Wydział	**Imię i nazwisko**	**Rok**	**Grupa**
WIMiIP	Mateusz Witkowski	II	4
Temat: Układy równań liniowych - metoda Thomasa.			**Prowadzący** dr hab. inż. Hojny Marcin, prof. AGH
Data ćwiczenia 14.05.2020	**Data oddania** 21.05.2020	**Data zaliczenia**	**OCENA**

1. Cel ćwiczenia

Celem ćwiczenia było zapoznanie się oraz implementacja wybranej przez siebie metody, w moim przypadku - rozwiązywania układów równań liniowych metodą Thomasa.

2. Wprowadzenie teoretyczne

Metoda Thomasa jest to uproszczona wersja metody eliminacji Gaussa, którą można zastosować gdy układ ma specyficzną strukturę, jest to tak zwany układ równań trójprzekątniowy. Wykorzystuje się ją gdyż jej złożoność w porównaniu z metodą Gaussa jest znacznie mniejsza (Thomas = O(n), Gauss = $O(n^3)$). Wspomniany układ równań wygląda następująco:

$$a_1 = 0 \begin{bmatrix} b_1 & c_1 & 0 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & 0 & \cdots & 0 \\ 0 & a_3 & b_3 & c_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & 0 & 0 & 0 & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix} c_n = 0$$

Można go również przedstawić w inny sposób:

$$\boldsymbol{a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i}$$

Gdzie:

- $\mathbf{a_1} = 0$
- $\mathbf{c_n} = 0$
- $\mathbf{i} = 1,2,\ldots,n$

Algorytm sprowadza się do wyzerowania współczynników $\boldsymbol{a_i}$ takich że $\boldsymbol{i} = 2,3,\ldots,n$, w taki sposób aby ostatnie równanie było w postaci $b_n x_n = d_n$, z którego łatwo można wyznaczyć $\boldsymbol{x_n}$. Wykorzystując poznane rozwiązanie wyznaczamy $\boldsymbol{x_{n-1}}$, postępując w ten sposób rozwiązujemy cały układ.

Przy wyzerowanej wartości współczynników a_i jesteśmy w stanie zapisać rekurencyjny wzór określający niewiadome układu:

$$x_n = \frac{d_n}{b_n}, \qquad x_i = \frac{d_i - c_i x_{i+1}}{b_i}, \qquad gdzie\ i = n-1, n-2, \dots, 1.$$

3. Kod programu

Zdefiniowano globalnie wektor przechowujący dane, nazwę pliku z którego dane zostaną pobrane oraz funkcje odpowiedzialną za proces pobrania danych i zapisu ich do przekazanego wektora.

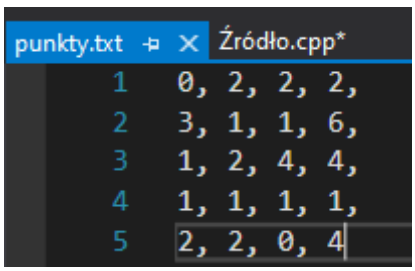
```
vector < double > dane;          //Pojemnik na dane z pliku
string nazwaPliku = "punkty.txt";//Nazwa pliku tekstowego

//Funkcja pobierjica dane z pliku tekstowego i zapisujice je do wektora.
int pobranie_z_pliku(vector <double>* data, string fileName) {
    string linia;
    fstream plik;

    plik.open(fileName, ios::in);       //otwarcie pliku
    if (plik.good() == true)            //sprawdzenie poprawnosci pliku
    {
        while (!plik.eof())
        {
            getline(plik, linia, ',');      //zapisuje slowa odzielane przecinkami
            double d = atof(linia.c_str()); //konwersja stringa na double
            data->push_back(d);             //zapis slowa parsowanego na typ double do wektora
        }
        plik.close();       //zamkniecie pliku
    }
    if (data->size() % 4 != 0) {    //zabezpiecznie w razie zle wypelnionego pliku tekstowego
        cout << "bledne dane w pliku" << endl;
        return -1;
    }
    return data->size();    //funkcja zwraca wielkosc wektora potrzebna do stworzenia tablic.
}
```

Rysunek 1.Funkcja pobierająca dane z pliku txt.

Utworzono plik txt i wypełniono go danym w następujący sposób: pierwsza kolumna w całości składa się ze współczynników „a”, druga ze współczynników „b”, trzecia z „c” i czwarta z wyrazów wolnych.



Rysunek 2. Plik txt przechowujący parametry układu równań

W funkcji main następuje sprawdzenie poprawności pliku pod względem zawartości, przygotowanie oraz wypełnienie dynamicznej tablicy dwuwymiarowej pełniącej rolę nośnika dla tablic przechowujących parametry „a”, „b”, „c” i wyrazy wolne „d”, wypełnienie ów tablic danymi pobranym z pliku. Przygotowano również tablicę przeznaczoną na rozwiązania układu.

```
int main() {
    ////////////SPRAWDZZANIE POPRAWNOSCI PLIKU////////////
    int wielkosc_wektora = pobranie_z_pliku(&dane, nazwaPliku);
    if (wielkosc_wektora == -1) //program zostanie przerwany jesli plik tekstowy byl bledny
    {
        return -1;
    }
    ////////////////////////////////////////////////////

    ////////////WYPEŁNIANIE TABLIC////////////////////////
    double** uklad_rownan;
    size_t liczba_wspolcznnikow = 4;
    size_t dlugosci_tablic = wielkosc_wektora / liczba_wspolcznnikow;
    double* Wyniki = new double[dlugosci_tablic];       //tablica rozwiazan x

    uklad_rownan = new double* [liczba_wspolcznnikow];      //utworzenie macierzy
    for (int i = 0; i < liczba_wspolcznnikow; i++) {
        uklad_rownan[i] = new double[dlugosci_tablic]; //tworzenie wiersza
    }
    int x = 0;

    for (size_t j = 0; j < dlugosci_tablic; j++)
    {
        for (size_t i = 0; i < liczba_wspolcznnikow; i++)   //wypelnienie tablic
        {
            uklad_rownan[i][j] = dane[x];
            x++;
        }
    }
    ////////////////////////////////////////////////////
```

Rysunek 3. Utworzenie tablic, wypełnienie macierzy współczynników, sprawdzenie pliku.

W dalszej części funkcji main wyświetlono stan początkowy macierzy współczynników oraz wywołano funkcję realizującą algorytm Thomasa. Przed i po wywołaniu funkcji zdefiniowano zmienne pobierające moment czasowy w celu obliczenia czasu potrzebnego do zrealizowania algorytmu.

```
//////////WYPISANIE POCZATKOWEGO UKLADU//////////
cout << "a" << setw(6) << "b" << setw(6) << "c" << setw(6) << "| " << "d"<< endl;
for (size_t i = 0; i < 20; i++)
{
    cout << "-";
}
cout << endl;
for (size_t j = 0; j < dlugosci_tablic; j++)
{
    for (size_t i = 0; i < liczba_wspolcznnikow; i++)
{
        if (i == liczba_wspolcznnikow-1) {
            cout << "| " << uklad_rownan[i][j];
        }
        else {
            cout << uklad_rownan[i][j] << setw(6);
        }
    }
    cout << endl;
}
cout << endl;

auto start = std::chrono::steady_clock::now();      //punkt startowy pomiaru czasu
int result = Thomas(uklad_rownan[0], uklad_rownan[1], uklad_rownan[2], uklad_rownan[3], Wyniki, dlugosci_tablic);
auto end = std::chrono::steady_clock::now();        //punkt koncowy pomiaru czasu
std::chrono::duration<double, milli> elapsed_seconds = end - start;     //wyliczamy czas ktory uplynal
std::cout << "Czas wykonania algorytmu: " << elapsed_seconds.count() << "ms\n";
```

Rysunek 4. Wypisanie stanu początkowego, wywołanie funkcji, pobranie czasu.

Funkcja Thomas przyjmuje sześć argumentów, którymi są wskaźniki do tablic współczynników, wskaźnik do tablicy potrzebnej na przechowania wyników oraz zmienną opisującą wielkość każdej z tych tablic. W odróżnieniu do algorytmu Gaussa nie mamy tutaj bardzo skomplikowanego mechanizmu sprowadzania macierzy do postaci schodkowej.
W każdym obiegu pętli zerowany jest kolejny współczynnik „a” i obliczane są wartości współczynników „b”, ”c” i „d”. Następnie z ostatniego równania wyliczana jest wartość „x”, na podstawie której wyliczane są następne wartości „x-ów” w kolejnej pętli.

```
int Thomas(double *a, double* b, double* c, double* d, double* Wyniki, int dlugosci_tablic) {
    for (int i = 1; i < dlugosci_tablic; i++) //reukowanie wartosci wspolczynnikow 'a'
    {                                         //wyliczanie wartosci wspolczynnikow 'b','c' i 'd'
        double l = a[i] / b[i - 1];
        a[i] = 0.0;
        b[i] -= l * c[i - 1];
        d[i] -= l * d[i - 1];
    }
    Wyniki[dlugosci_tablic - 1] = d[dlugosci_tablic - 1] / b[dlugosci_tablic - 1]; //wyliczenie pierwszego rozwiazania
    for (int i = dlugosci_tablic - 2; i >= 0; i--)
    {
        Wyniki[i] = (d[i] - c[i] * Wyniki[i + 1]) / b[i];   //wyliczanie reszty rozwiazan
    }
    return 1;
}
```

Rysunek 5. Funkcja realizująca metodę Thomasa.

Po zrealizowaniu algorytmu pozostaje wypisać wyniki:

```
for (int i = 0; i < dlugosci_tablic; i++)
{
    cout << "x["<<i+1<<"] = " << Wyniki[i] << endl;
}
```

Rysunek 6. Wypisanie wyników w funkcji main.

Cały kod:

```
using namespace std;
vector < double > dane;          //Pojemnik na dane z pliku
string nazwaPliku = "punkty.txt";//Nazwa pliku tekstowego

//Funkcja pobierjıca dane z pliku tekstowego i zapisujıce je do wektora.
int pobranie_z_pliku(vector <double>* data, string fileName) {
    string linia;
    fstream plik;

    plik.open(fileName, ios::in);       //otwarcie pliku
    if (plik.good() == true)            //sprawdzenie poprawnosci pliku
    {
        while (!plik.eof())
        {
            getline(plik, linia, ',');      //zapisuje slowa odzielane przecinkami
            double d = atof(linia.c_str()); //konwersja stringa na double
            data->push_back(d);             //zapis slowa parsowanego na typ double do wektora
        }
        plik.close();       //zamkniecie pliku
    }
    if (data->size() % 4 != 0) {    //zabezpiecznie w razie zle wypelnionego pliku tekstowego
        cout << "bledne dane w pliku" << endl;
        return -1;
    }
    return data->size();    //funkcja zwraca wielkosc wektora potrzebna do stworzenia tablic.
}
int Thomas(double *a, double* b, double* c, double* d, double* Wyniki, int dlugosci_tablic) {
    for (int i = 1; i < dlugosci_tablic; i++) //reukowanie wartosci wspolczynnikow 'a'
    {                                         //wyliczanie wartosci wspolczynnikow 'b','c' i 'd'
        double l = a[i] / b[i - 1];
        a[i] = 0.0;
        b[i] -= l * c[i - 1];
        d[i] -= l * d[i - 1];
    }
    Wyniki[dlugosci_tablic - 1] = d[dlugosci_tablic - 1] / b[dlugosci_tablic - 1]; //wyliczenie pierwszego rozwiazania
    for (int i = dlugosci_tablic - 2; i >= 0; i--)
    {
        Wyniki[i] = (d[i] - c[i] * Wyniki[i + 1]) / b[i];   //wyliczanie reszty rozwiazan
    }
    return 1;
}
```

Rysunek 7. Cały kod cz.1

```cpp
int main() {
    ////////////SPRAWDZZANIE POPRAWNOSCI PLIKU////////////
    int wielkosc_wektora = pobranie_z_pliku(&dane, nazwaPliku);
    if (wielkosc_wektora == -1) //program zostanie przerwany jesli plik tekstowy byl bledny
    {
        return -1;
    }
    ////////////////////////////////////////////////////
    ////////////WYPEŁNIANIE TABLIC//////////////////////
    double** uklad_rownan;
    size_t liczba_wspolczynnikow = 4;
    size_t dlugosci_tablic = wielkosc_wektora / liczba_wspolczynnikow;
    double* Wyniki = new double[dlugosci_tablic];       //tablica rozwiazan x
    uklad_rownan = new double* [liczba_wspolczynnikow];      //utworzenie macierzy
    for (int i = 0; i < liczba_wspolczynnikow; i++) {
        uklad_rownan[i] = new double[dlugosci_tablic]; //tworzenie wiersza
    }
    int x = 0;
    for (size_t j = 0; j < dlugosci_tablic; j++)
    {
        for (size_t i = 0; i < liczba_wspolczynnikow; i++)   //wypelnienie tablic
        {
            uklad_rownan[i][j] = dane[x];
            x++;
        }
    }
    ////////////////////////////////////////////////////
   ////////////WYPISANIE POCZATKOWEGO UKLADU////////////
   cout << "a" << setw(6) << "b" << setw(6) << "c" << setw(6) << "| " << "d"<< endl;
   for (size_t i = 0; i < 20; i++)
   {
       cout << "-";
   }
   cout << endl;
   for (size_t j = 0; j < dlugosci_tablic; j++)
   {
       for (size_t i = 0; i < liczba_wspolczynnikow; i++)
   {
           if (i == liczba_wspolczynnikow-1) {
               cout << "| " << uklad_rownan[i][j];
           }
           else {
               cout << uklad_rownan[i][j] << setw(6);
           }
       }
       cout << endl;
   }
   cout << endl;

   auto start = std::chrono::steady_clock::now();      //punkt startowy pomiaru czasu
   int result = Thomas(uklad_rownan[0], uklad_rownan[1], uklad_rownan[2], uklad_rownan[3], Wyniki, dlugosci_tablic);
   auto end = std::chrono::steady_clock::now();        //punkt koncowy pomiaru czasu
   std::chrono::duration<double, milli> elapsed_seconds = end - start;     //wyliczamy czas ktory uplynal
   std::cout << "Czas wykonania algorytmu: " << elapsed_seconds.count() << "ms\n";

   for (int i = 0; i < dlugosci_tablic; i++)
   {
       cout << "x["<<i+1<<"] = " << Wyniki[i] << endl;
   }
   getchar(); getchar();
   return 0;
}
```

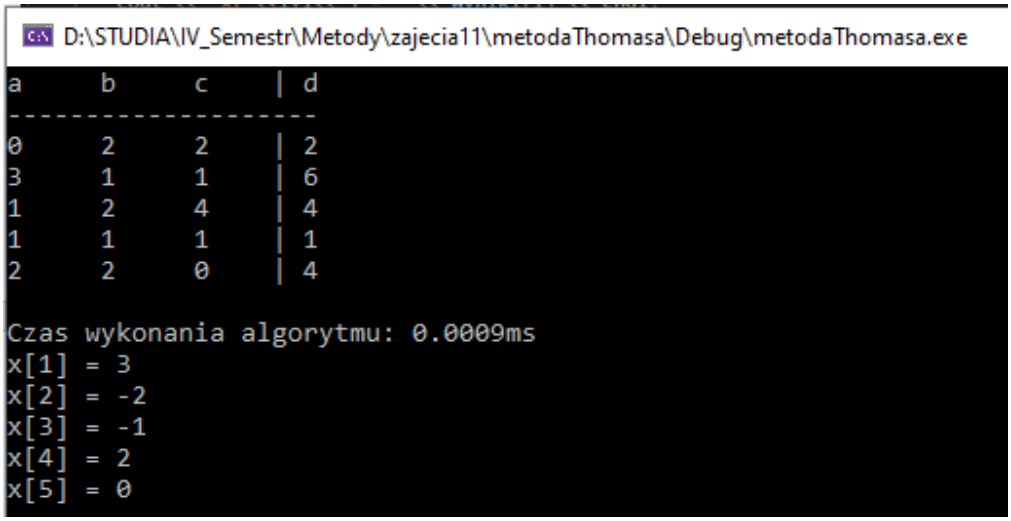
Rysunek 8.Cały kod cz.2

4. Testy

W celu zweryfikowania wyników programu dokonano trzech testów z wykorzystaniem programu z wcześniejszych zajęć oraz kalkulatora znajdującego się na stronie https://calcoolator.pl/metoda_gaussa.html#solve-usingGaussian-elimination%28%7B%7B4,-2,4,-2,8%7D,%7B3,1,4,2,7%7D,%7B2,4,2,1,10%7D,%7B2,2,4,2,2%7D%7D%29. W przypadku programu dotyczącego algorytmu eliminacji Gaussa porównano czasy wykonania obu funkcji.

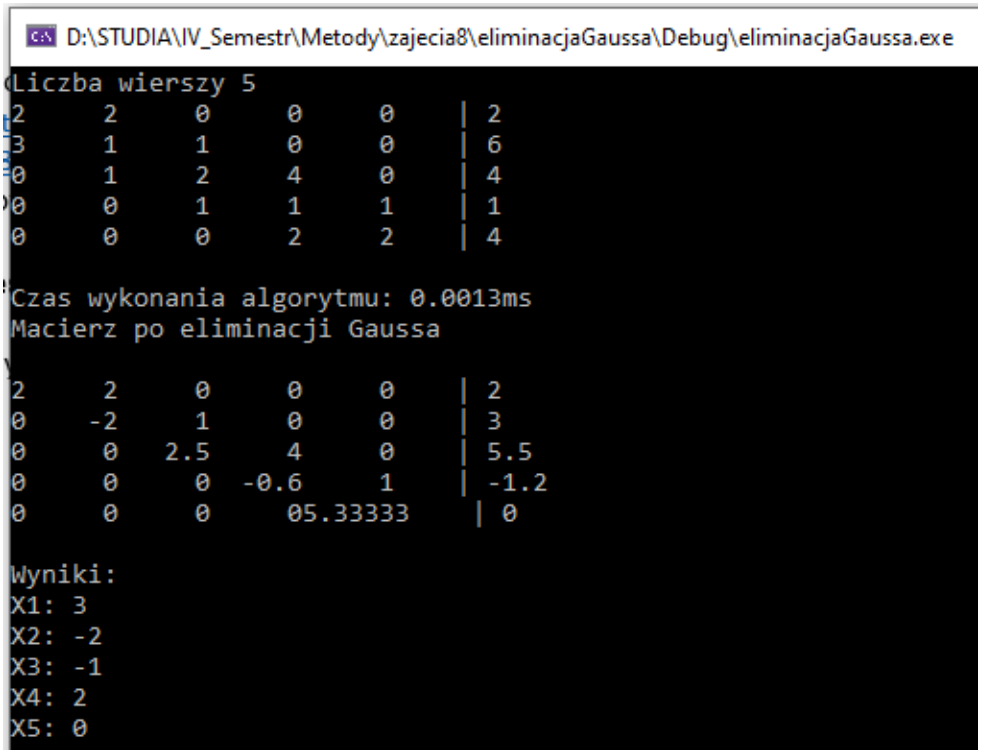
Test 1.

Wynik programu korzystającego z metody Thomasa:



Rysunek 9. Program realizujący algorytm Thomasa.

Wynik programu korzystającego z algorytmu Gaussa:



Rysunek 10.Program realizujący algorytm Gaussa.

Wynik kalkulatora z podanej strony internetowej:

Wynik:

$x_1 = 3$,

$x_2 = -2$,

$x_3 = -1$,

$x_4 = 2$,

$x_5 = 0$

Test 2.

Wynik programu korzystającego z metody Thomasa:

```
D:\STUDIA\IV_Semestr\Metody\zajecia11\metodaThomasa\Debug\metodaThomasa.exe
a     b     c     | d
--------------------
0     1     3     | 1
1     2     1     | 5
1     2     1     | 3
1     2     1     | 2
1     1     0     | 2

Czas wykonania algorytmu: 0.0009ms
x[1] = 2.5
x[2] = -0.5
x[3] = 3.5
x[4] = -3.5
x[5] = 5.5
```

Rysunek 11. Program realizujący algorytm Thomasa.

Wynik programu korzystającego z algorytmu Gaussa:

```
D:\STUDIA\IV_Semestr\Metody\zajecia8\eliminacjaGaussa\Debug\eliminacjaGaussa.exe
Liczba wierszy 5
1     3     0     0     0     | 1
1     2     1     0     0     | 5
0     1     2     1     0     | 3
0     0     1     2     1     | 2
0     0     0     1     1     | 2

Czas wykonania algorytmu: 0.0013ms
Macierz po eliminacji Gaussa

1     3     0     0     0     | 1
0    -1     1     0     0     | 4
0     0     3     1     0     | 7
0     0     01.66667     1     | -0.333333
0     0     0     0   0.4     | 2.2

Wyniki:
X1: 2.5
X2: -0.5
X3: 3.5
X4: -3.5
X5: 5.5
```

Rysunek 12.Program realizujący algorytm Gaussa.

Wynik kalkulatora z podanej strony internetowej:

Wynik:

$$x_1 = \frac{5}{2},$$

$$x_2 = \frac{-1}{2},$$

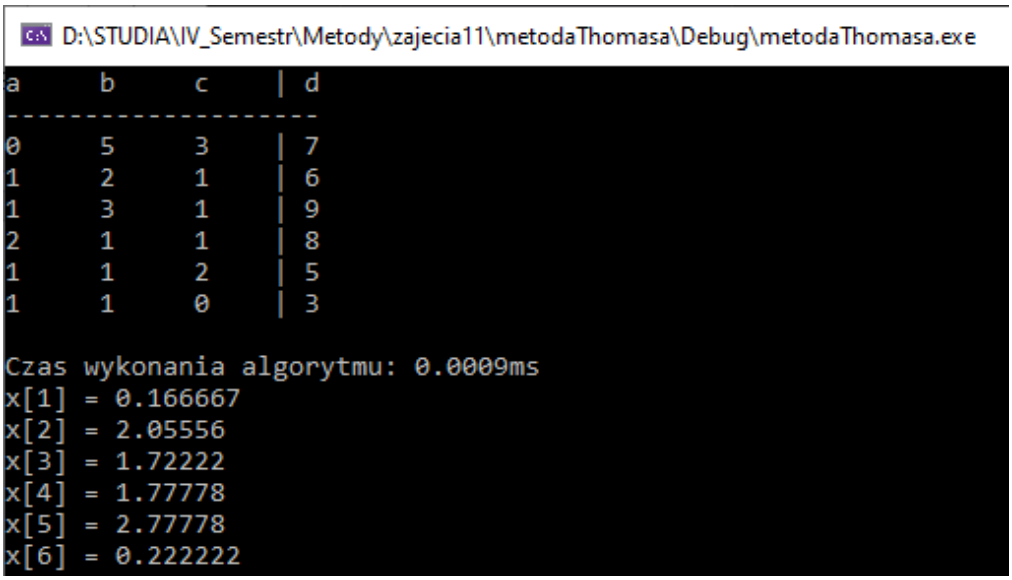
$$x_3 = \frac{7}{2},$$

$$x_4 = \frac{-7}{2},$$

$$x_5 = \frac{11}{2}$$

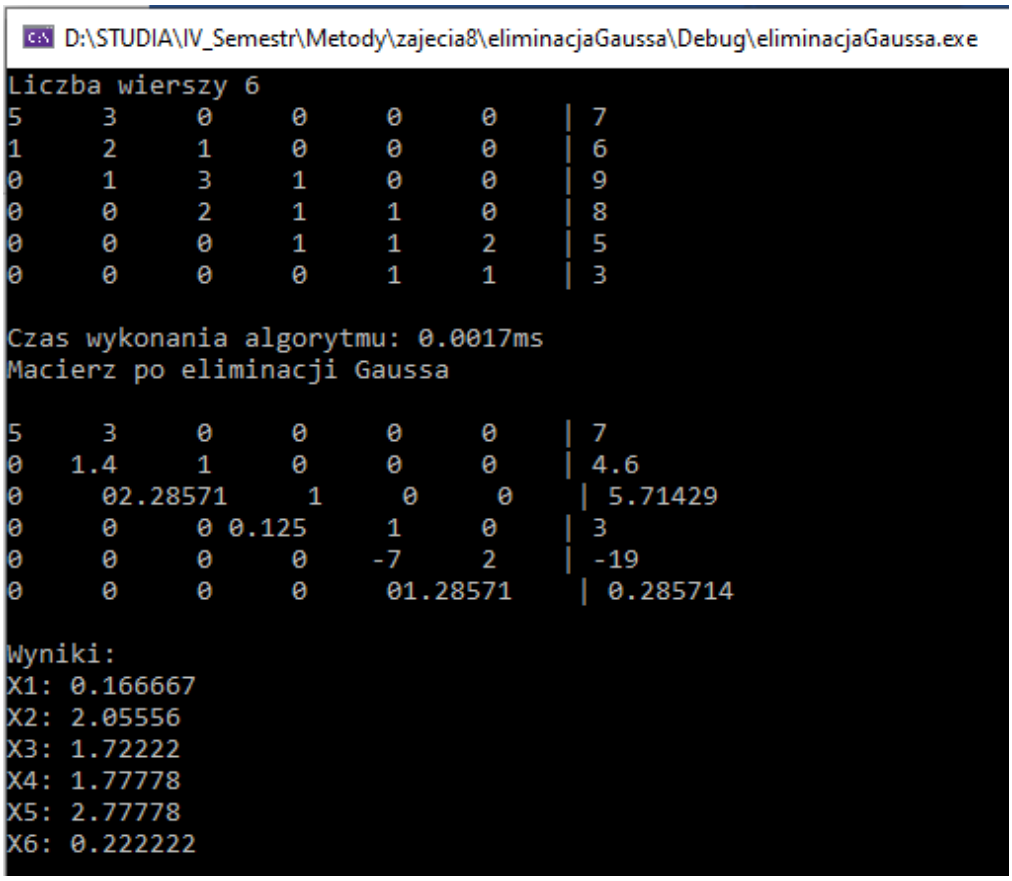
Test 3.

Wynik programu korzystającego z metody Thomasa:



Rysunek 11. Program realizujący algorytm Thomasa.

Wynik programu korzystającego z algorytmu Gaussa:



Rysunek 12.Program realizujący algorytm Gaussa.

Wynik kalkulatora z podanej strony internetowej:

Wynik:

$$x_1 = \frac{1}{6},$$

$$x_2 = \frac{37}{18},$$

$$x_3 = \frac{31}{18},$$

$$x_4 = \frac{16}{9},$$

$$x_5 = \frac{25}{9},$$

$$x_6 = \frac{2}{9}$$

5. Wnioski

Wykorzystana w ćwiczeniu metoda Thomasa może nie jest tak bardzo popularna jak algorytm Gaussa, ale na pewno znajduje zastosowanie w wielu specyficznych przypadkach. Jej główną przewagą jest jej liniowa złożoność obliczeniowa, która jak można było zauważyć podczas testów przekładała się znacząco na szybkość wykonania algorytmu. W każdym przypadku miała ona lepszy czas od metody Gaussa, a czas wykonania nie zmieniał się w zależności od wielkości macierzy wejściowej. Główną wadą natomiast jest to, że nie jest tak uniwersalna jak eliminacja Gaussa, macierz musi być zawsze odpowiednio wypełniona. Wyniki otrzymane w programie pokazały, że działa on bez błędów i jest w stanie policzyć dowolnie dużą macierz. Tak zaimplementowany algorytm bardzo dobrze sprawdzi się przy rozwiązywaniu bardzo rozbudowanych macierzy trójdiagonalnych.