Perbandingan Solusi Sistem Persamaan Linear Menggunakan Metode Iterasi *Jacobi* dan Metode Iterasi *Gauss-Seidel* dengan Menggunakan Komputasi *Python*

Prima Yanuaristya[1], Nurul Imamah Ah[2], Chusnul Khotimah Galatea[3]

[1]Universitas Muhammadiyah Jember
primayanuaristya@gmail.com

[2]Universitas Muhammadiyah Jember
nurulimamah@unmuhjember.ac.id

[3] Universitas Muhammadiyah Jember
chusnulkhotimah@unmuhjember.ac.id

Abstrak

Sistem persamaan linier mudah diselesaikan melalui analitik namun bisa juga menggunakan metode numerik. Masalah pada penelitian ini ialah bagaimana efisiensi mencari solusi sistem persamaan linier(SPL) menggunakan metode iterasi Gauss-Seidel dan metode iterasi Jacobi melalui komputasi python. Tujuan dari penelitian ini ialah mengetahui efisiensi mencari solusi sistem persamaan linier menggunakan metode iterasi Gauss-Seidel dan metode iterasi Jacobi melalui komputasi python.

Jenis penelitian ini ialah penelitian kualitatif. Metode yang digunakan adalah studi pustaka. Penelitian studi pustaka membatasi hingga bahan atau data koleksi perpustakaan saja sehingga tanpa perlu melakukan riset lapangan. Data yang digunakan pada penelitian ini meliputi persamaan linier, sistem persamaan linier, matriks dan data lainnya yang sesuai.

Solusi sistem persamaan linier(SPL) dengan mengubah kedalam bentuk Matriks berordo 4×4. Beberapa langkah yang dilakukan ialah menghitung solusi sistem persamaan linier (SPL) menggunakan metode iterasi Jacobi dan metode iterasi Gauss-Seidel. Dapat disimpulkan bahwa perbandingan solusi sistem persamaan linier (SPL) menggunakan metode iterasi Jacobi dan metode iterasi Gauss-Seidel waktu pemrosesan komputasi python Iterasi Gauss-Seidel lebih efisien dibandingkan dengan metode Iterasi Jacobi.

Kata Kunci: Metode Iterasi *Jacobi*, Metode Iterasi *Gauss-Seidel*, Komputasi *Python*.

Abstract

Systems of linear equations are easily solved through analytics but can also be solved using numerical methods. The problem in this research is how to efficiently find a solution to a system of linear equations using the Gauss-Seidel iteration method and the Jacobi iteration method through python computation. The purpose of this study is to determine the efficiency of finding a solution to a system of linear equations using the Gauss-Seidel iteration method and the Jacobi iteration method through python computation.

This type of research is qualitative research. The method used is literature study. Literature study research is limited to materials or library collection data, so there is no need to conduct field research. The data used in this study include linear equations, systems of linear equations, matrices and other appropriate data.

Solution to a system of linear equations by changing it into a 4×4 matrix form. Several steps were taken to calculate the solution of the system of linear equations using the Jacobi iteration method and the Gauss-Seidel iteration method. It can be

concluded that the comparison of solutions of systems of linear equations using the Jacobi iteration method and the Gauss-Seidel iteration method python computing processing time Gauss-Seidel iteration is more efficient than the Jacobi iteration method.
Keywords: Jacobi Iteration Method, Gauss-Seidel Iteration Method, Python Computing.

PENDAHULUAN

Sistem persamaan linier(SPL) mudah diselesaikan melalui analitik namun bisa juga menggunakan metode numerik. Sistem persamaan linier(SPL) dapat diselesaikan dengan operasi matriks menggunakan metode eliminasi Gauss dan Gauss-Jordan atau biasa lebih dikenal dengan Operasi Baris Elementer(OBE). Menurut [1] Rahmi dan Mulia Suryan Sistem persamaan linier adalah kumpulan persamaan linier yang memiliki variabel yang sama. Materi ini merupakan salah satu materi dari aljabar linier yang dipelajari dalam matematika. Materi ini digunakan dalam berbagai bidang dan dalam kehidupan sehari-hari, misalnya menentukan laba maksimum, perkiraan keuntungan, dan lain sebagainya. Ditingkat SMA kita telah belajar sistem persamaan linear(SPL) dengan tiga variabel sehingga materi ini sudah tidak asing namun pada penelitian ini lebih ditingkatkan menjadi empat hingga lima variabel. Sistem persamaan linear(SPL) secara garis besar bisa diselesaikan dengan cara langsung menggunakan konsep aljabar seperti eliminasi, substitusi dan campuran keduanya yang menghasilkan nilai eksak sedangkan penyelesaian secara tidak langsung atau biasa dikenal dengan nama iterasi seperti metode iterasi *Gauss-Seidel*, metode iterasi *Jacobi* dan metode iterasi lainnya yang menghasilkan nilai pendekatan atau perkiraan.

Python adalah bahasa pemrograman berorientasi objek yang efisien untuk mengembangkan aplikasi di bidang sains, teknik, dan bidang lainnya. Script Python tidak perlu dikompilasi menjadi kode mesin karena script Python dapat dijalankan cukup dengan bantuan interpreter. Menurut Rodiah [4] Keuntungan dari script yang bisa dijalankan dengan interpreter adalah ia dapat diuji dan didebug dengan cepat, sehingga programmer bisa lebih berkonsentrasi pada algoritma dibalik script yang sedang dibangunnya. Software python memiliki beberapa kelebihan yaitu free(gratis), open source, dan tidak perlu di compile sehingga bisa lebih cepat dalam mengetahui hasil ataupun perubahan ketika menulis dan menjalankan script.

Penyelesaian sistem persamaan linear (SPL) menggunakan metode iterasi Jacobi dan metode iterasi Gauss-Seidel pernah dibahas oleh peneliti sebelumnya diantaranya pada tahun 2016 dengan judul Perbandingan Metode Iterasi Jacobi dan Iterasi Gauss-Seidel dalam Penyelesaian Sistem Persamaan Linier dengan Menggunakan Simulasi Komputasi Oleh Shella Niyyaka disimpulkan Pemroresan hasil output iterasi Gauss-Seidel lebih sedikit dibandingkan iterasi Jacobi, selanjutnya Pada tahun 2015 dengan judul Penerapan Skema Jacobi dan Gauss Seidel Pada Penyelesaian Numerik Persamaan oleh Trija Fayeldi disimpulkan bahwa suatu SPL $Ax = b$ dapat diselesaikan secara iteratif dengan Metode Jacobi ataupun Metode Gauss-Seidel, dalam hal ini Metode Gauss-Seidel jauh lebih cepat konvergen dibandingkan dengan Metode Jacobi. Pada penelitian terdahulu

mengenai Perbandingan Metode Iterasi Jacobi dan Iterasi Gauss-Seidel disimpulkan bahwa jumlah iterasi Gauss-Seidel lebih sedikit dibandingkan iterasi Jacobi. Penelitian mengenai Perbandingan Metode Iterasi Jacobi dan metode Iterasi Gauss-Seidel menggunakan komputasi Python masih jarang, maka dari itu peneliti menggunakan komputasi Python.

Berdasarkan penelitian tersebut penulis tertarik untuk mengulas dengan membandingkan dua metode yaitu Perbandingan Metode Iterasi Jacobi dan metode Iterasi Gauss-Seidel namun menggunakan komputasi Phyton, oleh karena itu penulis tertarik mengambil judul "Perbandingan Solusi Sistem Persamaan Linear Menggunakan Metode Iterasi Jacobi dan Metode Iterasi Gauss-Seidel dengan Menggunakan Komputasi Python".

Fokus penelitian ini diantaranya untuk mengetahui perbandingan solusi sistem persamaan linier (SPL) menggunakan metode Iterasi *Gauss-Seidel* dan menggunakan metode Iterasi *Jacobi* serta efisiensi melalui komputasi *Pyhton* menggunakan modul *Numpy*. Tujuan penelitian ini untuk mengetahui efisiensi mencari solusi sistem persamaan linier menggunakan metode iterasi Gauss Seidel dan metode iterasi Jacobi melalui komputasi python.

BAHAN DAN METODE

Jenis penelitian ini adalah deskriptif kualitatif. Pendekatan yang digunakan adalah pendekatan kualitatif dengan metode kepustakaan(Library Research). Menurut Sugiyono [2] mengatakan bahwa studi kepustakaan berkaitan dengan kajian secara teori melalui referensi-referensi terkait dengan nilai, budaya, dan norma yang berkembang pada situasi sosial yang diteliti. Sedangkan menurut Zed [3] adalah serangkaian kegiatan penelitian yang berkenaan dengan metode pengumpulan data pustaka, kemudian membaca dan mencatat serta mengolah bahan penelitian tersebut.

Data yang digunakan penulis dalam penelitian ini adalah data-data yang meliputi persamaan linier, sistem persamaan linier, matriks dan data-data lain yang sesuai.

Sumber data dalam penelitian ini diperoleh dari buku-buku antara lain Rodiah dengan judul Modul Kuliah Komputasi dengan Python, Seymour dan Marc dengan judul Buku Aljabar Linier dan sumber lainnya yang relevan.

Teknik menganalisis data penulis dalam melakukan persiapan dengan menghitung menggunakan metode iterasi *Jacobi* serta menggunakan metode iterasi *Gauss-seidel* selanjutnya menyimpulkan mana yang lebih efisian apakah metode iterasi *Gauss-Seidel* atau metode iterasi Jacobi dalam menemukan solusi persamaan linier(SPL)

HASIL DAN PEMBAHASAN

Sistem persamaan linier(SPL) empat variabel dibuat dalam bentuk matriks berordo 4×4, Matriks tersebut dianalisis menggunakan determinan untuk mengetahui apakah memiliki solusi tunggal. Selanjutnya matriks tersebut di analisis secara eksak menggunakan Operasi Baris Elementer(OBE). Menghitung secara numerik menggunakan metode iterasi *Jacobi* dan metode iterasi *Gauss-Seidel* selanjutnya dibandingkan mana yang lebih efisien dari iterasi tersebut metode iterasi *Jacobi* atau metode iterasi *Gauss-Seidel*.

Pada Sistem persamaan linier berikut dibentuk kedalam bentuk Matriks berordo 4×4 sebagai berikut

$$8x_1 + x_2 + 3x_3 - 3x_4 = 49$$

$$3x_1 - 7x_2 + 2x_3 - x_4 = 27$$

$$2x_1 + 3x_2 + 8x_3 - x_4 = 25$$

$$3x_1 + 2x_2 + x_3 + 7x_4 = -16$$

Sistem persamaan linier(SPL) diatas diubah kedalam bentuk matriks menjadi bentuk $Ax = B$ seperti dibawah ini

$$\begin{pmatrix} 8 & 1 & 3 & -3 \\ 3 & -7 & 2 & -1 \\ 2 & 3 & 8 & -1 \\ 3 & 2 & 1 & 7 \end{pmatrix} \times \begin{pmatrix} x1 \\ x2 \\ x3 \\ x4 \end{pmatrix} = \begin{pmatrix} 49 \\ 27 \\ 25 \\ -16 \end{pmatrix}$$

Menentukan jumlah solusi pada matriks A sebagai berikut

$$A = \begin{pmatrix} 8 & 1 & 3 & -3 \\ 3 & -7 & 2 & -1 \\ 2 & 3 & 8 & -1 \\ 3 & 2 & 1 & 7 \end{pmatrix}$$

banyaknya solusi sistem persamaan linier ada 3 kemungkinan yakni memiliki tidak memiliki solusi, tepat satu solusi, dan tak hingga(banyak) solusi. Untuk mengetahui jumlah solusi maka bisa menggunakan determinan. Untuk menentukan determinan ada beberapa metode seperti metode Aturan *Sarrus*,

Metode Minor-Kofaktor, atau metode lainnya. Pada penelitian ini dalam menghitung determinan menggunakan operasi baris elementer(OBE) sehingga menghasilkan Determinan matriks A sebagai berikut

$$\begin{vmatrix} 8 & 1 & 3 & -3 \\ 3 & -7 & 2 & -1 \\ 2 & 3 & 8 & -1 \\ 3 & 2 & 1 & 7 \end{vmatrix} = \begin{vmatrix} 8 & 1 & 3 & -3 \\ 0 & \frac{-59}{8} & \frac{7}{8} & -1 \\ 0 & 0 & \frac{447}{59} & -1 \\ 0 & 0 & 0 & 7 \end{vmatrix} = 8 \times \left(\frac{-59}{8}\right) \times \left(\frac{447}{59}\right) \times \left(\frac{1215}{149}\right) = -3645$$

Hasil dari perhitungan determinan diatas adalah $\det(A) = -3645$, karena nilai $det(A) \neq 0$ (nol) maka sistem persamaan linier(SPL) tersebut merupakan sistem persamaan linier(SPL) non homogen sehingga sistem persamaan linier(SPL) tersebut memiliki solusi tunggal.

Pada penelitian ini jumlah diagonal utama harus lebih dari jumlah mutlak dari bilangan lainnya disajikan pada tabel berikut,

Tabel 1 Diagonal utama dan Jumlah Mutlak

Baris ke	Diagonal utama	Jumlah mutlak
1	8	| 1+3-3 | = 1
2	-7	| 3+2-1 | = 4
3	8	| 2+3-1 | = 4
4	7	| 3+2+1 | = 6

Jumlah mutlak pada tabel 1 masing-masing baris lebih dari bilangan pada diagonal pertama maka selanjutnya dilakukan perhitungan menggunakan metode iterasi *Jacobi* dan metode iterasi *Gauss-Seidel* program komputasi *Python*.

Metode Iterasi *Jacobi* matriks berordo 4×4 menggunakan program komputasi Python dengan menggunakan modul numpy sebagai berikut

Mengimport modul numpy dengan alias pr

```
import numpy as pr
```

Selanjutnya Matriks A dengan ordo 4×4 ditulis dalam bentuk kode array menjadi

```
A = pr.array([  [8., 1., 3., -3.],
                [3., -7., 2., -1.],
                [2., 3., 8., -1.],
                [3., 2., 1., 7]])
```

Matriks B dengan ordo 1×4 ditulis dalam bentuk kode array menjadi

```
b = pr.array([[49., 27., 25., -16.]])
```

Matriks x dengan ordo 1×4 merupakan nilai awal atau x_0 ditulis dalam bentuk kode menjadi

```
x = pr.zeros(len(A[0]))
```

Untuk memeriksa kode yang diketik, yakni mariks A, Matriks B, dan Matriks x ditulis dalam kode menjadi

```
print("Matriks A: \n", A)
print("Matriks b: \n", b)
print("Matriks x: \n", x)
```

untuk membuat baris baru atau *newline* maka menggunakan " \n " seperti pada kode diatas.
Selanjutnya setelah tampilan matriks-matriks sudah sesuai, maka dibuat kode *looping*, iterasi, atau perulangan. Pada penelitian ini menggunakan batas perulangan hingga 1000 perulangan maka ditulis dalam bentuk kode sebagai berikut

```
for h in range(1000):
```

Pada komputasi Python, tabulasi memiliki pengaruh sehingga untuk kode yang termasuk dalam perulangan diberi tab maka ditulis dalam bentuk kode sebagai berikut

```
for h in range(1000):
        D = pr.diag(pr.diag(A))
        R = A - D
        y = pr.round((b - pr.dot(R, x)) * pr.linalg.inv(D), 5)
        x_d = pr.diag(y)

        if pr.allclose(x, x_d, rtol=1e-10, atol=0.):
                break
        x = x_d
        print("iterasi- ", h + 1)
        print(x_d)
```

kode D = pr.diag(pr.diag(A)) digunakan untuk membuat matriks A dalam bentuk matriks Segitiga Atas dan matriks segitiga bawah sehingga angka-angka diluar diagonal utama menjadi 0 sebagai berikut

$$D = \begin{bmatrix} 8 & 0 & 0 & 0 \\ 0 & -7 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 7 \end{bmatrix}$$

Kode R = A – D merupakan pengurangan Matriks A dan B sebagai berikut

$$A - D = \begin{bmatrix} 8 & 1 & 3 & -3 \\ 3 & -7 & 2 & -1 \\ 2 & 3 & 8 & -1 \\ 3 & 2 & 1 & 7 \end{bmatrix} - \begin{bmatrix} 8 & 0 & 0 & 0 \\ 0 & -7 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 7 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 3 & -3 \\ 3 & 0 & 2 & -1 \\ 2 & 3 & 0 & -1 \\ 3 & 2 & 1 & 0 \end{bmatrix}$$

Nilai perhitungan terkadang ada banyak angka dibelakang koma oleh karena itu dibutuhkan batas angka dibelakang koma, pada penelitian ini dibatasi hingga 5(lima) angka dibelakang koma maka kode komputasi python sebagai berikut

pr.round((b - pr.dot(nilai, banyak angka dibelakang koma)

kode pr.dot(R, x) merupakan perkalian matriks R dan X sebagai berikut

$$x \times R = [0 \quad 0 \quad 0 \quad 0] \times \begin{bmatrix} 0 & 1 & 3 & -3 \\ 3 & 0 & 2 & -1 \\ 2 & 3 & 0 & -1 \\ 3 & 2 & 1 & 0 \end{bmatrix} = [0 \quad 0 \quad 0 \quad 0]$$

Kode b - pr.dot(R, x) merupakan pengurangan dari Matrik b dan hasil perkaliman matriks $x \times R$ sebagai berikut

$$b - (x \times R) = [49 \quad 27 \quad 25 \quad -16] - [0 \quad 0 \quad 0 \quad 0] = [0 \quad 0 \quad 0 \quad 0]$$

Kode pr.linalg.inv(D) merupakan invers perkalian dari Matriks D sebagai berikut

$$invers(D) = \begin{bmatrix} 0.125 & 0 & 0 & 0 \\ 0 & -0.14285714 & 0 & 0 \\ 0 & 0 & 0.125 & 0 \\ 0 & 0 & 0 & 0.14285714 \end{bmatrix}$$

kode (b - pr.dot(R, x)) * pr.linalg.inv(D) merupakan perkalian dua matriks namun perkalian tersebut berdasarkan elemen di kali elemen seperti pada penjumlahan atau pengurangan matriks, hasil dari kode tersebut sebagai berikut

$$\begin{bmatrix} 6.125 & 0 & 0 & 0 \\ 0 & -3.85714286 & 0 & 0 \\ 0 & 0 & 3.125 & 0 \\ 0 & 0 & 0 & -2.28571429 \end{bmatrix}$$

Selanjutnya dibulatkan hingga 5 angka dibelakang koma menggunakan kode

y = pr.round((b - pr.dot(R, x)) * pr.linalg.inv(D), 5)

Sehingga menghasilkan sebagai berikut

$$\begin{bmatrix} 6.125 & 0 & 0 & 0 \\ 0 & -3.85714 & 0 & 0 \\ 0 & 0 & 3.125 & 0 \\ 0 & 0 & 0 & -2.28571 \end{bmatrix}$$

Selanjutnya mengubah hasil diatas kebentuk matriks berordo 1×4 dengan nama variabel x_d. Kode x_d = pr.diag(y) digunakan untuk membuat matriks y menjadi matriks 1(satu) baris berisi nilai diagonal utama menghasilkan sebagai berikut

$$[6.125 \quad -3.85714 \quad 3.125 \quad -2.28571]$$

Nilai tersebut merupakan nilai pada iterasi pertama, selanjutnya nilai tersebut menggantikan nilai matriks x yang awalnya matriks 0(nol).
Iterasi perlu diberi kondisi berdasarkan toleransi sehingga iterasi bisa berhenti setelah mencapai itu dengan kode sebagai berikut

```
if pr.allclose(x, x_d, rtol=1e-10, atol=0.):
        break
```

Kode lengkap ditampilkan pada gambar berikut
Input

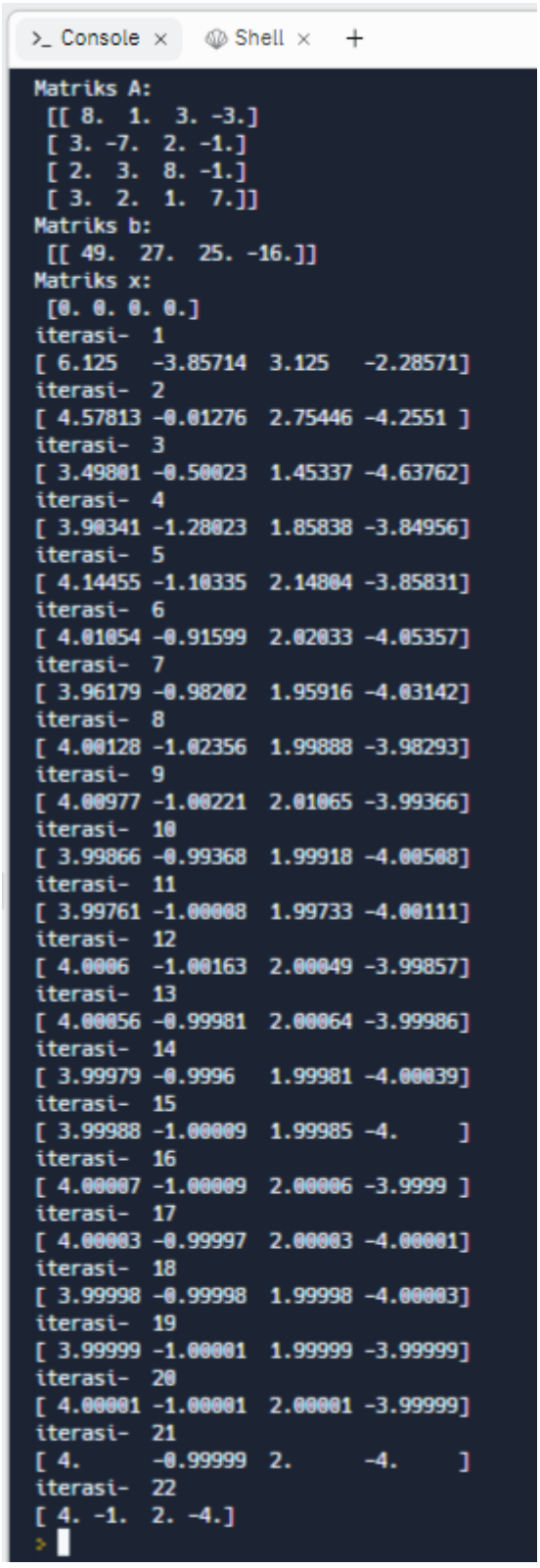
main.py

```
import numpy as pr

A = pr.array([[8., 1., 3., -3.],
              [3., -7., 2., -1.],
              [2., 3., 8., -1.],
              [3., 2., 1., 7]])

b = pr.array([[49., 27., 25., -16.]])

x = pr.zeros(len(A[0]))

print("Matriks A: \n", A)
print("Matriks b: \n", b)
print("Matriks x: \n", x)

for h in range(1000):
  D = pr.diag(pr.diag(A))
  R = A - D
  y = pr.round((b - pr.dot(R, x)) * pr.linalg.inv(D), 5)
  x_d = pr.diag(y)

  if pr.allclose(x, x_d, rtol=1e-10, atol=0.):
    break
  x = x_d
  print("iterasi- ", h + 1)
  print(x_d)
```

Gambar 1 Input Iterasi Jacobi 4×4

Output

Hasil dari program komputasi Python Iterasi Jacobi 4×4 ditampilkan pada gambar dibawah



Gambar 2 Output Iterasi Jacobi 4×4

Metode Iterasi Gauss-Seidel matriks berordo 4×4 menggunakan program komputasi Python dengan menggunakan modul numpy sebagai berikut
Mengimport modul numpy dengan alias pr

```
import numpy as pr
```

Selanjutnya Matriks A dengan ordo 4×4 ditulis dalam bentuk kode array menjadi

```
A = pr.array([  [8., 1., 3., -3.],
                [3., -7., 2., -1.],
                [2., 3., 8., -1.],
                [3., 2., 1., 7]])
```

Matriks B dengan ordo 1×4 ditulis dalam bentuk kode array menjadi

```
b = pr.array([[49., 27., 25., -16.]])
```

Pada metode Iterasi *Gauss-Seidel* dibutuhkan mengetahui jumlah kolom, untuk mengetahui jumlah kolom dapat menggunakan kode sebagai berikut

```
column=pr.shape(A)[0]
```

Matriks x dengan ordo 1×4 merupakan nilai awal atau x_0 ditulis dalam bentuk kode menjadi

```
x = pr.zeros(len(A[0]))
```

Untuk memeriksa kode yang diketik, yakni mariks A, Matriks B, dan Matriks x ditulis dalam kode menjadi

```
print("Matriks A: \n", A)
print("Matriks b: \n", b)
print("Matriks x: \n", x)
```

untuk membuat baris baru atau *newline* maka menggunakan " \n " seperti pada kode diatas.

Membuat matriks A dalam bentuk matriks Segitiga Atas dan matriks segitiga bawah sehingga angka-angka diluar diagonal utama menjadi 0 dapat menggunakan kode sebagai berikut

```
D = pr.diag(pr.diag(A))
```

Output dari kode tersebut sebagai berikut

$$D = \begin{bmatrix} 8 & 0 & 0 & 0 \\ 0 & -7 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 7 \end{bmatrix}$$

Membuat pengurangan Matriks A dan B dengan kode sebagai berikut

R = A – D

Output dari kode tersebut sebagai berikut

$$A - D = \begin{bmatrix} 8 & 1 & 3 & -3 \\ 3 & -7 & 2 & -1 \\ 2 & 3 & 8 & -1 \\ 3 & 2 & 1 & 7 \end{bmatrix} - \begin{bmatrix} 8 & 0 & 0 & 0 \\ 0 & -7 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 7 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 3 & -3 \\ 3 & 0 & 2 & -1 \\ 2 & 3 & 0 & -1 \\ 3 & 2 & 1 & 0 \end{bmatrix}$$

Selanjutnya setelah tampilan matriks-matriks sudah sesuai, maka dibuat kode *looping*, iterasi, atau perulangan. Pada penelitian ini menggunakan batas perulangan hingga 1000 perulangan maka ditulis dalam bentuk kode sebagai berikut

```
for h in range(1000):
```

Pada komputasi Python, tabulasi memiliki pengaruh sehingga untuk kode yang termasuk dalam perulangan diberi tab maka ditulis dalam bentuk kode sebagai berikut

```
for h in range(1000):
    for i in range(column):
        y=pr.round(((b-pr.dot(R,x))* pr.linalg.inv(D)),5)
        x_d=pr.diag(y)
        x[i]=x_d[i]
    print("iterasi ke-",h+1)
    print(x)
    if pr.allclose(x,x_d,rtol=1e-10,atol=0.):
        break
```

Pada Metode Iterasi Gauss-Seidel ada dua *Looping* atau perulangan, pada kode diatas ditandai dengan h dan i, awal *looping* pada variabel h, selanjutnya variabel i. Setelah looping variabel i selesai maka lanjut ke looping h hal ini karena perhitunga pada Metode Iterasi *Gauss-Seidel* untuk nilai/hasil nilai *x* dihitung langsung tanpa menunggu perhitungan selesai seperti pada Metode Iterasi Jacobi.

Nilai perhitungan terkadang ada banyak angka dibelakang koma oleh karena itu dibutuhkan batas angka dibelakang koma, pada penelitian ini dibatasi hingga 5(lima) angka dibelakang koma maka kode komputasi python sebagai berikut

```
pr.round((b - pr.dot(nilai, banyak angka dibelakang koma)
```

kode pr.dot(R, x) merupakan perkalian matriks R dan X sebagai berikut

$$x \times R = [0 \quad 0 \quad 0 \quad 0] \times \begin{bmatrix} 0 & 1 & 3 & -3 \\ 3 & 0 & 2 & -1 \\ 2 & 3 & 0 & -1 \\ 3 & 2 & 1 & 0 \end{bmatrix} = [0 \quad 0 \quad 0 \quad 0]$$

Kode b - pr.dot(R, x) merupakan pengurangan dari Matrik b dan hasil perkaliman matriks $x \times R$ sebagai berikut

$$b - (x \times R) = [49 \quad 27 \quad 25 \quad -16] - [0 \quad 0 \quad 0 \quad 0] = [0 \quad 0 \quad 0 \quad 0]$$

Kode pr.linalg.inv(D) merupakan invers perkalian dari Matriks D sebagai berikut

$$invers(D) = \begin{bmatrix} 0.125 & 0 & 0 & 0 \\ 0 & -0.14285714 & 0 & 0 \\ 0 & 0 & 0.125 & 0 \\ 0 & 0 & 0 & 0.14285714 \end{bmatrix}$$

kode (b - pr.dot(R, x)) * pr.linalg.inv(D) merupakan perkalian dua matriks namun perkalian tersebut berdasarkan elemen di kali elemen seperti pada penjumlahan atau pengurangan matriks, hasil dari kode tersebut sebagai berikut

$$\begin{bmatrix} 6.125 & 0 & 0 & 0 \\ 0 & -3.85714286 & 0 & 0 \\ 0 & 0 & 3.125 & 0 \\ 0 & 0 & 0 & -2.28571429 \end{bmatrix}$$

Selanjutnya dibulatkan hingga 5 angka dibelakang koma menggunakan kode

```
y = pr.round((b - pr.dot(R, x)) * pr.linalg.inv(D), 5)
```

Sehingga menghasilkan sebagai berikut

$$\begin{bmatrix} 6.125 & 0 & 0 & 0 \\ 0 & -3.85714 & 0 & 0 \\ 0 & 0 & 3.125 & 0 \\ 0 & 0 & 0 & -2.28571 \end{bmatrix}$$

Selanjutnya mengubah hasil diatas kebentuk matriks berordo 1×4 dengan nama variabel x_d. Kode x_d = pr.diag(y) digunakan untuk membuat matriks y menjadi matriks 1(satu) baris berisi nilai diagonal utama menghasilkan sebagai berikut

$$[6.125 \quad -3.85714 \quad 3.125 \quad -2.28571]$$

Pada perhitungan diatas untuk nilai $x_{1,1} = 6.125$ untuk menghitung selanjutnya $x_{1,2}$ berbeda dari metode iterasi Jacobi, jika iterasi Jacobi menggunakan x awal yakni 0, pada iterasi gauss seidel menggunakan nilai terbaru misalnya pada iterasi

ini ada nilai terbaru yakni nilai $x_{1,1} = 6.125$. Pada kode x[i]=x_d[i] iterasi i awal menghasilkan x[0]=x_d[0] nilainya 6.125, x[1]=x_d[1] nilainya -1.23214, x[2]=x_d[2]. Pada komputasi python, untuk iterasi atau nomor kolom dimulai dari 0. Jika tertulis kolom 0 maka itu merupakan kolom 1 begitu seterusnya. Pada iterasi pertama akan menghasilkan sebagai berikut

$$[\,6.125 \quad -1.23214 \;\; 2.0558 \quad -4.85236]$$

Nilai tersebut merupakan nilai pada iterasi pertama, selanjutnya nilai tersebut menggantikan nilai matriks x yang awalnya matriks 0(nol) namun berbeda dari metode iterasi *Jacobi*, pada metode iterasi *Gauss-Seidel* selalu menggunakan hasil terbaru untuk selanjutnya tanpa menunggu satu iterasi selesai. Iterasi perlu diberi kondisi berdasarkan toleransi sehingga iterasi bisa berhenti setelah mencapai itu dengan kode sebagai berikut

```
if pr.allclose(x, x_d, rtol=1e-10, atol=0.):
        break
```

Kode lengkap ditampilkan pada gambar berikut

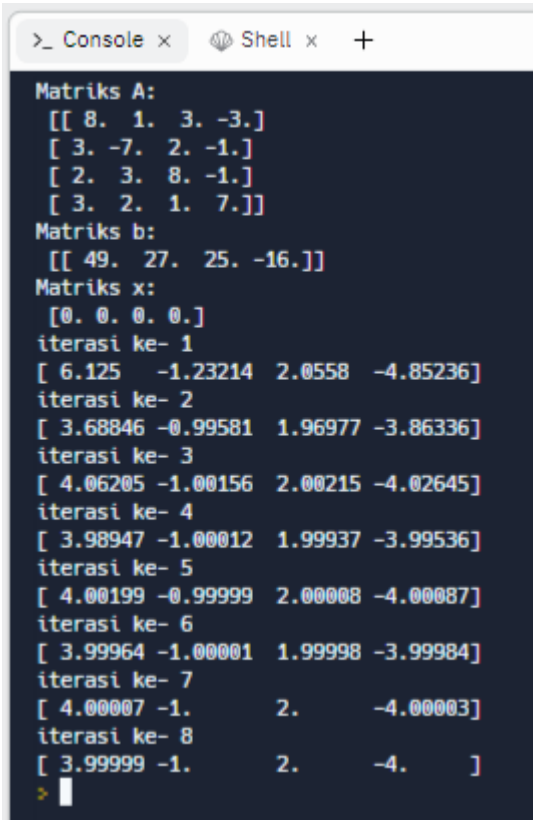
Input

main.py

```
import numpy as pr

A = pr.array([[8., 1., 3., -3.],
              [3., -7., 2.,-1.],
              [2., 3., 8.,-1.],
              [3., 2., 1.,7]])

b = pr.array([[49., 27., 25., -16.]])

column=pr.shape(A)[0]

x=pr.zeros(len(A[0]))

print("Matriks A: \n",A)
print("Matriks b: \n",b)
print("Matriks x: \n",x)

D=pr.diag(pr.diag(A))
R=A-D

for h in range(1000):
    for i in range(column):
        y=pr.round(((b-pr.dot(R,x))* pr.linalg.inv(D)),5)
        x_d=pr.diag(y)
        x[i]=x_d[i]
    print("iterasi ke-",h+1)
    print(x)
    if pr.allclose(x,x_d,rtol=1e-10,atol=0.):
        break

```

Gambar 3 Input Iterasi Gauss-Seidel 4 × 4

Output
Hasil dari program komputasi Python Iterasi Gauss-Seidel 4×4 ditampilkan pada gambar dibawah



Gambar 4 Output Iterasi Gauss-Seidel 4×4

Analisis antara penggunaan metode iterasi jacobi dengan metode iterasi gauss-seidel melalui progran komputasi *pyhton* lebih cepat menemukan solusi menggunakan metode iterasi Gauss-Seidel, hal ini bisa dilihat pada gambar nomor 2 dan pada gambar nomor 4.

KESIMPULAN

Berdasarkan hasil yang dibahas dalam penelitian ini, dapat disimpulkan bahwa yang pertama Program komputasi Python dapat digunakan untuk menghitung metode iterasi *Gauss-Seidel* dan metode iterasi *Jacobi*; kedua dalam pemrosesan program komputasi Python dengan menggunakan metode iterasi *Gauss-Seidel* banyaknya iterasi lebih sedikit dibandingkan dengan menggunakan metode iterasi *Jacobi*, *output* metode iterasi Jacobi 4×4 sampai iterasi ke 22 terlihat pada gambar 2 sedangkan *output* metode iterasi *Gauss-Seidel* 4×4 sampai iterasi ke 8 pada gambar 4; dan ketiga mengenai waktu pemrosesan program komputasi *Python* menggunakan metode iterasi *Gauss-Seidel* lebih cepat dibandingkan dengan menggunakan metode Iterasi *Jacobi* hal ini dapat dilihat dari jumlah iterasi pada gambar nomor 2 dan gambar nomor 4.

Berdasarkan hasil penelitian mengenai perbandingan solusi sistem persamaan linear(SPL) menggunakan metode iterasi *Jacobi* dan menggunakan metode iterasi *Gauss-Seidel* dengan menggunakan komputasi *Python*, maka diberikan beberapa saran untuk penelitian selanjutnya yaitu yang pertama untuk Peneliti berikutnya dapat menggunakan program komputasi lain seperti seperti *Matlab*, program R, *Maple*, atau program lainnya yang relevan; kedua untuk Peneliti berikutnya dapat melakukan analisis numerik pada sistem persamaan linier dengan Matriks ordo lebih tinggi; dan ketiga untuk Peneliti berikutnya dapat menggunakan metode iterasi numerik yang lain.

DAFTAR RUJUKAN

[1] Rahmi dan Mulia Suryan. (2018). *Buku Ajar Program Linier*. Yogyakarta: Deepublish.
[2] Sugiyono. (2018). *Metode Penelitian Kuntitatif, Kualitatif, dan R&D*. Bandung: Alfabeta.
[3] Zed, M. (2014). *Metode penelitian kepustakaan*. Jakarta: Yayasan Pustaka Obor Indonesia.
[4] Rodiah(2018). *Modul Kuliah Komputasi dengan Python*. Depok : Penerbit Gunadarma