Laporan Tugas Besar 1

Aljabar Linear dan Geometri

Anggota Kelompok :

1. Hafid Abi Daniswara (13519028)
2. Fabian Savero Diaz Pranoto (13519140)
3. M. Alfandavi Aryo Utomo (13519211)

BAB I

Deskripsi Masalah

A. Abstraksi

Sistem persamaan linier (SPL) $Ax = b$ dengan n peubah (*variable*) dan m persamaan adalah berbentuk

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$

$$\cdot\cdot \qquad \cdot\cdot$$

$$\cdot\cdot \qquad \cdot\cdot$$

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$

yang dalam hal ini x_i adalah peubah, a_{ij} dan b_i adalah koefisien ∈ R. 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, atau hanya satu (unik/tunggal).

Sebuah matriks M berukuran $n \times n$

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

determinannya adalah

$$\det(M) = \begin{vmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nn} \end{vmatrix}$$

Determinan matriks M berukuran $n \times n$ dapat dihitung dengan beberapa cara: reduksi baris dan ekspansi kofaktor.

SPL memiliki banyak aplikasi dalam bidang sains dan rekayasa, dua diantaranya diterapkan pada tugas besar ini, yaitu interpolasi polinom dan regresi linier.

BAB II

Teori Singkat

A. Metode Eliminasi Gauss

Metode eliminasi Gauss ditemukan oleh Carl Freidrich Gauss. Metode ini dapat digunakan untuk memecahkan sebuah sistem persamaan linear. SPL direpresentasikan menjadi matriks teraugmentasi kemudian diubah menjadi sebuah matriks eselon baris melalui OBE (Operasi Baris Elementer). Sebuah matriks memiliki bentuk eselon baris jika memenuhi 4 kriteria sebagai berikut :

1. Pada baris yang terdapat elemen yang tidak nol, bilangan tak nol pertama di dalam baris tersebut adalah 1 atau 1 utama.
2. Pada baris yang elemennya nol semua, baris tersebut harus diletakkan di bagian bawah matriks.
3. Pada dua baris berurutan yang memenuhi kriteria pertama, baris yang lebih rendah adalah baris yang 1 utamanya di kolom yang lebih kanan daripada baris di atasnya.
4. Di bawah 1 utama harus nol.

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

1. Contoh matriks eselon baris

Setelah ditemukan bentuk eselon baris suatu SPL, dicari nilai setiap variabel melalui metode substitusi.

B. Metode Eliminasi Gauss-Jordan

Metode eliminasi Gauss-Jordan merupakan pengembangan dari metode eliminasi Gauss. Metode ini dicetuskan oleh Wilhelm Jordan, seorang insinyur Jerman, pada tahun 1887. Pada metode ini, SPL direpresentasikan menjadi matriks teraugmentasi kemudia diubah menjadi sebuah matriks eselon baris tereduksi melalui OBE. Matriks eselon baris tereduksi memiliki kriteria yang sama seperti matriks eselon baris namun di atas dan di bawah 1 utama harus nol.

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

Gauss ***Gauss-Jordan***

2. Perbedaan eselon baris dan eselon baris tereduksi

Keuntungan dari pemakaian metode eliminasi Gauss-Jordan adalah tidak perlu melakukan substitusi pada matriks eselon baris tereduksi untuk menentukan nilai setiap variabel.

C. Matriks Kofaktor

Misalkan A adalah matriks n x n dan C_{ij} adalah kofaktor entri a_{ij}. Maka matriks kofaktor dari A adalah

$$\begin{bmatrix} C_{11} & C_{12} & \dots & C_{1n} \\ C_{21} & C_{22} & \dots & C_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ C_{n1} & C_{n2} & \dots & C_{nn} \end{bmatrix}$$

Untuk mendapatkan kofaktor, perlu dicari minor sebuah matriks terlebih dahulu. Minor suatu matriks A dilambangkan dengan M_{ij} adalah determinan matriks bagian dari matriks A yang diperoleh dengan cara menghilangkan elemen-elemennya pada baris ke-i dan elemen-elemen pada kolom ke-j. Misalkan diketahui matriks A yang berordo 3 x 3

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Untuk mencari minor M_{11}, ditarik garis vertikal dan horizontal melalui baris ke-1 dan kolom ke -1 sehingga mendapatkan minornya

$$\begin{bmatrix} \cancel{a} & \cancel{b} & \cancel{c} \\ \cancel{d} & e & f \\ \cancel{g} & h & i \end{bmatrix} \qquad \begin{bmatrix} e & f \\ h & i \end{bmatrix}$$

Setelah ditemukan minor, dapat dicari kofaktor. Kofaktor suatu elemen baris ke-i dan kolom ke-j dari matriks A dilambangkan dengan $C_{ij} = (-1)^{i+j} M_{ij}$ sehingga matriks kofaktor A di atas menjadi

$$[|e\ f\ h\ i| \ -|d\ f\ g\ i| \ |d\ e\ g\ h| \ -|b\ c\ h\ i| \ |a\ c\ g\ i| \ -|a\ b\ g\ h| \ |b\ c\ e\ f| \ -|a\ c\ g\ i| \ |a\ b\ d\ e| \ ]$$

D. Matriks Adjoin

Adjoin matriks merupakan transpose dari matriks kofaktor. Transpose sendiri adalah pertukaran elemen pada baris menjadi kolom dan sebaliknya. Adjoin sering dilambangkan dengan lambang *adj*. Misalkan ada suatu matriks A, maka adjoin dari matriks tersebut dilambangkan $adj(A)$. Adjoin matriks sering digunakan dalam menentukan invers matriks.

E. Determinan

Determinan matriks dapat diartikan sebagai nilai yang mewakili sebuah matriks bujur sangkar. Simbol nilai determinan sebuah matriks A dinyatakan sebagai $det\,(A)$ atau $|A|$. Cara menghitung determinan sebuah matriks bergantung pada ukuran matriks tersebut.

Sifat-sifat determinan :

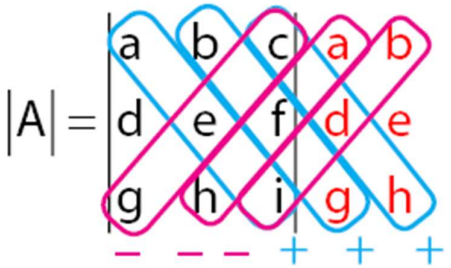
1. Jika matriks A mengandung sebuah baris atau kolom nol, $det\ det\,(A) = 0$.
2. Jika A^T adalah matriks transpose dari A, $det\ det\,(A) = det\,(A^T)$.
3. Jika A = BC maka $det\ det\,(A) = det\ det\,(B)\ det\,(C)$.
4. Sebuah matriks hanya mempunyai balikan jika dan hanya jika $det\,(A) \neq 0$.
5. $det\ det\left(A^{-1}\right) = 1/det\,(A)$.
6. Jika barisan pertama matriks A dikali k, $det\ det\,(B) = k\ det\,(A)$.
7. Jika terdapat pertukaran 2 baris, $det\ det\,(B) = -det\,(A)$.
8. Jika sebuah kelipatan baris ditambahkan ke baris lain, $det\ det\,(B) = det\,(A)$.

Pada matriks 2x2, cara untuk mencari determinan sebagai berikut :

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

Pada matriks 3x3 dan ke atas, terdapat berbagai cara untuk mencari determinan sebuah matriks. Metode-metodenya adalah :

1. Metode Sarrus
   Pada metode sarrus, dua kolom pertama sebuah matriks ditambahkan kembali menjadi dua kolom terakhir seperti pada gambar di bawah agar dapat dilakukan perhitungan determinan matriks sebagai berikut,



$$|A| = aei + bfg + cdh - ceg - afh - bdi$$

2. Metode Reduksi Baris

Determinan sebuah matriks dapat ditentukan melalui pengubahan sebuah matriks menjadi matriks segitiga menggunakan OBE (Operasi Baris Elementer). Untuk setiap n operasi pertukaran baris, dikalikan $(-1)^n$ pada hasil akhir perhitungan determinan. Apabila matriks telah diubah menjadi matriks segitiga, determinan dapat dicari melalui rumus:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix} \longrightarrow \det(A) = a_{11}a_{22}\,a_{33}a_{44}$$

$$A = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \longrightarrow \det(A) = a_{11}a_{22}\,a_{33}a_{44}$$

3. Ekspansi Kofaktor
   Determinan suatu matriks juga dapat ditemukan melalui metode ekspansi kofaktor. Dengan menggunakan kofaktor, maka determinan matriks

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

Dapat dihitung dengan salah satu dari persamaan berikut :

$\det(A) = a_{11}C_{11} + a_{12}C_{12} + \dots + a_{1n}C_{1n}$
$\det(A) = a_{21}C_{21} + a_{22}C_{22} + \dots + a_{2n}C_{2n}$
$\vdots$
$\det(A) = a_{n1}C_{n1} + a_{n2}C_{n2} + \dots + a_{nn}C_{nn}$

Secara baris

$\det(A) = a_{11}C_{11} + a_{21}C_{21} + \dots + a_{n1}C_{n1}$
$\det(A) = a_{12}C_{12} + a_{22}C_{22} + \dots + a_{n2}C_{n2}$
$\vdots$
$\det(A) = a_{1n}C_{1n} + a_{2n}C_{2n} + \dots + a_{nn}C_{nn}$

Secara kolom

Didefinisikan :
$C_{ij} = (-1)^{i+j}M_{ij}$ = Kofaktor entri a_{ij}

F. Kaidah Cramer

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

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

yang dalam hal ini, A_j adalah matriks yang diperoleh dengan mengganti entri pada kolom ke-j dari A dengan entri dari matriks

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

G. Matriks Balikan

Matriks balikan atau matriks invers adalah kebalikan dari sebuah matriks. Misalkan ada sebuah matriks A maka matriks balikannya adalah A^{-1}. Matriks A dikatakan memiliki balikan apabila $det\ (A) \neq 0$. Balikan dari sebuah matriks 2 x 2 dapat diperoleh melalui cara berikut :

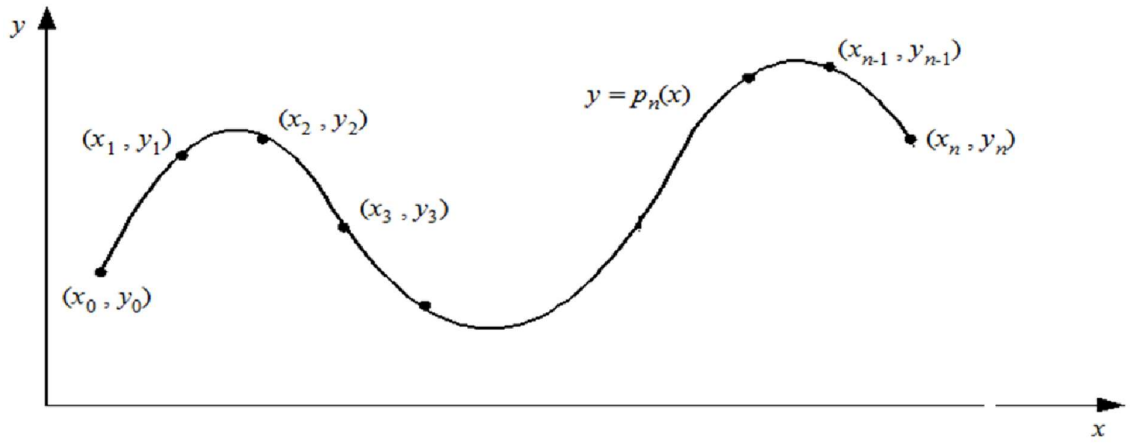
$$A^{-1} = \frac{1}{ad-bc}\ [d\ -b\ -c\ a\ ] \text{ dengan nilai determinan A} \neq 0\ .$$

Balikan dari matriks 3 x 3 ke atas dapat diperoleh melalui pencarian determinan dan adjoin lalu memakai rumus berikut :

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

H. Interpolasi Polinom

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.



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 $p_n(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$. 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 kami 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 $p_n(x) = a_0 + a_1x + a_2x^2 + \ldots + anxn$ untuk $i = 0, 1, 2, \ldots, n$, akan diperoleh n buah sistem persamaan lanjar dalam $a_0, a_1, a_2, \ldots, a_n$,

$$a_0 + a_1x_0 + a_2x_0^2 + \cdots + a_nx_0^n = y_0$$

$$a_0 + a_1x_1 + a_2x_1^2 + \cdots + a_nx_1^n = y_1$$

.....

$$a_0 + a_1x_n + a_2x_n^2 + \cdots + a_nx_n^n = y_n$$

Solusi sistem persamaan lanjar ini, yaitu nilai $a_0, a_1, \ldots, a_n$, 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 sisten 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: $p_2(9.2) = 0.6762 + 0.2266(9.2) - 0.0064(9.2)^2 = 2.2192$.

I. Regresi Linear Berganda

Regresi Linear (akan dipelajari lebih lanjut di Probabilitas dan Statistika) merupakan salah satu metode untuk memprediksi nilai selain menggunakan Interpolasi Polinom. Meskipun sudah ada rumus jadi untuk menghitung regresi linear sederhana, terdapat rumus umum dari regresi linear yang bisa digunakan untuk regresi linear berganda, yaitu.

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

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

$$nb_0 + b_1 \sum_{i=1}^{n} x_{1i} + b_2 \sum_{i=1}^{n} x_{2i} + \cdots + b_k \sum_{i=1}^{n} x_{ki} = \sum_{i=1}^{n} y_i$$

$$b_0 \sum_{i=1}^{n} x_{1i} + b_1 \sum_{i=1}^{n} x_{1i}^2 + b_2 \sum_{i=1}^{n} x_{1i}x_{2i} + \cdots + b_k \sum_{i=1}^{n} x_{1i}x_{ki} = \sum_{i=1}^{n} x_{1i}y_i$$

....

....

$$b_0 \sum_{i=1}^{n} x_{ki} + b_1 \sum_{i=1}^{n} x_{ki}x_{1i} + b_2 \sum_{i=1}^{n} x_{ki}x_{2i} + \cdots + b_k \sum_{i=1}^{n} x_{ki}^2 = \sum_{i=1}^{n} x_{ki}y_i$$

Sistem persamaan linier tersebut diselesaikan dengan menggunakan metode eliminasi Gauss.

BAB III

Implementasi dalam Java

Pada tugas ini, kelompok kami membuat dua *public class* yaitu class Matriks dan MatriksGauss. Penjelasan untuk kelas Matriks sebagai berikut:

```
public class Matriks {
    int baris;
    int kolom;
    double determinan;
    double[][] mtrx;
```

Class Matriks memiliki atribut integer baris, kolom dan double determinan serta bentuk mtrx ([][]).

```
/* KONSTRUKTOR */
public Matriks(int baris, int kolom, boolean diIsiJuga){
    this.baris  = baris;
    this.kolom = kolom;
    this.mtrx = new double[baris][kolom];
    if(diIsiJuga){
        isiMatriks();
        this.determinan = this.getDeterminanLokal(mtrx);
    }
}

public Matriks(int baris, int kolom, double determinan){
    this.baris  = baris;
    this.kolom = kolom;
    this.mtrx = new double[baris][kolom];
    this.determinan = determinan;
}
```

Bagian konstruktor matriks, yaitu memiliki 2 buah versi yang memiliki perbedaan pada parameter di ujungnya, pada yang pertama memiliki parameter boolean diIsi Juga yang apabila bernilai true maka matriks tersebut akan diisi melalui prosedur isiMatriks() dan assign determinannya dengan fungsi getDeterminanLokal(), apabila bernilai false maka tidak dilakukan pengisian pada matriks. Pada yang kedua memiliki parameter double determinan untuk assign nilai determinan matriks tersebut secara manual.

```java
/* CLASS RESET*/
public void resetMatriks(int baris,int kolom){
    this.baris = baris;
    this.kolom = kolom;
    this.mtrx = new double[baris][kolom];
    this.isiMatriks();
    this.determinan = this.getDeterminanLokal(mtrx);
}

public void resetMatriks(){
    this.isiMatriks();
    this.determinan = this.getDeterminanLokal(mtrx);
}
```

Bagian Class Reset, terdapat 2 buah versi dari prosedur resetMatriks, yaitu pertama yang memiliki parameter integer baris dan kolom, dan versi yang tanpa parameter. Untuk versi pertama yang menggunakan parameter baris dan kolom, apabila kami ingin mengubah nilai integer baris dan kolomnya, namun apabila kami menginginkan agar nilai baris dan kolomnya tetap, maka kami menggunakan versi 2.

```java
/* BASIC GETTER AND SETTER */

public boolean isMatrixSquare(){
    return (kolom==baris)&&(kolom>0&&baris>0);
}

private boolean isInversable(){
    return (this.getDeterminan()!=0) && (this.getBaris()==this.getKolom());
}

public int getBaris(){
    return this.baris;
}

public int getKolom(){
    return this.kolom;
}

public double getElement(int row,int col){
    return this.mtrx[row][col];
}

public void setElement(int row, int col, double value){
    this.mtrx[row][col] = value;
}
```

Bagian Basic Getter and Setter berisi beberapa fungsi sederhana:

1. **isMatriksSquare()**
   Fungsi yang mengembalikan kondisi boolean yang bernilai true apabila memiliki jumlah kolom dan baris yang sama dan keduanya lebih besar daripada 0.
2. **isInversable()**
   Fungsi yang mengembalikan kondisi boolean yang bernilai true apabila nilai determinan matriks tersebut bernilai tidak sama dengan 0, dan memiliki ukuran baris serta kolom yang sama.

3. **getBaris()**
   Fungsi yang mengembalikan nilai integer berupa jumlah baris matriks tersebut.
4. **getKolom()**
   Fungsi yang mengembalikan nilai integer berupa jumlah kolom matriks tersebut.
5. **getElement(int row, int col)**
   Fungsi yang mengembalikan nilai double berupa data matriks pada baris dan kolom keberapa.
6. **setElement(int row, int col, double value)**

   Prosedur yang mengubah nilai pada baris dan kolom tertentu dengan nilai double value.

```
public void tampilinMatriks(){
    for(int x=0;x<baris;x++){
        for(int y=0; y<kolom;y++){
            DecimalFormat df = new DecimalFormat("##.##");
            System.out.print(df.format(this.mtrx[x][y])+" ");
        }
        System.out.println("");
    }
}
```

Prosedur tampilinMatriks() digunakan untuk menampilkan data matriks dengan format desimal yang telah ditentukan dan jarak spasi antar data.

```
public void KaliBaris ( int i, double num){
    // Kita validasi Barisnya dulu
    // Baris tidak valid bila berada dibawah 0 dan diatas baris
    if (i < 0 || i > baris){
        System.out.printf("Baris tidak valid");
    }
    else{
        for (int j = 0; j < kolom; j ++ ){
            this.mtrx[i][j] *= num;
            // Jadi baris tersebut akan dikali dengan double num untuk semua kolomnya dari 0 sampai jumlahKolom-1
        }
    }
}
```

Prosedur KaliBaris (int i, double num) melakukan perkalian antara baris ke (i+1) yang diawali dengan pengecekan nilai i, apabila nilai i tidak valid maka mengeluarkan pesan “Baris tidak valid” dan apabila nilai i valid maka, dilakukan perkalian pada setiap kolom dalam baris tersebut dengan nilai double num yang telah di-input.

```
public Matriks KaliMatriks(Matriks M1, Matriks M2){
    Matriks res = new Matriks(M1.baris, M1.kolom,M1.determinan);
    for (int i = 0; i < res.baris; i++){
        for (int j = 0; j < res.kolom; j++){
            res.mtrx[i][j] = 0;
            for (int k = 0; k < this.kolom; k++){
                res.mtrx[i][j] += M1.mtrx[i][k] * M2.mtrx[k][j];
            }
        }
    }
    res.determinan = getDeterminan();
    return res;
}
```

Fungsi KaliMatriks(Matriks M1, Matriks M2) melakukan perkalian antara 2 buah matriks, yang diawali dengan membuat matriks penampung res dengan nilai baris dan kolom yang sama dengan M1 (asumsi ukuran M1 dan M2 sama), kemudian dilakukan iterasi untuk melakukan perkalian 2 buah matriks dengan 3 buah indeks (i, j, k), dan diakhiri dengan update nilai determinan matriks hasil, serta mengembalikan matriks res sebagai hasil akhir fungsi.

```
public double SumKolom (int i){
    double temp = 0;
    for (int j = 0; j < this.baris;j++){
        temp += this.mtrx[j][i];
    }
    return temp;
}
```

Fungsi SumKolom (int i) melakukan penjumlahan semua elemen pada kolom ke-(i+1) yang mereturn hasil penjumlahan tersebut.

```
public Matriks KaliKolom (int i,int j){
    Matriks res = new Matriks(this.baris, 1, false);
    for (int k = 0; k < res.baris; k++){
        res.mtrx[k][0] = this.mtrx[k][i] * this.mtrx[k][j];
    }
    return res;
}
```

Fungsi KaliKolom (int i, int j) melakukan iterasi selama jumlah baris matriks tersebut dan dilakukan perkalian antara 2 buah kolom dan mengembalikan sebuah matriks dengan ukuran this.baris dan 1 buah kolom sebagai hasilnya.

```
public void TambahBaris ( int i, int j){
    // Kita validasi Barisnya dulu
    // Baris tidak valid bila berada dibawah 0 dan diatas baris
    if ((i < 0 || i > baris) && (j < 0 || j > baris)){
        System.out.printf("Baris tidak valid");
    }
    else{
        for (int k = 0; k < kolom; k ++ ){
            this.mtrx[i][k] += this.mtrx[j][k];
        }
    }
}
```

Prosedur TambahBaris (int i, int j) melakukan penjumlahan pada setiap kolom pada baris ke-(i+1) dan (j+1) yang diawali dengan melakukan pengecekan pada nilai i dan j.

```
public Matriks TambahMatriks(Matriks M1, Matriks M2){
    Matriks res = new Matriks(M1.baris, M1.kolom,M1.determinan);
    for (int i = 0; i < res.baris; i++){
        for (int j = 0; j < res.kolom; j++){
            res.mtrx[i][j] = 0;
            res.mtrx[i][j] += M1.mtrx[i][j] + M2.mtrx[i][j];

        }
    }
    res.determinan = getDeterminan();
    return res;
}
```

Fungsi TambahMatriks (Matriks M1, Matriks M2) melakukan penjumlahan antara 2 buah matriks dan mengembalikan sebuah matriks hasil (res) yang memiliki nilai berupa penjumlahan elemen pada baris dan kolom yang bersesuaian dan memiliki determinan yang telah diupdate.

```
public void KurangBaris ( int i, int j){
    // Kita validasi Barisnya dulu
    // Baris tidak valid bila berada dibawah 0 dan diatas baris
    if (i < 0 || i > baris){
        System.out.printf("Baris tidak valid");
    }
    else{
        for (int k = 0; k < kolom; k ++ ){
            this.mtrx[i][k] -= this.mtrx[j][k];
        }
    }
}
```

Prosedur KurangBaris(int i, int j) melakukan pengurangan pada setiap kolom pada baris ke-(i+1) dan (j+1) yang diawali dengan melakukan pengecekan pada nilai i dan j.

```
public Matriks KurangMatriks(Matriks M1, Matriks M2){
    Matriks res = new Matriks(M1.baris, M1.kolom,M1.determinan);
    for (int i = 0; i < res.baris; i++){
        for (int j = 0; j < res.kolom; j++){
            res.mtrx[i][j] = 0;
            res.mtrx[i][j] += M1.mtrx[i][j] - M2.mtrx[i][j];
        }
    }
    res.determinan = getDeterminan();
    return res;
}
```

Fungsi KurangMatriks(Matriks M1, Matriks M2) melakukan pengurangan antara 2 buah matriks dan mengembalikan sebuah matriks hasil (res) yang memiliki nilai berupa pengurangan elemen pada baris dan kolom yang bersesuaian dan memiliki determinan yang telah diupdate.

```java
public double getDeterminan(){
    if(this.isMatrixSquare()){
        this.determinan = this.getDeterminanLokal(this.mtrx);
        return this.determinan;
    }else{
        System.out.println("matriks tidak memiliki determinan");
        return 0;
    }
}
```

Fungsi getDeterminan mengembalikan nilai determinan yang didapat dengan memanggil fungsi getDeterminanLokal pada matriks tersebut, yang diawali dengan pengecekan apakah matriks tersebut persegi dengan fungsi isMatrixSquare, yang apabila bernilai false akan mengeluarkan pesan tersebut.

```java
public Matriks getInverse(){

    if(this.isInversable()){
        Matriks result = new Matriks(this.baris,this.kolom,(1/determinan));
        result = this.getAdjoin();
        int l = this.baris;
        int a,b;
        for(a=0;a<l;a++){
           for(b=0;b<l;b++){
               result.mtrx[a][b] = result.mtrx[a][b]/this.determinan;
           }
       }

       return result;
    }else{
        //System.out.println("tidak bisa di Invers, karena matriks tidak persegi atau determinan==0 adalah benar");
        return new Matriks(0,0,0);
    }
}
```

Fungsi getInverse mengembalikan matriks yang telah di invers, proses diawali dengan memanggil fungsi isInversable() yang apabila bernilai true, maka kami bisa melanjutkan proses. Selanjutnya dibuat matriks baru dengan determinan baru bernilai 1/determian awal, dan setiap nilai matriks baru (result) diisi dengan adjoin matriks lama dengan fungsi getAdjoin(), kemudian dilakukan iterasi selama panjang baris yang membagi setiap elemen di matriks baru (result) dengan determinan matriks lama (cara invers matriks metode adjoin) dan mengembalikan matriks baru tersebut.

```java
public Matriks getAdjoin(){
   Matriks res = new Matriks(this.baris,this.kolom,this.determinan);
    if(this.isMatrixSquare()){
        res.mtrx = this.getTranspose(getKofaktor().mtrx);
        return res;
    }else{
        //System.out.println("tidak punya adjoin karena tidak ada minor dan kofaktor");
        return null;
    }
}
```

Fungsi getAdjoin() diawali dengan membuat matriks baru dengan ukuran baris,kolom, nilai determinan sama persis dengan matriks lama, kemudian dilakukan pengecekan apabila matriks

tersebut persegi dengan fungsi isMatrixSquare() yang apabila bernilai true, maka setiap elemen matriks baru tadi akan di replace dengan this.getTranspose(getKofaktor().mtrx) dan mengembalikan matriks baru tadi. Namun apabila matriks lama tersebut bukanlah matriks persegi, maka mereturn null karena tidak bisa diproses.

```
public Matriks getMinor(){
    if(this.isMatrixSquare()){
        Matriks temp = new Matriks(this.baris,this.kolom,this.determinan);
        int brs = temp.baris;
        int kol = temp.kolom;

        for(int a=0;a<brs;a++){
            for(int b=0;b<kol;b++){
                //debugarr(this.getMatriksNonSejajar(a, b));
                temp.mtrx[a][b] = this.getDeterminanLokal(this.getMatriksNonSejajar(a, b));
            }
        }
    return temp;
    }else{
        return null;
    }
}
```

Fungsi getMinor() diawali dengan pengecekan apakah matriks tersebut persegi, kemudian dibuat matriks baru dengan ukuran baris,kolom, nilai determinan sama persis dengan matriks lama, kemudian dilakukan iterasi selama jumlah baris dan kolom matriks baru yaitu assign nilai matriks baru dengan mencari matriks non sejajar dulu, kemudian mengambil determinan lokalnya (cara mencari minor).

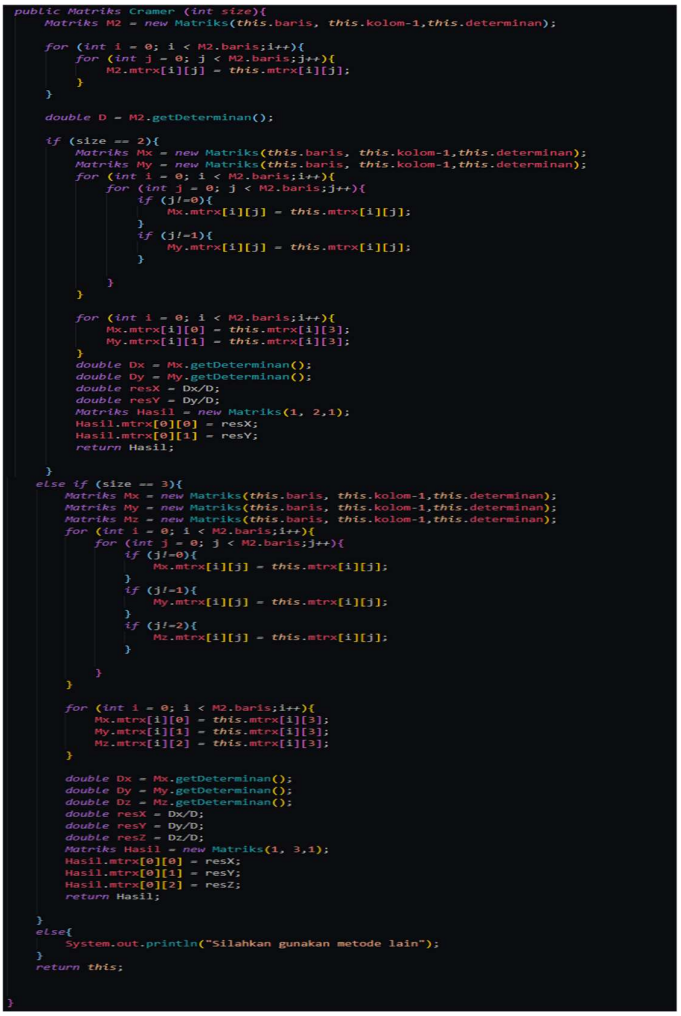
```
public Matriks getKofaktor(){
    Matriks temp=null;
    temp = this.getMinor();

    if(temp!=null && this.isMatrixSquare()){
        int brs = temp.baris;
        int klm = temp.kolom;
        int a,b;

        for(a=0;a<brs;a++){
            for(b=0;b<klm;b++){
                if((a+b)%2==1 && temp.mtrx[a][b]!=0){
                    temp.mtrx[a][b]= temp.mtrx[a][b]*(-1);
                }
            }
        }

    return temp;
}
```

Fungsi getKofaktor() diawali dengan membuat sebuah matriks baru yang bernilai null dan diisi dengan minor dari matriks lama, kemudian apabila nilai matriks baru tidak null dan matriks tersebut persegi, maka dilakukan iterasi selama ukuran baris dan kolom matriks baru, dan

apabila indeks a+b bernilai ganjil dan elemen matriks tidak bernilai 0, maka nilai elemen matriks tersebut akan dikali dengan -1. Diakhiri dengan mengembalikan matriks baru.

```
public Matriks getTranspose(){
    Matriks res = new Matriks(this.kolom,this.baris,this.determinan);
    int a,b;
    for(a=0;a<res.baris;a++){
        for(b=0;b<res.kolom;b++){
            res.mtrx[a][b]= this.mtrx[b][a];
        }
    }

    return res;
}
```

Fungsi getTranspose() diawali dengan membuat matriks baru yang memiliki ukuran kolom, baris, nilai determinan sama dengan matriks lama, dan dilakukan iterasi selama panjang baris dan kolom matriks baru dan mengisikan nilai matriks baru yang mana indeks [a][b] dibalik menjadi [b][a] karena definisi transpose, dan mengembalikan matriks baru yang merupakan transpose dari matriks lama.



```
public Matriks Cramer (int size){
    Matriks M2 = new Matriks(this.baris, this.kolom-1,this.determinan);

    for (int i = 0; i < M2.baris;i++){
        for (int j = 0; j < M2.baris;j++){
            M2.mtrx[i][j] = this.mtrx[i][j];
        }
    }

    double D = M2.getDeterminan();

    if (size == 2){
        Matriks Mx = new Matriks(this.baris, this.kolom-1,this.determinan);
        Matriks My = new Matriks(this.baris, this.kolom-1,this.determinan);
        for (int i = 0; i < M2.baris;i++){
            for (int j = 0; j < M2.baris;j++){
                if (j!=0){
                    Mx.mtrx[i][j] = this.mtrx[i][j];
                }
                if (j!=1){
                    My.mtrx[i][j] = this.mtrx[i][j];
                }

            }
        }

        for (int i = 0; i < M2.baris;i++){
            Mx.mtrx[i][0] = this.mtrx[i][3];
            My.mtrx[i][1] = this.mtrx[i][3];
        }
        double Dx = Mx.getDeterminan();
        double Dy = My.getDeterminan();
        double resX = Dx/D;
        double resY = Dy/D;
        Matriks Hasil = new Matriks(1, 2,1);
        Hasil.mtrx[0][0] = resX;
        Hasil.mtrx[0][1] = resY;
        return Hasil;

    }
    else if (size == 3){
        Matriks Mx = new Matriks(this.baris, this.kolom-1,this.determinan);
        Matriks My = new Matriks(this.baris, this.kolom-1,this.determinan);
        Matriks Mz = new Matriks(this.baris, this.kolom-1,this.determinan);
        for (int i = 0; i < M2.baris;i++){
            for (int j = 0; j < M2.baris;j++){
                if (j!=0){
                    Mx.mtrx[i][j] = this.mtrx[i][j];
                }
                if (j!=1){
                    My.mtrx[i][j] = this.mtrx[i][j];
                }
                if (j!=2){
                    Mz.mtrx[i][j] = this.mtrx[i][j];
                }

            }
        }

        for (int i = 0; i < M2.baris;i++){
            Mx.mtrx[i][0] = this.mtrx[i][3];
            My.mtrx[i][1] = this.mtrx[i][3];
            Mz.mtrx[i][2] = this.mtrx[i][3];
        }

        double Dx = Mx.getDeterminan();
        double Dy = My.getDeterminan();
        double Dz = Mz.getDeterminan();
        double resX = Dx/D;
        double resY = Dy/D;
        double resZ = Dz/D;
        Matriks Hasil = new Matriks(1, 3,1);
        Hasil.mtrx[0][0] = resX;
        Hasil.mtrx[0][1] = resY;
        Hasil.mtrx[0][2] = resZ;
        return Hasil;

    }
    else{
        System.out.println("Silahkan gunakan metode lain");
    }
    return this;

}
```

Fungsi Kaidah Cramer digunakan untuk matriks 2x2 dan 3x3, karena untuk ukuran diatasnya lebih baik menggunakan metode gauss maupun gauss jordan. Kaidah Cramer digunakan untuk mencari nilai atau akar2 persamaan yang berbentuk matriks dengan membagi determinan tiap2 akar seperti Dx,Dy,Dz dengan determinan utama matriks D. Proses diawali dengan membuat matriks baru M2, yang berukuran baris sama dengan matriks lama, namun kolomnya dikurangi 1, dan determinan sama dengan matriks lama (tidak perlu dipikirkan). Kemudian iterasi untuk assign semua nilai matriks lama dengan batas tersebut ke dalam matriks baru, ini dilakukan untuk proses selanjutnya yaitu mencari D (Determinan utama matriks). Kemudian dari parameter fungsi (int size) ditentukan apakah matriks kami ini 2x2 atau 3x3, apabila int size bernilai 2, maka matriks diproses dengan cara 2x2, dan 3 untuk 3x3. Untuk ukuran / size 2 (2x2),

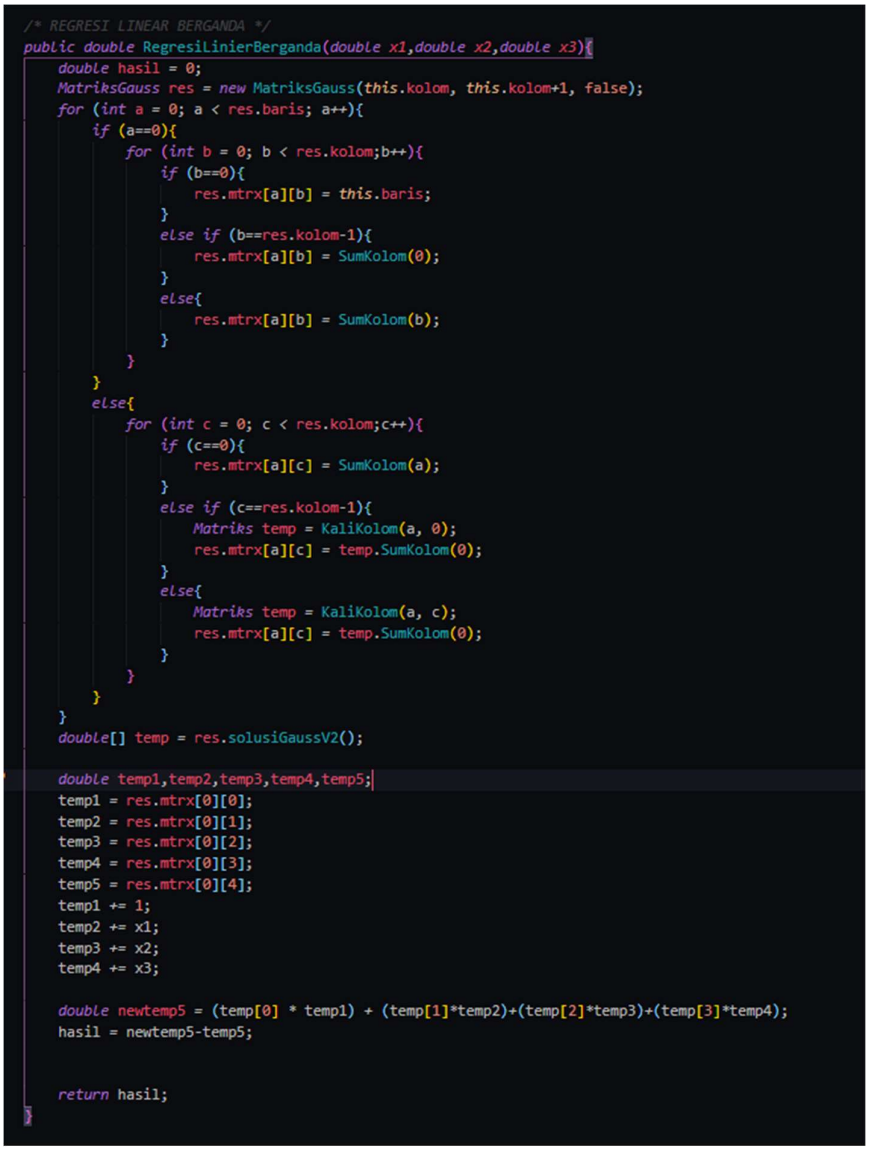
kami membuat matriks baru yang mana memiliki ukuran baris,kolom,determinan (tidak perlu dipikirkan) sama persis dengan M2. Kemudian dilakukan iterasi selama panjang dan kolom baris M2 (karena matriks persegi jadi anggap saja sama), apabila nilai indeks j (kolom) sama dengan 0, maka untuk Mx kami skip karena akan digunakan dalam rangkaian proses selanjutnya, dan apabila nilai indeks j (kolom) sama dengan 1, maka untuk My kami skip. Langkah selanjutnya yaitu kami mengisikan setiap nilai pada kolom yang kami skip tadi, yaitu apabila Mx kami isikan pada indeks [i][0] dengan setiap nilai pada ujung kanan matriks lama (indeks kolom ke 3/ paling ujung kanan), demikian pula pada My kami isikan pada indeks [i][1] dengan cara yang sama dengan Mx. Kemudian kami cari Dx, dan Dy kemudian didapat solusi (resX dan resY) berupa hasil bagi antara Dx/D dan Dy/D dan hasil ini kami masukkan pada sebuah matriks baru berukuran (1x2) [resx,resY] dan mengembalikan matriks baru ini sebagai hasil Kaidah Cramer kami. Untuk matriks dengan size (3) atau 3x3 sama dengan cara matriks size (2) atau 2x2 tapi ditambah Dz karena 3 variabel, dan dilakukan skip pada indeks 2 untuk Dz, serta mengembalikan matriks baru berukuran (1x3) [resX,resY,resZ].

```
/* INTERPOLASI */
public double Interpolasi(double num){
    double hasil = 0;
    MatriksGauss res = new MatriksGauss(this.baris, this.baris+1, false);
    for (int i = 0; i < res.baris;i++){
        for (int j= 0 ; j < res.kolom; j++){
            if (j==0){
                res.mtrx[i][j] = 1;
            }
            else if (j == this.baris){
                res.mtrx[i][j] = this.mtrx[i][1];
            }
            else{
                res.mtrx[i][j] = (double) Math.pow(this.mtrx[i][0],j);
            }


        }
    }
    double[] temp = res.solusiGaussV2();
    for (int k = 0; k < res.baris;k++){
        hasil += (temp[k] * (Math.pow(num, k)));
    }

    return hasil;
}
```

Fungsi Interpolasi dengan parameter (double num) merupakan sebuah fungsi yang digunakan untuk mengembalikan estimasi nilai f(num) apabila dimasukkan kedalam sebuah fungsi f yang mana kami tidak mengetahui fungsi f tersebut namun kami memiliki beberapa data sebelumnya sehingga kami bisa mengestimasi nilai f(num) tersebut. Diawali dengan membuat sebuah matriks baru dengan ukuran baris sama dengan matriks lama, namun kali ini ukuran kolom berubah menjadi ukuran baris lama + 1, dan false karena kami tidak mengisi nilai matriks secara manual. Iterasi dilakukan sepanjang ukuran baris dan kolom matriks baru, dan terdapat 3 pecabangan:

1. Apabila (j==0) maka bernilai 1 karena x^0.
2. Apabila (j == this.baris) maka kami pindahkan data pada matriks pertama yang berupa (8.0, 2.0794), (9.0, 2.1972), dan (9.5, 2.2513) yang di stabilo kuning dipindahkan ke dalam matriks baru pada ujung kanan matriks.
3. Apabila tidak memenuhi (j == 0 dan j == this.baris) maka, elemen matriks lama yang berupa (8.0, 2.0794), (9.0, 2.1972), dan (9.5, 2.2513) yang di stabilo oren dijadikan nilai matriks baru yang dipangkatkan j, jadi seperti [8^0,8^1,8^2...].

Kemudian dibuat array penampung hasil berupa fungsi solusiGaussV2, dan setiap elemen pada array baru tersebut digunakan untuk penyelesaian metode interpolasi ini. Dilakukan iterasi sepanjang ukuran barus matriks baru, yang mana nilai hasil akan ditambahkan dengan elemen array penampung ke-k yang dikalikan dengan hasil pangkat dari num^k. Hasil akhirnya dikembalikan nilai hasil sebagai hasil estimasi metode interpolasi tersebut untuk nilai double num.



```
/* REGRESI LINEAR BERGANDA */
public double RegresiLinierBerganda(double x1,double x2,double x3){
    double hasil = 0;
    MatriksGauss res = new MatriksGauss(this.kolom, this.kolom+1, false);
    for (int a = 0; a < res.baris; a++){
        if (a==0){
            for (int b = 0; b < res.kolom;b++){
                if (b==0){
                    res.mtrx[a][b] = this.baris;
                }
                else if (b==res.kolom-1){
                    res.mtrx[a][b] = SumKolom(0);
                }
                else{
                    res.mtrx[a][b] = SumKolom(b);
                }
            }
        }
        else{
            for (int c = 0; c < res.kolom;c++){
                if (c==0){
                    res.mtrx[a][c] = SumKolom(a);
                }
                else if (c==res.kolom-1){
                    Matriks temp = KaliKolom(a, 0);
                    res.mtrx[a][c] = temp.SumKolom(0);
                }
                else{
                    Matriks temp = KaliKolom(a, c);
                    res.mtrx[a][c] = temp.SumKolom(0);
                }
            }
        }
    }
    double[] temp = res.solusiGaussV2();

    double temp1,temp2,temp3,temp4,temp5;
    temp1 = res.mtrx[0][0];
    temp2 = res.mtrx[0][1];
    temp3 = res.mtrx[0][2];
    temp4 = res.mtrx[0][3];
    temp5 = res.mtrx[0][4];
    temp1 += 1;
    temp2 += x1;
    temp3 += x2;
    temp4 += x3;

    double newtemp5 = (temp[0] * temp1) + (temp[1]*temp2)+(temp[2]*temp3)+(temp[3]*temp4);
    hasil = newtemp5-temp5;


    return hasil;
}
```

$$20b_0 + 863.1b_1 + 1530.4b_2 + 587.84b_3 = 19.42$$

$$863.1b_0 + 54876.89b_1 + 67000.09b_2 + 25283.395b_3 = 779.477$$

$$1530.4b_0 + 67000.09b_1 + 117912.32b_2 + 44976.867b_3 = 1483.437$$

$$587.84b_0 + 25283.395b_1 + 44976.867b_2 + 17278.5086b_3 = 571.1219$$

Fungsi RegresiLinierBerganda merupakan sebuah metode untuk menentukan estimasi nilai suatu variabel dengan data2 yang telah diinputkan sebelumnya, fungsi ini menggunakan 3 buah parameter yaitu double x1,x2,x3 (menyesuaikan studi kasus) yang mana akan dicari nilai y (sesuai dengan data yang diberikan). Proses diawali dengan membuat variabel hasil untuk menyimpan hasilnya dengan nilai 0, kemudian membuat sebuah matriks baru yang mana ukuran barisnya sama dengan ukuran kolom matriks lama, dan ukuran kolomnya sama dengan ukuran kolom matriks lama + 1 serta false karena kami tidak mengisikan setiap elemen matriks baru ini secara manual. Kemudian kami lakukan iterasi a sepanjang baris matriks baru, apabila nilai a:

1. Jika a == 0 (baris paling atas) maka kami lakukan iterasi b sepanjang ukuran kolom matriks baru :
    a. Untuk nilai b == 0 (pojok kiri atas) kami assign dengan jumlah data / ukuran baris matriks lama.
    b. Untuk b == kolom matriks baru - 1 (pojok kanan atas), kami masukkan dengan jumlah semua baris pada kolom indeks 0 pada matriks lama, dengan fungsi SumKolom(0).
    c. Selain itu, kami assign nilai matriks baru [a][b] dengan jumlah semua baris pada kolom indeks b pada matriks lama, dengan fungsi SumKolom(b).
2. Jika a != 0 (baris-baris di bawahnya), kami lakukan iterasi c sepanjang ukuran kolom matriks baru:
    a. Untuk nilai c == 0 (pojok kiri setiap baris) kami assign dengan jumlah semua baris pada kolom ke a dengan fungsi SumKolom(a).
    b. Untuk c == kolom matriks baru - 1 (pojok kanan setiap baris), kami masukkan dengan hasil perkalian kolom indeks a dengan kolom indeks 0 (misalnya a=1 maka perkalian kolom x1(1) dengan kolom y(0)) dengan fungsi KaliKolom(a,0), dan kami assign nilai elemen indeks tersebut dengan SumKolom matriks hasil KaliKolom(a,0) tersebut.
    c. Selain itu, kami assign nilai matriks baru dengan hasil Penjumlahan setiap baris hasil perkalian setiap baris pada kolom a dan c, atau SumKolom hasil KaliKolom (a,c) tersebut.

Kami buat matriks penampung hasil (temp) dengan menggunakan fungsi solusiGaussV2(), dan membuat 5 buah nilai yaitu temp1,2,3,4,5.

- temp1 kami assign dengan indeks matriks lama [0][0]
- temp2 kami assign dengan indeks matriks lama [0][1]
- temp3 kami assign dengan indeks matriks lama [0][2]
- temp4 kami assign dengan indeks matriks lama [0][3]

- temp5 kami assign dengan indeks matriks lama [0][4]

Kemudian, kami tambahkan temp1 (yang memuat jumlah data) dengan 1 karena kami menambahkan 1 data lagi yaitu data yang akan kami estimasi nilainya.

Variabel temp2 ditambahkan dengan parameter x1, temp3 dengan x2, temp4 dengan x3. Kami buat sebuah variabel double baru yaitu newtemp5 yaitu hasil penjumlahan perkalian setiap elemen pada array hasil penampung dengan temp1-temp4, setelah itu kami assign nilai hasil dengan selisih nilai antara newtemp5 dengan temp5. Proses berakhir dengan mengembalikan nilai hasil sebagai hasil estimasi dengan metode regresi linier berganda.

Untuk class GaussMatriks, kelas tersebut merupakan sebuah *subclass* dari class Matriks sehingga memiliki atribut yang sama dengan class Matriks. Class ini berisi fungsi-fungsi yang berkaitan dalam penyelesaian Sistem Persamaan Linear menggunakan metode eliminasi Gauss dan Gauss-Jordan. Fungsi-fungsi yang ada di kelas ini antara lain:

1. **Fungsi getGauss**
   Pada fungsi getGauss, dilakukan pengubahan matriks menjadi sebuah matriks Gauss melalui metode eliminasi Gauss. Pada awalnya, kami membuat loop dari baris pertama sampai terakhir untuk mencari nilai dan index yang akan ditukar memakai fungsi swap. Setelah ditukar barisnya, dilakukan operasi baris elementer menggunakan *loop* untuk menghitung variabel ratio dari dua baris dan menggunakan *nested loop* untuk melakukan pengurangan nilai elemen dalam satu baris. Apabila *loop* awal sudah berhenti dilakukan pembagian setiap baris dengan koefisien utamanya/*lead coeffiecient.*

2. **Fungsi getGaussJordan**
   Pada fungsi getGaussJordan, algoritma yang digunakan hampir sama dengan fungsi getGauss. Pada awal dilakukan operasi swap agar baris terurut lalu dilakukan loop untuk melakukan operasi baris elementer sehingga menghasilkan sebuah matriks eselon baris tereduksi. Pada akhir loop dilakukan operasi bagiLeadCoef agar bilangan pertama setiap barisan adalah 1 utama.

3. **Fungsi getType**
   Pada fungsi getType, diasumsikan setiap matriks yang melalui fungsi ini adalah matriks yang telah melewat fungsi getGauss atau getGaussJordan. Nilai *return* fungsi ini adalah *integer* yang menandakan jenis solusi yang dimiliki oleh suatu matriks Gauss atau Gauss-Jordan. Tipe 1 untuk solusi unik, tipe 2 untuk banyak solusi, dan tipe 3 untuk tidak ada solusi. Cara pencarian adalah menentukan tipe setiap baris terlebih dahulu. Dibuatkan array untuk menyimpan angka tipe setiap baris. Apabila suatu baris merupakan baris kosong maka disimpen ke array angka 0. Jika nilai index *lead coeffiecient*-nya di kolom terakhir, disimpan ke array angka 3. Angka 2 disimpan untuk elemen baris setelah bilangan utama tidak 0 dan angka 1 disimpan untuk baris yang mengandung nilai bilangan utama dan bilangan kolom terakhirnya. Array lalu dibaca memakai loop, jika ada angka 3 otomatis tipe matriks tidak ada solusi, jika ada angka 2 tanpa ada angka 1 matriks banyak solusi, jika ada angka 1 ada solusi unik.

4. **Fungsi solusiGauss**
   Fungsi ini untuk mengeluarkan solusi untuk matriks hasil eliminasi Gauss. Pertama dilakukan penentuan tipe matriks. Jika matriks memiliki solusi unik, dilakukan *back substitution* dan mencetak hasil ke layar. Jika matriks memiliki banyak solusi, dicetak nilai setiap variabel dalam bentuk parametrik. Jika matriks tidak memiliki solusi, dicetak "SPL tidak memiliki solusi".

5. **Fungsi solusiGaussJordan**
   Fungsi ini sama seperti fungsi solusiGauss namun tidak ada algoritma *back substitution* karena tidak diperlukan.

6. **Fungsi getDetRowReduced**
   Fungsi ini digunakan untuk mencari determinan suatu matriks menggunakan metode reduksi baris. Algoritma yang dipakai adalah algoritma eliminasi Gauss seperti pada fungsi getGaussMatriks. Terdapat perbedaan yaitu penghitungan jumlah kejadian swap untuk menghitung perkalian elemen diagonal matriks dengan $(-1)^n$ dengan n adalah jumlah swap.

7. **Fungsi copyMatriks**
   Fungsi ini berfungsi untuk meng-*copy* sebuah matriks input dan me-*return* sebuah matriks duplikat.

8. **Fungsi swap**
   Fungsi ini menerima dua parameter yaitu i dan j yang merupakan indeks sebuah matriks baris. Elemen di kedua baris akan ditukar satu sama lain.

9. **Fungsi getLeadCoef**
   Fungsi getLeadCoef menerima parameter suatu nilai baris dan mencari bilangan utama di baris tersebut.

10. **Fungsi idxLeadCoef**
    Fungsi idxLeadCoef memiliki algoritma sama seperti fungsi getLeadCoef namun *return* berbentuk indeks bilangan utama.

11. **Fungsi bagiLeadCoef**

    Fungsi bagiLeadCoef akan membagi setiap elemen baris pada matriks dengan bilangan utama setiap baris untuk menghasilkan matriks yang memiliki 1 utama di setiap baris.

BAB IV

Eksperimen

A. Hasil Eksekusi Program

1. Studi Kasus SPL

```
1 1 -1 -1 1
2 5 -7 -5 -2
2 -1 1 3 4
5 2 -4 2 6
Bentuk Gauss:
1 0 0 0.67 1.67
0 3 0 -8 1
0 0 -2 2 -2
0 0 0 0 1
SPL tidak memiliki solusi.
```

Nomor 1a

```
1 -1 0 0 1 3
1 1 0 -3 0 6
2 -1 0 1 -1 5
-1 2 0 -2 -1 -1
Bentuk Gauss:
1 0 0 -1.5 0.5 4.5
0 2 0 -3 -1 3
0 0 0 2.5 -2.5 -2.5
0 0 0 -0.5 0.5 0.5
Solusi X3 = 1.00X4 - 1.00
Solusi X3 = 1.00X4 - 1.00
Solusi X1 = 1.50X3 + 0.50X4 + 1.50
Solusi X0 = 1.50X3 - 0.50X4 + 4.50
```

Nomor 1b

```
0 1 0 0 1 0 2
0 0 0 1 1 0 -1
0 1 0 0 0 1 1
Bentuk Gauss:
0 1 0 0 1 0 2
0 0 0 1 1 0 -1
0 1 0 0 0 1 1
Solusi X1 = - 1.00X5 + 1.00
Solusi X3 = - 1.00X4 - 1.00
Solusi X1 = - 1.00X4 + 2.00
```

Nomor 1c

```
1       -1      2       -1      -1
2       1       -2      -2      -2
-1      2       -4      1       1
3       0       0       -3      -3
========= Bentuk gauss =========
1 -1 2 -1 -1
0 3 -6 0 0
0 0 0 0 0
0 0 0 0 0
Solusi X1 = 2.00X2 + 0.00
Solusi X0 = 1.00X1 - 2.00X2 + 1.00X3 - 1.00
PS E:\HMIF\Algeo\TugasBesar\src>
```

Nomor 2a

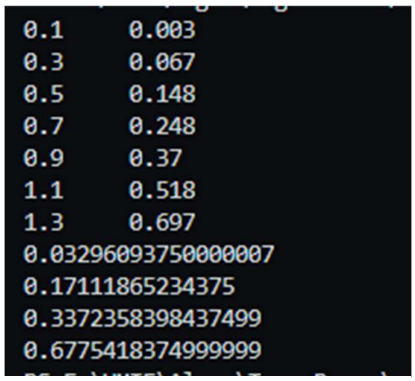
```
2       0       8       0       8
0       1       0       4       6
-4      0       6       0       6
0       -2      0       -3      -1
2       0       -4      0       -4
0       1       0       -2      0
========= Bentuk gauss =========
2 0 8 0 8
0 1 0 4 6
0 0 22 0 22
0 0 0 5 11
0 0 0 0 0
0 0 0 0 7.2
SPL tidak memiliki solusi.
```

Nomor 2b

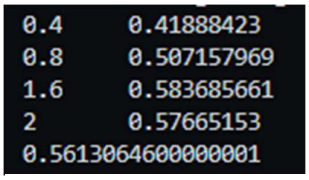
```
8       1       3       2       0
2       9       -1      -2      1
1       3       2       -4      2
1       0       6       4       3
========= Bentuk gauss =========
1 0.12 0.38 0.25 0
0 1 -0.2 -0.29 0.11
0 0 1 -1.56 0.76
0 0 0 1 -0.1
Solusi X0 = -0.23
Solusi X1 = 0.21
Solusi X2 = 0.60
Solusi X3 = -0.10
```

Nomor 3a

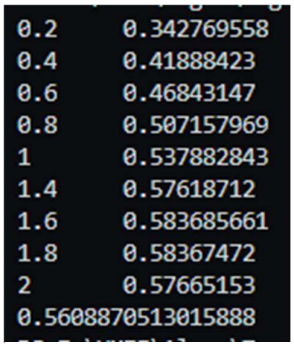
2. Studi Kasus Interpolasi



Nomor 5

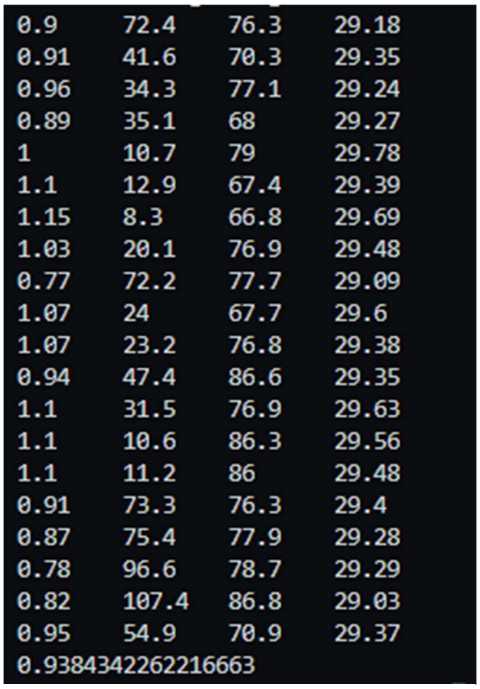


Nomor 7 dengan n = 5



Nomor 7 dengan n = 10

3. Studi Kasus Regresi Linear Berganda



Nomor 8

B. Analisis

Pada studi kasus SPL, terlihat bahwa program dapat mengidentifikasi SPL yang memiliki solusi unik, solusi banyak, dan tidak ada solusi. Pada nomor 1a dan 2b, SPL tidak memiliki solusi. Pada nomor 1b, 1c, dan 2a, SPL memiliki banyak solusi sehingga output dalam bentuk persamaan parametrik. Pada nomor 3a, SPL memiliki 1 solusi unik.

Pada studi kasus interpolasi, program sudah dapat melakukan kalkuklasi interpolasi dengan baik. Pada nomor 5, program melakukan pengujian pada 4 nilai *default* yang berbeda. Pada nomor 7, dilakukan penyederhanaan fungsi dengan n = 5 dan n = 10. Terlihat bahwa hasil dari kedua kalkulasi hampir sama. Pada studi kasus regresi linier berganda, telah dihitung regresi linear dari data di dalam spesifikasi tugas dan dikeluarkan output sebesar 0.9384.

BAB V

Kesimpulan dan Saran

A. Kesimpulan

Dari tugas besar ini, dapat disimpulkan bahwa dalam melakukan operasi Sistem Persamaan Linear dapat digunakan metode eliminasi Gauss atau metode eliminasi Gauss-Jordan untuk memecahkan SPL. Selain metode Gauss dan Gauss-Jordan, terdapat metode Cramer yang juga dapat menyelesaikan SPL melalui perhitungan determinan matriks. Dari metode-metode ini, dapat melakukan perhitungan lain seperti Interpolasi Polinom dan Regresi Linear Berganda. Hasil perhitungan-perhitungan dapat digunakan untuk mengolah dan memprediksi suatu data.

B. Saran

Dalam penentuan banyaknya solusi suatu SPL dapat digunakan teorema Rouché–Capelli yang menggunakan perhitungan *rank* matriks teraugmentasi dan *rank* matriks koefisien. Juga dapat dilakukan pengurangan penggunaan *loop* dalam fungsi untuk mengurangi waktu terjalannnya suatu perhitungan. Ada beberapa fungsi yang memiliki fungsi yang hampir sama sehingga ada kemungkinan untuk dilakukan penggabungan beberapa fungsi agar jumlah fungsi tidak terlalu banyak.

REFERENSI

1. Anton, Howard, dan Chris Rorres. *Elementary Linear Algebra: Applications Version*. John Wiley & Sons, 2013.
2. Meyer, Carl D. *Matrix Analysis and Applied Linear Algebra*. Vol. 71. Siam, 2000.
3. Demmel, James W. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.
4. Noble, Ben, dan James W. Daniel. *Applied Linear Algebra*. Vol. 3. Englewood Cliffs, NJ: Prentice-Hall, 1977.
5. Aitken, Alexander Craig. *Determinants and Matrices*. Read Books Ltd, 2017.
6. Varyani, Yash. *Gaussian Elimination to Solve Linear Equations*, GeeksforGekks, diakses pada 25 September 2020. <https://www.geeksforgeeks.org/gaussian-elimination>
7. Stapel, Elizabeth. *Systems of Linear Equations Solving by Gaussian Elimination*, PurpleMath, diakses pada 25 September 2020. <https://www.purplemath.com/modules/systlin6.htm>
8. Hidayat, Anwar. *Penjelasan dan Tutorial Regresi Linear Berganda*, Statistikian, diakses pada 29 September 2020. <https://www.statistikian.com/2018/01/penjelasan-tutorial-regresi-linear-berganda.html>
9. Rosidi, Mohammad. *Interpolasi dan Ekstrapolasi*, Bookdown, diakses pada 29 September 2020. <https://bookdown.org/moh_rosidi2610/Metode_Numerik/interpolation.html>