PENGGUNAAN PYTHON DALAM IMPLEMENTASI MATERI SAINS KOMPUTASI

Dosen Pengampu :

Dr. Yuni Yamasari, S.Kom., M.Kom

Disusun oleh :

Kelompok 9

Sandhika Lyandra Prasetyo	23051204074
Rayhan Ramadhani Hendra Atmadja	23051204075
Adriano Emmanuel	23051204082
Cornelius Louis Nathan	23051204085

PROGRAM STUDI (S1) TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI SURABAYA

2024

KATA PENGANTAR

Puji syukur kami panjatkan kepada Tuhan Yang Maha Esa. Karena atas limpahan rahmat-Nya kami dapat menyelesaikan makalah ini yang berjudul "*PENGGUNAAN PYTHON DALAM IMPLEMENTASI MATERI SAINS KOMPUTASI*" dengan tepat waktu tanpa ada halangan yang dan sesuai dengan harapan.

Adapun tujuan dari penulisan laporan ini adalah untuk memahami dan memperdalam bagaimana pengimplementasian sains komputasi dalam banyak bidang dengan menggunakkan bahasa pemrograman *python* menggunakkan *library* Numpy, Matplotlib. Pembuatan laporan ini juga menjadi tugas yang mengisi nilai Ujian Akhir Semester dari Mata Kuliah Sains Komputasi.

Penyusunan makalah ini tidak lepas dari bantuan dan dukungan. Sehingga, kami menyampaikan terima kasih kepada pihak yang sudah mendukung pembuatan makalah ini. Terutama kepada Ibu Yuni Yamasari, S.Kom, M.Kom., selaku dosen pengampu mata kuliah Sains Komputasi, yang telah membantu memberikan arahan dan pemahaman dalam penyusunan makalah ini.

Kami menyadari bahwa dalam penyusunan makalah ini masih banyak kekurangan karena keterbatasan kami. Maka dari itu kami mengharapkan kritik dan saran dari pembaca untuk menyempurnakan laporan ini. Semoga apa yang ditulis dapat bermanfaat bagi semua pihak yang membutuhkan.

Surabaya, 1 Desember 2024

Tertanda,

Kelompok 9

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Latar Belakang

Sains komputasi adalah bidang keilmuan yang menggabungkan sains, teknik, dan matematika sebagai pengetahuan, keterampilan, dan landasan ilmiah (Nasution, dkk. 2021 : 1146). Sains komputasi memungkinkan para peneliti untuk memodelkan, menganalisa, dan mensimulasikan fenomena kompleks, sehingga tidak perlu mengeluarkan biaya yang mahal maupun resiko yang besar.

Sains Komputasi dapat diimplementasikan dalam banyak bahasa pemrograman seperti MATLAB, R, C++, dan Python. Dimana setiap bahasa pemrograman memiliki keunggulan dan karakteristiknya masing-masing. Sebagai contoh, python banyak dipilih karena python merupakan bahasa pemrograman yang fleksibel dan mudah digunakan. Banyaknya *library* yang tersedia dalam python seperti Numpy, Scipy, Matplotlib memudahkan pengguna untuk mempelajari dan mengimplementasikan sains komputasi.

Banyak sekali bidang-bidang sains komputasi yang dapat diimplementasikan menggunakkan bahasa pemrograman Python, seperti sistem persamaan linear, matriks, metode numerik, simulasi, dan iterasi. Pada makalah ini, materi-materi tersebut terbagi menjadi dua belas bagian, yaitu :

1. Sistem persamaan linear dengan cara triangularisasi dan substitusi mundur,
2. Sistem persamaan linear dengan cara eliminasi gauss,
3. Praktik matriks dan operasinya,
4. Invers matriks,
5. *LU Decomposition*,
6. Metode numerik persamaan linear
7. Metode numerik persamaan non-linear,
8. Dasar-dasar simulasi,
9. Metode Iterasi Jacobi,
10. Metode Iterasi *Gauss-Seidel*,
11. Simulasi Monte-Carlo,
12. Rantai Markov.

Kami menyadari bahwasanya beberapa materi tersebut masih dapat disatukan karena ada yang berada dalam satu konteks yang sama, tetapi disini kami membagi materi tersebut

berdasarkan pada jadwal mengajar yang telah diadakan oleh Ibu Yuni Yamasari sekaligus juga pada bagian materi yang diajar oleh Ibu Yuni Yamasari. Sebagai contoh Sistem persamaan linear dengan cara triangularisasi dan substitusi mundur diajarkan pada minggu ke tiga perkuliahan, praktik matriks dan operasinya diajarkan pada minggu ke empat.

Makalah ini bertujuan untuk membedah, menjelaskan dan mengimplementasikan setiap materi tersebut. Sehingga diharapkan pembaca dapat memahami bagaimana python dapat digunakan untuk memecahkan masalah komputasi dan memperjelas setiap materi yang ada pada sains komputasi.

1.2 Rumusan Masalah

1. Apa saja materi sains komputasi yang dapat diimplementasikan menggunakkan python?
2. Bagaimana pengimplementasian setiap materi sains komputasi menggunakkan python?

1.3 Tujuan

1. Mengetahui apa saja materi sains komputasi yang dapat diimplementasikan menggunakkan python.
2. Mengetahui bagaiman pengimplementasian setiap materi sains komputasi menggunakkan python.

BAB II

LANDASAN TEORI

2.1 Sains Komputasi

Sains Komputasi adalah bidang keilmuan yang menggabungkan sains, teknik, dan matematika sebagai pengetahuan, keterampilan, dan landasan ilmiah. Sehingga, sejak awal sains komputasi melibatkan masalah yang berhubungan dengan *hardware*, *software*, dan *brainware*, dan data. (Nasution, dkk. 2021 : 1146). Sebagai contoh sederhana, pada pensimulasian peluncuran roket luar angkasa, *hardware* yang digunakan adalah *supercomputer* yang terhubung dengan server maupun *supercomputer* lain. *Software* yang digunakan bisa menggunakkan aplikasi *Computational Fluid Dynamics,* seperti OpenFOAM dan ANSYS, ataupun aplikasi pendukung lain. *Brainware* adalah orang yang menggunakkan *hardware*, dalam konteks ini adalah insinyur roket, ilmuwan, fisikawan dan matematikawan. Data yang digunakan mencakup parameter seperti massa roket, gaya dorong roket, banyak bahan bakar, suhu di lapisan atmosfer, estimasi jarak tempuh, gaya gravitasi bumi, dan masih banyak lagi.

Baron (2013 : 14), berpendapat bahwa sains komputasi berfokus pada hal-hal buatan seperti angka, grafik, fungsi, dan daftar. Kata "buatan" disini merujuk pada angka-angka yang digunakan dalam sains komputasi biasanya diambil dari kejadian atau objek di dunia nyata. Bisa dikatakan bahwa sains komputasi juga berperan sebagai penghubung antara teori matematika, eksperimen ilmiah, dan pemrosesan data menggunakkan komputer.

Jika ingin menjabarkan bagian dari sains komputasi, sistem persamaan linear, matriks, metode numerik, simulasi, dan iterasi adalah bagian kecil dari sains komputasi. Materi-materi yang sudah terbagi pada bagian latar belakang adalah pecahan dari bagian-bagian tersebut. Sebagai contoh, praktik matriks dan operasi matriks, invers matriks dan *LU Decomposition* adalah bagian dari implementasi matriks.

Bagian-bagian dari sains komputasi juga saling melengkapi satu dengan yang lain. Sistem persamaan linear dapat diselesaikan menggunakkan triangularisasi dan eliminasi gauss, yang tentunya menggunakkan matriks. Iterasi *Gauss-Seidel* menggunakkan sistem persamaan linear untuk menyatakan bentuk indeks. *Markov Chain* menggunakkan matriks probabilitas transisi sebagai tahap awal dalam perhitungan probabilitas untuk sistem stokastik.

2.2 Sistem Persamaan Linear

Sistem persamaan linear merupakan himpunan dari beberapa persamaan linear yang melibatkan sejumlah variabel. Solusi dari suatu sistem ini adalah-nilai-nilai variabel yang memenuhi semua persamaan secara simultan. Dalam realitas, penyelesaian sistem persamaan linear sangat sering dipakai dalam bidang apapun, seperti fisika, teknik, ekonomi, dan komputasi.

Sistem persamaan linear dapat diselesaikan dengan dua metode, yaitu metode langsung (Eliminasi gauss, teknik faktorisasi) dan metode iterasi (jacobi, *Gauss-Seidel, SOR* (*Successive Over Relaxation),* gradien konjugasi, dan multigrid). Metode langsung lebih banyak dipakai, tetapi jika permasalahannya terlalu besar, metode iterasi sangat disarankan karena waktu komputasi yang lebih baik (Derzie,dkk. 2022 : 1). Adapun penjelasan setiap metode yaitu sebagai berikut:

2.3 Metode Triangularisasi dan Substitusi Mundur

Metode triangularisasi dan substitusi mundur adalah metode langsung yang digunakan untuk menyelesaikan sistem persamaan linear. Dalam metode ini, matriks koefisien sistem persamaan diubah menjadi matriks segitiga atas sehingga substitusi mundur dapat langsung dilakukan. Berikut adalah langkah-langkah dari metode triangularisasi dan substitusi mundur: triangularisasi dan substitusi mundur :

Triangularisasi.

1. Ubah matriks persamaan kedalam bentuk matriks *augmented* yaitu gabungan matriks koefisien dan vektor konstanta
2. Salah satu tahap konversi yaitu menggunakan operasi baris elementer untuk mengubah bentuk matriks *augmented* menjadi bentuk segitiga atas. Misalkan pengurutan, penjumlahan atau pembagian baris.

Substitusi Mundur.

1. Setelah berada pada bentuk segitiga atas, gunakan metode substitusi mundur untuk mencari solusi variabel duluan dari sudut terakhir, kemudian sudut kedua dan seterusnya hingga sudut pertama.

2.4 Eliminasi Gauss

Metode eliminasi Gauss menggunakan pendekatan langsung juga untuk mencari solusi dari sistem persamaan linear. Metode ini memperlakukan sistem dengan proses triangularisasi yang dilakukan melalui operasi baris-elementer sehingga sistem persamaan linear sistem menjadi pada bentuk segitiga atas. Proses cara eliminasi Gauss sangat mirip dengan metode triangularisasi akan tetapi dilakukan lebih sistematis. Adapun langkah-langkah yang harus dilakukan dalam metode cara Gauss adalah sebagai berikut:

1. Sisipkan sistem persamaan linear pada bentuk matriks *augmented*.
2. Gunakan operasi baris-elementer untuk mengelemen di bawah pivot dalam setiap kolom, sehingga menghasilkan matriks segitiga atas.
3. Substitusi mundur dicari solusinya.

Kelebihan Metode Eliminasi Gauss:

1. Metode yang sistematis dan mudah dipahami.
2. Dapat diterapkan untuk sistem persamaan linear dengan ukuran berapa pun.

Kekurangan Metode Eliminasi Gauss:

1. Untuk sistem persamaan linear yang besar, perhitungannya bisa menjadi kompleks dan memakan waktu.
2. Rentan terhadap kesalahan pembulatan.

2.5 Matriks

Matriks adalah himpunan skalar bilangan *real* atau kompleks yang disusun dalam bentuk dua dimensi dengan panjang dan lebar. Perlu dicatat bahwa setiap baris matriks harus memiliki entri yang sama dengan baris lainnya, dan begitu juga dengan setiap kolom matrik (Nathaniel 2021 : 20). Baris, yang didefinisikan dengan m, dan kolom, yang didefinisikan dengan n, harus sejajar dan setiap titik pada matriks harus memiliki nilai. Berikut adalah contoh matriks yang benar:

$$\begin{bmatrix} 4 & 7 & 8 \\ 9 & 1 & 5 \\ 3 & 4 & 2 \end{bmatrix} \qquad \begin{bmatrix} 2 \\ 1 \\ 8 \end{bmatrix} \begin{bmatrix} 9 & 2 & 7 \end{bmatrix}$$

Matriks memiliki banyak macam tipe yang setiap tipenya memiliki karakteristiknya masing-masing. Berikut adalah macam-macam tipenya :

1. Matriks nol

   Matriks yang semua elemennya bernilai nol dan dinotasikan dengan A = 0.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

2. Matriks baris

   Matriks yang hanya memiliki satu baris

$$\begin{bmatrix} 4 & 8 & 2 \end{bmatrix}$$

3. Matriks kolom

   Matriks yang hanya memiliki satu kolom

$$\begin{bmatrix} 9 \\ 3 \\ 5 \end{bmatrix}$$

4. Matriks bujur sangkar (simetri):

   Matriks yang dimana panjang dari baris dan kolomnya adalah sama, dinotasikan dengan A = (a_{ij}) dengan i = 1,2,..., n dan j = 1,2,..., n.

$$\begin{bmatrix} 8 & 3 & 2 \\ 1 & 5 & 7 \\ 4 & 2 & 3 \end{bmatrix}$$

5. Matriks persegi panjang:

   Matriks yang dimana panjang dari baris dan kolomnya tidak sama, dinotasikan dengan A = (a_{ij}) dengan i = 1,2,..., n dan j = 1,2,..., m.

$$\begin{bmatrix} 3 & 4 & 9 \\ 5 & 2 & 6 \end{bmatrix}$$

6. Matriks diagonal:

   Matriks persegi yang tiap elemen pada diagonal utamanya bernilai *real* sedangkan semua entri lain di luar diagonal utama bernilai nol (Nathaniel 2021 : 41). Matriks diagonal dinotasikan A =(a_{ij}) dengan $a_{ij} = 0$ untuk $i \neq j$, a_{ij} = *real* untuk i = j.

$$\begin{bmatrix} 4 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

7. Matrik satuan (identitas):

   Matriks persegi yang tiap elemen pada diagonal utamanya bernilai satu sedangkan elemen lainnya bernilai nol, dinotasikan A =(a_{ij}) dengan $a_{ij} = 0$ untuk $i \neq j$, $a_{ij} = 1$ untuk $i = j$.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

8. Matrik skalar:

   Matriks persegi yang tiap elemen pada diagonal utamanya bernilai real dan sama sedangkan elemen lainnya bernilai nol, dinotasikan A =(a_{ij}) dengan $a_{ij} = 0$ untuk $i \neq j$, $a_{ij} = c$ untuk $i = j$.

$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

Matriks juga memiliki sifat-sifat operasi matematika yang mendasari berbagai aplikasi dalam sains. Jika A, B, C adalah matriks berordo mxm dan k, l adalah skalar, berikut adalah macam-macam sifatnya :

1. A + B = B + A, yang merupakan bentuk adisi komutatif,
2. A + (B + C) = (A + B) + C, yang merupakan bentuk adisi asosiatif,
3. k * (A + B) = kA + kB, yang merupakan bentuk multiplikasi distributif dengan skalar,
4. (k+l) * A = kA + lA, yang merupakan bentuk multiplikasi distributif dengan skalar,
5. (kl) A = k(lA), yang merupakan bentuk multiplikasi asosiatif dengan skalar,
6. k(A * B) = kA(B) = A(kB), yang merupakan bentuk multiplikasi distributif dengan skalar,
7. A(BC) = (AB)C, yang merupakan bentuk multiplikasi asosiatif,
8. A(B + C) = AB + AC, yang merupakan bentuk adisi distributif,
9. (A + B)C = AC + AB, yang merupakan bentuk adisi distributif,
10. A * B ≠ B * A, yang berarti tidak berbentuk multiplikasi komutatif
11. Jika A * B = A * C, bukan berarti B = C,

12. Jika A * B = 0 , bisa jadi A = 0 dan B = 0 atau A ≠ 0 dan B ≠ 0.

2.6 Invers Matriks

Invers matriks adalah matriks yang, jika dikalikan dengan matriks asalnya, menghasilkan matriks identitas. Matriks invers hanya ada untuk **matriks persegi** (n × n) dan jika determinannya tidak sama dengan nol:

$$(det(A) \neq 0)$$

Jika *A* adalah matriks, maka invers *A*, dilambangkan sebagai *A^-1*, memenuhi:

$$A.A^{-1} = A^{-1}.A = I$$

Dengan *I* adalah matriks identitas.

Langkah-langkah menentukan matriks terdapat beberapa, namun berikut merupakan 2 metode yang sering digunakan:

1. Menggunakan determinan dan adjoin

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

   a. *det(A)*: Determinan matriks *A*.
   b. *adj(A)*: Matriks adjoin *A* (transpose kofaktor A).

2. Menggunakan operasi baris elementer: Mengubah *A* menjadi matriks identitas *I*, sambil menerapkan operasi baris pada *I*, yang nantinya akan menjadi A^-1.

Contoh soal:

Misalkan, diberikan sebuah matriks *A* sebagai berikut:

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

Berikut langkah-langkah penyelesaiannya:

1. Menghitung determinan matriks:

   Untuk matriks 2x2 seperti diatas, rumus yang digunakan ialah

$$det(A) = (a.d) - (b.c)$$

   Dimana,

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Sehingga apabila dimasukkan kedalam rumus

$$det(A) = (2.4) - (1.3) = 8 - 3 = 5$$

Dengan nilai det(*A*) != 0, maka matriks *A* memiliki invers.

2. Mencari kofaktor matriks:

   Untuk matriks 2x2, rumus kofaktor ialah

$$Cof\left(A\right) = \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

   Sehingga apabila disubtitusikan dengan *A*

$$Cof\left(A\right) = \begin{bmatrix} 4 & -1 \\ -3 & 2 \end{bmatrix}$$

3. Selanjutnya, mencari transpose kofaktor matriks(Adjoin) dengan hasil:

$$adj\left(A\right) = \begin{bmatrix} 4 & -3 \\ -1 & 2 \end{bmatrix}$$

4. Langkah terakhir yakni menentukan nilai invers dengan rumus diatas yang disubstitusikan dengan *A*,

   Maka hasilnya ialah,

$$A^{-1} = \frac{1}{5} . \begin{bmatrix} 4 & -3 \\ -1 & 2 \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} \frac{4}{5} & -\frac{3}{5} \\ -\frac{1}{5} & \frac{2}{5} \end{bmatrix}$$

2.7 LU Decomposition

LU Decomposition adalah metode untuk memfaktorkan sebuah matriks persegi *A* menjadi dua matriks:

1. ***L***: Matriks segitiga bawah (*Lower triangular matrix*) dengan elemen diagonal sama dengan 1.
2. ***U***: Matriks segitiga atas (*Upper triangular matrix*).

Bentuk LU Decomposition sebagai berikut:

$$A = L.U$$

Metode ini sering digunakan untuk menyelesaikan sistem persamaan linear, menghitung determinan, atau mencari invers matriks.

Langkah-langkah penyelesaian LU Decomposition:

1. Pastikan matriks *A* adalah matriks persegi (*n x n*).
2. Faktorkan *A* menjadi dua buah matriks, yakni *L* dan *U* melalui eliminasi Gauss:

   L: Berisi faktor eliminasi (koefisien yang digunakan untuk menghilangkan elemen di bawah diagonal).

   U: Berisi hasil eliminasi, yaitu matriks segitiga atas.

Contoh soal:

- Terdapat matriks *A*:

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{bmatrix}$$

Berikut langkah penyelesaiannya:

1. Inisialisasi *L* dan *U*,

   L merupakan matriks segitiga bawah dengan elemen diagonal =1.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix}$$

   Lalu, matriks *U* merupakan matriks segitiga atas diinisialisasi sama dengan *A*, yang akan diubah melalui eliminasi).

$$U = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{bmatrix}$$

2. Eliminasi baris pertama,

   Hilangkan tiap-tiap elemen dibawah diagonal pertama (U_{21} dan U_{32}):

$$l_{21} = \frac{U_{21}}{U_{11}} = \frac{4}{2} = 2,\ l_{31} = \frac{U_{31}}{U_{11}} = \frac{8}{2} = 4$$

Perbarui baris kedua dan ketiga dari matriks *U*:

$$U_2 = [4 \quad 3 \quad 3] - 2.[2 \quad 1 \quad 1] = [0 \quad 1 \quad 1]$$

$$U_3 = [8 \quad 7 \quad 9] - 4.[2 \quad 1 \quad 1] = [0 \quad 3 \quad 5]$$

Sehingga matriks *U* dan *L* menjadi:

$$U = \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 3 & 5 \end{bmatrix} \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 0 & 1 \end{bmatrix}$$

3. Dilanjutkan dengan eliminasi baris kedua

Hilangkan elemen dibawah diagonal kedua (U_{32}):

$$l_{32} = \frac{U_{32}}{U_{22}} = \frac{3}{1} = 3$$

Lalu, perbarui baris ketiga *U*:

$$U_3 = [0 \quad 3 \quad 5] - 3.[0 \quad 1 \quad 1] = [0 \quad 0 \quad 2]$$

Sehingga matriks *U* dan *L* menjadi:

$$U = \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix} \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 3 & 1 \end{bmatrix}$$

Maka, hasil akhir dari LU Decomposition untuk matriks *A* adalah

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 3 & 1 \end{bmatrix} . \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

Untuk memverifikasinya, kami akan mengalikan matriks *L* dengan *U*, sehingga apabila hasilnya ialah matriks *A*, maka kedua matriks tersebut telah sesuai:

$$L.U = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 3 & 1 \end{bmatrix} . \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{bmatrix}$$

2.8 Metode Numerik

A. Metode numerik persamaan linear

Metode ini digunakan untuk menyelesaikan sistem persamaan linear berbentuk:

$$Ax = b$$

Dimana *A* adalah matriks koefisien, *x* adalah vektor solusi, dan *b* adalah konstanta.

Contoh metode-metode untuk numerik persamaan linear:

1. **Eliminasi Gauss**: Mengubah sistem menjadi matriks segitiga atas untuk menyelesaikannya melalui substitusi balik.
2. **Iterasi Jacobi/Gauss-Seidel**: Metode iteratif untuk mendekati solusi sistem linear.

Contoh soal:

Diberikan sistem persamaan yang nantinya akan diselesaikan menggunakan metode Eliminasi Gauss:

$$2_x + y - z = 8$$
$$-3_x - y + 2_z = -11$$
$$-2_x + y + 2_z = -3$$

Berikut merupakan langkah-langkah penyelesaiannya:

1. Pertama-tama, susun matriks *augmented* seperti dibawah:

$$\left(\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array}\right)$$

2. Lalu, eliminasi baris ke-2 dan ke-3 dengan baris 1:

$$R_2 \rightarrow R_2 + \frac{3}{2}R_1, R_3 \rightarrow R_3 + R_1$$

Maka hasil yang didapatkan:

$$\left(\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & 0.5 & 0.5 & 1 \\ 0 & 2 & 1 & 5 \end{array}\right)$$

3. Selanjutnya eliminasi baris 3 dengan baris 2, maka hasilnya:

$$\left(\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & 0.5 & 0.5 & 1 \\ 0 & 0 & -1 & 1 \end{array}\right)$$

4. Sehingga apabila dilakukan substitusi balik:

$$z = -1, \qquad y = 3, \qquad x = 2$$

B. Metode numerik persamaan non-linear

Metode ini digunakan untuk menyelesaikan persamaan non-linear berbentuk:

$$f(x) = 0$$

dimana *f(x)* adalah fungsi non-linear.

Contoh metode-metode untuk numerik persamaan non-linear:

1. **Metode Bagi Dua (Bisection)**: Membagi interval menjadi dua bagian untuk menemukan akar.
2. **Metode Newton-Raphson**: Menggunakan turunan fungsi untuk mendekati akar secara iteratif.

Contoh soal dengan menggunakan *Bisection Method*:

Diberikan sistem persamaan sebagai berikut:

$$f(x) = x3 - x - 2 = 0$$

Goals nya ialah mencari akar persamaan dalam interval [1,2] dengan toleransi 0.01.

Berikut merupakan langkah-langkah penyelesaiannya menggunakan metode *Bisection*:

1. Periksa terlebih dahulu nilai awal:

   Hitung *f*(*a*) dan *f*(*b*) untuk *a* = 1 dan *b* = 2:

   $$f(1) = 13 - 1 - 2 = -2$$

   $$f(2) = 23 - 2 - 2 = 4$$

   Karena $f(1) \,.\, f(2) < 0 (-2 \,.\, 4 = -8)$, maka ada akar di interval [1,2].

2. Selanjutnya, menentukan rumus Bisection:

   Rumus titik tengah:

$$C = \frac{a + b}{2}$$

   Hitung *f*(*c*), lalu perbarui interval:

   1. Jika f(a).f(c) <0, maka akar ada di [a,c].
   2. Jika f(c).f(b) <0, maka akar ada di [c,b].
   3. Ulangi hingga |b-a|<.

3. Lakukan iterasi dengan detail sebagai berikut:

Iterasi	a	b	c	f(a)	f(b)	f(c)	Interval baru
1	1.0000	2.0000	1.5000	-2.0000	4.0000	-0.8750	[1,1.5]
2	1.5000	2.0000	1.7500	-0.8750	4.0000	1.6094	[1.5,1.75]
3	1.5000	1.7500	1.6250	-0.8750	1.6094	0.6660	[1.5,1.625]
4	1.5000	1.6250	1.5625	-0.8750	0.6660	-0.2559	[1.5625,1.625]
5	1.5625	1.6250	1.5938	-0.2559	0.6660	0.1978	[1.5625,1.5938]
6	1.5625	1.5938	1.5781	-0.2559	0.1978	-0.0314	[1.5781,1.5938]
7	1.5781	1.5938	1.5859	-0.0314	0.1978	0.0811	[1.5781,1.5859]
8	1.5781	1.5859	1.5820	-0.0314	0.0811	0.0248	[1.5781,1.5820]
9	1.5781	1.5820	1.5801	-0.0314	0.0248	-0.0033	[1.5801,1.5820]

4. Hentikan iterasi ketika telah mencapai kondisi berikut:

$$|b - a| < \varepsilon :$$

$$|1.5820 - 1.5801| = 0.0019 < 0.01$$

   Akan mendekati:

$$X \approx 1.5801$$

2.9 Simulasi

Simulasi menurut KBBI adalah metode pelatihan yang meragakan sesuatu dalam bentuk tiruan yang mirip dengan keadaan yang sesungguhnya. Tujuan utama dari simulasi adalah mengestimasi kuantitas yang dibutuhkan untuk komputasi yang rumit, beresiko, konsumtif, mahal atau bahkan mustahil (Baron 2013 : 101). Sebagai contoh, kita pasti lebih memilih untuk menguji perhitungan dan keamanan peluncuran roket melalui simulasi komputer dibandingkan pengujian secara langsung.

Simulasi akan lebih bermanfaat jika kita mengetahui apa yang sesungguhnya terjadi. Memahami cara kerja simulasi membantu kita untuk menerapkan fungsi dengan benar dan memahami output yang diberikan.

Simulasi terbagi menjadi tiga tipe :

1. Berdasarkan lintasan waktu
    a. Simulasi statik

       Simulasi yang tidak didasarkan oleh waktu dan sampel data diambil secara acak. Biasanya disebut juga sebagai simulasi Monte Carlo. Contoh dari simulasi statik adalah memilih portofolio untuk saham
    b. Simulasi dinamik

       Simulasi yang didasarkan pada waktu dan memiliki suatu mekanisme waktu (*clock mechanism*) yang dapat merubah variabel status.
2. Berdasarkan input variabel
    a. Simulasi stokastik

       Simulasi yang salah satu atau semua input variabelnya adalah acak, sehingga output yang diberikan juga acak. Simulasi stokastik memberikan hanya satu titik data untuk mengetahui bagaimana sistem berperilaku
    b. Simulasi deterministik

       Simulasi yang semua input variabelnya tidak bersifat acak. Semua *upcoming state* ditentukan setelah data input dan *early state* didefinisikan.
3. Berdasarkan distribusi nilai
    a. Simulasi diskrit

       Simulasi yang merepresentasikan perubahan nilai dalam bentuk nilai yang memiliki beberapa kemungkinan dan biasanya terjadi pada titik-titik waktu tertentu. Contoh dari simulasi diskrit adalah jumlah barang dalam suatu lot dan jumlah individu dalam suatu kelompok.

b. Simulasi kontinu

Simulasi yang merepresentasikan perubahan nilai secara berkelanjutan dalam suatu rentang tertentu. Contoh dari simulasi kontinu adalah kerja mesin dengan waktu siklus yang terdistribusi secara seragam.

Beberapa simulasi yang disebutkan di atas membutuhkan input variabel acak untuk menghasilkan output yang realistis. Untuk menghasilkan angka acak, maka dibuatlah *random number generator* (RNG). RNG adalah algoritma yang bertujuan menghasilkan angka secara acak yang tidak dapat diprediksi. Kebanyakan bahasa komputer memiliki generator bilangan acak yang hanya mengembalikan nilai variabel acak independen yang terdistribusi secara acak (Baron 2013 : 104).

Tetapi, ada juga RNG yang menghasilkan angka terlihat secara acak, tetapi pada kenyataannya, angka dapat diprediksi jika diketahui rumus, parameter, *seed* nya. RNG tersebut disebut sebagai *pseudo random number generator* (PRNG). Algoritma PRNG bekerja berdasarkan algoritma deterministik yang menggunakkan parameter awal dan seed. PRNG memiliki suatu periode tertentu yang dimana jika PRNG sudah mencapai periode tersebut, maka urutan angka yang dihasilkan akan kembali berulang dari awal, menghasilkan pola sebelumnya. Contoh dari PRNG adalah algoritma *Linear Congruential Generators* (LCG) dan *Mersenne Twister*.

Linear Congruential Generators dibutuhkan ketika kita membutuhkan angka acak dengan cara yang sederhana dan efisien. LCG biasa digunakan pada simulasi stokastik. Efisiensinya dalam hal komputasi dan memori membuat LCG menjadi salah satu PRNG yang banyak digunakan. LCG menghasilkan angka acak berdasarkan rumus rekursi berikut :

$$Zi = (\mathrm{a} * \mathrm{Zi} - 1 + \mathrm{c}) \bmod (\mathrm{m})$$

a = *multiplier*

c = *increment*

m = *modulus*

$Z0$ = *start value*

LCG menghasilkan angka acak dalam rentang [0 , m-1] dan kualitas keacakan angka bergantung pada pemilihan nilai a,c, dan m. Jika parameter dipilih dengan benar, LCG dapat menghasilkan urutan angka yang terlihat acak dengan periode panjang.

Sebagai contoh, jika kita mengambil parameter sebagai berikut :

$$a = 23, \quad c = 5, \quad m = 11, \quad Z0 = 12$$

maka,

$$Z1 = (23 * 12 + 5)\ mod\ 11$$
$$= (276 + 5)\ mod\ 11$$
$$= 281\ mod\ 11$$
$$= 6$$
$$Z2 = (23 * 6 + 5)\ mod\ 11$$
$$= (138 + 5)\ mod\ 11$$
$$= 143\ mod\ 11$$
$$= 0$$

Jika diteruskan, maka hasilnya adalah sebagai berikut

iteration	calc	mod	Zi	Ui = Zi/m
1	281	11	6	0.5455
2	143	11	0	0.0
3	5	11	5	0.4545
4	120	11	10	0.9091
5	235	11	4	0.3636
6	97	11	9	0.8182
7	212	11	3	0.2727
8	74	11	8	0.7273
9	189	11	2	0.1818
10	51	11	7	0.6364
11	166	11	1	0.0909
12	28	11	6	0.5455
13	143	11	0	0.0
14	5	11	5	0.4545

Gambar 2.5.1 Hasil perhitungan *LGC*

*Keterangan : calc = (a * Zi-1 + c)

Jika diperhatikan dengan seksama, pada iterasi ke 11, hasil perhitungan Zi dan Ui akan sama seperti iterasi ke 1. Hasil perhitungan Zi , Ui, dan calc pada iterasi ke 12 akan sama seperti iterasi ke 2, dan seterusnya. Fenomena ini menunjukkan bahwa urutan angka yang dihasilkan akan berulang setelah mencapai periode tertentu. Periode tersebut biasanya berulang setelah mencapai m (*modulus*) atau maksimal m - 1 setelah iterasi, tergantung pada pemilihan parameter seperti *multiplier*, *increment*, dan *start point.*

Dalam simulasi, Nilai yang biasa digunakan adalah nilai dari Zi, sedangkan nilai dari Ui adalah hasil mengubah nilai bilangan Zi menjadi bilangan desimal dalam rentang [0,1). Ui biasa digunakan pada simulasi monte-carlo, distribusi statistik, ataupun model probabilistik lainnya.

2.10 Simulasi Monte-Carlo

Simulasi Monte Carlo adalah metode numerik yang digunakan untuk menyelesaikan masalah matematika atau fisika melalui pengambilan sampel acak. Teknik ini digunakan dalam berbagai bidang seperti keuangan, teknik, dan fisika untuk memodelkan ketidakpastian dan memperkirakan solusi.

Prinsip dasar Monte Carlo adalah menggunakan pengulangan eksperimen acak untuk mendekati solusi. Hasil dihitung sebagai rata-rata dari pengamatan acak.

Contoh soal:

Estimasikan nilai dari dengan eksperimen simulasi. Misalkan kami akan menembakkan sejumlah titik acak ke dalam sebuah persegi dengan sisi 2, yang berpusat di sekitar lingkaran dengan jari-jari r = 1.

Berikut langkah penyelesaiannya:

1. Dengan konsep matematika:
   A. Lingkaran dengan jari-jari $r = 1$ memiliki luas $A = \pi r^2 = \pi$.
   B. Persegi dengan panjang sisi 2 memiliki luas $A = 4$.
   C. Rasio antara luas lingkaran dan luas persegi adalah

$$\frac{Luas\ Lingkaran}{Luas\ Persegi} = \frac{\pi}{4}$$

Oleh karena itu:

$$\pi = 4 \times \frac{Jumlah\ Titik\ Dalam\ Lingkaran}{Jumlah\ Total\ Titik}$$

2. Dengan simulasi membuat titik:
   A. Membuat titik acak x dan y dalam interval [-1,1].
   B. Periksa posisi titik dengan menghitung $d = x^2+y^2$.
   C. Jika $d \leq 1$, titik berada dalam lingkaran.
   D. Lalu, menghitung rasio titik dalam lingkaran terhadap total titik digunakan untuk mendekati nilai.

2.11 Rantai Markov

Rantai Markov atau Markov Chain merupakan teknik perhitungan yang digunakan untuk melakukan berbagai macam pemodelan dengan berbagai kondisi. Teknik ini digunakan untuk membantu memperkirakan perubahan yang bisa terjadi di masa yang akan datang. Model Rantai Markov pertama kali ditemukan oleh ilmuwan Rusia bernama Andrey Andreyevich pada tahun 1906.

Model ini berfokus untuk menghitung probabilitas berdasarkan kondisi kejadian sebelumnya. Dalam proses pembuatan Rantai Markov, ada dua prosedur yaitu menyusun matriks probabilitas transisi dan juga menghitung kemungkinan di waktu yang akan datang. Probabilitas transisi merupakan kemungkinan pergantian yang mungkin pada salah satu state, contoh:

a. Cerah → Cerah = 30% (0.3)
b. Cerah → Hujan = 70% (0.7)
c. Hujan → Cerah = 20% (0.2)
d. Hujan → Hujan = 80% (0.8)

Ketika Dibuat dalam matriks atau tabel, menjadi seperti berikut

	Cerah	Hujan
Cerah	0.3	0.7
Hujan	0.2	0.8

Dalam contoh ini, kondisi Cerah ke Cerah memiliki kemungkinan sebesar 30% atau ketika ditulis dalam bentuk desimal menjadi 0.3. Kemudian kondisi dari Cerah ke Hujan memiliki kemungkinan sebesar 0.7, kemungkinan yang berawal dari Cerah ketika dijumlah harus menghasilkan angka satu, karena probabilitas terbagi menjadi dua (0.3 + 0.7 = 1). Dan kondisi yang sama berlaku untuk Hujan.

Dibawah ini merupakan Formula dari Rantai Markov, dimana hasil dari kondisi di masa yang akan datang bergantung pada kondisi sekarang.

$$\mathbb{P}(X_{t+1} = s \,|\, X_t = s_t, X_{t-1} = s_{t-1}, \ldots, X_0 = s_0) \;=\; \mathbb{P}(X_{t+1} = s \,|\, X_t = s_t),$$

2.12 Metode Iterasi

Metode iterasi adalah metode perhitungan matematika yang menemukan jawaban dalam suatu masalah melalui proses pengulangan. Setiap iterasi yang dilakukan menghasilkan jawaban yang mendekati solusi yang diinginkan hingga mencapai tingkat toleransi tertentu.

Sesuai yang disebutkan sebelumnya, metode iterasi dapat digunakan untuk mencari nilai variabel dari suatu persamaan linear kompleks. Dari sekian banyak metode iterasi, metode iterasi jacobi dan metode iterasi gauss-*seidel* merupakan metode iterasi yang paling sering digunakan (Ihsan, dkk 2024 : 35).

Prinsip kerja metode iterasi jacobi adalah menghitung nilai baru untuk setiap variabel secara bersamaan dalam setiap iterasi dan nilai tersebut menggantikan nilai yang lama. Metode iterasi akan mempunyai solusi (konvergen) jika matriks koefisiennya memiliki sifat dominan diagonal. Metode iterasi jacobi memiliki rumus sebagai berikut :

$$x_i^{(k)} = \frac{1}{a_{ii}}\left[-\sum_{\substack{j=1 \\ j\neq i}}^{n}\left(a_{ij}x_j^{(k-1)}\right) + b_i\right]$$

Gambar 2.6.1 Rumus perhitungan metode iterasi jacobi

Untuk menghitung keakuratan dari suatu jawaban, maka bisa menggunakkan rumus sebagai berikut :

$$\left|\frac{x_s^{(k)} - x_s^{(k-1)}}{x_s^{(k)}}\right|$$

Gambar 2.6.2 Rumus galat relatif untuk mengetahui hasil keakuratan perhitungan

*Keterangan: Ɛ = keakuratan yang diinginkan

Sebagai contoh, jika terdapat tiga persamaan linear sebagai berikut:

$$4x - y + z = 7$$
$$4x - 8y + z = -21$$
$$-2x + y + 5z = 15$$

Langkah pertama yang dilakukan pada metode iterasi jacobi adalah menulis ulang setiap variabel dalam bentuk dirinya sendiri:

$$x = 7 + y - z / 4$$

$$y = -21 - 4x - z / -8$$

$$z = 15 + 2x - y / 5$$

Langkah kedua adalah memberi nilai berupa *initial guess* yaitu x(0)= 0, y(0)= 0, z(0)=0 untuk memulai iterasi:

$$x(1) = 7 + 0 - 0 / 4$$

$$y(1) = -21 - 4(0) - 0 / -8$$

$$z(1) = 15 + 2(0) - 0 / 5$$

Lalu dilakukan perhitungan untuk menggantikan nilai *initial guess* :

$$x(1) = 1.75$$

$$y(1) = 2.625$$

$$z(1) = 3$$

Setelah itu, lakukan iterasi berikutnya dengan mengganti nilai variabel menggunakkan nilai iterasi sebelumnya :

$$x(2) = 7 + 2.625 - 3 / 4$$

$$y(2) = -21 - 4(1.75) - 3 / -8$$

$$z(2) = 15 + 2(1.75) - 2.625 / 5$$

$$x(2) = 1.65625$$

$$y(2) = 3.875$$

$$z(2) = 3.175$$

Untuk mengukur keakuratan jawaban tersebut, kita dapat menghitung menggunakkan rumus galat relatif lalu dikali 100%.

$$Persentase\ x(2) = 1.65625 - 1.75 / 1.6565 * 100\%$$

$$Persentase\ y(2) = 3.0625 - 2.625 / 3.0625 * 100\%$$

$$Persentase\ z(2) = 3.175 - 3 / 3.175 * 100\%$$

$$Persentase\ x(2) = -5.66\%$$

$$Persentase\ y(2) = 32.26\%$$

$$Persentase\ z(2) = 5.51\%$$

Lakukan iterasi terus menerus sampai semua persentase menyentuh angka 0.00%

$$x(13) = 7 + 4.0000 - 3.0000 \,/\, 4$$
$$y(13) = -21 - 4(2{,}00000) - 3.0000 \,/\, -8$$
$$z\,(13) = 15 + 2(2{,}00000) - 4.0000/\,5$$

$$x(13) = 2{,}00000$$
$$y(13) = 4.0000$$
$$z\,(13) = 3.0000$$

$$Persentase\ x(13) = 2{,}00000 - 2{,}00000/\,2{,}00000 * 100\%$$
$$Persentase\ y(13) = 4{,}00000 - 3{,}99999 \,/\, 4{,}00000 * 100\%$$
$$Persentase\ z(13) = 3.0000 - 2{,}99999 \,/\, 3.0000 * 100\%$$

$$Persentase\ x(13) = 0.00\%$$
$$Persentase\ y(13) = 0.00\%$$
$$Persentase\ z(13) = 0.00\%$$

Persentase 0.00% juga dipengaruhi dengan banyaknya angka di belakang koma. Semakin sedikit angka di belakang koma, iterasi semakin sedikit, tetapi akurasi juga semakin rendah. Sebaliknya, semakin banyak angka di belakang koma, iterasi semakin banyak dan akurasi juga semakin tinggi.

2.13 Metode Iterasi Gauss-Seidel

Metode **Gauss-Seidel** adalah teknik numerik untuk menyelesaikan sistem persamaan linear berbentuk:

$$\overline{A_x = b}$$

Metode ini adalah salah satu metode iterasi di mana solusi diperbarui secara bertahap berdasarkan nilai yang telah diperoleh dalam iterasi sebelumnya. Ciri khas Gauss-Seidel adalah nilai terbaru dari variabel langsung digunakan dalam iterasi berikutnya.

Rumus iterasi Gauss-Seidel untuk sistem *n* persamaan :

$$a_{11}x_1 + a_{12} + \cdots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22} + \cdots + a_{2n}x_n = b_2$$
$$\vdots$$
$$a_{n1}x_1 + a_{n2} + \cdots + a_{nn}x_n = b_n$$

adalah :

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_i^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)})$$

Dimana:

- $x_i^{(k+1)}$: Nilai terbaru untuk variabel xi.
- $x_j^{(k+1)}$: Nilai yang telah diperbaruh dalam iterasi saat ini.
- $x_j^{(k)}$: Nilai dari iterasi sebelumnya.

BAB III

PEMBAHASAN

Untuk mengimplementasikan semua materi di atas, maka kami menggunakkan bahasa pemrograman python dengan bantuan library NumPy dan Matplotlib untuk membantu proses perhitungan matriks. Program kami dirancang dengan pendekatan *functional programming*, di mana setiap fungsinya dikelompokkan berdasarkan 12 materi yang sudah disebutkan pada bagian latar belakang, yaitu:

1. Sistem persamaan linear dengan cara triangularisasi dan substitusi mundur,
2. Sistem persamaan linear dengan cara eliminasi gauss,
3. Praktik matriks dan operasinya,
4. Invers matriks,
5. *LU Decomposition*,
6. Metode numerik persamaan linear
7. Metode numerik persamaan non-linear,
8. Dasar-dasar simulasi,
9. Metode Iterasi Jacobi,
10. Metode Iterasi *Gauss-Seidel*,
11. Simulasi Monte-Carlo,
12. Rantai Markov.

Kami membagi materi tersebut berdasarkan pada jadwal mengajar yang telah diadakan oleh Ibu Yuni Yamasari sekaligus juga berdasarkan pada materi yang diberikan oleh Ibu Yuni Yamasari. Sebagai contoh Sistem persamaan linear dengan cara triangularisasi dan substitusi mundur diajarkan pada minggu ke tiga perkuliahan, praktik matriks dan operasinya diajarkan pada minggu ke empat.

```
def main():
    print("================== Group 7 ==================")
```

```
    print("| 1. Rayhan Hendra Atmadja     075 - TI 23 C |")
    print("| 2. Adriano Emmanuel          082 - TI 23 C |")
    print("| 3. Cornelius Louis Nathan    085 - TI 23 C |")
    print("=============================================")

    while True:
        print("\n========== Science Comp Implementation ========")
        print("1. Systems of Linear Equations") # Sistem Persamaan Linear
        print("2. Matrix Practices")
        print("3. Invertible Matrix") # Invers Matrix
        print("4. LU Decomposition")
        print("5. Linear Equations")  # Persamaan linear
        print("6. Non Linear Equations")
        print("7. Interpolation") # Interpolasi
        print("8. Basic Simulation")
        print("9. Iterasi Theory") # Teori iterasi
        print("10. Monte-Carlo Simulation")
        print("11. Markov Chain")
        print("12. Exit\n")

        print("Choose from 1-11")
```

3.1 Sistem Persamaan Linear

Pada implementasi Sistem Persamaan Linear kali ini, kami merancang untuk menyelesaikan Sistem Persamaan Linear dengan menggunakan matriks dan vektor. Namun, perlu ditekankan bahwa metode yang kami gunakan dalam kode dibawah ini, yaitu eliminasi Gauss-Jordan, lalu kami juga menerapkan metode triangularisasi dan substitusi mundur.

```
def func1() :
    rowsfunc1, colsfunc1 = map(int, input("Enter size of matrix (rows cols): ").split())

    if rowsfunc1 < 2 or colsfunc1 < 2:
        print("Matrix size must be at least 2x2.")
        return  # Kembali jika ukuran tidak valid

    if rowsfunc1 != colsfunc1:
        print("Matrix must be square (rows == cols) to solve a system of linear equations.")
        return  # Kembali jika matriks bukan persegi
```

```
    matrix = []
    y = []

    print(f"Enter the elements for a {rowsfunc1}x{colsfunc1} matrix:")
    for i in range(rowsfunc1):
        while True:
            try:
                row = list(map(int, input(f"Enter row #{i+1}: ").split()))
                if len(row) != colls:
                    raise ValueError(f"Row must have exactly {colsfunc1} elements.")
                matrix.append(row)
                break
            except ValueError as e:
                print(e)

    print("Enter elements for the vector (y):")
    for i in range(colsfunc1):
        while True:
            try:
                el = int(input(f"Enter element #{i+1}: "))
                y.append(el)
                break
            except ValueError:
                print("Please enter a valid integer.")

    A = np.array(matrix)
    y = np.array(y)

    # Periksa apakah determinan matriks adalah nol
    if np.linalg.det(A) == 0:
        print("The matrix is singular and cannot be solved.")
        return

    # Menyelesaikan sistem persamaan linear
    x = np.linalg.solve(A, y)
    print("\nSolution (x):")
    print(x)
```

3.2 Praktik Matriks

Pada pengimplementasian praktik matriks dan operasinya (menu kedua), kami mengimplementasikan bentuk dan sifat dari matriks yang sudah disebutkan pada bab landasan teori.

```
while True:
        print("\nMatrix Properties")
        print("1. Types of matrices")
        print("2. Behavior of matrix")
        print("3. Exit")
        try:
            choice = int(input("> "))
        except ValueError:
            print("\nInvalid input, choose from 1-2!")
            enter()
            continue

        if choice == 1:
            func2_1()
        elif choice == 2:
            func2_2()
        elif choice == 3:
            break
```

Untuk bagian bentuk matriks, kami menampilkan delapan tipe matriks sesuai dengan tipenya, seperti matriks nol, matriks persegi, matriks diagonal. Matriks ini dibuat dengan ordo acak menggunakkan np.random.randint(1,9), sehingga matriks akan memiliki baris dan kolom yang beragam. Hal ini dilakukan agar ukuran matriks dapat bervariasi. Tak lupa juga kami memberikan penjelasan singkat pada tiap matriks.

```
def func2_1():
    while True:
        print("\nThere are a lot of types of matrices")
        print("1. Zero Matrix")
        print("2. Square Matrix")
        print("3. Rectangular Matrix")
        print("4. Diagonal Matrix")
        print("5. Identity Matrix")
        print("6. Scalar Matrix")
        print("7. Row Matrix")
        print("8. Column Matrix")
        print("7. Exit")
```

```
        print("Choose one to see the example")


        try:
            choice = int(input("> "))
        except ValueError:
            print("\nInvalid input, choose from 1-2!")
            enter()
            continue

        if choice == 1:
            print("\nZero Matrix or Null Matrix is a matrix with all its elements")
            print("having a VALUE of ZERO.")

            zeroMatrix = np.zeros((rows,colls))
            print("\n", zeroMatrix)
            enter()


        elif choice == 2:
            print("\nSquare Matrix is a matrix where the lengths")
            print("of row and the columns ARE the SAME.")


            squareMatrix = np.random.randint(9, size=(4,4))
            print("\n", squareMatrix)
            enter()


        elif choice == 3:
            print("\nRectangular Matrix is a matrix where the lengths")
            print("of row and the column are NOT the SAME.")


            rectangularMatrix = np.random.randint(9, size=(rows,colls))
            print("\n", rectangularMatrix)
            enter()



        elif choice == 4:
            print("\nDiagonal Matrix is a square matrix with the elements")
            print("on the MAIN DIAGONAL having real values while the others are ZERO")
            diagonalMatrix = np.diag([np.random.randint(1,9),np.random.randint(1,9),
np.random.randint(1,9)])
            print("\n", diagonalMatrix)
            enter()
```

```
        elif choice == 5:
            print("\nIdentity Matrix is a square matrix with the elements")
            print("on the MAIN DIAGONAL having a value of '1' while the others are ZERO")

            identityMatrix = np.eye(4)
            print("\n", identityMatrix)
            enter()

        elif choice == 6:
            print("\nScalar Matrix is a square matrix with the elements")
            print("on the MAIN DIAGONAL having the same values while the others are
ZERO")

            scalarMatrix = np.eye(4) * np.random.randint(1,9)
            print("\n", scalarMatrix)
            enter()

        elif choice == 7:
            print("\nRow Matrix is a matrix that only consist of one row")

            rowMatrix = np.array([np.random.randint(1,9), np.random.randint(1,9),
np.random.randint(1,9)])
            print("\n", rowMatrix)
            enter()

        elif choice == 8:
            print("\nColumn Matrix is a matrix that only consist of one column")

            columnMatrix = np.array([[np.random.randint(1,9)], [np.random.randint(1,9)],
[np.random.randint(1,9)]])
            print("\n", columnMatrix)
            enter()

        elif choice == 9:
            break

        else:
            print("\nInvalid choice! Please choose from 1-9.")
            enter()
            continue
```

Pada bagian sifat matriks, kami menampilkan dua belas sifat matriks beserta pembuktian dari sifat-sifat tersebut. Matriks ini dibuat dengan ordo 3x3 dengan tiap elemen bernilai acak menggunakkan np.random.randint(1,9), sehingga matriks akan memiliki nilai yang beragam. Kami juga menambahkan variabel k dan l sebagai skalar untuk mendukung pembuktian dari sifat-sifat matriks seperti asosiatif, distributif, komutatif.

```
def func2_2():
    A = np.array([[np.random.randint(1,9), np.random.randint(1,9)],
[np.random.randint(1,9),np.random.randint(1,9)]])
    B = np.array([[np.random.randint(1,9), np.random.randint(1,9)],
[np.random.randint(1,9),np.random.randint(1,9)]])
    C = np.array([[np.random.randint(1,9), np.random.randint(1,9)],
[np.random.randint(1,9),np.random.randint(1,9)]])
    k = np.random.randint(1,5)
    l = np.random.randint(1,5)

    while True:
        print("\nMatrix has a lot of behavior")
        print("Suppose A,B,C are matrix and k and l are scalar")
        print("1. A + B = B + A\t\tCommutative Addition")
        print("2. A + (B + C) = (A + B) + C\tAssociative Addition")
        print("3. k * (A + B) = kA + kB\tDistributive Multiplication With Scalar")
        print("4. (k+l) A = kA + lA\t\tDistributive Multiplication With Scalar")
        print("5. (kl) A = k(lA)\t\tAssociative Multiplication With Scalar")
        print("6. k(A * B) = kA(B) = A(kB)\tDistributive Multiplication With Scalar")
        print("7. A(BC) = (AB)C\t\tAssociative Multiplication")
        print("8. A(B + C) = AB + AC\t\tDistributive Addition")
        print("9. (A + B)C = AC + BC\t\tDistributive Addition")
        print("10. A * B != B * A\t\tNot Commutative Multiplication")
        print("11. If A * B = A * C, does not mean B = C ")
        print("12. If A * B = 0, it's either A = 0 and B = 0")
        print("                  or A != 0 and B != 0")
        print("13. Exit")
        print("Choose one to see the example")

        try:
            choice = int(input("> "))
        except ValueError:
            print("\nInvalid input, choose from 1-2!")
            enter()
```

```
            continue


        if choice == 1:
            print("\n1. A + B = B + A\t\tCommutative Addition")
            print("\nA + B")
            print("\n",A), print("\n+ "), print("\n",B)
            print("\n= "), print("\n",A + B)


            print("\nwhile B + A")
            print("\n",B), print("\n+ "), print("\n",A)
            print("\n= "), print("\n",B + A)


            print("\nTherefore, A + B = B + A is TRUE")
            enter()


        elif choice == 2:
            print("\n2. A + (B + C) = (A + B) + C\tAssociative Addition")
            print("\nA + (B + C)")
            print("\n",A), print("\n+ ( "), print("\n",B), print("\n+ "), print("\n",C),
print("\n)")
            print("\n= "), print("\n",A + (B + C))


            print("\nwhile (A + B) + C")
            print("\n( "), print("\n",A), print("\n+ "), print("\n",B), print("\n)"),
print("\n+ "),print("\n",C),
            print("\n= "), print("\n",(A + B) + C)


            print("\nTherefore, A + (B + C) = (A + B) + C is TRUE")
            enter()


        elif choice == 3:
            print("\n3. k * (A + B) = kA + kB\tDistributive Multiplication With Scalar")
            print("\nk * (A + B)")
            print("\n",k," *"), print("\n( "), print("\n",A), print("\n+ "),
print("\n",B), print("\n )")
            print("\n= "), print("\n",k * (A + B))


            print("\nwhile kA + kB")
            print("\n",k," *"), print("\n",A), print("\n+ "), print("\n",k," *"),
print("\n",B),
```

```
            print("\n= "), print("\n",k*A + k*B)

            print("\nTherefore, k * (A + B) = kA + kB is TRUE")
            enter()

        elif choice == 4:
            print("\n4. (k+l) A = kA + lA\t\tDistributive Multiplication With Scalar")
            print("\n(k+l) A")
            print("\n(",k," + ",l,")"), print("\n* "), print("\n",A)
            print("\n= "), print("\n",(k+l) * A)

            print("\nwhile kA + lA")
            print("\n",k," *"), print("\n",A), print("\n+ "), print("\n",l," *"),
print("\n",A),
            print("\n= "), print("\n",k*A + l*A)

            print("\nTherefore, (k+l) A = kA + lA is TRUE")
            enter()

        elif choice == 5:
            print("\n5. (kl) A = k(lA)\t\t\tAssociative Multiplication With Scalar")
            print("\n(kl) A")
            print("\n(",k," * ",l,")"), print("\n",A)
            print("\n= "), print("\n",(k * l) * A)

            print("\nwhile k(lA)")
            print("\n",k," *"), print("\n(",l," *"), print("\n",A), print("\n)")
            print("\n= "), print("\n",k*(l*A))

            print("\nTherefore, (kl) A = k(lA) is TRUE")
            enter()

        elif choice == 6:
            print("\n6. k(A * B) = (kA) * B = A * (kB)\tDistributive Multiplication With
Scalar")
            print("\nk(A * B)")
            print("\n",k," * "), print("\n("), print("\n",A), print("\n* "),
print("\n",B)
            print("\n= "), print("\n",k * np.dot(A, B))
```

```
            print("\nwhile (kA) * B")
            print("\n(",k," * "), print("\n",A), print("\n)"), print("\n* "),
print("\n",B)
            print("\n= "), print("\n",np.dot(k * A, B))

            print("\nwhile A * (kB)")
            print("\n",A), print("\n*"), print("\n("), print("\n(",k," * "),
print("\n",B), print("\n)")
            print("\n= "), print("\n",np.dot(A, k * B))
            print("\nTherefore, k(A * B) = (kA) * B = A * (kB) is TRUE")
            enter()

        elif choice == 7:
            print("\n7. A(BC) = (AB)C\t\tAssociative Multiplication")
            print("\nA(BC)")
            print("\n",A), print("\n*"), print("\n("), print("\n",B), print("\n* "),
print("\n",C), print("\n)")
            print("\n= "), print("\n",np.dot(A, np.dot(B, C)))

            print("\nwhile (AB)C")
            print("\n("), print("\n",A), print("\n*"), print("\n",B), print("\n)"),
print("\n* "),print("\n",C)
            print("\n= "), print("\n",np.dot(np.dot(A, B), C))
            print("\nTherefore, A(BC) = (AB)C is TRUE")
            enter()

        elif choice == 8:
            print("\n8. A(B + C) = AB + AC\t\tDistributive Addition")
            print("\nA(B+C)")
            print("\n",A), print("\n*"), print("\n("), print("\n",B), print("\n+ "),
print("\n",C), print("\n)")
            print("\n= "), print("\n",np.dot(A, B + C))

            print("\nwhile AB + AC")
            print("\n",A), print("\n*"), print("\n",B), print("\n+"), print("\n",A),
print("\n*"), print("\n",C)
            print("\n= "), print("\n",np.dot(A, B) + np.dot(A, C))
            print("\nTherefore, A(B+C) = AB + AC is TRUE")
            enter()

        elif choice == 9:
```

```
            print("\n9. (A + B)C = AC + BC\t\tDistributive Addition")
            print("\n(A+B)C")
            print("\n("), print("\n",A), print("\n+"), print("\n",B), print("\n)"),
print("\n* "), print("\n",C),
            print("\n= "), print("\n",np.dot(A + B, C))


            print("\nwhile AC + BC")
            print("\n",A), print("\n*"), print("\n",C), print("\n+"), print("\n",B),
print("\n*"), print("\n",C)
            print("\n= "), print("\n",np.dot(A, C) + np.dot(B, C))
            print("\nTherefore, (A+B)C = AC + BC is TRUE")
            enter()


        elif choice == 10:
            print("\n10. A * B != B * A\t\tNot Commutative Multiplication")
            print("\nA * B")
            print("\n",A), print("\n*"), print("\n",B)
            print("\n= "), print("\n",np.dot(A, B))


            print("\nwhile B * A")
            print("\n",B), print("\n*"), print("\n",A)
            print("\n= "), print("\n",np.dot(B, A))
            print("\nSince the results are different")
            print("Therefore, A * B != B * A is TRUE")
            enter()


        elif choice == 11:
            print("11. If A * B = A * C, does not mean B = C ")
            AKhusus = np.array([[0, 100], [0, 0]])
            BKhusus = np.array([[-6, 28], [0, 0]])
            CKhusus = np.array([[48, -19], [0, 0]])


            print("\nA * B")
            print("\n",AKhusus), print("\n*"), print("\n",BKhusus)
            print("\n= "), print("\n",np.dot(AKhusus, BKhusus))


            print("\nA * C")
            print("\n",AKhusus), print("\n*"), print("\n",CKhusus)
            print("\n= "), print("\n",np.dot(AKhusus, CKhusus))


            print("\nEven the results are same, but B != C")
```

```
            print("Therefore, if A * B = A * C, does not mean B = C is TRUE")
            enter()


        elif choice == 12:
            print("12. If A * B = 0, it's either A = 0 and B = 0")
            print("                  or A != 0 and B != 0")
            AKhusus = np.array([[1, 2], [2, 4]])
            BKhusus = np.array([[6, -4], [-3, 2]])


            print("\nSuppose A = 0 * B = 0")
            print(np.array([[0, 0], [0, 0]])), print("\n*")
            print(np.array([[0, 0], [0, 0]]))


            print("\nBut if A != 0 * B != 0")
            print("\n",AKhusus), print("\n*"), print("\n",BKhusus)
            print("\n= "), print("\n",np.dot(AKhusus, BKhusus))


            print("\nTherefore, if A * B = 0, it's either A = 0 and B = 0")
            print("                  or A != 0 and B != 0")
            enter()

        elif choice == 13:
            break


        else:
            print("\nInvalid choice! Please choose from 1-13.")
            enter()
            continue
```

3.4 Invers Matriks

Pada pengaplikasian invers matriks kali ini, kami menggunakan bantuan *third-party library* yang cukup populer pada kalangan *python developer*, yakni *numpy*. Kami membuat sebuah fungsi dengan nama *inverse_matrix()* yang menerima *parameter* yakni elemen matriks. Berikut merupakan fungsi tersebut:

```
def inverse_matrix(matrix_el):
    A = np.array(matrix_el, dtype=np.float64)


    n = A.shape[0]
```

```
    print("n: ", n)

    if n != A.shape[1]:
        raise ValueError("Matrix must be square")

    augmented = np.column_stack((A, np.eye(n)))

    for i in range(n):
        max_element = abs(augmented[i:, i]).argmax() + i

        if max_element != i:
            augmented[[i, max_element]] = augmented[[max_element, i]]

        if np.isclose(augmented[i, i], 0):
            raise ValueError("Matrix is not invertible")

        augmented[i] = augmented[i] / augmented[i, i]

        for j in range(n):
            if i != j:
                augmented[j] -= augmented[i] * augmented[j, i]

    return augmented[:, n:]
```

Baris pertama pada fungsi tersebut bertujuan untuk melakukan konversi elemen matriks yang telah diterima sebagai parameter sebelumnya menjadi sebuah array dengan tipa data *float64*. Lalu, pada baris selanjutnya dimensi dari matriks akan dimasukkan ke dalam variabel *n* agar dapat dilakukan pengecekan apakah matriks tersebut merupakan sebuah matriks persegi/*square*, karena sesuai dengan sifat invers berikut:

Syarat Invers Matriks

Sebelum kita mempelajari cara menghitung invers matriks, penting untuk memahami syarat-syarat yang harus dipenuhi agar suatu matriks memiliki invers. Tidak semua matriks dapat diinverskan, dan mengetahui syarat-syarat ini akan membantu kita menghemat waktu dan usaha dalam perhitungan.

Berikut adalah syarat-syarat agar suatu matriks memiliki invers:

1. **Matriks harus persegi**: Hanya matriks persegi (jumlah baris sama dengan jumlah kolom) yang dapat memiliki invers. Matriks yang tidak persegi tidak akan pernah memiliki invers.

Gambar 3.4.1 - Syarat Invers Matriks - https://www.liputan6.com/feeds/read/5789115/cara-invers-matriks-dalam-aljabar-untuk-pemula-berikut-syarat-dan-metodenya?page=3

Selanjutnya, matriks identitas/*augmented* akan dibangun sesuai dengan ukuran matriks A. Matriks identitias ini bertujuan untuk nantinya dikalikan ke tiap-tiap elemen didalam matriks A. Sehingga, nantinya bentuk akhir dari matriks identitas/*augmented* yang terdapat dalam variabel *augmented* akan di *return* sebagai hasil invers dari matriks A.

Sehingga, apabila fungsi di *run* dengan matriks yang diberikan:

```
A = [
    [4, 7, 2],
    [2, 6, 1],
    [1, 5, 3]
    ]
```

Akan menghasilkan invers matriks sebagai berikut:

```
Original matrix:
  4.0000   7.0000   2.0000
  2.0000   6.0000   1.0000
  1.0000   5.0000   3.0000
Inverse matrix:
  0.5200  -0.4400  -0.2000
 -0.2000   0.4000   0.0000
  0.1600  -0.5200   0.4000
```

Gambar 3.4.2 - Hasil Invers Matriks

Tidak cukup sampai situ, untuk mem-verifikasi apakah hasil dari inverse matriks tersebut benar atau tidak, saya menggunakan bantuan *package numpy* dari *python*, lebih tepatnya method `np.linalg.inv(matrix)`:

```
print("Original matrix:")
print_matrix(A)
print("Inverse matrix:")
print_matrix(inverse)
print("Inverse matrix with numpy:")
print_matrix(np.linalg.inv(A))
```

Dengan memverifikasi apakah hasil inverse yang dihasilkan sama dengan fungsi yang telah kami buat, maka berikut hasil inverse dari kedua metode:

```
Original matrix:
  4.0000   7.0000   2.0000
  2.0000   6.0000   1.0000
  1.0000   5.0000   3.0000
Inverse matrix:
  0.5200  -0.4400  -0.2000
 -0.2000   0.4000   0.0000
  0.1600  -0.5200   0.4000
Inverse matrix with numpy:
  0.5200  -0.4400  -0.2000
 -0.2000   0.4000   0.0000
  0.1600  -0.5200   0.4000
```

Gambar 3.4.3 - Hasil Invers Matriks 2 Metode

3.5 LU Decomposition

LU Decomposition yang kami implementasikan didalam program kami cukup *straightforward*, simpel dan mudah dimengerti. Dimana fungsi dengan nama *lu_decomposition* menerima sebuah parameter, yakni sebuah matriks *A* yang nantinya akan digunakan untuk menentukan :

- Dimensi matriks(*n*),
- Matriks *L*, hasil dari perkalian tiap-tiap elemen matriks *A* dengan 0,
- Matriks *U*, hasil dari perkalian tiap-tiap elemen matriks *A* dengan 0,

```
def lu_decomposition(A):
    n = len(A)
    L = [[0]*n for _ in range(n)]
    U = [[0]*n for _ in range(n)]
    print("L: ", L)
    print("U: ", U)
```

```
    for i in range(n):
        for j in range(i, n):
            U[i][j] = A[i][j] - sum(L[i][k]*U[k][j] for k in range(i))
        for j in range(i, n):
            if i == j:
                L[i][i] = 1
            else:
                L[j][i] = (A[j][i] - sum(L[j][k]*U[k][i] for k in range(i))) / U[i][i]

    return L, U
```

Sehingga, perhitungan didalam perulangan bertujuan untuk mendapatkan nilai dari *Upper Triangular Matrix(U)* dan *Lower Triangular Matrix(L)*, dengan detail sebagai berikut:

1. Perulangan pertama, `for i in range(n):` menghitung seluruh baris dalam matriks A. Dalam tiap-tiap iterasi, bertujuan untuk menghitung baris dalam *U* dan kolom dalam *L*.

2. Menghitung *Upper Triangular Matrix(U)*, perulangan dalam yakni `for j in range(i, n)` menghitung baris ke-*i* dalam *U*, rumus yang digunakan:

$$U[i][j] = A[i][j] - \sum_{k=0}^{i-1} (L[i][k].U[k][j])$$

   Dengan rumus diatas, nantinya akan didapatkan nilai 0 untuk seluruh elemen dibawah diagonal.

3. Menghitung *Lower Triangular Matrix(L)*, perulangan dalam ke-2 yakni `for j in range(i, n)` untuk diagonal matriks (i=j):

$$L[i][i] = 1$$

   Memastikan bahwa diagonal matriks L bernilai 1. Untuk elemen non diagonal (i!=j): Agar didapatkan nilai 0 untuk elemen diatas diagonal.

$$L[j][i] = \frac{A[j][i] - \sum_{k=0}^{i-1} (L[j][k] \cdot U[k][i])}{U[i][i]}$$

4. Fungsi mengembalikan kedua matriks, yakni *L* dan *U* sebagai *return value*.

Dari langkah-langkah rumus yang dijabarkan diatas, berikut adalah contoh eksekusi program LU Decomposition dengan matriks A, yakni:

```
A = [
     [4, 7, 2],
     [2, 6, 1],
     [1, 5, 3]
     ]
```

Dan format keluaran sebagai berikut:

```
L, U = lu_decomposition(A)
print("L:")
print_matrix(L)
print("U:")
print_matrix(U)
enter()
```

Sehingga didapatkan hasil sebagai berikut:

```
LU decomposition:
L:  [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
U:  [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
L:
  1.0000   0.0000   0.0000
  0.5000   1.0000   0.0000
  0.2500   1.3000   1.0000
U:
  4.0000   7.0000   2.0000
  0.0000   2.5000   0.0000
  0.0000   0.0000   2.5000
```

Gambar 3.5.1 - Hasil LU Decomposition Matriks A (Dokumentasi penulis)

3.6 Persamaan Linear

Pengaplikasian persamaan linear kali ini, kami akan menggunakan *point-slope form* untuk mendemonstrasikan garis yang terbentuk dari *slope(m)*, *y-intercept(b)* maupun *(X1, Y1)* dari sebuah persamaan. Dimana untuk *slope* sendiri merupakan point yang akan menjadi penentu arah garis.

$$m = \frac{rise}{run}$$

Point-slope form ini dapat dijumpai dalam dua bentuk persamaan, diantara lain:

$$y = mx + b$$

Dimana dalam bentuk ini, selain *m*, *y-intercept(b)* akan menjadi penentu arah garis linear yang dibentuk, dan bentuk selanjutnya ialah:

$$y - y_1 = m(x - x_1)$$

Dalam bentuk ini, *(y1, x1)* akan menjadi indikator penentu arah garis disamping *m*.

Untuk praktiknya, kami menggunakan persamaan dengan bentuk pertama, yang masing-masing variabelnya memiliki nilai:

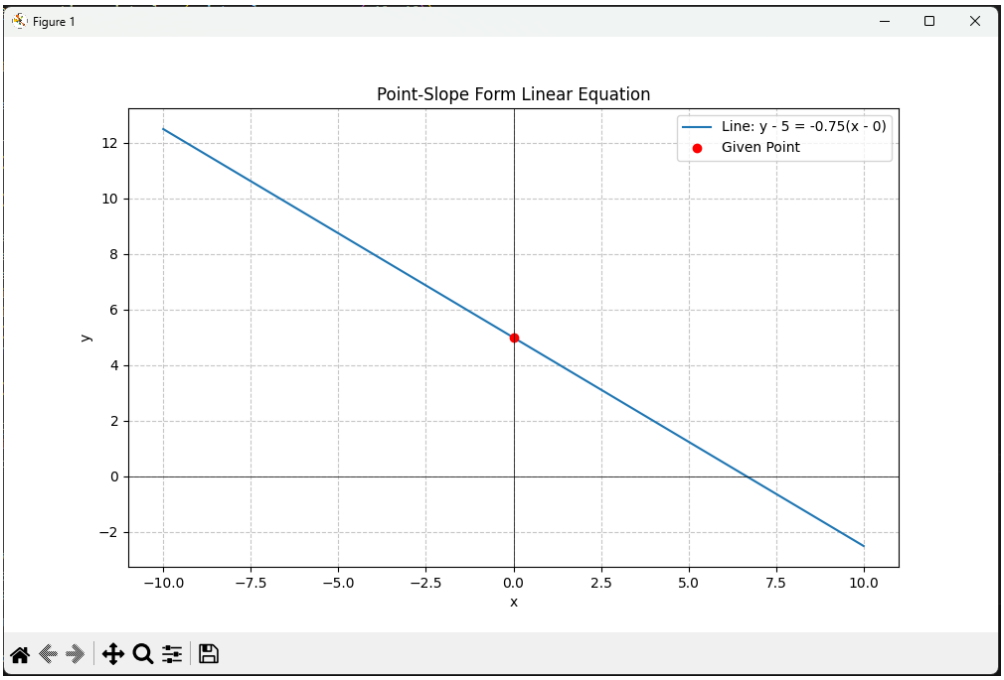
1. *y-intercept(b)*: 5, sehingga titik yang didapatkan menjadi (0, 5)
2. *m*: -¾

Variabel-variabel tersebuk dimasukkan sebagai parameter kedalam fungsi *plot_linear_equation_point_slope*(), yang juga berfungsi untuk mensimulasikan terbentuknya garis linear menggunakan bantuan *library* yang dinamakan *matplotlib*. Berikut bentuk fungsi yang telah kami buat untuk persamaan linear dengan *point-slope form*:

```
def plot_linear_equation_point_slope(point, slope, x_range=(-10, 10)):
    x0, y0 = point

    x = np.linspace(x_range[0], x_range[1], 100)

    y = slope * (x - x0) + y0

    plt.figure(figsize=(10, 6))
    plt.plot(x, y, label=f'Line: y - {y0} = {slope}(x - {x0})')

    plt.plot(x0, y0, 'ro', label='Given Point')

    plt.title('Point-Slope Form Linear Equation')
    plt.xlabel('x')
```

```
    plt.ylabel('y')
    plt.axhline(y=0, color='k', linewidth=0.5)
    plt.axvline(x=0, color='k', linewidth=0.5)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend()

    plt.show()
```

Sehingga, dengan variabel-variabel yang telah diberikan sebelumnya, garis yang akan terbentuk kurang lebih akan terlihat sebagai berikut:



Gambar 3.6.1 - Hasil Garis Linear Dengan poin-slope form (Sumber : Dokumentasi penulis)

3.7 Metode Numerik Persamaan Non Linear Dengan Metode Biseksi

Kami melakukan perhitungan Persamaan Non Linear dengan menggunakan Metode Biseksi. Metode Biseksi merupakan salah satu cara untuk melakukan perhitungan fungsi non linear f(x) = 0 dengan f merupakan fungsi kontinu dengan interval [a, b] dan f(a) dan f(b) memiliki tanda yang berlawanan. Progam ini menggunakan metode ini untuk menyelesaikan persamaan yang dimasukkan oleh pengguna. Pengguna akan perlu memasukkan persamaan, interval, dan juga batas toleransi untuk menghentikan iterasi.

Pengguna akan diminta untuk melakukan *input* fungsi pada variabel *self.function input*. Kemudian pengguna akan diminta untuk memasukkan titik interval bawah dan atas yang

kemudian disimpan pada variabel *inter1* dan *inter2*. Kemudian pengguna diminta untuk memasukkan toleransi berapa angka di belakang koma yang disimpan pada variabel *tol*. Toleransi digunakan sebagai batas iterasi yang dilakukan, jadi iterasi akan berhenti ketika nilai *c* sesuai dengan toleransi.

Fungsi *bisection1* akan melakukan perhitungan hingga ditemukannya nilai *c* sesuai dengan batas toleransi. Fungsi *bisec_func* digunakan untuk melakukan pemecahan *input* fungsi yang sebelumnya dilakukan oleh pengguna. Hasil *output* akan langsung menunjukkan akar atau nilai *c* dan juga jumlah iterasi yang telah dilakukan untuk mendapat nilai *c*.

```
class nonlinear(): # func6
    def main_non_linear(self):
        while True:
            print()
            print("Non Linear Equation Calculator")
            print("1. Quadratic Equation ( ax^2 + bx + c = 0 )")
            print("2. Exponential Equations ( e^x = 3x )")
            print("3. Trigonometric Equations ( e^x = 0 )")
            print("0. Exit")
            choice = input("> ")
            if choice == '1':
                self.quadratic()
            elif choice == '2':
                self.exponential()
            elif choice == '3':
                self.trigonometric()
            elif choice == '0':
                return
            else:
                input("Invalid choice! Please choose from 1-3.")
                enter()

    def bisection(self):
        clear()
        print("Bisection Method Calculator")
        print("Input Function (e.g x**2 - 4 / x***3 + 2)")
        self.function_input = input("> ")

        inter1 = int(input("Enter interval 1: "))
        inter2 = int(input("Enter interval 2: "))
        tol = float(input("Enter tolerance (e.g 0.00001): "))
```

```
        root = self.bisection1(self.bisec_func, inter1, inter2, tol)

        if root is not None:
            print(f'Approximate root is {root} after {self.iteration}
iteration')
            enter()
        else:
            print('Invalid Interval')
            enter()


    def bisection1(self, lin_func, inter1, inter2, tol):
        x1 = lin_func(inter1)
        x2 = lin_func(inter2)

        if x1 * x2 >= 0:
            return None
        self.iteration = 1
        while True:
            midpoint = (inter1 + inter2) / 2
            x3 = lin_func(midpoint)

            if abs(x3) < tol :
                return midpoint


            if x1 * x3 < 0:
                inter2 = midpoint
            else:
                inter1 = midpoint
            self.iteration += 1
            if abs(midpoint) - abs(x2) > tol:
                return midpoint

    def bisec_func(self, inter):
        x = inter
        return eval(self.function_input)
```

3.8 Dasar Simulasi

Untuk pengaplikasian dasar simulasi, kami memutuskan untuk menampilkan hasil perhitungan dari *Linear Congruential Generator*. Pada bagian awal, kami menjelaskan apa itu simulasi secara singkat dan menjelaskan apa korelasi dari simulasi dengan *LCG*.

```
def func8() :
    print()
    print("Simulation is a training method that demonstrates something")
    print("AS AN IMITATION form that is SIMILAR with real situation")
    print()
    print("Some simulation require RANDOM VARIABLE INPUT")
    print("to produce realistic OUTPUTS")
    print()
    enter()

    print()
    print("Random variable input can be generated with PSEUDO RANDOM NUMBER GENERATOR (PRNG)")
    print("One example of PRNG that can be used is Linear Congruential Generator (LCG)")
    print()
```

Setelah menjelaskan korelasi dari simulasi dengan LCG, kami menampilkan rumus perhitungan LCG dan meminta pengguna untuk menginputkan empat parameter utama yang menentukan output dari LCG, yaitu *start state*, *multiplier*, *increment*, *modulus*, *iteration*.

```
    print("The Formula of LGC is : ")
    print("Zi = (a * Zi-1 + c) * mod m")
    print("a = multiplier")
    print("c = increment")
    print("m = modulus")
    print()
    enter()


    print()
    print("Input start state ")
    start = int(input("> "))
    print("Input a (multiplier) ")
    a = int(input("> "))
    print("Input c (increment) ")
    c = int(input("> "))
    print("Input m (modulus) ")
```

```
    m = int(input("> "))
    print("Input iteration ")
    iteration = int(input("> "))
    print()
```

Lalu, dibuatlah *dictionary* data untuk menyimpan data hasil perhitungan dari LGC. Untuk *key dictionary* yang disimpan adalah *iteration*, *calc*, *mod*, *Zi*, dan *Ui*. *Iteration* untuk menyimpan berapa kali iterasi dilakukan, *calc* untuk menampilkan hasil perhitungan (a * Zi-1 + c), *mod* untuk menyimpan data modulus, *Zi* untuk menyimpan hasil perhitungan LGC, dan Ui untuk menyimpan hasil pembagian Zi/m. Setelah itu, dilakukan perhitungan dengan menggunakkan *for loop* dengan batas sesuai jumlah *iteration* dan setiap hasil perhitungan disimpan dalam *value dictionary* secara urut.

```
  data = {'iteration': [], 'calc': [], 'Zi': [], 'mod': [], 'Ui = Zi/m': []}

    print("a = ", a)
    print("c = ", c)
    print("m = ", m)
    print("start = ", start)
    print()


    print("Zi = (a * Zi-1 + c) * mod m")
    print("calc = (a * Zi-1 + c)")

    for i in range(iteration):
        calc = (a * start + c)
        mod = m
        zi = calc % m
        ui = zi / m

        data['iteration'].append(i+1)
        data['calc'].append(calc)
        data['mod'].append(m)
        data['Zi'].append(zi)
        data['Ui = Zi/m'].append(round(ui, 4))


        start = zi
```

Setelah hasil perhitungan *for loop* selesai, maka data *dictionary* akan ditampilkan dalam bentuk tabel sederhana. Untuk membuat kepala tabel, digunakanlah *f-string* untuk

membuat format *dynamic string*, sehingga memungkinkan untuk mengatur tata letak setiap kolom dan memberi jarak yang sesuai. Untuk membuat isi tabel, digunakan perulangan *for loop* dengan parameter *value dictionary* dari hasil perhitungan sebelumnya. Agar tabel terlihat rapi, *value dictionary* ditampilkan menggunakkan *f-string*.

Kami juga menjelaskan bahwa hasil perhitungan pada iterasi *mod* atau *mod + 1* akan menampilkan hasil yang serupa dengan iterasi pertama. Hasil perhitungan *calc*, *Zi*, dan *Ui* pada iterasi kedua akan sama dengan hasil iterasi *mod + 1* atau *mod + 2*. Ini menunjukkan bahwa setiap *pseudo random number generator* memiliki periode, yang apabila hasil perhitungan mencapai periode tersebut, hasil perhitungan akan berulang ke awal, mengulang siklus yang serupa.

```
  print(f"{'iteration':<12}{'calc':<10}{'mod':<7}{'Zi':<8}{'Ui = Zi/m':<12}")
    print("-" * 40)
    for i, calc, mod, zi, ui in zip(data['iteration'], data['calc'], data['mod'],
data['Zi'],data['Ui = Zi/m']):
        print(f"{i:<12}{calc:<10}{mod:<7}{zi:<8}{ui:<12}")
    print()
    enter()


    print()
    print("if you look carefully, around iteration", mod , "or", mod + 1, ", the result
of Zi and Ui with iteration 1 are similar")
    print("The calc, Zi, Ui of iteration number 2 are similar with iteration" , mod + 1,
"or" , mod + 2,)
    print("You might recognize other pattern as well")


    print()
    print("PNRG has a certain period where if it has reached that period,")
    print("the resulting sequence of numbers will repeat from the beginning, starting
the previous pattern")
    enter()
```

3.9 Iterasi Jacobi

Untuk memperindah program, kami menghubungkan dua fungsi ke dalam satu fungsi yang sama, yaitu fungsi iterasi jacobi dan fungsi iterasi *gauss-seidel*. Fungsi tersebut dihubungkan dalam suatu fungsi yaitu `def func9()`. Pada bagian awal fungsi, kami

menjelaskan bahwa metode iterasi adalah salah satu cara untuk menyelesaikan persamaan linear kompleks dengan perulangan untuk mencari jawaban terdekat.

Kami juga menjelaskan bahwa persamaan linear tersebut harus dikonversikan menjadi dua matriks, yaitu matriks *A* yang berfungsi untuk menyimpan koefisien dari suatu persamaan linear dengan ukuran *n x n*, dan matriks *b* yang berfungsi untuk menyimpan konstanta pada sisi kanan dari suatu persamaan linear.

```
def func9() :
    print()
    print("Iterative method is one of the ways to solve complex linear equations")
    print()
    print("As its name suggests, the iterative method finds the solution by using
REPEATED ITERATIONS ")
    print("until the solution CONVERGES with desired accuracy")
    print()
    enter()



    print("To solve the complex linear equation, we have to convert the linear equation
into 2 matrixes")
    print("Matrix A stores the coefficients of the variables")
    print()
    A = [
        [4, 1, 1],
        [4, -8, 1],
        [-2, 1, 5],
        ]
    print("A: ", A)
    print()


    print("Matrix b stores the constants on the right side of the equations")
    b = [7, -21, 15]
    print()
    print("b: ",b)
    print()
    enter()
```

Agar lebih efisien, pengimplementasian iterasi jacobi dan *gauss-seidel* menggunakkan matriks *A* dan matriks *b* yang sama, sehingga menghindari duplikasi data dan dapat dilakukan

perbandingan pada kedua iterasi. Pada contoh diatas, matriks *A* dan *b* diperoleh dari konversi persamaan linear sebagai berikut :

$$4x - y + z = 7$$
$$4x - 8y + z = -21$$
$$-2x + y + 5z = 15$$

Setelah selesai, matriks tersebut dikalkulasi dengan fungsi iterasi jacobi. Fungsi iterasi jacobi membutuhkan lima parameter, yaitu matriks *A*, matriks *b*, variabel *x0* sebagai pengali awal, variabel *tolerance* untuk menentukan batas berhentinya suatu iterasi, dan variabel *max_iteration* untuk menentukan batas iterasi dapat dieksekusi.

```
def jacobi(A,b, x0=None, tolerance=1e-10,max_iterations = 1000):
    A = np.array(A, dtype=float)
    b = np.array(b, dtype=float)

    n = len(A)

    if x0 is None:
        x = np.zeros_like(b)
    else:
        x = np.array(x0, dtype=float)

    for i in range(max_iterations):
        x_new = np.zeros_like(x)

        for j in range(n):
            sum1 = np.dot(A[j, :], x) - A[j, j] * x[j]
            x_new[j] = (b[j] - sum1) / A[j, j]

        if np.linalg.norm(x_new - x, ord=np.inf) < tolerance:
            return x_new, i + 1
        x = x_new
    raise ValueError(f"Method did not converge after {max_iterations} iterations")
```

Setelah dilakukan inisialisasi variabel, selanjutnya iterasi terluar akan melakukan eksekusi dengan tujuan untuk menyempurnakan solusi hingga salah satu kondisi terpenuhi:

1. Jawaban yang ada telah berhasil terkonvergensi, dengan toleransi nilai yaitu 1e-10.

2. Iterasi telah mencapai batas *max_iterations*, yaitu 1000.

Variabel *tolerance* dan *max_iterations* dapat diubah sesuai keinginan pengguna. Kami memilih nilai tersebut agar hasil yang dihasilkan mendekati solusi sebenarnya dengan akurasi yang tinggi.

Langkah pertama yang dilakukan adalah memberi nilai nol pada matriks *x* sebagai pengali dengan mengambil nilai dari matriks *x0*, ukurannya disesuaikan dengan panjang dari matriks *b*. Lalu proses penghitungan dimulai dengan mencari nilai *x_new* dengan cara menghitung jumlah perkalian antara elemen matriks koefisien A[j , :] dengan nilai *x* saat ini, yang bernilai nol, kecuali elemen diagonal utama, A[j,j]·x[j]. Yang merupakan implementasi rumus dari

$$x_i^{(k)} = \frac{1}{a_{ii}}\left[-\sum_{\substack{j=1 \\ j\neq i}}^{n}\left(a_{ij}x_j^{(k-1)}\right) + b_i\right]$$

Nilai hasil kemudian digunakan untuk memperbarui nilai *x_new[j]* dengan rumus

$$x_{\text{new}}[j] = \frac{b[j] - \text{sum1}}{A[j,j]}$$

Setelah itu, fungsi akan memeriksa apakah *x_new* telah mendekati jawaban sebelumnya (*x*) dengan bantuan fungsi *numpy.linalg.norm* untuk mengukur perbedaan antar nilai. Jika perbedaan ini lebih kecil dari nilai *tolerance*, atau dalam kasus ini disebut konvergen, maka fungsi akan berhenti dan mengembalikan nilai *x_new* dan nilai iterasinya (untuk memberitahu iterasi ke berapa nilai ditemukan). Sebaliknya, jika belum konvergen, maka fungsi akan melakukan iterasi sampai nilai *x_new* dan *x* konvergen.

Setelah fungsi `def func9()` mengembalikan nilai *x_new* dan iterasi, program akan menampilkan nilai *x_new* dalam bentuk matriks, dan jumlah iterasi sebagai angka. Program juga memberikan pembuktian dengan mengalikan matriks *A* dengan solusi yang ditemukan. Selain itu, program juga mengkalkulasi dan menampilkan sistem persamaan linear dengan metode *gauss-seidel*, yang akan dijelaskan secara rinci pada sub bab selanjutnya.

```
print()
    print("After some iterations, here's the result of the calculation using Jacobi
iterative and Gauss-Seidel iterative")
    print()
    try:
        solutionJacobi, iterationJacobi = jacobi(A, b)
```

```
        solutionGauss, iterationGauss = gauss_seidel(A, b)
        print("Solution with Jacobi:", solutionJacobi)
        print("Iterations with Jacobi:", iterationJacobi)
        print()

        print("Solution with Gauss-Seidel:", solutionGauss)
        print("Iterations with Gauss-Seidel:", iterationGauss)

        print("\nVerification:")
        print("A * Solution Jacobi\t\t =", np.dot(A, solutionJacobi))
        print("A * Solution Gauss-Seidel\t =", np.dot(A, solutionGauss))

        print("B\t\t\t\t= ", b)
        print()
        print("Usually, Jacobi iterative requires more iteration than Gauss-Seidel
iterative")
        print()
        print("The results are close to the expected values, differing only by small
amounts ")
        print("If we want a MORE ACCURATE solution, we can decrease the TOLERANCE
VALUE")
        print()
```

Dari sistem persamaan linear yang disebutkan sebelumnya, berikut adalah hasil perhitungan menggunakkan iterasi jacobi dan pembuktiannya

```
A:  [[4, 1, 1], [4, -8, 1], [-2, 1, 5]]

Matrix b stores the constants on the right side of the equations

b:  [7, -21, 15]

Press Enter to continue

After some iterations, here's the result of the calculation using Jacobi iterative and Gauss-Seidel iterative

Solution with Jacobi: [0.34343434 3.11111111 2.51515152]
Iterations with Jacobi: 37

Solution with Gauss-Seidel: [0.34343434 3.11111111 2.51515152]
Iterations with Gauss-Seidel: 14

Verification:
A * Solution Jacobi             = [  7. -21.  15.]
A * Solution Gauss-Seidel       = [  7. -21.  15.]
B                               =  [7, -21, 15]

Usually, Jacobi iterative requires more iteration than Gauss-Seidel iterative

The results are close to the expected values, differing only by small amounts
If we want a MORE ACCURATE solution, we can decrease the TOLERANCE VALUE

Press Enter to continue
```

Gambar 3.9.1 - Hasil Iterasi Jacobi (Sumber : Dokumentasi penulis)

3.10 Iterasi *Gauss-Seidel*

Pada implementasi iterasi *gauss-seidel* kali ini, kami akan mengalikan matriks *A* yang berfungsi sebagai koefisien dari matriks berukuran *n x n* dengan *b* atau bisa disebut juga sebagai konstanta:

$$A_x = b$$

Diperlukan juga sebuah variabel yang akan bekerja sebagai pengali awal untuk iterasi ini. Variabel ini dinamakan *x0* yang biasanya diberikan nilai 0 atau *None* sebagai nilai awal. Lalu, terdapat juga nilai *tolerance*, yakni *convergence threshold*(batas konvergensi), yang mana akan menentukan berhentinya sebuah iterasi apabila syarat berikut terpenuhi:

$$\left(\left|\left|x_{new} - x_{old}\right|\right|\right)$$

Terdapat variabel *max_iterations* yang menentukan nilai maksimum dari sebuah iterasi yang akan dieksekusi. Adapun 2 variabel terakhir yakni *n* dan *x*, dimana variabel *n* yang bernilai sebagai panjang/*length* dari *A*, lalu *x* sebagai matriks bernilai 0.

```python
def gauss_seidel(A, b, x0=None, tolerance=1e-10, max_iterations=1000):
    A = np.array(A, dtype=float)
    b = np.array(b, dtype=float)

    n = len(A)

    if x0 is None:
        x = np.zeros_like(b)
    else:
        x = np.array(x0, dtype=float)

    for iteration in range(max_iterations):
        x_old = x.copy()

        for i in range(n):
            sigma = sum(A[i][j] * x[j] for j in range(n) if j != i)

            x[i] = (b[i] - sigma) / A[i][i]

        if np.linalg.norm(x - x_old) < tolerance:
            return x, iteration + 1
```

```
    raise ValueError(f"Method did not converge after {max_iterations} iterations")
```

Setelah dilakukannya inisialisasi variabel-variabel yang dibutuhkan di atas, selanjutnya iterasi terluar `for iteration in range(max_iterations):` akan dieksekusi dengan tujuan untuk menyempurnakan solusi hingga dua kondisi dibawah ini terpenuhi:

1. Solusi yang ada telah berhasil terkonvergensi (perubahan antara tiap-tiap iterasi cukup kecil, seperti 0.00001).
2. Iterasi telah mencapai batas *max_iterations*.

Selanjutnya, iterasi dalam `for i in range(n):` akan memperbarui tiap-tiap komponen dari *Xi* dari variabel *x* menggunakan rumus gauss-seidel:

$$x_i = \frac{b_i - \sum_{j \neq i} A_{ij} x_j}{A_{ii}}$$

Berikut langkah-langkah mendetail dari eksekusi iterasi dalam menggunakan rumus diatas:

1. Menghitung penjumlahan dengan rumus:

$$\sigma = \sum_{j \neq i} A_{ij} x_j$$

yang mana:

- Merepresentasikan kontribusi dari tiap-tiap variabel kepada iterasi ke *i-th* dari persamaan.
- Penjumlahan ini menggunakan nilai terupdate dari *Xj*

2. Update *Xi*:

$$x_i = \frac{b_i - \sigma}{A_{ii}}$$

Membagi *bi* dengan koefisien diagonal dari *Aii*.

3. Setelah meng*update* semua komponen dari variabel *Xi*, fungsi diatas akan melakukan pengecekan apakah solusi yang didapat telah terkonvergensi dengan kondisi:

$$\left|\left|x_{new} - x_{old}\right|\right| < tolerance$$

Pengecekan konvergensi ini menggunakan bantuan *library numpy*, lebih tepatnya *euclidean norm* (`np.linalg.norm`).

Apabila perubahan lebih kecil dari *tolerance*, maka fungsi tersebut akan me-*return* solusi yang ada beserta variabel *x* sebagai jumlah iterasi yang telah dilakukan.

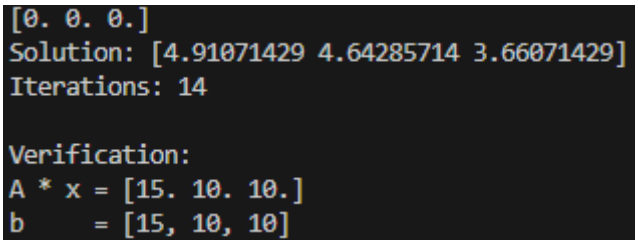
Berikut adalah matriks *A* dan *b* yang akan dimasukkan sebagai *parameter* pada fungsi `gauss_seidel` diatas:

```
A = [
    [4, -1, 0],
    [-1, 4, -1],
    [0, -1, 4]
    ]
 b = [15, 10, 10]
```

Disamping itu, kami juga melakukan verifikasi dari solusi yang didapatkan dari fungsi tersebut menggunakan `np.dot(A, solution)` dan juga menampilkan jumlah iterasi yang telah dilakukan:

```
try:
   solution, iterations = gauss_seidel(A, b)

   print("Solution:", solution)
   print("Iterations:", iterations)

               print("\nVerification:")
               print("A * x =", np.dot(A, solution))
               print("b     =", b)
               enter()

           except ValueError as e:
               print(e)
               enter()
```

Sehingga, output yang didapatkan ialah:



Gambar 3.10.1 - Hasil Iterasi Gauss-Seidel (Sumber : Dokumentasi penulis)

Dengan output seperti gambar diatas, maka dapat dikatakan solusi yang kami terima dari fungsi `gauss_seidel` sudah benar, dan juga iterasi yang dilakukan tidak terlalu jauh dan cukup baik dari segi performa.

3. 11 Monte Carlo

Monte Carlo kami gunakan untuk melakukan perhitungan kemungkinan sebuah kemenangan dalam Casino Roulette. Monte Carlo menggunakan nilai acak yang dihasilkan terlebih dahulu di awal yang kemudian digunakan untuk memperhitungkan kemungkinan dari hasil nilai tersebut. Dalam program ini, nilai acak yang dihasilkan yaitu nilai dari tiap kemenangan roulette dengan batasan dari 0 hingga 36, berdasarkan angka roulette yang ada pada casino roulette eropa. Pengguna hanya perlu memasukkan pilihannya dari pilihan *red*, *green*, *black*, *even*, dan *odd*. Pengguna juga akan diminta untuk memasukkan jumlah berapa kali simulasi atau iterasi. Hasil dari perputaran roulette akan muncul dan ketika dijumlah, akan sesuai dengan jumlah iterasi atau simulasi yang sebelumnya dimasukkan.

```
class MonteCarlo():
    def monte_carlo_main(self):
        while True:
            clear()
            print("Monte Carlo Simulation Calculator\n")
            print("Monte Carlo is a simulation technique used to approximate the
probability of an event\n")
            print("by running a large number of simulations and then analyzing the
results\n")
            print("This Program will implement Monte Carlo \
Simulation to calculate the probability of winning a roulette\n")
```

```
            print("1. Roulette Simulation")
            print("0. Exit")
            choice = input("> ")
            if choice == '1':
                while True:
                    clear()
                    print("Roulette Simulation Calculator")
                    print("Bet options: red, green, black, even, odd")
                    bet_option = input("Enter your bet option: ").strip().lower()
                    while bet_option not in {"red", "green", "black", "even", "odd"}:
                        print("Invalid option. Please choose from: red, green, black,
even, odd.")
                        bet_option = input("Enter your bet option: ").strip().lower()

                    try:
                        iterations = int(input("Enter the number of iterations for the
simulation: "))
                        if iterations <= 0:
                            raise ValueError("Number of iterations must be greater than
0.")
                    except ValueError as e:
                        print(f"Invalid input: {e}")
                        enter()
                    # Run simulation
                    win_percentage, tot_red, tot_black, tot_green, tot_even, \
                tot_odd = self.simulate_roulette(bet_option, iterations)

                    # Display results
                    print(f"Bet option\t\t: {bet_option.capitalize()}")
                    print(f"Number of iterations\t: {iterations}")
                    print(f"Total Red Spins\t\t: {tot_red}")
                    print(f"Total Black Spins\t: {tot_black}")
                    print(f"Total Green Spins\t: {tot_green}")
                    print(f"Total Even Spins\t: {tot_even}")
                    print(f"Total Odd Spins\t\t: {tot_odd}")
                    print(f"Win percentage\t\t: {win_percentage:.4f}%")
                    print("Do you want to run another simulation? (y/n)")
                    answer = input("> ")
                    if answer.lower() == "y":
                        continue
                    else:
                        return
            elif choice == '0':
```

```
                return
            else:
                print("Invalid choice! Please choose from 1-3.")
                enter()


    def simulate_roulette(self, bet_option, iterations):
        # Define roulette wheel segments
        red_numbers = {1, 3, 5, 7, 9, 12, 14, 16, 18, 19, 21, 23, 25, 27, 30, 32, 34, 36}
        black_numbers = {2, 4, 6, 8, 10, 11, 13, 15, 17, 20, 22, 24, 26, 28, 29, 31, 33,
35}
        green_numbers = {0}

        wins = 0
        tot_red, tot_black, tot_green, tot_even, tot_odd = 0, 0, 0, 0, 0

        for _ in range(iterations):
            # Simulate a spin of the roulette wheel
            spin_result = random.randint(0, 36)

            # Check if the spin result matches the bet
            if bet_option == "red" and spin_result in red_numbers:
                wins += 1
            elif bet_option == "black" and spin_result in black_numbers:
                wins += 1
            elif bet_option == "green" and spin_result in green_numbers:
                wins += 1
            elif bet_option == "even" and spin_result != 0 and spin_result % 2 == 0:
                wins += 1
            elif bet_option == "odd" and spin_result % 2 == 1:
                wins += 1


            if spin_result in red_numbers:
                tot_red += 1
            elif spin_result in black_numbers:
                tot_black += 1
            elif spin_result in green_numbers:
                tot_green += 1

            if spin_result % 2 == 0:
                tot_even += 1
            elif spin_result % 2 == 1:
```

```
                tot_odd += 1

        win_percentage = (wins / iterations) * 100
        return win_percentage, tot_red, tot_black, tot_green, tot_even, tot_odd
```

```
Enter your bet option: green
Enter the number of iterations for the simulation: 1000000
Bet option              : Green
Number of iterations    : 1000000
Total Red Spins         : 485411
Total Black Spins       : 487355
Total Green Spins       : 27234
Total Even Spins        : 514039
Total Odd Spins         : 485961
Win percentage          : 2.7234%
```

Dalam contoh ini, kami memasukkan green sebagai pilihan kemenangan dan 1 juta kali jumlah iterasi atau simulasi. Hasilnya, green memiliki persentase kemenangan 2.7234% dari jumlah putaran green kemudian dibagi dengan jumlah iterasi.

Ketika menjumlah total dari red, black, dan green, maka akan mendapat jumlah 1 juta dan juga ketika menjumlah jumlah putaran even dan odd, maka akan mendapat jumlah 1 juta.

3. 12 Rantai Markov

Rantai Markov merupakan salah satu simulasi yang menggunakan probabilitas. Berbeda dengan Monte Carlo yang menggunakan angka acak, Rantai Markov menggunakan initial state berdasarkan nilai yang ada pada dunia nyata. Hasil pada masa yang akan datang juga ditentukan berdasarkan hasil yang ada pada sekarang.

Program ini menggunakan Rantai Markov untuk melakukan prediksi cuaca. Dengan dua *initial state* yaitu Cerah dan juga Hujan. Pengguna akan diminta untuk memasukkan matriks transisi dalam bentuk bilangan desimal dan juga memasukkan probabilitas *initial state*.

```
def markov() :
    while True:
        clear()
        print("Markov Chain Calculator")
        print("1. Markov Chain")
        print("0. Exit")
        choice = input("> ")
```

```
        if choice == '1':
            clear()
            print("Markov Chain Weather Prediction")
            print("Enter transition matrix (e.g 0.3):")
            print("Sunny -> Sunny: a")
            print("Sunny -> Rainy: b")
            print("Rainy -> Sunny: c")
            print("Rainy -> Rainy: d")
            a = float(input("Enter a: "))
            b = float(input("Enter b: "))
            c = float(input("Enter c: "))
            d = float(input("Enter d: "))

            print("Enter starting probabilities (e.g 0.4):")
            e = float(input("Sunny : "))
            f = float(input("Rainy : "))
            iteration = int(input("Enter total iteration: "))
            transition_matrix = {
                'Sunny': {'Sunny': a, 'Rainy': b},
                'Rainy': {'Sunny': c, 'Rainy': d}
            }
            starting_probabilities = {'Sunny': e, 'Rainy': f}

            # Choose the starting state randomly based on starting probabilities
            current_state = random.choices(
                population=list(starting_probabilities.keys()),
                weights=list(starting_probabilities.values())
            )[0]

            # Generate a sequence of states using the transition matrix
            num_iterations = iteration
            states_sequence = [current_state]

            for _ in range(num_iterations):
                next_state = random.choices(
                    population=list(transition_matrix[current_state].keys()),
                    weights=list(transition_matrix[current_state].values())
                )[0]
                states_sequence.append(next_state)
                current_state = next_state

            print(states_sequence)
            input("Press Enter to continue...")
```

```
        elif choice == '0':
            break
        else:
            print("Invalid choice. Please try again.")
            input("Press Enter to continue...")
```

DAFTAR PUSTAKA

Nasution, Mahyuddin & Hidayat, Rahmat & Syah, B R. (2022). *Computer Science*. 10.48550/arXiv.2207.07901.

Baron, Michael. 2013. *Probability And Statistics For Computer Science.* Boca Raton: CRC Press.

Belete, Eshetu & Gemechu, Tekle. (2022). *Comparative Study of Some Iterative Methods For Solving Poisson Equations*. 10.21203/rs.3.rs-1572658/v1.

Ihsan, Hasyam, Wahyuni, Maya Sari, Waode, Yully Sofyah. 2024. *Penerapan Metode Iterasi Jacobi dan Gauss-Seidel dalam Menyelesaikan Sistem Persamaan Linear Kompleks*. Makassar : Universitas Negeri Makassar.

Johnston, Nathaniel. 2021. *Introduction to Linear and Matrix Algebra*. Cham : Springer.

Markov Chain. (2013, June 30). https://socs.binus.ac.id/2013/06/30/markov-chain/
How to build and visualise a Monte Carlo simulation with Python and Plotly Shedload Of Code. (2023, January 6). Shedload of Code.

GeeksforGeeks.16 Januari 2022. *Plot mathematical expressions in Python using Matplotlib*. https://www.geeksforgeeks.org/plot-mathematical-expressions-in-python-using-matplotlib/, diakses pada tanggal 13 Desember 2024, pukul 18.37.

Math Is Fun. (n.d.). *Line equation: Point-slope*. https://www.mathsisfun.com/algebra/line-equation-point-slope.html, diakses pada tanggal 13 Desember, pukul 19.14.

Shedload of Code. 7 Januari 2023. *How to build and visualise a Monte Carlo simulation with Python and Plotly*. https://www.shedloadofcode.com/blog/how-to-build-and-

visualise-a-monte-carlo-simulation-with-python-and-plotly, diakses pada tanggal 13 Desember, pukul 19.53.

Wikipedia. 25 September 2024. *Gauss–Seidel method*. https://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel_method, diakses pada tanggal 14 Desember, pukul 10.35

iMathEQ. (n.d.). *iMathEQ Math Equation Editor*. https://www.imatheq.com/imatheq/com/imatheq/math-equation-editor.html, diakses pada tanggal 15 Desember, pukul 13.32.

Study Session. 9 Februari 2021. *Monte Carlo simulation explained*. Diakses dari https://www.youtube.com/watch?v=qd4jXumIy9E, diakses pada tanggal 15 Desember, pukul 16.13.

LAMPIRAN