Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. І. Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

Лабораторна робота №2
з дисципліни «Чисельні методи в моделюванні енергетичних процесів»
«Розв’язання систем лінійних алгебраїчних рівнянь (СЛАР) ітераційними методами. Метод простої ітерації. Метод Зейделя»
Варіант №1

<u>Виконав:</u>
студент 2-го курсу, ТЕФ
групи ТР-15
Руденко В.І.

КИЇВ-2022

Мета роботи Якщо матриця не є матрицею із діагональною перевагою, привести систему до еквівалентної, у якій є діагональна перевага (письмово). Можна, наприклад, провести одну ітерацію метода Гауса, зкомбінувавши рядки з метою отримати нульовий недіагональний елемент у стовпчику. Розробити програму, що реалізує розв'язання за ітераційним методом, який відповідає заданому варіанту. Обчислення проводити з з кількістю значущих цифр m = 6. Для кожної ітерації розраховувати нев'язку r = b – Ax, де x - отриманий розв'язок. Розв'язати задану систему рівнянь за допомогою програмного забезпечення Mathcad. Навести результат перевірки: вектор нев'язки r = b – Axm, де xm - отриманий у Mathcad розв'язок. Порівняти корені рівнянь, отримані у Mathcad, із власними результатами за допомогою методу середньоквадратичної похибки.

Варіант завдання (Вихідна система рівнянь).

№ вар.	Матриця системи A	Вектор правої частини b
1-4	$\begin{pmatrix} 5{,}18+\alpha & 1{,}12 & 0{,}95 & 1{,}32 & 0{,}83 \\ 1{,}12 & 4{,}28-\alpha & 2{,}12 & 0{,}57 & 0{,}91 \\ 0{,}95 & 2{,}12 & 6{,}13+\alpha & 1{,}29 & 1{,}57 \\ 1{,}32 & 0{,}57 & 1{,}29 & 4{,}57-\alpha & 1{,}25 \\ 0{,}83 & 0{,}91 & 1{,}57 & 1{,}25 & 5{,}21+\alpha \end{pmatrix}$ $\alpha = 0{,}25k,\ k = №вар-1$	$\begin{pmatrix} 6{,}19+\beta \\ 3{,}21 \\ 4{,}28-\beta \\ 6{,}25 \\ 4{,}95+\beta \end{pmatrix}$ $\beta = 0{,}35k,\ k = №вар-1$

Теоретична частина

Ітераційними методами є такі, що навіть у припущенні, що обчислення ведуться без округлень, дозволяють отримати розв'язок системи лише із заданою точністю. До таких методів відносяться метод простої ітерації (метод Якобі) та метод Зейделя. Будемо розглядати системи вигляду Ax = b,(1) де A (n×n) - матриця системи, b - вектор правої частини, x - вектор розв'язку.

Деяка модифікація методу простої ітерації. Основна ідея в тому, що при обчисленні -го наближення невідомої враховуються уже обчислені раніше наближення невідомих, тобто виконуютьсяпослідовні ітерації.

Схема методу Зейделя для системи

$$
\begin{aligned}
x_1^{k+1} &= \phi_1(x_1^k, x_2^k, \ldots, x_n^k) \\
x_2^{k+1} &= \phi_2(x_1^{k+1}, x_2^k, \ldots, x_n^k) \\
&\ldots \\
x_n^{k+1} &= \phi_n(x_1^{k+1}, x_2^{k+1}, \ldots, x_{n-1}^{k+1}, x_n^k)
\end{aligned}
$$

Умова закінчення ітерацій $\Delta^{k+1} = \max_i \left| x_i^{k+1} - x_i^k \right| \le \varepsilon$.

Вказана вище теорема збірностей для простих ітерацій залишається вірною для ітерацій за методом Зейделя.

Ітераційний процес **методу Зейделя** необхідно продовжувати до тих пір, поки не буде виконуватись умова , де   задана точність обчислювального процесу.

Цей метод має кращу збіжність, ніж метод простих ітерацій, але приводить до більш об’ємних обчислень. Процес Зейделя може бути збіжним навіть у тому випадку, коли процес ітерацій розбіжний. Можливі випадки, коли метод Зейделя збігається і повільніше процесу ітерації, і навіть розбіжний по Зейделю.

Проміжні результати виконання:

```
-------------------------------------------
Basic Array:
   5.180000   1.120000   0.950000   1.320000   0.830000   6.190000
   1.120000   4.280000   2.120000   0.570000   0.910000   3.210000
   0.950000   2.120000   6.130000   1.290000   1.570000   4.280000
   1.320000   0.570000   1.290000   4.570000   1.250000   6.250000
   0.830000   0.910000   1.570000   1.250000   5.210000   4.950000
-------------------------------------------
        Diagonal advantage - row(1): Passed
        Diagonal advantage - row(2): Fail
        Diagonal advantage - row(3): Passed
        Diagonal advantage - row(4): Passed
        Diagonal advantage - row(5): Passed
-------------------------------------------
Matrix with diagonal advantage: false
Fix Log:
        Deduction Row (2)  with multiplier:  0.216216
        Divide diagonal in row (2) to "one"
-------------------------------------------
Fixed Array:
   5.180000   1.120000   0.950000   1.320000   0.830000   6.190000
   0.000000   1.000000   0.474163   0.070482   0.180924   0.463521
   0.950000   2.120000   6.130000   1.290000   1.570000   4.280000
   1.320000   0.570000   1.290000   4.570000   1.250000   6.250000
   0.830000   0.910000   1.570000   1.250000   5.210000   4.950000
-------------------------------------------
```

```
-------------------------------------------
Divide diagonals to "one"
   1.000000   0.216216   0.183398   0.254826   0.160232   1.194981
   0.000000   1.000000   0.474163   0.070482   0.180924   0.463521
   0.154976   0.345840   1.000000   0.210440   0.256117   0.698206
   0.288840   0.124726   0.282276   1.000000   0.273523   1.367615
   0.159309   0.174664   0.301344   0.239923   1.000000   0.950096
-------------------------------------------
        Diagonal advantage - row(1): Passed
        Diagonal advantage - row(2): Passed
        Diagonal advantage - row(3): Passed
        Diagonal advantage - row(4): Passed
        Diagonal advantage - row(5): Passed
-------------------------------------------
Matrix with diagonal advantage: true
-------------------------------------------
Approximation (1.000000) : 1.194981 0.463521 0.352709 0.865082 0.364925
Approximation (2.000000) : 0.751156 0.169283 0.247738 0.959791 0.495932
Approximation (3.000000) : 0.788900 0.188679 0.181697 0.929278 0.513753
Approximation (4.000000) : 0.801739 0.218919 0.171106 0.919913 0.511864
Approximation (5.000000) : 0.799832 0.224943 0.171773 0.920041 0.510884
Approximation (6.000000) : 0.798531 0.224795 0.172249 0.920569 0.510847
Approximation (7.000000) : 0.798347 0.224539 0.172265 0.920659 0.510895
Approximation (8.000000) : 0.798369 0.224516 0.172238 0.920650 0.510905
Approximation (9.000000) : 0.798379 0.224528 0.172232 0.920645 0.510905
Approximation (10.000000) : 0.798380 0.224531 0.172232 0.920645 0.510904
```

```
Rusult:
        X(1) : 0.798380
        X(2) : 0.224531
        X(3) : 0.172232
        X(4) : 0.920645
        X(5) : 0.510904

Accuracy (delta X):
        X(1) : 0.000000
        X(2) : 0.000004
        X(3) : 0.000000
        X(4) : 0.000000
        X(5) : 0.000001
```

```
SQRT Accuracy (delta X):0.000002
```

Розв’язок задачі у середовищі MathCAD 14

$$ORIGIN := 1$$

$$Ar := augment(A, B) \qquad Ar = \begin{pmatrix} 5.18 & 1.12 & 0.95 & 1.32 & 0.83 & 6.19 \\ 1.12 & 4.28 & 2.12 & 0.57 & 0.91 & 3.21 \\ 0.95 & 2.12 & 6.13 & 1.29 & 1.57 & 4.28 \\ 1.32 & 0.57 & 1.29 & 4.57 & 1.25 & 6.25 \\ 0.83 & 0.91 & 1.57 & 1.25 & 5.21 & 4.95 \end{pmatrix}$$

$$Ag = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0.798379 \\ 0 & 1 & 0 & 0 & 0 & 0.224531 \\ 0 & 0 & 1 & 0 & 0 & 0.172232 \\ 0 & 0 & 0 & 1 & 0 & 0.920645 \\ 0 & 0 & 0 & 0 & 1 & 0.510904 \end{pmatrix}$$

$$a = \begin{pmatrix} 0.798379 \\ 0.224531 \\ 0.172232 \\ 0.920645 \\ 0.510904 \end{pmatrix} \qquad P := \begin{pmatrix} 0.798380 \\ 0.224531 \\ 0.172232 \\ 0.920645 \\ 0.510904 \end{pmatrix}$$

$$A \cdot P - B = \begin{pmatrix} 5.24 \times 10^{-6} \\ 4.1 \times 10^{-7} \\ 2.1 \times 10^{-7} \\ 1.2 \times 10^{-6} \\ -1.06 \times 10^{-6} \end{pmatrix} \qquad stderr(a, P) = 5.573 \times 10^{-7}$$

MathCad 1 Вектор Нев'язки фінальних результатів

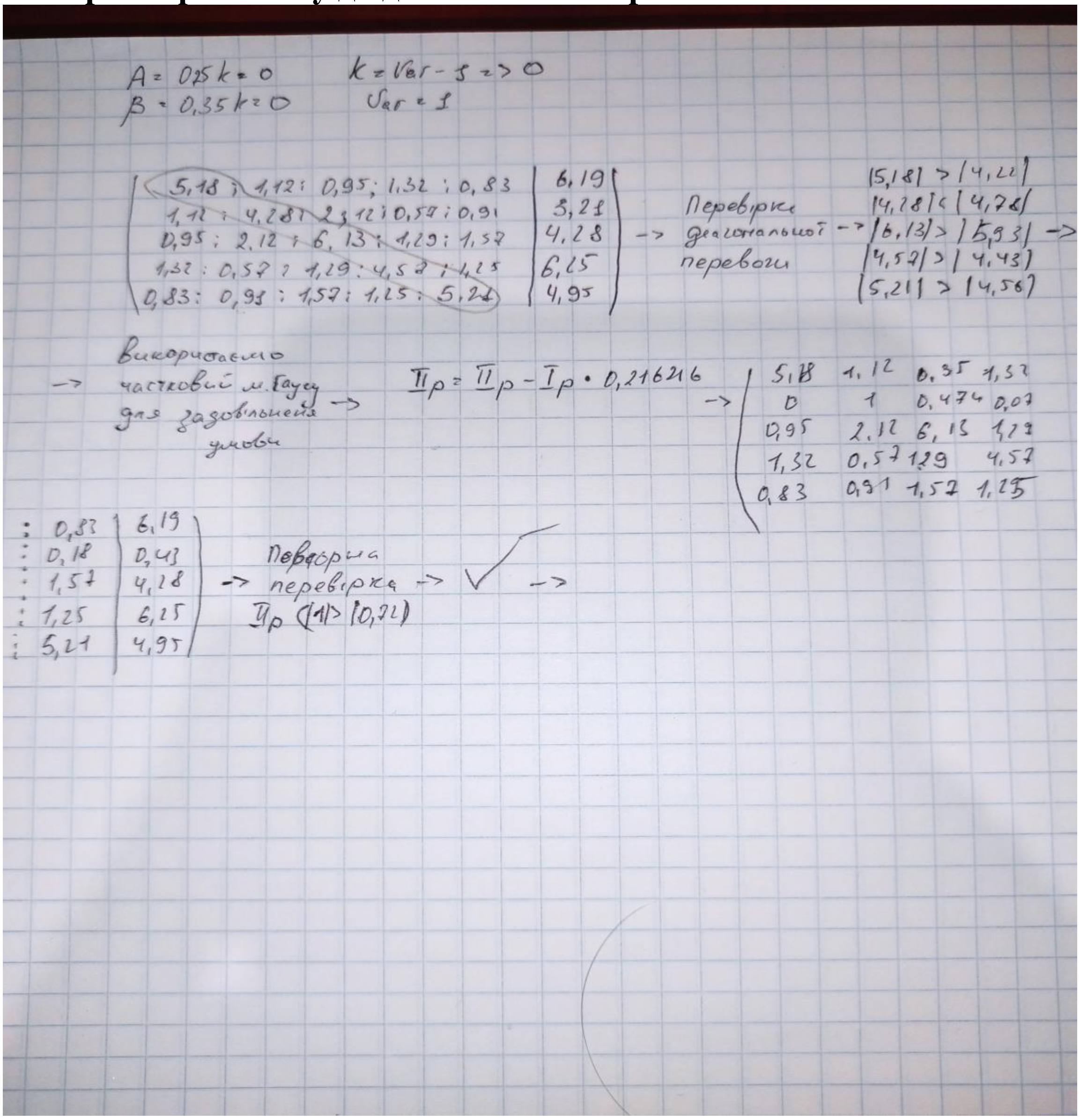
$$P := \begin{pmatrix} 1.194981 \\ 0.463521 \\ 0.352709 \\ 0.865082 \\ 0.364925 \end{pmatrix} \qquad A \cdot P - B = \begin{pmatrix} 2.299 \\ 1.685 \\ 1.689 \\ 0.456 \\ 3.22 \times 10^{-6} \end{pmatrix} \qquad stderr(a, P) = 0.252$$

$$P := \begin{pmatrix} 0.751156 \\ 0.169283 \\ 0.247738 \\ 0.959791 \\ 0.495932 \end{pmatrix} \qquad A \cdot P - B = \begin{pmatrix} -0.196 \\ -0.121 \\ 0.328 \\ 0.164 \\ 1.4 \times 10^{-7} \end{pmatrix} \qquad stderr(a, P) = 0.065$$

$$P := \begin{pmatrix} 0.788900 \\ 0.188679 \\ 0.181697 \\ 0.929278 \\ 0.513753 \end{pmatrix} \qquad A \cdot P - B = \begin{pmatrix} -0.067 \\ -0.136 \\ -0.011 \\ 0.022 \\ -1.9 \times 10^{-7} \end{pmatrix} \qquad stderr(a, P) = 0.021$$

MathCad 2 Вектор нев'язки перших трьох наближень

Власний варіант розв’язку до діагональної переваги:



Порівняння результатів

	Програма	**MatchCad**
Середньоквадратична похибка	**2*10^7**	**5.57*10^7**

Лістинг програми:

```
#include <iomanip>
#include <iostream>


using namespace std;

#define k 0
#define A 0.25*k
```

```cpp
#define B 0.35*k

#define COLUMN 6
#define ROW 5

#define TEST_off

void ShowArray(float Array[5][6], string Massage)
{
    cout.setf(ios::fixed);
    cout.precision(6);
    cout << "---------------------------------------------" << endl <<  Massage <<
endl;
    for(int i=0;i<ROW;i++)
    {
        for(int j=0;j<COLUMN;j++)
            cout << setw(11) << Array[i][j];
        cout << endl;
    }
    cout << "---------------------------------------------" << endl;
}

bool CheakIterationMethod(float Array[ROW][COLUMN], bool (&BadRowMark)[ROW]);
void UpperGauseFix(float (&Array)[ROW][COLUMN], bool (&BadRowMark)[ROW]);
void MatrixFixer(float (&Array)[ROW][COLUMN], bool (&BadRowMark)[ROW]);
void ZeidelSolve(float (&Array)[ROW][COLUMN],float (&XResultArray)[3][ROW]);

int main(int argc, char* argv[])
{
    bool BadRowMark[ROW];
    float XResultArray[3][ROW];
    float Array[ROW][COLUMN] =
        {
            {5.18+A,1.12,0.95,1.32,0.83,6.19+B},
            {1.12,4.28-A,2.12,0.57,0.91, 3.21},
            {0.95,2.12,6.13+A,1.29,1.57,4.28-B},
            {1.32,0.57,1.29,4.57-A,1.25,6.25},
            {0.83,0.91,1.57,1.25,5.21+A,4.95+B}
        };
        ShowArray(Array,"Basic Array: ");
        if(!CheakIterationMethod(Array, BadRowMark))
        {
            MatrixFixer(Array, BadRowMark);
        }
        ZeidelSolve(Array,XResultArray);
}

bool CheakIterationMethod(float Array[ROW][COLUMN], bool (&BadRowMark)[ROW])
{
    bool reCheck = true;
    float ChecValue;

    for(int i=0;i<ROW;i++)
    {
        float sum=0;
        for(int j=0;j<ROW;j++)
        {
            if(i==j)
                ChecValue = abs(Array[i][j]);
            else
                sum += abs(Array[i][j]);
```

```
        }
        if(ChecValue>sum)
        {
            BadRowMark[i]=false;
            if(reCheck)
                reCheck = true;
        }
        else
        {
            reCheck = false;
            BadRowMark[i]=true;
        }

/////////////////////////////////////////
#ifndef TEST
        cout << "\tDiagonal advantage - row(" << i+1 << "): " << (ChecValue>sum ?
"Passed" : "Fail") << endl;
#endif
    }
#ifndef TEST
    cout << "--------------------------------------------" << endl;
#endif
/////////////////////////////////////////
    cout << "Matrix with diagonal advantage: " << (reCheck ? "true" : "false") <<
endl;
    return (reCheck ? true : false);
}

void UpperGauseFix(float (& Array)[ROW][COLUMN], bool (& BadRowMark)[ROW])
{
    double Temp[COLUMN];
    for (int i = ROW - 1; i >= 0; i--)
    {
        for (int j = i - 1; j >= 0; j--) {
            if(!BadRowMark[j]) continue;
            double multiplier = Array[j][i] / Array[i][i];
            cout << "\tDeduction Row (" << i+1 << ")  with multiplier:  " <<
multiplier << endl;
            for (int n = 0; n < COLUMN; n++)
            {
                Temp[n] = Array[i][n] * multiplier;
                Array[j][n] -= Temp[n];
            }
        }
    }
    ShowArray(Array, "*Second option* Fixed Array:");
}


void MatrixFixer(float (&Array)[ROW][COLUMN], bool (&BadRowMark)[ROW])
{
    float temp[COLUMN];
    cout << "Fix Log:" << endl;
    for (int i = 0; i < COLUMN - 1; i++)
    {
        float divisor = Array[i][i];

        if(BadRowMark[i])
        {
            cout << "\tDivide diagonal in row ("<< i+1 <<") to \"one\"" << endl;
            for(int j=0;j<COLUMN;j++)
```

```
                Array[i][j] /= divisor;
        }

        if(i+1==ROW || !BadRowMark[i+1]) continue;
        float multiplier = Array[i+1][i] / Array[i][i];
        cout << "\tDeduction Row (" << i+2 << ")  with multiplier:  " << multiplier
<< endl;
        for (int n = 0; n < COLUMN; n++)
        {
            temp[n] = Array[i][n] * multiplier;
            Array[i+1][n] -= temp[n];
        }
    }
    ShowArray(Array, "Fixed Array:");


for(int i=0;i<ROW;i++)
{
    float del = Array[i][i];
    for(int j=0;j<COLUMN;j++)
        Array[i][j]/=del;
}
    ShowArray(Array, "Divide diagonals to \"one\"");
    if(!CheakIterationMethod(Array, BadRowMark))
        UpperGauseFix(Array,BadRowMark);
}

void ZeidelSolve(float (&Array)[ROW][COLUMN],float (&XResultArray)[3][ROW])
{
    cout << "----------------------------------------------" << endl;
    for(int i=0;i<3;i++)
        for(int j=0;j<ROW;j++)
            XResultArray[i][j]=0;

    float XSUM,p=0;
    do
    {
        XSUM=0;
        cout << "Approximation (" << ++p << ") : ";
        for(int i=0;i<ROW;i++)
        {
            float SUM=0;
            for(int j=0;j<ROW;j++)
            {
                if(i>j) SUM+=Array[i][j]*XResultArray[2][j];
                else if(i<j) SUM+=Array[i][j]*XResultArray[1][j];
            }
            XResultArray[2][i] = Array[i][COLUMN-1]-SUM;
            XResultArray[0][i] = abs(XResultArray[2][i]-XResultArray[1][i]);
            XResultArray[1][i]=XResultArray[2][i];
            cout << XResultArray[1][i] << " ";
            XSUM += XResultArray[0][i];
        }
        cout << endl;
    }while(XSUM >= 0.00001);

    cout << endl << "Rusult:" << endl;
    for(int i=0;i<ROW;i++)
        cout << "\tX(" << i+1 << ") : " << XResultArray[1][i] << endl;
    cout << endl << "Accuracy (delta X):" << endl;
    for(int i=0;i<ROW;i++)
```

```
        cout << "\tX(" << i+1 << ") : " <<  XResultArray[0][i] << endl;
    XSUM=0;
    for(int i=0;i<ROW;i++)
        XSUM+=pow(XResultArray[0][i],2);
    double AccuracyResult = sqrt(XSUM/ROW);
    cout << endl << "SQRT Accuracy (delta X):" << AccuracyResult << endl;
}
```