Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки
Кафедра Обчислювальної Техніки

Лабораторна робота № 3

з дисципліни «Чисельні методи»

на тему

«Розв’язання систем лінійних алгебраїчних рівнянь (СЛАР) ітераційними методами. Метод простої ітерації. Метод Зейделя»

Виконав:
студент гр. ІП-93
Домінський Валентин

Викладач:
доц. Рибачук Л.В.

Київ – 2021

Зміст

1 Постановка задачі

Розв’язати систему рівнянь методом Зейделя з кількістю значущих цифр m = 6.

$$\begin{bmatrix} 3.81 & 0.25 & 1.28 & 2.75 \\ 2.25 & 1.32 & 6.58 & 0.49 \\ 5.31 & 8.28 & 0.98 & 1.04 \\ 11.39 & 2.45 & 3.35 & 2.28 \end{bmatrix} \cdot X = \begin{bmatrix} 4.21 \\ 8.47 \\ 2.38 \\ 12.48 \end{bmatrix}$$

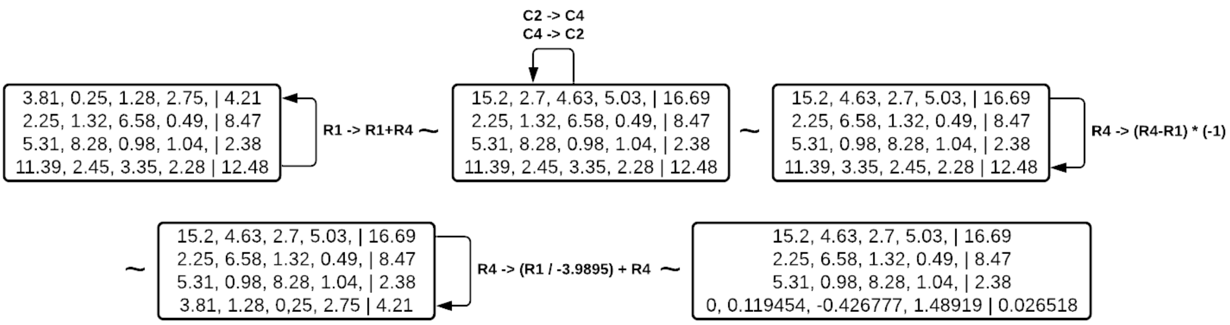
Вивести всі проміжні результати та розв’язок системи. Навести результат перевірки: вектор нев’язки $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Ax}$, де $\boldsymbol{x}$ - отриманий розв’язок.

Розв’язати задану систему рівнянь за допомогою програмного забезпечення Mathcad. Навести результат перевірки: вектор нев’язки $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Ax_m}$, де $\boldsymbol{x_m}$ - отриманий у Mathcad розв’язок.

Порівняти корені рівнянь, отримані у Mathcad, із власними результатами за допомогою методу середньоквадратичної похибки.

2 Розв’язок

Оскільки матриця не є матрицею із діагональною перевагою, я приводжу систему до еквівалентної, у якій є діагональна перевага:

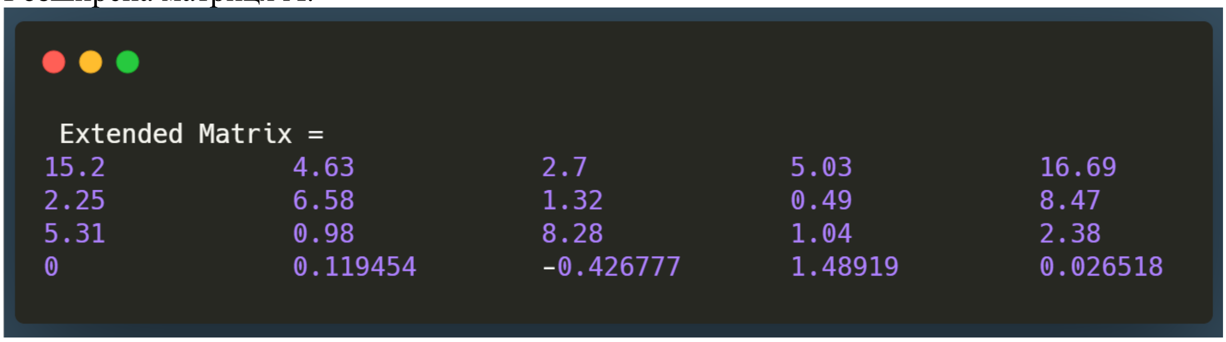


Остаточний варіант системи рівнянь:

$$\begin{bmatrix} 15.2 & 4.63 & 2.7 & 5.03 \\ 2.25 & 6.58 & 1.32 & 0.49 \\ 5.31 & 0.98 & 8.28 & 1.04 \\ 0 & 0.119454 & -0.426777 & 1.48919 \end{bmatrix} \cdot X = \begin{bmatrix} 16.69 \\ 8.47 \\ 2.38 \\ 0.026518 \end{bmatrix}$$

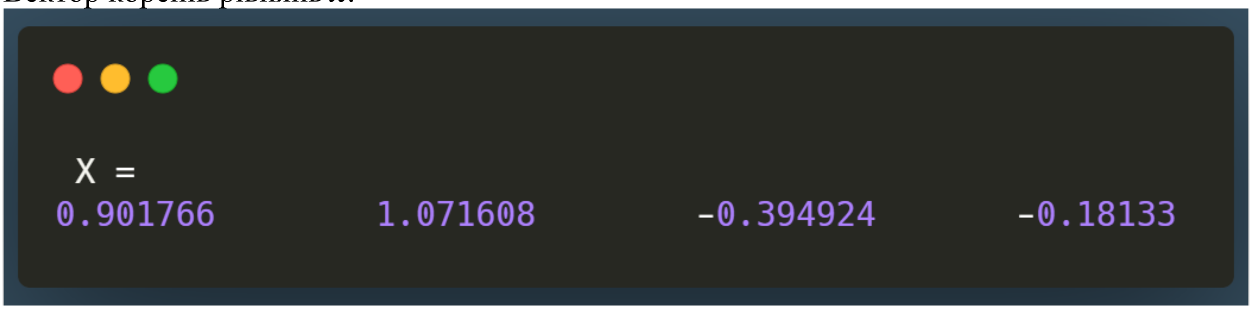
Нижче наведені результати виконання програми.

Розширена матриця А:



```
 Extended Matrix =
15.2        4.63        2.7         5.03        16.69
2.25        6.58        1.32        0.49        8.47
5.31        0.98        8.28        1.04        2.38
0           0.119454    -0.426777   1.48919     0.026518
```

Вектор коренів рівнянь ***x***:



```
 X =
0.901766      1.071608      -0.394924      -0.18133
```

Вектор нев'язки ***r* = *b* – *Ax*:**

```
R =
 [-0.000004 -0.000003  0.000001  0.       ]
```

Перші три та остання ітерації:

```
 X =
1.098026         0.911769         -0.524644        -0.205684
R =
 [-1.770356  0.793317  0.213912  0.       ]

 X =
0.981555         1.072161         -0.443099        -0.19518
R =
 [-1.015619 -0.112789 -0.010928  0.       ]

 X =
0.914738         1.077868         -0.402244        -0.18393
R =
 [-0.19332  -0.059444 -0.011702  0.000001]

  ...

R =
 [-0.000004 -0.000003  0.000001  0.       ]

 X =
0.901766         1.071608         -0.394924        -0.1813
```

Увесь вивід:

```
 Start Matrix =
15.2            4.63            2.7             5.03
2.25            6.58            1.32            0.49
5.31            0.98            8.28            1.04
0               0.119454        -0.426777       1.48919

 Right Part =
16.69   8.47    2.38    0.026518

 Rows = 4
 Columns = 4
 Exteneded Rows = 4
 Exteneded Columns = 5
 n = 4

 Extended Matrix =
15.2            4.63            2.7             5.03            16.69
2.25            6.58            1.32            0.49            8.47
5.31            0.98            8.28            1.04            2.38
0               0.119454        -0.426777       1.48919         0.02651
```

```
 X =
1.098026        0.911769        -0.524644       -0.205684
R =
 [-1.770356  0.793317  0.213912  0.       ]

 X =
0.981555        1.072161        -0.443099       -0.19518
R =
 [-1.015619 -0.112789 -0.010928  0.       ]

 X =
0.914738        1.077868        -0.402244       -0.18393
R =
 [-0.19332  -0.059444 -0.011702  0.000001]
R =
 [-0.010425 -0.010842 -0.002563  0.000001]
R =
 [ 0.004738 -0.000492 -0.00021   0.       ]
```

```
R =
 [0.001606 0.000288 0.000046 0.       ]
R =
 [ 0.000223  0.000095  0.000016 -0.000001]
R =
 [-0.000002  0.000016  0.000007  0.       ]
R =
 [-0.000014  0.000002 -0.000004  0.000001]
R =
 [-0.000004 -0.000003  0.000001  0.       ]
R =
 [-0.000004 -0.000003  0.000001  0.       ]

 X =
0.901766        1.071608        -0.394924       -0.18133
Total iterations =  11
```

З вигляду вектору нев’язки випливає, що метод Зейделя є доволі точним для розв’язання систем з матрицею A, але лише з певною точністю.

3 Розв’язок у Mathcad

Нижче наведено розв’язок системи у Mathcad

$k := 9 - 5 = 4 \quad \alpha := 0.5 \; k = 2 \quad \beta := \alpha = 2 \quad \mathbf{ORIGIN} := 1 \quad N := 4$

$$matrix := \begin{bmatrix} 3.81 & 0.25 & 1.28 & 0.75+\alpha \\ 2.25 & 1.32 & 4.58+\alpha & 0.49 \\ 5.31 & 6.28+\alpha & 0.98 & 1.04 \\ 9.39+\alpha & 2.45 & 3.35 & 2.28 \end{bmatrix} = \begin{bmatrix} 3.81 & 0.25 & 1.28 & 2.75 \\ 2.25 & 1.32 & 6.58 & 0.49 \\ 5.31 & 8.28 & 0.98 & 1.04 \\ 11.39 & 2.45 & 3.35 & 2.28 \end{bmatrix} \qquad X := \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$rightPart := \begin{bmatrix} 4.21 \\ 6.47+\beta \\ 2.38 \\ 10.48+\beta \end{bmatrix} = \begin{bmatrix} 4.21 \\ 8.47 \\ 2.38 \\ 12.48 \end{bmatrix} \qquad extenededMatrix := \mathrm{augment}(matrix, rightPart)$$

$$extenededMatrix = \begin{bmatrix} 3.81 & 0.25 & 1.28 & 2.75 & 4.21 \\ 2.25 & 1.32 & 6.58 & 0.49 & 8.47 \\ 5.31 & 8.28 & 0.98 & 1.04 & 2.38 \\ 11.39 & 2.45 & 3.35 & 2.28 & 12.48 \end{bmatrix}$$

$$matrixDiagonal := \begin{bmatrix} 15.2 & 4.63 & 2.7 & 5.03 \\ 2.25 & 6.58 & 1.32 & 0.49 \\ 5.31 & 0.98 & 8.28 & 1.04 \\ 0 & 0.119454 & -0.426777 & 1.48919 \end{bmatrix} \qquad rightPartDiagonal := \begin{bmatrix} 16.69 \\ 8.47 \\ 2.38 \\ 0.026518 \end{bmatrix}$$

$$extenededMatrixDiagonal := \mathrm{augment}(matrixDiagonal, rightPartDiagonal)$$

$$extenededMatrixDiagonal = \begin{bmatrix} 15.2 & 4.63 & 2.7 & 5.03 & 16.69 \\ 2.25 & 6.58 & 1.32 & 0.49 & 8.47 \\ 5.31 & 0.98 & 8.28 & 1.04 & 2.38 \\ 0 & 0.119454 & -0.426777 & 1.48919 & 0.026518 \end{bmatrix}$$

Guess Values

$$X$$

Constraints

$$matrixDiagonal \cdot X = rightPartDiagonal$$

Solver

$$\mathbf{X} := \mathbf{find}(X) = \begin{bmatrix} 0.901766 \\ 1.071608 \\ -0.394924 \\ -0.18133 \end{bmatrix}$$

$$r := rightPartDiagonal - (matrixDiagonal \cdot \mathbf{X})$$

$$r = \begin{bmatrix} 0 \\ 0 \\ 0.000000000000001 \\ 0 \end{bmatrix}$$

$$x := \begin{bmatrix} 0.901766 \\ 1.071608 \\ -0.394924 \\ -0.18133 \end{bmatrix} \qquad xm := \mathbf{X}$$

$$\delta := \sqrt{\frac{1}{N} \sum_{k=1}^{N} \left(x_k - xm_k\right)^2}$$

$$\delta = 0.000000354520393$$

$$\delta = 0$$

Вектор нев’язки з округлення:

$$r = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Вектор нев’язки без округлення:

$$r = \begin{bmatrix} 0 \\ 0 \\ 0.000000000000001 \\ 0 \end{bmatrix}$$

Порівняння отриманого результату (п. 2) із результатом з Mathcad за допомогою методу середньоквадратичної похибки без округлення:

$$x := \begin{bmatrix} 0.901766 \\ 1.071608 \\ -0.394924 \\ -0.18133 \end{bmatrix} \qquad xm := \mathbf{X}$$

$$\delta := \sqrt{\frac{1}{N} \sum_{k=1}^{N} \left(x_k - xm_k\right)^2}$$

$$\delta = 0.000000354520393$$

Та з округленням:

$$x := \begin{bmatrix} 0.901766 \\ 1.071608 \\ -0.394924 \\ -0.18133 \end{bmatrix} \qquad xm := \mathbf{X}$$

$$\delta := \sqrt{\frac{1}{N} \sum_{k=1}^{N} \left(x_k - xm_k\right)^2}$$

$$\delta = 0$$

У технічних розрахунках точність вимірювань характеризують відносною похибкою. Результат вважають гарним, якщо відносна похибка не перевищує 0,1 %. Отже Наш результат є гарним

4 Лістинг програми

Lab3.py

```
# region Starting Values
# need for multiplicating matrices in the end
import numpy as np
np.set_printoptions(suppress=True)

matrix = [[3.81, 0.25, 1.28, 2.75],
        [2.25, 1.32, 6.58, 0.49],
        [5.31, 8.28, 0.98, 1.04],
        [11.39, 2.45, 3.35, 2.28]]

rightPart = [4.21, 8.47, 2.38, 12.48]

matrixDiagonal =    [[15.2, 4.63, 2.7, 5.03],
                [2.25, 6.58, 1.32, 0.49],
                [5.31, 0.98, 8.28, 1.04],
                [0, 0.119454, -0.426777, 1.48919]]

rightPartDiagonal = [16.69, 8.47, 2.38, 0.026518]

N = len(matrixDiagonal)
difference = [0] * N
X = [0] * N
rounding = 6
vectorToShow = 3
doOperations = True
epsilonValue = 0.000001 # 10^(-6)
iterations = 0

# endregion Starting Values

# copy matrix to create extended matrix
extendedMatrix = list(map(list, matrixDiagonal))

# region Prints

# Print vector
def PrintVector(vectorName,vector):
   print("\n",vectorName,"=")
   for i in vector:
      print(i, end = " \t")
   print()

# print matrix
def PrintMatrix(matrixName,matrix):
   print("\n",matrixName,"=")
   for i in matrix:
      for j in i:
         print(j, end=" \t")
         if len(str(j)) <= 5:
            print(end=" \t")
      print()

# print additional parametrs
```

```
def PrintParametrs():
    print("\n Rows =", rows)
    print(" Columns =", columns)
    print(" Exteneded Rows =", extendedRows)
    print(" Exteneded Columns =", extendedColumns)
    print(" n =", N)

# just printing
def PrintAll():
    PrintMatrix("Start Matrix",matrix)

    PrintVector("Right Part", rightPart)

    PrintParametrs()

    PrintMatrix("Extended Matrix",extendedMatrix)

# endregion Prints

# region Check the results
def Residual():
    multiplied = np.round(np.dot(matrixDiagonal,X),rounding)

    R = np.round(np.subtract(rightPartDiagonal,multiplied),rounding)

    print("R =\n",R)

# endregion Check the results

# add right part to main matrix
RPCounter = 0
while RPCounter < len(rightPartDiagonal):
    extendedMatrix[RPCounter].append(rightPartDiagonal[RPCounter])
    RPCounter += 1

# region Getting rows and columns

rows = len(matrixDiagonal)
columns = len(matrixDiagonal)

extendedRows = len(extendedMatrix)
extendedColumns = len(extendedMatrix)+1

# endregion Getting rows and columns

PrintAll()

while doOperations:
    tempX = X.copy()
    for c in range(0, N):
        #temporal variable to store rightPart element
        element  = rightPartDiagonal[c]

        # calculate every element in array
        for z in range(0, N):
            if(c != z):
                element  = element - (matrixDiagonal[c][z] * X[z])
        # create new value
```

```
        X[c] = round(element  / matrixDiagonal[c][c], rounding)

    # compare two vectors
    for k in range(N):
        difference[k] = abs(tempX[k] - X[k])
        if max(difference) < epsilonValue:
            doOperations = False

    # show first three iterations of vector X
    if iterations < vectorToShow:
        PrintVector("X", X)
    iterations = iterations + 1
    Residual()

PrintVector("X", X)
print("Total iterations = ", iterations)
```

Висновок:

Я навчився розв’язувати СЛАР ітераційними методами (метод Зейделя), зрозумів, що метод Зейделя можна вважати покращеним методом Якобі, оскільки значення x відразу підставляються, отримав більше знань для роботи з MathCad та на практиці з’ясував, як порівнювати результати методом середньоквадратичної похибки.