Laporan Praktikum
Komputasi Numerik

KN - 03

SISTEM PERSAMAAN LINEAR

Nama	: Rendy Putra Pratama
NPM	: 140310230037
Hari/Tanggal	: Jumat/ 27 Maret 2024
Waktu	: 08.00 WIB
Asisten	: M. Naufaldi D.

LABORATORIUM KOMPUTASI

DEPARTEMEN FISIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS PADJADJARAN

2024

LEMBAR PENGESAHAN

KN - 03

SISTEM PERSAMAAN LINEAR

Nama : Rendy Putra Pratama

NPM : 140310230037

Hari / Tanggal : Jumat/ 27 Maret 2024

Waktu / Sesi : 08.00 WIB

Asisten : M. Naufaldi D.

Pretest	Laporan Akhir

Jatinangor, 27 Maret 2024
Asisten

(M. Naufaldi D.)

1. Tugas Percobaan Praktikum

   Buatlah program dengan menggunakan bahasa Python untuk menentukan solusi persamaan linier dari persamaan-persamaan berikut menggunakan metode Iterasi Jacobi, Gauss-Seidel, dan SOR :

   a. $2x_1 + 3x_2 = 5$
      $x_1 - 6x_2 = 9$
   b. $-3x_1 - 0.1x_2 - 0.2x_3 = 7.85$
      $0.1x_1 + 7x_2 - 0.3x_3 = -19.3$
      $0.3x_1 - 0.2x_2 + 10x_3 = 71.4$
   c. $a + 3b - c + 0.3d = 28$
      $b - 9.8c = 12$
      $3a - 4b + 2c = -2$
      $1.4a - 2.4c = 3$

1.1 Percobaan A

1.1.1 Listing Program

1.1.1.1 Metode Jacobi

```
import numpy as np

# Flowchart

# Buat array berdasarkan persamaan yang akan dicari
solusinya
A = np.array([[2,3],[1,-6]])
# Array hasil dari persamaan sebelumnya
Y = np.array([5,9])
# Array untuk hasil x1,x2
x = np.array([0,0])

# inisiasi awal
itr = 0
e = 1

while e > 0.000001:
    solveA = []
    n = len(x)
    for i in range(n):
        sigma = 0
        for j in range(0,n):
            if j != i:
                sigma += A[i][j]*x[j]
        xNow = (Y[i]-sigma)/A[i][i]
        solveA.append(xNow) # diappend = biar array
solveA kosong, dan bisa di input lagi dengan nilai
baru
    e = abs(max(x)-max(solveA))
    x = np.array(solveA)
    itr += 1
print("Perhitungan dengan Metode Jacobi")
print(f"Solusi x1 = {x[0]:.3f}; x2 = {x[1]:.3f}")
print(f"error = {e}")
print(f"Matriks solusi X1,X2 = {x} \t iterasi =
{itr}")
```

1.1.1.2 Metode Gauss Seidell

```
import numpy as np
A = np.array([[2,3],[1,-6]])

x = np.array([0,0])

y = np.array([5,9])
e = 1
while e>1e-6:
    solutionAx = [] # wadah solusi x1 dan x2
    n = len(x)
    for i in range(n):
        sigma = 0
        for j in range(0,i):
            sigma += A[i][j]*solutionAx[j] #
Perbedaan dengan Jacobi
        for j in range(i+1,n):
            sigma += A[i][j]*x[j]
        xn = (y[i]-sigma)/A[i][i]
        solutionAx.append(xn)
    e = abs(max(x)-max(solutionAx))
    x = np.array(solutionAx)
    i+=1
print("Penyelesaian Dengan Metode Seidell")
print(f"Solusi x1 = {x[0]:.3f}; x2 = {x[1]:.3f}")
print(f"error = {e}")
print(f"Hasil akar-akar = {x} iterasi - {i}")
```

1.1.2 Output Program

1.1.2.1 Metode Jacobi

```
Perhitungan dengan Metode Jacobi
Solusi x1 = 3.800; x2 = -0.867
error = 5.960464477539062e-07
Matriks solusi X1,X2 = [ 3.80000031 -0.86666652]
       iterasi = 23
```

1.1.2.2 Metode Gauss Seidell

```
Penyelesaian Dengan Metode Seidell
Solusi x1 = 3.800; x2 = -0.867
error = 3.8743019104003906e-07
Hasil akar-akar = [ 3.79999992 -0.86666668] iterasi
- 2
```

1.1.3 Analisis

Berdasarkan percobaan, pada setiap metode praktikan melakukan inisiasi awal untuk variabel-variabel yang akan mempengaruhi perhitungan seperti menentukan matriks yang diketahui, array kosong untuk nilai solusi x_1 dan x_2, dan nilai $e = 1$ sebagai kondisi true pada looping while.

Kemudian praktikan menggunakan library numpy, library ini untuk memudahkan dalam membuat matriks yaitu dengan menggunakan fungsi array – np.array().

Selanjutnya praktikan membuat variabel kosong sebagai wadah sementara dalam iterasi while yaitu solutionAx dan menghitung panjang data dari matriks x – matriks kosong pada global scope. Langkah tersebut dilakukan pada kedua metode – Jacobi dan Seidell.

Setelah melakukan inisiasi pada global scope dan local scope pada looping while, praktikum menggunakan flowchart yang ada pada jacobi dan seidel untuk menghitung nilai solusi x_1 dan x_2.

Pada metode jacobi pengulangan – *for i in range* dimulai dari 0 sampai n (panjang matriks x yaitu 2), selanjutnya didalam looping for tersebut diinisiasikan nilai awal sigma = 0 – wadah penjumlahan dari rumus jacobi. Setelah itu, berdasarkan rumus jacobi perhitungan sigma dimulai dari j = 0 sampai ke i dengan i teriterasi pada for i in range() dengan kondisi j≠i. Kemudian, jika program telah masuk kedalam pengondisian selanjutnya program akan menghitung nilai dari matriks A pada elemen ke-i dan ke-j yang kemudian nilai elemen matriks $A_{\{ij\}}$ dikalikan dengan x_j akan tetapi elemen x_j diinisiasikan dengan nilai awal nol.

Contohnya ketika nilai x_1 yang dicari maka, persamaannya akan menjadi :

$$x_1 = \frac{5 - 3x_2}{2}$$

Dengan nilai awal x_2 adalah nol, kemudian hal tersebut juga dilakukan pada persamaan kedua dan hasilnya akan diperbarui dan disimpan dalam variabel sigma. Setelahnya, nilai sigma tersebut akan dikurangkan dengan nilai dari hasil yang telah ada pada variabel Y.

$$\frac{Y - sigma}{A_{ii}}$$

Hal yang menjadi perbedaan dengan seidel adalah pada seidel tidak ada kondisi $j \neq i$, akan tetapi terdapat dua pengulangan untuk menjumlahkan nilai dari sigma, yang pertama for j in range(0,i) dengan penjumlahan didalam loop for tersebut elemen A_{ij} dengan nilai yang telah diperbarui yaitu pada variabel solutionAx. Maka, ketika nilai dari solusi awal telah diketahui, maka looping for yang pertama akan memasukkan nilai solusi x dengan hasil terbaru. Kemudian, nilai x_1 dan x_2 pada kondisi awal akan disubstitusikan dengan nilai inisiasi awal nol. Sehingga hasil dari perhitungan pada for loop kedua, akan dijumlahkan dengan hasil sigma yang telah dilakukan pada loop pertama.

Setelah melakukan sejumlah iterasi pada for j in range yang pertama dan kedua, nilai sigma yang diperbarui dari perhitungan tersebut akan dikurangkan dengan nilai konstanta pada setiap persamaan. Hasil perhitungan tersebut akan disimpan didalam variabel xn dan nilai didalam xn akan disubstitusikan kedalam solutionAx.

Hasilnya adalah percobaan menggunakan metode Seidel memiliki iterasi yang jauh lebih efisien dibandingkan dengan metode jacobi. Faktor utama dari metode seidel adalah iterasi dilakukan dengan memasukkan nilai baru dari x_1 dan x_2, sedangkan pada jacobi nilai x_1 dan x_2 semuanya diinisiasikan dengan nilai nol. Kemudian setelah nilai x_1 dan x_2 telah tertentu, selanjutnya akan disubstitusi nilai x_1 dan x_2 terbaru untuk mendapatkan nilai error paling rendah untuk x_1 dan x_2 pada iterasi selanjutnya.

1.2 Percobaan B

1.2.1 Listing Program

1.2.1.1 Metode Gauss-Seidell

```
import numpy as np
A = np.array([[-3,-0.1,-0.2],
              [0.1,7,-0.3],
              [0.3,-0.2,10]])

x = np.array([0.0,0.0,0.0])

y = np.array([7.85,-19.3,71.4])
e = 1
while e>1e-6:
    solutionAx = [] # wadah solusi x1 dan x2
    n = len(x)
    for i in range(n):
        sigma = 0
        for j in range(0,i):
            sigma += A[i][j]*solutionAx[j] #
Perbedaan dengan Jacobi
        for j in range(i+1,n):
            sigma += A[i][j]*x[j]
        xn = (y[i]-sigma)/A[i][i]
        solutionAx.append(xn)
    e = abs(max(x)-max(solutionAx))
    x = np.array(solutionAx)
    i+=1
print("Penyelesaian Dengan Metode Seidell")
print(f"Solusi x1 = {x[0]:.3f}; x2 = {x[1]:.3f};
x3 = {x[2]:.3f}")
print(f"error = {e}")
print(f"Hasil akar-akar = {x} iterasi - {i}")
```

1.2.1.2 Metode SOR

```
# SOR METHODS
import numpy as np
# buat matriks persamaan linear [x,y]
A = np.array([[-3,-0.1,-0.2],
              [0.1,7.0,-0.3],
              [0.3,-0.2,10]])
# matriks hasil persamaan linear
y = np.array([7.85,-19.3,71.4])

# wadah kosong
x = np.array([0.0,0.0,0.0])

# inisiasi awal
e = 1
i = 0
w = 0.8
while e>1e-6:
    n = len(x)
    solutionAx = []

    for i in range(n):
        sigma = 0
        for j in range(0,i):
            sigma += A[i][j]*solutionAx[j]
        for j in range(i+1,n):
            sigma += A[i][j]*x[j]
        xn = (1-w)*x[i]+(w/A[i][i])*(y[i]-sigma)
        solutionAx.append(xn)
    e = abs(max(x)-max(solutionAx))
    x = np.array(solutionAx)
    i += 1
print(f"nilai w = {w}")
print(f"error = {e}")
print(f"Solusi X1,X2, dan X3 dengan SOR = {x} \t
iterasi = {i}")
```

1.2.2 Output Program

1.2.2.1 Metode Gauss-Seidell

```
Penyelesaian Dengan Metode Seidell
Solusi x1 = -3.015; x2 = -2.406; x3 = 7.182
error = 2.2413931155540467e-08
Hasil akar-akar = [-3.01528045 -2.40625314
7.18233335] iterasi - 3
```

1.2.2.2 Metode SOR

```
nilai w = 0.8
error = 9.53758438448915e-07
Solusi X1,X2, dan X3 dengan SOR = [-3.01527971 -
2.40625347  7.18233308]        iterasi = 3
```

1.2.3 Analisis

Pada percobaan kedua praktikan menggunakan metode seidel dan SOR, dengan langkah yang sebagaimana percobaan pertama yaitu melakukan pendefinisian untuk variabel yang telah diketahui serta variabel yang digunakan untuk melakukan pengondisian untuk pengulangan while.

Metode yang berbeda pada percobaan ini adalah metode SOR. Metode SOR secara algoritma sesuai dengan metode Seidell akan tetapi pada metode SOR terdapat faktor w. Faktor w merupakan relaksasi dari error pada iterasi. Faktor w berada pada rentang 0 sampai 1.

Oleh sebab itu, iterasi dari metode SOR dengan relaksasi pada percobaan ini adalah sama. Hal ini dikarenakan kesamaan algoritma yang terdapat pada metode SOR dan Seidell.

Jika dilihat dari hasil solusi untuk x_1 dan x_2, metode SOR memiliki nilai ketetilitian yang lebih akurat dibandingkan dengan metode SOR. Hal ini didukung karena pada metode SOR terdapat nilai relaksasi.

1.3 Percobaan C

1.3.1 Listing Program

1.3.1.1 Metode Jacobi

```
import numpy as np

# Flowchart

# Buat array berdasarkan persamaan yang akan
dicari solusinya
A = np.array([[1.4,0,-2.4,0],[3,-4,2,0],[0,1,-
9.8,0],[1,3,-1,0.3]])
# Array hasil dari persamaan sebelumnya
Y = np.array([[3],[12],[-2],[28]])
# Array untuk hasil x1,x2,x3
x = np.array([0,0,0,0])

# inisiasi awal
itr = 0
e = 1

while e > 0.000001:
    solveA = []
    n = len(x)
    for i in range(n):
        sigma = 0
        for j in range(0,i):
                sigma += A[i][j]*x[j]
        for j in range(i+1,n):
            sigma += A[i][j]*x[j]
        xNow = (Y[i]-sigma)/A[i][i]
        solveA.append(xNow) # diappend = biar
array solveA kosong, dan bisa di input lagi dengan
nilai baru
    e = abs(max(x)-max(solveA))
    x = np.array(solveA)
    itr += 1
print("Perhitungan dengan Metode Jacobi")
print(f"Matriks solusi X1,X2,X3, dan X4 = {x}
iterasi {itr}")
```

1.3.1.2 Metode SOR

```
# SOR METHODS
import numpy as np
# buat matriks persamaan linear [x,y]
A = np.array([[1.4,0,-2.4,0],[3,-4,2,0],[0,1,-
9.8,0],[1,3,-1,0.3]])
# matriks hasil persamaan linear
y = np.array([[3],[12],[-2],[28]])

# wadah kosong
x = np.array([0.0,0.0,0.0,0.0])

# inisiasi awal
e = 1
i = 0
w = 0.8
while e>1e-6:
    n = len(x)
    solutionAx = []

    for i in range(n):
        sigma = 0
        for j in range(0,i):
            sigma += A[i][j]*solutionAx[j]
        for j in range(i+1,n):
            sigma += A[i][j]*x[j]
        xn = (1-w)*x[i]+(w/A[i][i])*(y[i]-sigma)
        solutionAx.append(xn)
    e = abs(max(x)-max(solutionAx))
    x = np.array(solutionAx)
    i += 1
print(f"nilai w = {w}")
print(f"error = {e}")
print(f"Solusi X1,X2, dan X3 dengan SOR = {x} \t
iterasi = {i}")
```

1.3.2 Output Program

1.3.2.1 Metode Jacobi

```
Perhitungan dengan Metode Jacobi
error = [1.65610359e-07]
Matriks solusi X1,X2,X3, dan X4 = [[
2.27272721e+00]
 [-1.25757596e+00]
 [ 7.57575361e-02]
 [ 9.85858631e+01]] iterasi 25
```

1.3.2.2 Metode SOR

```
nilai w = 0.8
error = [5.28055921e-07]
Solusi X1,X2, dan X3 dengan SOR = [[
2.27272725e+00]
 [-1.25757578e+00]
 [ 7.57575719e-02]
 [ 9.85858590e+01]]      iterasi = 4
```

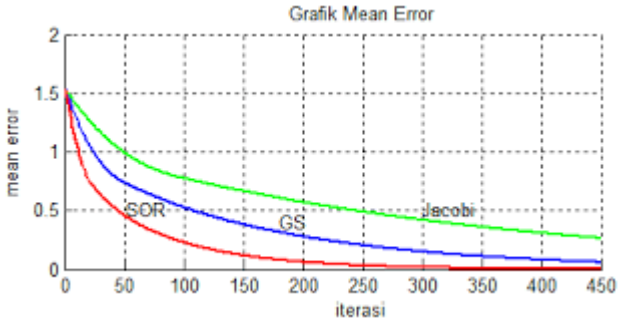
1.3.3 Analisis

Pada percobaan terakhir praktikan akan mencari solusi dari nilai x_1, x_2, dan x_3. Praktikan membandingkan metode Jacobi dengan metode SOR. Sebagaimana pada percobaan sebelumnya praktikan, melakukan inisiasi nilai awal untuk variabel-variabel yang telah diketahui.

Pada metode SOR praktikan menginput nilai relaksasi senilai 0.8, dengan algoritma yang sama seperti pada metode Seidel akan tetapi dengan perbedaan memasukkan nilai dari relaksasi kedalam perhitungan didapat hasil perhitungan pada program dengan metode SOR memiliki iterasi sebesar 4, sedangkan pada metode Jacobi iterasi mencapai nilai error dibawah dari nilai toleransi pada iterasi ke-24.

Dari percobaan diatas mengindikasikan bahwa metode SOR merupakan metode yang paling efisien pada percobaan 3. Hal ini menunjukkan bahwa metode SOR memiliki tingkat iterasi yang cepat untuk menentukan nilai eksak dari solusi x_1, x_2 dan x_3. Sedangkan metode Jacobi memerlukan 24 iterasi atau lebih banyak pengulangan atau memori dalam memecahkan solusi dari tiga persamaan.

Kesimpulan sementara dari percobaan 1,2, dan 3 adalah metode SOR akan sangat membantu pada kasus kasus kompleks sebagaimana yang tertera pada percobaan 3. Namun, untuk kasus sederhana pada percobaan 1, metode Jacobi dan Seidell menjadi metode yang paling efektif karena tidak membutuhkan memori yang tidak terlalu banyak. Jika dapat digambarkan dengan grafik, grafik penurunan nilai error terhadap tingkat iterasi antara metode Jacobi, Gauss Seidell dan SOR adalah sebagai berikut :

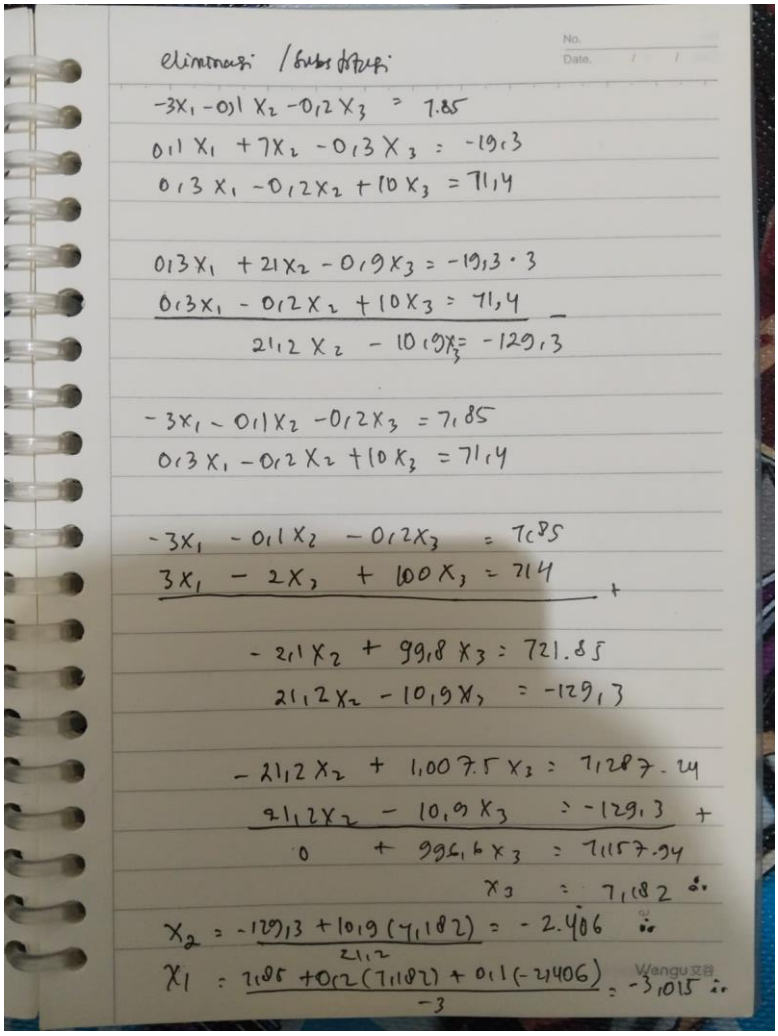


Nilai relaksasi pada metode SOR berada pada rentang 0 sampai 1 bertujuan agar hasil dari perhitungan mendapat nilai yang konvergen

2. Tugas Akhir Praktikum
    a. Selidiki perbedaan norm jika diaplikasikan pada Percobaan untuk metode iterasi Jacobi dan Gauss-Seidel.
    b. Apa yang akan terjadi jika kita menggunakan metode SOR untuk menyelesaikan percobaan dengan nilai $w > 2$ atau $w < 0$?
    Pada pembahasan percobaan ketiga, ketika nilai relaksasi berada diluar rentang 0 sampai 1, maka iterasi akan semakin membesar atau tidak mencapai nilai konvergen. Hal ini dikarenakan nilai tersebut berada pada kondisi overrelaxation, sedangkan rentang 0 sampai 1 bertujuan untuk mencari nilai konvergen dari suatu persamaan yang akan mengubah nilai yang awalnya divergen menjadi konvergen. Kesimpulannya adalah nilai relaksasi yang tidak berada pada rentang 0 sampai 1 menyebabkan solusi persamaan tersebut semakin membesar (overrelaxation).
    c. Hitung solusi dari sistem persamaan linier di bawah ini menggunakan cara analitik (eliminasi/substitusi). Lalu hitung solusi dari sistem permaan linear dibawah ini menggunakan iterasi Jacobi hingga iterasi ke-2.

$$-3x_1 - 0.1x_2 - 0.2x_3 = 7.85$$
$$0.1x_1 + 7x_2 - 0.3x_3 = -19.3$$
$$0.3x_1 - 0.2x_2 + 10x_3 = 71.4$$

    Metode Eliminasi



Metode Jacobi hingga Iterasi kedua :

Iterasi pertama :

$$x_1 = \frac{7.85+0.2x_3+0.1x_2}{-3} = \frac{7.85+0.2(0)+0.1(0)}{-3} = -2.616$$

$$x_2 = \frac{-19.3+0.3x_3-0.1x_1}{7} = \frac{-19.3+0.3(0)-0.1(0)}{7} = -2.757$$

$$x_3 = \frac{71.4+0.2x_2-0.3x_1}{10} = \frac{71.4+0.2(0)-0.3(0)}{10} = 7.14$$

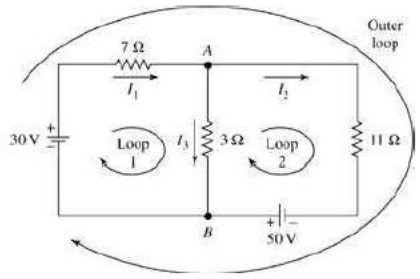
Iterasi kedua :

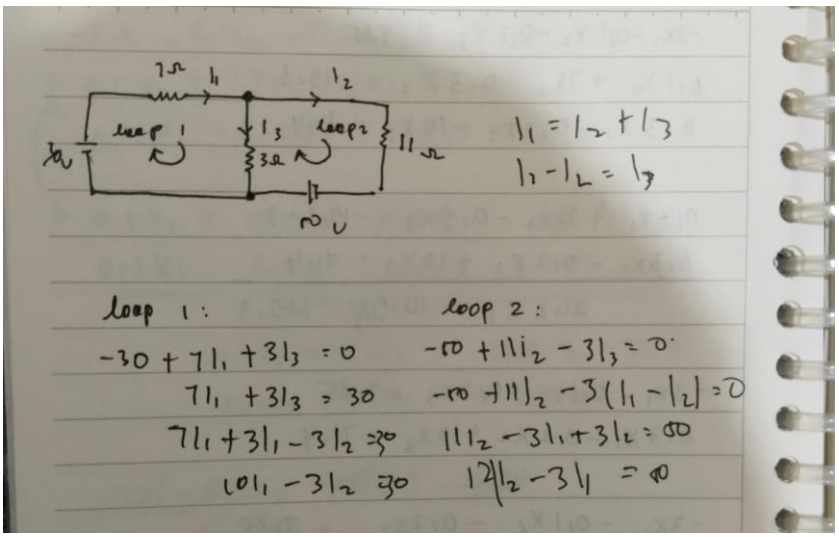
$$x_1 = \frac{7.85 + 0.2(7.14) + 0.1(-2.757)}{-3} = -3.0007$$

$$x_2 = \frac{-19.3 + 0.3(7.14) - 0.1(-2.616)}{7} = -2.413$$

$$x_3 = \frac{71.4 + 0.2(-2.757) - 0.3(-2.616)}{10} = 7.1633$$

d. Tentukan besar masing-masing arus listrik $(I_1, I_{2,}\ dan\ I_3)$ pada rangkaian dibawah ini, dengan menggunakan sistem persamaan linear.





Listing

```
import numpy as np
A = np.array([[10.0,-3.0],[-3.0,14]])

x = np.array([0.0,0.0])

y = np.array([30.0,50.0])
e = 1
w = 0.1
while e>1e-6:
    solutionAx = [] # wadah solusi x1 dan x2
    n = len(x)
    for i in range(n):
        sigma = 0
        for j in range(0,i):
            sigma += A[i][j]*solutionAx[j]
        for j in range(i+1,n):
            sigma += A[i][j]*x[j]
        xn = (1-w)*x[i]+(w/A[i][i])*(y[i]-sigma)
        solutionAx.append(xn)
    e = abs(max(x)-max(solutionAx))
    x = np.array(solutionAx)
    i+=1
print(f"i1 = {x[0]:.4f} i2 = {x[1]:.4f}")
print(f"i3 = {x[1]-x[0]}")
print(f"faktor relaksasi = {w}\t iterasi = {i}")
```

Output

```
i1 = 4.3511 i2 = 4.5038
i3 = 0.15267448062729638
faktor relaksasi = 0.1   iterasi = 2
```

3. Kesimpulan

Pada praktikum ini praktikan dapat menentukan nilai dari solusi sistem persamaan menggunakan 3 metode yaitu Jacobi, Seidel dan SOR. Berdasarkan percobaan metode SOR merupakan metode terbaik dalam mencari solusi dari banyak sistem persamaan. Nilai relaksasi pada metode SOR membantu dalam melakukan konvergensi dari nilai yang akan menjadi solusi untuk sistem persamaan linear.

DAFTAR PUSTAKA

C. Chapra, S., & P. Canale, R. (2015). *Numerical Methods in Engineering* (7th ed.). McGraw-Hill Education.

Kiusalaas, J. (2009). *NUMERICAL METHODS IN ENGINEERING* (2nd ed.). New York: Cambridge University Press.

Munir, Rinaldi. (2017). *Metode Numerik.* Bandung: Penerbit INFORMATIKA