Jakub Niżyński

Metody numeryczne

Zadanie 2
- rozkład LU macierzy z pivotingiem

Grupa poniedziałkowa

Dane wejściowe:

Macierz kwadratowa, dowolnego rozmiaru: 4 x 4, 5 x 5, 6 x 6, 7 x 7, itd.

Uruchamiając program podajemy rozmiar macierzy (tutaj: 4, ale program działa też dla większych) oraz wpisujemy elementy macierzy. Elementy macierzy oddzielamy spacjami, każdy wiersz w osobnej linii.

Przykładowe rozwiązanie dla macierzy z wykładu:

$$\begin{bmatrix} 2 & 4 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 0 & 1 & 2 & -1 \\ -1 & 1 & 0 & 1 \end{bmatrix}$$

Kod programu w języku Java:

```
import java.util.Scanner;

public class LUdecomposition {
    int matrixSize;
    double A[][];
    int[][] matrixOrder;
    double temp[][];
    double L[][];
    double U[][];


    void show(double table[][]) {
        for (double[] tab1 : table) {
            System.out.print("|");
            for (int j = 0; j < tab1.length; j++) {
                System.out.print(tab1[j]);
                if (j != tab1.length - 1) {
                    System.out.print("\t");
                }
            }
            System.out.print("|");
            System.out.println();
        }
    }
```

```
void showP() {
    for (int i = 0; i < matrixOrder.length; i++) {
        System.out.print("|");
        for (int j = 0; j < matrixOrder[i].length; j++) {
            System.out.print(matrixOrder[j][i]);
            if (j != matrixOrder[i].length - 1) {
                System.out.print("\t");
            }
        }
        System.out.print("|");
        System.out.println();
    }
}


LUdecomposition(double[][] macierz) {
    matrixSize = macierz.length;
    A = macierz;
    matrixOrder = new int[matrixSize][matrixSize];
    temp = new double[matrixSize][matrixSize];
    int i = 0;
    while (i < matrixSize) {
        System.arraycopy(A[i], 0, temp[i], 0, matrixSize);
        i++;
    }
    int j = 0;
    while (j < matrixSize) {
        matrixOrder[j][j] = 1;
        j++;
    }
    L = new double[matrixSize][matrixSize];
    U = new double[matrixSize][matrixSize];
    int k = 0;
    while (k < matrixSize) {
        L[k][k] = 1;
        k++;
    }
}


void findPermutation(int k) {
    double max = 0;
    int column = -1, line = -1;
    for (int i = k; i < matrixSize; i++) {
        for (int j = k; j < matrixSize; j++) {
            if (Math.abs(A[i][j]) > max) {
                line = i;
                column = j;
                max = Math.abs(A[i][j]);
            }
        }
    }
    if (line != -1) {
        doVersePermutation(k, line);
    }
    if (column != -1) {
        doColumnPermutation(k, column);
    }
}
```

```
void doVersePermutation(int a, int b) {
    double tempA[];
    tempA = A[a];
    A[a] = A[b];
    A[b] = tempA;
    int tempO[];
    tempO = matrixOrder[a];
    matrixOrder[a] = matrixOrder[b];
    matrixOrder[b] = tempO;
    int i = 0;
    while (i < Math.min(a, b)) {
        double tempL = L[a][i];
        L[a][i] = L[b][i];
        L[b][i] = tempL;
        i++;
    }
}


void doColumnPermutation(int a, int b) {
    int i = 0;
    while (i < matrixSize) {
        double tempA = A[i][a];
        A[i][a] = A[i][b];
        A[i][b] = tempA;
        int tempO = matrixOrder[i][a];
        matrixOrder[i][a] = matrixOrder[i][b];
        matrixOrder[i][b] = tempO;
        i++;
    }
    int j = Math.min(a, b);
    while (j < matrixSize) {
        double tempU = U[j][a];
        U[j][a] = U[j][b];
        U[j][b] = tempU;
        j++;
    }
}


public void printLU() {
    System.out.println("P:");
    showP();
    System.out.println("L:");
    show(L);
    System.out.println("U:");
    show(U);
}


public void doLU() {
    for (int i = 0; i < matrixSize; i++) {
        if (A[i][i] == 0) {
            findPermutation(i);
            for (int j = i + 1; j < matrixSize; j++) {
                L[j][i] = A[j][i] / A[i][i];
            }
            System.arraycopy(A[i], i, U[i], i, matrixSize - i);
            for (int j = i + 1; j < matrixSize; j++) {
                for (int k = i + 1; k < matrixSize; k++) {
                    A[j][k] -= L[j][i] * U[i][k];
```

```
                }
            }
            for (int l = i + 1; l < matrixSize; l++) {
                findPermutation(l);
                for (int j = l + 1; j < matrixSize; j++) {
                    L[j][l] = A[j][l] / A[l][l];
                }
                System.arraycopy(A[l], l, U[l], l, matrixSize - l);
                for (int j = l + 1; j < matrixSize; j++) {
                    for (int k = l + 1; k < matrixSize; k++) {
                        A[j][k] -= L[j][l] * U[l][k];
                    }
                }
            }
            break;
        }
        for (int j = i + 1; j < matrixSize; j++) {
            L[j][i] = A[j][i] / A[i][i];
        }
        System.arraycopy(A[i], i, U[i], i, matrixSize - i);
        for (int j = i + 1; j < matrixSize; j++) {
            for (int k = i + 1; k < matrixSize; k++) {
                A[j][k] -= L[j][i] * U[i][k];
            }
        }
    }
}

public static void main(String[] args) {
    int matrix_size;

    Scanner scanner = new Scanner(System.in);
    System.out.println("Podaj rozmiar macierzy:");
    matrix_size = scanner.nextInt();

    double matrix[][] = new double[matrix_size][matrix_size];
    System.out.println("Podaj współczynniki wiersza macierzy
oddzielone spacjami: ");

    for (int i = 0; i < matrix_size; i++) {
        for (int j = 0; j < matrix_size; j++) {
            matrix[i][j] = scanner.nextInt();
        }
    }

    LUdecomposition A = new LUdecomposition(matrix);
    A.doLU();
    A.printLU();
}
}
```

Wynik działania programu:

Na wyjściu otrzymujemy macierz permutacji P oraz szukane trójkątne macierze L,U.

```
P:
|1      0       0       0|
|0      0       1       0|
|0      0       0       1|
|0      1       0       0|
L:
|1.0    0.0     0.0     0.0|
|-0.5   1.0     0.0     0.0|
|0.5    0.0     1.0     0.0|
|0.0    0.3333333333333333      0.7333333333333333      1.0|
U:
|2.0    4.0     1.0     1.0|
|0.0    3.0     0.5     1.5|
|0.0    0.0     2.5     0.5|
|0.0    0.0     0.0     -1.8666666666666667|

Process finished with exit code 0
```

Macierz L posiada same 1 na diagonali.