Imię i nazwisko Meg Paskowski	Temat laboratorium Rozwiązywanie układów równań liniowych - metoda LU, Choleskiego	Data oddania 01.06.2024r.	Data ćwiczeń 25.05.2024r.
Prowadzący dr hab. inż. Marcin Hojny		Grupa laboratoryjna 4	

1. Cel ćwiczenia

Celem laboratorium nr. 10 było zapoznanie się z pojęciem rozwiązywania równań liniowych metodą eliminacji LU, Chleskiego oraz zaimplementowanie tych algorytmów w wybranym języku programowania.

2. Wstęp teoretyczny

Metoda LU (LU Decomposition) → jest to jedna z najważniejszych metod rozwiązywania układów liniowych. Polega na rozkładzie danej macierzy kwadratowej *A* na iloczyn dwóch macierzy trójkątnych: dolnej macierzy trójkątnej *L* oraz górnej macierzy trójkątnej U. Jeśli *A* jest macierzą n×n (liczbie niewiadomych n), to:

$$\mathbf{A = L \cdot U \text{ oraz zachodzi wtedy: } A \cdot X = L \cdot U \cdot X = D} \quad \textbf{(1)}$$

Gdzie:

- L jest macierzą dolnotrójkątną (wszystkie elementy powyżej głównej przekątnej są zerowe).
- U jest macierzą górnotrójkątną (wszystkie elementy poniżej głównej przekątnej są zerowe).
- D oznacza wektor prawych stron układu równań liniowych, który jest wynikiem mnożenia macierzy L·U przez wektor niewiadomych X. Reprezentuje wektor, który otrzymujemy po przekształceniu wektora niewiadomych X przez macierze L i U.

Algorytm Doolittle'a → proces ten polega na dekompozycji macierzy A na iloczyn dwóch macierzy trójkątnych: dolnej macierzy trójkątnej L i górnej macierzy trójkątnej U takich, że A =L·U. W trakcie tego procesu, korzystając z modyfikacji eliminacji Gaussa, obliczamy współczynniki, które zerują odpowiednie elementy macierzy A, a następnie zapisujemy te współczynniki w odpowiednich pozycjach macierzy L i U.

I. Inicjacja → Rozpoczynamy od macierzy A i tworzymy pustą macierz L (z jedynkami na diagonali) oraz macierz U, która jest kopią macierzy A.
II. Dekompozycja → Przechodzimy przez kolumny macierzy A, aby wyznaczyć elementy macierzy L i U. Tutaj każdy element macierzy U w kolumnie j i wierszach j,j+1,…,n obliczamy zgodnie z równaniem (2).

$$u_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ij} u_{kj} \quad \textbf{(2)}$$

Gdzie:

- l_{ik} i u_{kj} to elementy macierzy L i U odpowiednio, a a_{jj} to element macierzy A.

Dla macierzy L, elementy poniżej diagonali w każdej kolumnie obliczamy zgodnie z równaniem (3).

$$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right) \quad \textbf{(3)}$$

Gdzie:

- a_{ij} to element macierzy A na pozycji i-tego wiersza i j-tej kolumny.
- n to liczba niewiadomych w układzie równań,

- u_{jj} to element główny kolumny j i musi być różny od zera, aby uniknąć dzielenia przez zero.

Rozwiązanie układu równań:

Gdy mamy rozkład A=LU, możemy rozwiązać układ równań Ax=b:

I. Najpierw rozwiązujemy układ Ly=b za pomocą podstawiania w przód (forward substitution). → W trakcie eliminacji w przód obliczamy współczynniki m_{ij}, które mnożymy przez elementy poniżej głównej przekątnej w kolumnie j tak, aby je zniwelować. Każdy współczynnik m_{ij} obliczamy jako iloraz elementu macierzy a_{ij} (aktualnie przetwarzanego elementu) przez element macierzy a_{jj} (element główny kolumny).

$$m_{ji}=\frac{a_{ij}}{a_{jj}} \quad (4)$$

Gdzie:

- a_{ij} to element macierzy na i-tym wierszu i j-tej kolumnie (aktualnie przetwarzany element),
- a_{jj} to element główny kolumny j.

II. Następnie rozwiązujemy układ Ux=y za pomocą podstawiania wstecz (backward substitution).

Zalety stosowania tej metody rozwiązania równań liniowych:

- Efektywność → Rozkład LU jest bardziej efektywny obliczeniowo w porównaniu z innymi metodami, zwłaszcza gdy macierz A jest stała, a rozwiązujemy wiele różnych układów równań z różnymi wektorami prawych stron b. Po rozkładzie, rozwiązanie każdego kolejnego układu równań jest szybkie.
- Stabilność numeryczna → Wprowadzenie metody pivotingu (przemieszczanie wierszy lub kolumn) może poprawić stabilność numeryczną rozkładu LU.

Metoda Choleskiego → jest metodą rozkładu macierzy symetrycznej dodatnio określonej A na iloczyn macierzy trójkątnej dolnej L i jej transpozycji L^T. Metoda ta jest szczególnym przypadkiem rozkładu LU i jest efektywniejsza obliczeniowo dla macierzy symetrycznych.

$$A=LL^T \quad (5)$$

Wówczas zachodzi:

$$A \cdot X=L \cdot L^T \cdot X=B \quad (6)$$

Oraz

$$\begin{gathered} L^T \cdot X=Y \\ L \cdot Y=B \end{gathered} \quad (7)$$

Przebieg algorytmu:

I. Inicjalizacja → Tworzenie macierzy L jako macierz zerową o rozmiarze A.
II. Obliczanie elementów macierzy L:
    a. Dla elementu na diagonali:

$$l_{jj}=\sqrt{a_{ij}-\sum_{k=1}^{j-1} l_{ik} l^2{}_{jk}} \quad (8)$$

Gdzie j=1,2,…,n.

b. Dla pozostałych elementów poniżej diagonali:

$$l_{ij}=\frac{1}{l_{ij}}\left(a_{ij}-\sum_{k=1}^{j-1} l_{ik}l_{jk}\right) \tag{9}$$

Gdzie i=j+1,…,n.

III. Rozwiązanie układu równań Ax = b

a. Najpierw rozwiązujemy układ Ly = b metodą podstawienia wprzód

$$y_i=\frac{b_1}{l_{11}} \tag{10}$$

Gdzie i=2,…,n

b. Następnie rozwiązujemy układ $L^T x=y$ metodą podstawienia wstecz

$$x_n=\frac{y_n}{l_{nn}} \tag{11}$$

$$x_i=\frac{y_i-\sum_{j=i+1}^{n} l_{ji}x_j}{l_{ii}}$$

Gdzie

- i=n−1,…,1.
- l_{ii} to element główny kolumny i i musi być różny od zera, aby uniknąć dzielenia przez zero.

Zalety stosowania tej metody rozwiązania równań liniowych:

- Wymaga tylko n(n+1)/2 mnożeń i pierwiastkowań, podczas gdy metoda LU wymaga n^3 operacji.
- Metoda jest stabilna numerycznie, ponieważ nie występuje dzielenie przez zero.
- Macierz L jest łatwiejsza do przechowywania, ponieważ jest macierzą trójkątną dolną.

Metoda Choleskiego jest efektywną i stabilną numerycznie metodą rozwiązywania układów równań liniowych, gdy macierz współczynników A jest symetryczna i dodatnio określona. Pozwala ona na szybkie rozwiązywanie wielu układów równań z różnymi wektorami prawych stron b po wykonaniu rozkładu macierzy A.

3. Implementacja

W ramach ćwiczeń zaimplementowano program w języku C++, który rozwiązuje układ równań liniowych Ax = b metodami LU i Choleskiego, wczytując dane z pliku tekstowego. Funkcje *„LU_method_decomposition()”, „solve_LU_method()”, „Cholesky_method_decomposition()”* i solve_Cholesky_method odpowiadają za obliczenia, a funkcja *„readFile()”* wczytuje dane z pliku. Główna funkcja main koordynuje cały proces.

Funkcja *„LU_method_decomposition()”* wykonuje rozkład LU macierzy A na macierze L i U. Alokuje pamięć na macierze L i U jako tablice dynamiczne, inicjalizuje macierz L jako macierz jednostkową i macierz U jako macierz zerową. Następnie oblicza wartości macierzy U i L zgodnie z algorytmem rozkładu LU, wykorzystując podwójną pętlę for.

```
void LU_method_decomposition(double** A, double** L, double** U, int n) {
    for (int i = 0; i < n; i++) {
        L[i] = new double[n];
        U[i] = new double[n];

        for (int j = 0; j < n; j++) {
            if (i == j)
                L[i][j] = 1;
            else {
                L[i][j] = 0;
                U[i][j] = 0;
            }
        }
    }
```

```
    double sum = 0.0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (j >= i) {
                sum = 0;
                for (int k = 0; k < i; k++)
                    sum += L[i][k] * U[k][j];
                U[i][j] = A[i][j] - sum;
            } else {
                sum = 0;
                for (int k = 0; k < j; k++)
                    sum += L[i][k] * U[k][j];
                L[i][j] = (A[i][j] - sum) / U[j][j];
            }
        }
    }
}
```

Funkcja „solve_LU_*method()*" rozwiązuje układ równań Ax = b po rozkładzie LU. Alokuje pamięć na macierze L i U oraz wektory Y i X jako tablice dynamiczne. Wywołuje funkcję „*LU_method_decomposition()*", aby uzyskać macierze L i U. Następnie oblicza wektor Y metodą podstawienia wprzód, wykorzystując macierz L, a następnie wektor X metodą podstawienia wstecz, wykorzystując macierz U i wektor Y. Na koniec zwalnia pamięć zajętą przez macierze L i U oraz wektor Y i zwraca wskaźnik na wektor rozwiązań X.

```
double* solve_LU_method(double** A, double* B, int n) {
    //Dkleracja nowych macierzy U i L
    double** L = new double*[n];
    double** U = new double*[n];

    LU_method_decomposition(A, L, U, n);

    double sum = 0.0;
    double* Y = new double[n];

    for (int i = 0; i < n; i++) {
        sum = 0;
        for (int j = 0; j < i; j++)
            sum += L[i][j] * Y[j];
        Y[i] = (B[i] - sum) / L[i][i];
    }

    double* X = new double[n];
    for (int i = n - 1; i >= 0; i--) {
        sum = 0;
        for (int j = i + 1; j < n; j++)
            sum += U[i][j] * X[j];
        X[i] = (Y[i] - sum) / U[i][i];
    }

    for (int i = 0; i < n; i++) {
        delete[] L[i];
        delete[] U[i];
    }
    delete[] L;
    delete[] U;
    delete[] Y;
    return X;
}
```

Funkcja *„readFile()"* wczytuje dane z pliku tekstowego do macierzy A i wektora B. Otwiera plik o podanej nazwie *„fileName()"* za pomocą ifstream. Sprawdza, czy otwarcie pliku powiodło się. Jeśli nie, wypisuje komunikat o błędzie i kończy działanie funkcji. Wczytuje rozmiar układu równań n z pliku. Alokuje pamięć na macierz A i wektor B jako tablice dynamiczne. Wczytuje elementy macierzy A i wektora B z pliku do dynamicznie alokowanych tablic. Na koniec zamyka plik.

```
void readFile(string fileName, int& n, double**& A, double*& B) {
    ifstream file(fileName);

    if (!file) {
        cerr << "Cannot open this file :<." << endl;
        return;
    }

    file >> n;

    A = new double* [n];
    for (int i = 0; i < n; i++) {
```

```
        A[i] = new double[n];
        for (int j = 0; j < n; j++)
            file >> A[i][j];
    }

    B = new double[n];
    for (int i = 0; i < n; i++)
        file >> B[i];

    file.close();
}
```

Funkcja „*Cholesky_method_decomposition()*" wykonuje rozkład Choleskiego macierzy A na macierz L. Alokuje pamięć na macierz L jako tablicę dynamiczną. Następnie oblicza wartości macierzy L zgodnie z algorytmem rozkładu Choleskiego, wykorzystując podwójną pętlę for. Dla każdego elementu (i, j) macierzy L, jeśli j <= i, obliczana jest suma kwadratów odpowiednich elementów macierzy L powyżej przekątnej, odejmowana od odpowiedniego elementu macierzy A, a następnie obliczany jest pierwiastek kwadratowy wyniku. Dla pozostałych elementów poniżej przekątnej, obliczana jest suma iloczynów odpowiednich elementów macierzy L, odejmowana od odpowiedniego elementu macierzy A, a następnie wynik dzielony jest przez element na przekątnej w danej kolumnie.

```
void Cholesky_method_decomposition(double** A, double** L, int n) {
    double sum = 0.0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            sum = 0;
            if (j == i) { // Wartości na przekątnych
                for (int k = 0; k < j; k++)
                    sum += L[j][k] * L[j][k];
                L[j][j] = sqrt(A[j][j] - sum);
            }
            else { // Pozostałe wartości w macierzy
                for (int k = 0; k < j; k++)
                    sum += L[i][k] * L[j][k];
                L[i][j] = (A[i][j] - sum) / L[j][j];
            }
        }
    }
}
```

Funkcja „*solve_Cholesky_method()*" rozwiązuje układ równań Ax = b po rozkładzie Choleskiego. Alokuje pamięć na macierz L i wektory Y i X jako tablice dynamiczne. Wywołuje funkcję „*Cholesky_method_decomposition*()", aby uzyskać macierz L. Następnie oblicza wektor Y metodą podstawienia wprzód, wykorzystując macierz L. Dla każdego elementu i wektora Y, obliczana jest suma iloczynów odpowiednich elementów macierzy L i wektora Y, odejmowana od odpowiedniego elementu wektora B, a wynik dzielony przez odpowiedni element na przekątnej macierzy L. Następnie obliczana jest wartość elementu wektora X metodą podstawienia wstecz, wykorzystując macierz L i wektor Y. Dla każdego elementu i wektora X, obliczana jest suma iloczynów odpowiednich elementów macierzy L i wektora X, odejmowana od odpowiedniego elementu wektora Y, a wynik dzielony jest przez odpowiedni element na przekątnej macierzy L. Na koniec zwalniania jest pamięć zajętą przez macierz L oraz wektory Y i X, a zwracany jest wskaźnik na wektor rozwiązań X.

```
double* solve_Cholesky_method(double** A, double* B, int n) {
    double sum=0.0;
    double** L = new double* [n];

    for (int i = 0; i < n; ++i)
        L[i] = new double[n];
    Cholesky_method_decomposition(A, L, n); // Zamiana macierzy A na macierz L

    double* Y = new double[n];
    for (int i = 0; i < n; i++) { // Wyliczanie wektora Y
        sum = 0;
        for (int j = 0; j < i; ++j)
            sum += L[i][j] * Y[j];
        Y[i] = (B[i] - sum) / L[i][i];
    }

    double* X = new double[n]; // Wyliczanie wektora X
    for (int i = n - 1; i >= 0; i--) {
        sum = 0;
```

```
        for (int j = i + 1; j < n; j++)
            sum += L[j][i] * X[j];
        X[i] = (Y[i] - sum) / L[i][i];
    }

    for (int i = 0; i < n; ++i)
        delete[] L[i];
    delete[] L;
    delete[] Y;
    return X;
}
```

Funkcja main jest funkcją główną programu. Deklaruje zmienne n, A, A2, B i B2 do przechowywania rozmiaru układu równań oraz wskaźników na macierze A i A2 oraz wektory B i B2. Wywołuje funkcję readFile, przekazując nazwę pliku "macierz.txt" oraz zmienne n, A i B, aby wczytać dane z pliku. Tworzy kopie macierzy A i wektora B do A2 i B2, ponieważ funkcje *„solve_LU_method()"* i *„solve_Cholesky_method()"* modyfikują oryginalne dane. Następnie wywołuje funkcję *„solve_LU_method()",* przekazując macierz A, wektor B i rozmiar n, aby uzyskać wektor rozwiązań X_LU. Wywołuje również funkcję *„solve_Cholesky_method()"*, przekazując macierz A2, wektor B2 i rozmiar n, aby uzyskać wektor rozwiązań X_Cholesky. Następnie wypisuje rozwiązanie układu równań metodą LU i metodą Choleskiego, wyświetlając wartości wektorów X_LU i X_Cholesky. Na koniec zwalnia pamięć zajętą przez macierze A, A2, wektory B, B2 oraz wektory rozwiązań X_LU i X_Cholesky.

```
int main() {
    int n;
    double** A, **A2;
    double* B, *B2;

    readFile("macierz.txt", n, A, B);

    //Kopie macierzy
    A2 = new double* [n];
    for (int i = 0; i < n; i++) {
        A2[i] = new double[n];
        for (int j = 0; j < n; j++)
            A2[i][j] = A[i][j];
    }

    B2 = new double[n];
    for (int i = 0; i < n; i++)
        B2[i] = B[i];

    //Wywolanie funkcji
    double* X_LU = solve_LU_method(A, B, n);

    double* X_Cholesky = solve_Cholesky_method(A2, B2, n);

    cout << "Solve - method LU:" << endl;
    for (int i = 0; i < n; i++)
        cout << "x" << i + 1 << " = " << setw(10) << X_LU[i] << endl;

    cout << "Solve - method Choleskiego:" << endl;
    for (int i = 0; i < n; i++)
        cout << "x" << i + 1 << " = " << setw(10) << X_Cholesky[i] << endl;

    //Czyszczenie pamięci
    for (int i = 0; i < n; i++)
        delete[] A[i];
    delete[] A;
    delete[] B;
    delete[] X_LU;
    delete[] X_Cholesky;

    return 0;
}
```

5. Testy jednostkowe i opracowanie wyników

Testy zostały przeprowadzone dla kliku macierzy oraz porównane zostały do wyników otrzymanych przy wykorzystaniu biblioteki NumPy w języku Python oraz przykładów podanych przez prowadzącego.

1. Test dla macierzy o wymiarach 3x3:

   Macierz:

4 -2 2
-2 2 2
2 2 14

Wyraz wolny:

-6 4 0

Wyniki uzyskane:
- Metoda LU: -1 1 0
- Metoda Choleskiego: -1 1 0

Oczekiwany wynik: -1 1 0

```
Solve - method LU:
x1 =         -1
x2 =          1
x3 =          0
Solve - method Choleskiego:
x1 =         -1
x2 =          1
x3 =          0
```

Rysunek 1 Wyniki programu dla testu 1

2. Test 2 dla macierzy o wymiarach 3x3

Macierz:

3 -4 4 -4
1.5 -1 2 -2
1.5 -0.5 0 -3
4.5 -5.5 4 -9

Wyraz wolny:

-9 -3.5 -2 -14

Wyniki uzyskane:
- Metoda LU: 0.166667 0.0833333 0.333333
- Metoda Choleskiego: 0.166667 0.0833333 0.333333

Oczekiwany wynik: 0.166667 0.0833333 0.333333

```
Solve - method LU:
x1 =   0.166667
x2 = -0.0833333
x3 =   0.333333
Solve - method Choleskiego:
x1 =   0.166667
x2 = -0.0833333
x3 =   0.333333
```

Rysunek 2 Wyniki programu dla testu 2

3. Test dla macierzy o wymiarach 4x4

Macierz:

10 -1 2 3
-1 12 3 -1
2 3 13 2
3 -1 2 14

Wyraz wolny:

12 13 14 15

Wyniki uzyskane:

- Metoda LU: 0.939586 1.09792 0.545059 0.870646
- Metoda Choleskiego: 0.939586 1.09792 0.545059 0.870646

Oczekiwany wynik: 0.939586 1.09792 0.545059 0.870646

```
Solve - method LU:
x1 =   0.939586
x2 =    1.09792
x3 =   0.545059
x4 =   0.870646
Solve - method Choleskiego:
x1 =   0.939586
x2 =    1.09792
x3 =   0.545059
x4 =   0.870646
```

Rysunek 3 Wyniki programu dla testu 3

4. Test dla macierzy o wymiarach 8x8:

Macierz:

8.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	8.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	8.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	8.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	8.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	8.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	8.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0

Wyraz wolny:

8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0

Wyniki uzyskane:

- Metoda LU: 0.8   0.657143 0.514286 0.371429 0.228571 0.0857143 -0.0571429 -0.2
- Metoda Choleskiego: 0.8   0.657143 0.514286 0.371429 0.228571 0.0857143 -0.0571429 -0.2

Oczekiwany wynik: 0.8   0.657143 0.514286 0.371429 0.228571 0.0857143 -0.0571429 -0.2

```
Solve - method LU:
x1 =         0.8
x2 =    0.657143
x3 =    0.514286
x4 =    0.371429
x5 =    0.228571
x6 =   0.0857143
x7 =  -0.0571429
x8 =        -0.2
Solve - method Choleskiego:
x1 =         0.8
x2 =    0.657143
x3 =    0.514286
x4 =    0.371429
x5 =    0.228571
x6 =   0.0857143
x7 =  -0.0571429
x8 =        -0.2
```

Rysunek 4 Wyniki programu dla testu 4

5. Test dla macierzy o wymiarach 12x12:

Macierz:

14.659086	2.670755	2.203171	2.943375	2.118089	1.690437	2.808325	2.495304	1.837052	2.889184	2.385626	2.650907
2.670755	16.827431	4.174097	4.313477	3.379633	3.285787	3.328731	3.747904	2.331158	3.689706	4.311929	3.990032
2.203171	4.174097	16.410138	4.203170	3.707578	3.458344	2.946478	3.807931	2.458833	3.096718	4.786398	3.692646
2.943375	4.313477	4.203170	17.612428	4.671792	3.990023	3.650870	4.448078	2.955805	4.190907	4.817435	4.470329
2.118089	3.379633	3.707578	4.671792	16.757489	3.822856	2.925625	3.993420	2.584489	3.253510	4.348152	3.936254
1.690437	3.285787	3.458344	3.990023	3.822856	15.842648	3.044077	3.752313	2.108923	2.987046	4.123046	3.230894
2.808325	3.328731	2.946478	3.650870	2.925625	3.044077	16.407768	3.094135	2.586648	3.674168	3.502887	3.167853
2.495304	3.747904	3.807931	4.448078	3.993420	3.752313	3.094135	16.633206	2.154666	3.347823	4.426679	3.770747
1.837052	2.331158	2.458833	2.955805	2.584489	2.108923	2.586648	2.154666	15.248465	2.498262	3.244370	3.164541
2.889184	3.689706	3.096718	4.190907	3.253510	2.987046	3.674168	3.347823	2.498262	15.975393	3.508648	3.460893
2.385626	4.311929	4.786398	4.817435	4.348152	4.123046	3.502887	4.426679	3.244370	3.508648	17.616792	4.304444

Wyraz wolny:

0.545495 0.461573 0.858064 0.611172 0.901298 0.134769 0.814464 0.416905 0.637986 0.367165 0.565342 0.801276

Wyniki uzyskane:

- Metoda LU:

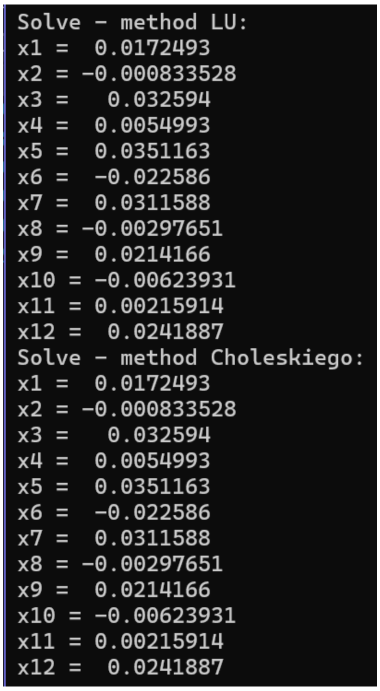
0.0172493 -0.000833528 0.032594 0.0054993 0.0351163 -0.022586 0.0311588- 0.00297651 0.0214166 -0.00623931 0.00215914 0.0241887

- Metoda Choleskiego:

0.0172493 -0.000833528 0.032594 0.0054993 0.0351163 -0.022586 0.0311588- 0.00297651 0.0214166 -0.00623931 0.00215914 0.0241887

Oczekiwane wyniki:

0.0172493 -0.000833528 0.032594 0.0054993 0.0351163 -0.022586 0.0311588- 0.00297651 0.0214166 -0.00623931 0.00215914 0.0241887



```
Solve - method LU:
x1 =  0.0172493
x2 = -0.000833528
x3 =   0.032594
x4 =  0.0054993
x5 =  0.0351163
x6 =  -0.022586
x7 =  0.0311588
x8 = -0.00297651
x9 =  0.0214166
x10 = -0.00623931
x11 = 0.00215914
x12 =  0.0241887
Solve - method Choleskiego:
x1 =  0.0172493
x2 = -0.000833528
x3 =   0.032594
x4 =  0.0054993
x5 =  0.0351163
x6 =  -0.022586
x7 =  0.0311588
x8 = -0.00297651
x9 =  0.0214166
x10 = -0.00623931
x11 = 0.00215914
x12 =  0.0241887
```

Rysunek 5 Wyniki programu dla testu 5

1. Opracowanie wyników

Jak można zauważyć po przeprowadzonych testach w wynikach nie występują różnice wynikające z założonych zaokrągleniach użytych w poszczególnych implementacjach. Natomiast gdyby występowały to mogą one prowadzić do akumulacji błędów numerycznych w trakcie wykonywania kolejnych operacji, co może wpłynąć na końcowe wyniki. Dlatego też, porównując wyniki testów, istotne jest uwzględnienie różnic w strategiach zaokrąglania.

Przedstawienie wyników poszczególnych przypadków zawartych w Tabelach 1-4.

Tabela 1 Wyniki algorytmów dla testu 1 macierzy 3x3

Wyniki przewidywane	Metoda LU	Metoda Choleskiego
-1	-1	-1
1	1	1
0	0	0

Tabela 2 Wyniki algorytmów dla testu 2 macierzy 3x3

Wyniki przewidywane	Metoda LU	Metoda Choleskiego
0.166667	0.166667	0.166667
0.0833333	0.0833333	0.0833333
0.333333	0.333333	0.333333

Tabela 3 Wyniki algorytmów dla testu 3 macierzy 4x4

Wyniki przewidywane	Metoda LU	Metoda Choleskiego
0.939586	0.939586	0.939586
1.09792	1.09792	1.09792
0.545059	0.545059	0.545059
0.870646	0.870646	0.870646

Tabela 4 Wyniki algorytmów dla testu 4 macierzy 8x8

Wyniki przewidywane	Metoda LU	Metoda Choleskiego
0.8	0.8	0.8
0.657143	0.657143	0.657143
0.514286	0.514286	0.514286
0.371429	0.371429	0.371429
0.228571	0.228571	0.228571
0.0857143	0.0857143	0.0857143
-0.0571429	-0.0571429	-0.0571429
-0.2	-0.2	-0.2

Tabela 5 Wyniki algorytmów dla testu 5 macierz 12x12

Wyniki przewidywane	Metoda LU	Metoda Choleskiego
0.0172493	0.0172493	0.0172493
-0.000833528	-0.000833528	-0.000833528
0.032594	0.032594	0.032594
0.0054993	0.0054993	0.0054993
0.0351163	0.0351163	0.0351163
-0.022586	-0.022586	-0.022586
0.0311588	0.0311588	0.0311588
-0.00297651	-0.00297651	-0.00297651
0.0214166	0.0214166	0.0214166
-0.00623931	-0.00623931	-0.00623931
0.00215914	0.00215914	0.00215914
0.0241887	0.0241887	0.0241887

2. Wnioski

Obie metody, LU i Choleskiego, są efektywnymi sposobami rozwiązywania układów równań liniowych, ale różnią się w kilku kluczowych aspektach:

Wymagania wstępne

- Metoda LU nie ma specjalnych wymagań wstępnych i może być stosowana do szerokiej gamy macierzy.
- Metoda Choleskiego wymaga, aby macierz była symetryczna i dodatnio określona, co ogranicza zakres jej zastosowań.

Zużycie pamięci

- Metoda LU wymaga większej ilości pamięci, ponieważ przechowuje dwie macierze (L i U) zamiast jednej.
- Metoda Choleskiego potrzebuje mniej pamięci, ponieważ przechowuje tylko jedną macierz (czynnik Choleskiego).

Szybkość obliczeń

- Metoda Choleskiego jest szybsza i bardziej optymalna obliczeniowo niż metoda LU, ponieważ wykonuje mniej operacji.
- Metoda LU jest nieco wolniejsza, ale ma szersze zastosowanie.

Obie metody mają swoje mocne strony i słabości. Metoda LU jest bardziej uniwersalna, ale wymaga więcej pamięci i jest nieco wolniejsza. Metoda Choleskiego jest szybsza i bardziej optymalna obliczeniowo, ale ma ograniczenia co do rodzaju macierzy, które może przetwarzać. Wybór między nimi zależy od konkretnego problemu i dostępnych zasobów.

3. Źródła

- Prezentacja autorstwa dr hab. inż. Marcina Hojnego „Rozwiązywanie układów równań liniowych – metoda LU, Choleskiego”.
- https://www.yumpu.com/xx/document/read/41674350/triangularyzacja-choleskyego-i-normy-macierzy