Laboratorium **Metod Obliczeniowych** Wydział Elektrotechniki Automatyki i Informatyki Politechnika Świętokrzyska	
Studia: **Stacjonarne I stopnia**	Kierunek: **Informatyka**
Data wykonania: **27.11.2017**	Grupa: **3ID13B**
Imię i nazwisko: **Bartłomiej Osak**	
Numer ćwiczenia: **7**	Temat ćwiczenia: **Układy równań liniowych**

1. Układy równań liniowych – metoda eliminacji Gaussa w środowisku MATLAB.

Polecenie: Rozwiąż układ równań metodą eliminacji Gaussa:

$$\begin{cases} 5.05x_1 - 6.92x_2 - 0.93x_3 = 17.9873 \\ 4.45x_1 + 4.34x_2 + 4.91x_3 = 23.2409 \\ -7.45x_1 + 9.96x_2 - 8.23x_3 = 30.7435 \end{cases}$$

Kod źródłowy:

```
function[X] = metoda_gaussa(A,B)
    C = [A B.’];
    n = size(A,2);
    for s = 1:n-1
        for i = s+1:n
            for j = s+1:n+1
                C(i,j) = C(i,j) - C(i,s) / C(s,s) * C(s,j);
            end
        end
    end
    A = C(1:n,1:n);
    B = C(1:n,n+1);
    X = zeros([n 1]);
    X(n) = B(n) / A(n,n);
    for i = n-1:-1:1
        sum = 0;
        for s = i+1:n
            sum = sum + A(i,s) * X(s);
        end
        X(i) = (B(i) - sum) / A(i,i);
    end
end
```

Wywołanie:

- **Przypisanie do macierzy A oraz wektora B wartości:**

  ```
  A=[[5.05 -6.92 -0.93];[4.45 4.34 4.91];[-7.45 9.96 -8.23]]
  B=[17.9873 23.2409 30.7435]
  ```

- **Wywołanie funkcji przypisując wyniki do dwóch zmiennych:**

  ```
  C = metoda_gaussa(A,B)
  ```

- **Wynik zadziałania:**

  ```
  C =
      7.9800
      4.0400
     -6.0700
  ```

Zrzut ekranu z działania programu:

```
Command Window
>> A=[[5.05 -6.92 -0.93];[4.45 4.34 4.91];[-7.45 9.96 -8.23]]

A =

    5.0500   -6.9200   -0.9300
    4.4500    4.3400    4.9100
   -7.4500    9.9600   -8.2300

>> B=[17.9873 23.2409 30.7435]

B =

   17.9873   23.2409   30.7435

>> C = metoda_gaussa(A,B)

C =

    7.9800
    4.0400
   -6.0700
```

Opis programu:

Program metoda_gaussa przyjmuje dwa argumenty wejściowe:

- A – macierz współczynników
- B – wektor wyników równań

Ponadto zwraca on wektor wynikowy dla każdego współczynnika x. Na początku do macierzy C przypisywane są macierz A oraz transponowany wektor B. Następnie do zmiennej n przypisujemy rozmiar pojedynczy macierzy A. Dalej deklarujemy pętle – trzy zagnieżdżone pętle iterujące zgodnie z ogólnym algorytmem metody eliminacji Gaussa:

$$\begin{cases} s = 1,2,\ldots,n-1 \\ \begin{cases} i = s+1, s+2, \ldots n \\ c_{ij}^{(s)} = c_{ij}^{(s-1)} - \dfrac{c_{is}^{(s-1)}}{c_{ss}^{(s-1)}} c_{sj}^{(s-1)}, j = s+1, s+2, \ldots, n+1 \end{cases} \end{cases}$$

Następnie wektor wynikowy X jest inicjowany zerowaniem go na rozmiarze n x 1. Następnie zgodnie z algorytmem obliczania układu równań trójkątnego wykonywane są obliczenia:

$$x_n = \frac{b_n}{a_{nn}}$$

$$x_i = \left(b_i - \frac{\sum_{s=i+1}^{n} a_{is} x_s}{a_{ii}}\right), i = n-1, n-1, \ldots, 1, gdzie\ a_{ii} \neq 0, i = 1,2,\ldots,n$$

Ostatecznie wypisywany jest transponowany wektor wynikowy X.

2. Układy równań liniowych – metoda eliminacji Gaussa w środowisku MATLAB.

Polecenie: Rozwiąż układ równań metodą eliminacji Gaussa:

$$\begin{cases} 5.05x_1 - 6.92x_2 - 0.93x_3 = 17.9873 \\ 4.45x_1 + 4.34x_2 + 4.91x_3 = 23.2409 \\ -7.45x_1 + 9.96x_2 - 8.23x_3 = 30.7435 \end{cases}$$

Kod źródłowy:

```
import java.util.Scanner;

public class Gauss {

    private static double[][] A;
    private static double[] B;
    private static double[] result;
    private static double x;
    private static double sum;
    private static int N;
    private static int max;
    private static Scanner in = new Scanner(System.in);

    public static void enterData() {
        boolean end = false;
        while (!end) {
            try {
                System.out.print("Wprowadź ilość niewiadomych: ");
                N = Integer.parseInt(in.next());
                A = new double[N][N];
                B = new double[N];
                System.out.println("Wprowadź " + N + " współczynników równań:");
                for (int i = 0; i < N; i++) {
                    for (int j = 0; j < N; j++) {
                        A[i][j] = Double.parseDouble(in.next());
                    }
                }
                System.out.println("Wprowadź " + N + " wyrazy wynikowe:");
                for (int i = 0; i < N; i++) {
                    B[i] = Double.parseDouble(in.next());
                }
                end = true;
            } catch (Exception e) {
                System.out.println("[BŁĄD] BŁĘDNIE WPISANO DANE!\n");
                for (int i = 0; i < 500; i++) {
                    System.out.println();
                }
            }
        }

    }

    public static void methodGauss() {
        N = B.length;
        for (int s = 0; s < N; s++) {
            max = s;
            for (int i = s + 1; i < N; i++) {
                if (Math.abs(A[i][s]) > Math.abs(A[max][s])) {
                    max = i;
                }
            }
            double[] tempA = A[s];
            A[s] = A[max];
            A[max] = tempA;
            double tempB = B[s];
            B[s] = B[max];
            B[max] = tempB;
            for (int i = s + 1; i < N; i++) {
                x = A[i][s] / A[s][s];
                B[i] -= x * B[s];
```

```java
                for (int j = s; j < N; j++) {
                    A[i][j] -= x * A[s][j];
                }
            }
        }
        result = new double[N];
        for (int i = N - 1; i >= 0; i--) {
            sum = 0.0;
            for (int s = i + 1; s < N; s++) {
                sum += A[i][s] * result[s];
            }
            result[i] = (B[i] - sum) / A[i][i];
        }
    }

    public static void printResult() {
        N = result.length;
        System.out.println("Wynik: ");
        for (int i = 0; i < N; i++) {
            System.out.println("x" + i + ": " + result[i]);
        }
    }

    public static void main(String[] args) {
        Gauss.enterData();
        Gauss.methodGauss();
        Gauss.printResult();
    }
}
```

Zrzut ekranu z działania programu:

```
Wprowadź ilość niewiadomych: 3
Wprowadź 3 współczynników równań:
5.05 -6.92 -0.93
4.45 4.34 4.91
-7.45 9.96 -8.23
Wprowadź 3 wyrazy wynikowe:
17.9873
23.2409
30.7435
Wynik:
x0: 7.98
x1: 4.04
x2: -6.07

Process finished with exit code 0
```

Opis programu:

Program składa się z jednej klasy publicznej o nazwie Gauss. Klasa ta posiada 8 pól statycznych. Są to:

- double[ ][ ] A – tablica dwuwymiarowa współczynników (macierz A)
- double[ ] B – tablica jednowymiarowa wyników układu współrzędnych (macierz B)
- double[ ] result – tablica jednowymiarowa przechowująca wynik
- double x – zmienna przechowująca wartość tymczasową
- int N – zmienna przechowująca ilość niewiadomych oraz długość wektora B
- int max – zmienna potrzebna na rzecz algorytmu
- Scanner in – zmienna typu Scanner inicjalizująca skaner do wprowadzania danych przez użytkownika

Ponadto klasa zawiera 3 metody statyczne:

- void enterData() – metoda do wprowadzania danych danych przez użytkownika. Program prosi użytkownika o podanie ilości niewiadomych, wprowadzanie współczynników równań oraz wprowadzanie wyrazów wynikowych. Metoda jest zabezpieczona przed podaniem danych w złym formacie – rzucany jest wyjątek ze stosownym komunikatem.

- methodGauss() – metoda odpowiedzialna za prawidłowe obliczenie niewiadomych. Bazuje ona na algorymie metody eliminacji Gaussa. Następnie rozwiązywany jest układ równań trójkątnych. Wzory zostały podane w punkcie 1 sprawozdania w opisie programu w środowisku MATLAB.
- printResult() – metoda odpowiedzialna za wypisanie zawartości tablicy result, która przechowuje ostateczny wynik, czyli wartości poszczególnych niewiadomych.

3. Metoda Jacobiego – przybliżona metoda rozwiązywania układu równań liniowych.

Biorąc $M = D$, gdzie D jest macierzą diagonalną składającą się z wyrazów stojących na głównej przekątnej macierzy A otrzymujemy (o ile na przekątnej macierzy A nie ma zera) metodę iteracyjną:

$$x_{k+1} = D^{-1}(b - (L + U)x_k),$$

zwaną **metodą Jacobiego.**

Rozpisując ją po współrzędnych, dostajemy układ rozszczepionych równań (numer iteracji wyjątkowo zaznaczamy w postaci górnego indeksu):

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}\right)$$

co znaczy dokładnie tyle, że w i-tym równaniu wyjściowego układu przyjmujemy za współrzędne x wartości poprzedniej iteracji i na tej podstawie wyznaczamy wartość x_i.

Widzimy więc, że metoda rzeczywiście jest prosta w implementacji, a dodatkowo jest w pełni równoległa: każdą współrzędną nowego przybliżenia możemy wyznaczyć niezależne od pozostałych.

Twierdzenie o zbieżności metody Jacobiego:

W metodzie Jacobiego warunek dostateczny, $||D^{-1}(L + U)||_{\infty} < 1$ jest spełniony np. wtedy, gdy macierz A ma dominującą przekątną, tzn. gdy:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \forall i = 1, \ldots, N$$

Niestety w wielu przypadkach metoda Jacobiego, choć zbieżna, będzie zbieżna zbyt wolno, by nas zadowolić.