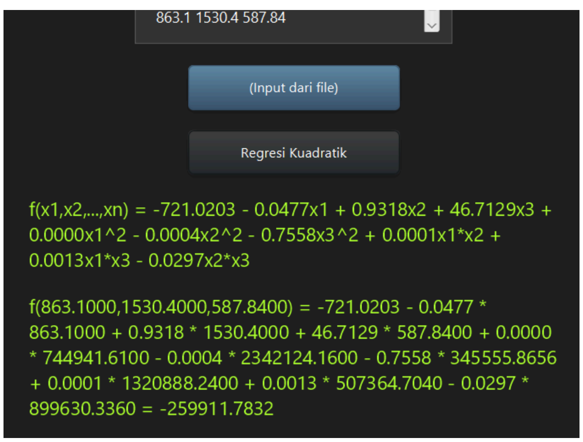


$470.7120 + 586.84\beta_0 = 568.4795,$ nilai aktual 571.1219 maka $error = \frac{(571.1219-568.4795)}{571.1219} x100\% = 0.4627$

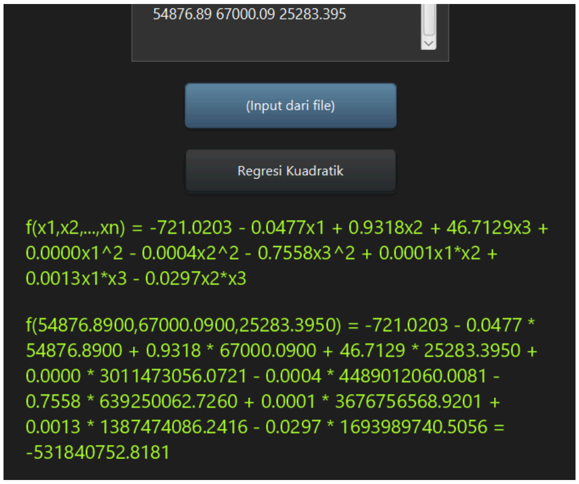


$\beta_0 = -\ 721.0203$

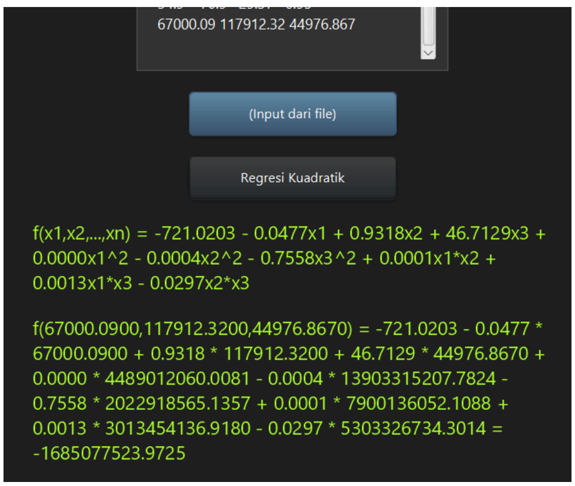
Menggunakan regresi kuadratik berganda $f(50, 76, 29.3)\ =\ 1.0909$



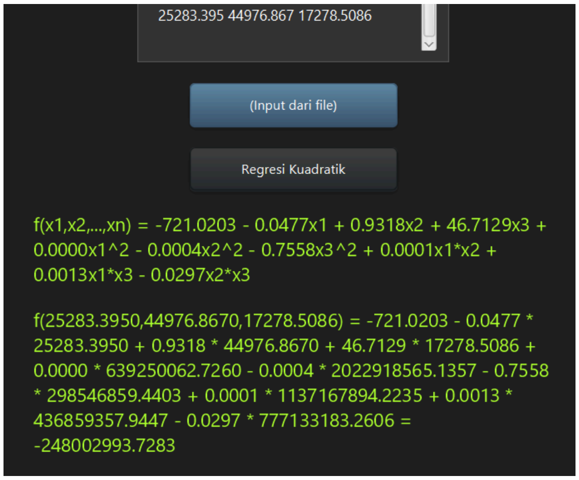
$-\ 259911.7832\ +\ 19\beta_0 = -\ 273611.1689,$ nilai aktual 19.42 maka $error\ =\ \frac{(19.42-(-273611.1689))}{19.42} x100\%\ =\ 1409014.36097\%$



$-\ 531840752.8181\ +\ 862.1\beta_0 =-\ 532462344.41873,$ nilai aktual 779.477 maka $error\ =\ \frac{(779.477-(-532462344.41873))}{779.477}x100\%\ =\ 68310305.9995\%$



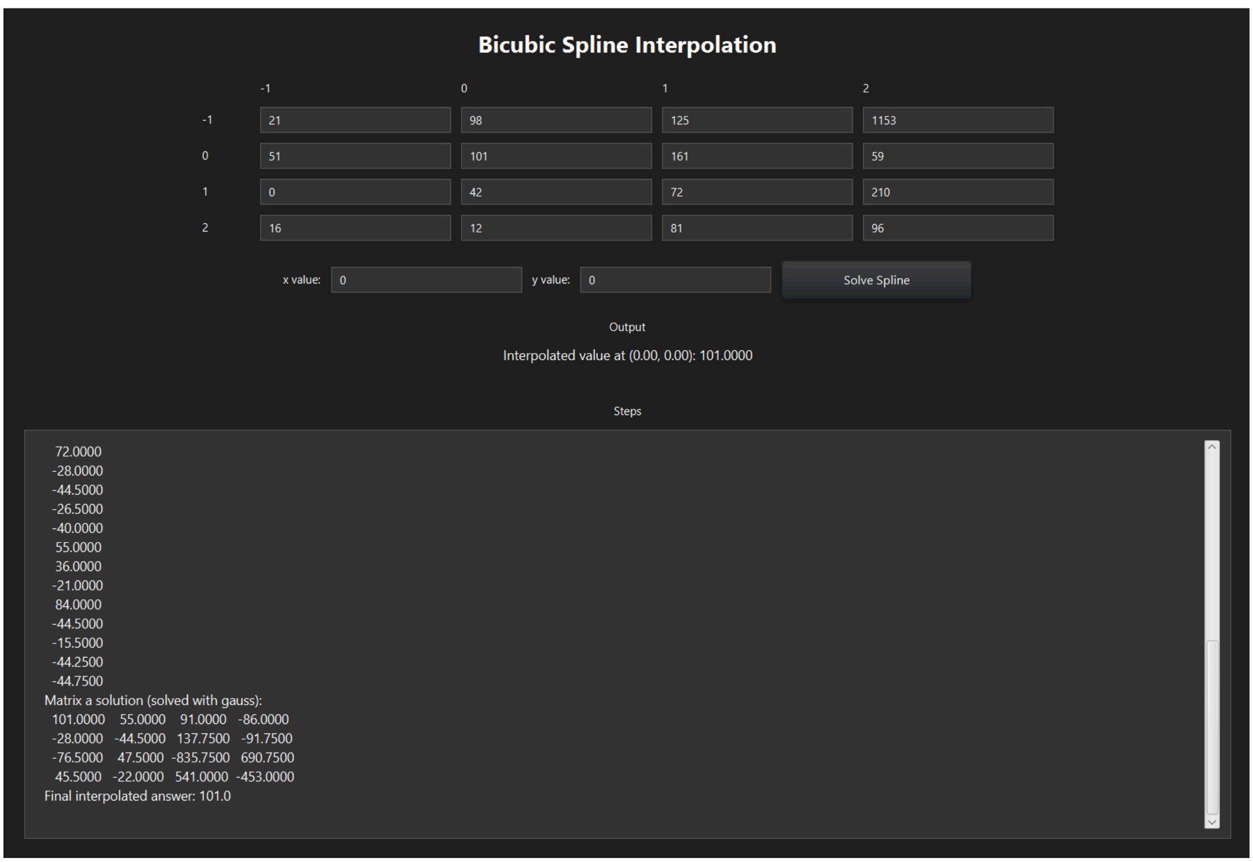
$-\ 1685077523.9725\ +\ 1529.4\beta_0 =-\ 1686180252.4193,$ nilai aktual 1483.437 maka $error\ =\ \frac{(1483.437-(-1686180252.4193))}{1483.437}x100\%\ =\ 113667229.269\%$

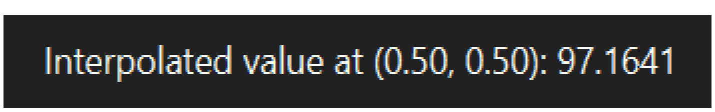


$-248002993.7283 + 586.84\beta_0 = -248426117.2811$, nilai aktual 571.1219 maka

$error = \frac{(571.1219-(-248426117.2811))}{571.1219} x100\% = 43498014.7676\%$

4.7





```
Interpolated value at (0.25, 0.75): 63.6230
```

```
Interpolated value at (0.10, 0.90): 50.1280
```

Bab 5. Kesimpulan, saran, komentar, dan refleksi

Kesimpulan

Matriks sangat berguna untuk menyelesaikan permasalahan yang akhirnya bisa diaplikasikan dalam keilmuan keinformatikaan

Saran

Test case yang panjang, kalau bisa diberi dalam bentuk teks atau file, kalau dalam bentuk gambar, ketidak telitiannya tinggi

Komentar

Untuk beberapa fitur pemberian contoh test case dan hasil sangat membantu, yang tidak ada membuat kami tidak yakin

Refleksi

Chisli

Lampiran

Referensi, tautan repository, tautan video (jika ada).

https://github.com/yonatan-nyo/Algeo01-23003 (repository)

https://www.mssc.mu.edu/~daniel/pubs/RoweTalkMSCS_BiCubic.pdf (referensi)

https://youtu.be/Rb-0uBOxrDQ (video demo dan penjelasan)