


Paulina Grabowska 	Metody Numeryczne
Sprawozdanie 9 Rozwiązywanie układów równań liniowych Metoda LU i Metoda Choleskiego	

1. Wstęp

Układy równań liniowych to fundamentalny element algebry liniowej i mają szerokie zastosowanie w wielu dziedzinach nauki, inżynierii, ekonomii, oraz informatyki. Celem rozwiązywania układów równań liniowych jest znalezienie wartości zmiennych, które spełniają wszystkie równania jednocześnie.

W praktyce, układy równań liniowych stosuje się do modelowania i rozwiązywania rzeczywistych problemów. Dzięki swojej prostocie i uniwersalności, układy równań liniowych stanowią narzędzie, które jest nieodzowne w rozwiązywaniu złożonych problemów w różnych obszarach wiedzy.

2. Opis metod numerycznych

2.1. Metoda LU

Metoda LU jest techniką stosowaną do rozwiązywania układów równań liniowych, które mają postać $AX = B$, gdzie A jest macierzą współczynników, X jest wektorem niewiadomych, a B jest wektorem wyrazów wolnych. Metoda ta polega na dekompozycji macierzy A na iloczyn dwóch macierzy: trójkątnej dolnej macierzy L oraz trójkątnej górnej macierzy U , czyli $A = LU$.

Pierwszym krokiem jest rozkład macierzy A na macierze L i U . Macierz L ma elementy niezerowe tylko na i pod główną przekątną, podczas gdy macierz U ma elementy niezerowe tylko na i powyżej głównej przekątnej. Po uzyskaniu macierzy L i U , układ równań $AX = B$ można przekształcić w dwa prostsze układy:

- $LY = B$ - najpierw rozwiązywany jest układ z macierzą trójkątną dolną L i wektorem Y . Polega to na zastosowaniu podstawiania w przód.

- $UX = Y$ - następnie rozwiązywany jest układ z macierzą trójkątną górną U i wektorem X . Polega to na zastosowaniu podstawiania wstecz.

Metoda LU jest szczególnie użyteczna w przypadkach, gdy trzeba rozwiązać wiele układów równań liniowych z tą samą macierzą współczynników A , ale z różnymi wektorami wyrazów wolnych B . Dzięki jednorazowemu przeprowadzeniu dekompozycji A na L i U , można szybko rozwiązywać kolejne układy za pomocą prostych operacji podstawiania. Jest to metoda efektywna i numerycznie stabilna, co czyni ją popularnym wyborem w obliczeniach inżynierskich, ekonomicznych i naukowych.

2.2. Metoda Choleskiego

Metoda Choleskiego, znana również jako rozkład Choleskiego, jest specjalistyczną techniką stosowaną do rozwiązywania układów równań liniowych, które mają macierze symetryczne i dodatnio określone. Rozkład ten przekształca macierz współczynników A w iloczyn dwóch macierzy: trójkątnej dolnej macierzy L i jej transpozycji L^T . W związku z tym, macierz A można zapisać jako $A = LL^T$.

Pierwszym krokiem jest rozkład macierzy A na macierze L i L^T . Macierz L jest trójkątna dolna, co oznacza, że ma elementy niezerowe tylko na i pod główną przekątną, natomiast L^T jest jej transpozycją, czyli trójkątną górną macierzą.

Proces dekompozycji polega na wyznaczeniu elementów macierzy L w taki sposób, aby iloczyn LL^T dał pierwotną macierz A . Wzory na elementy macierzy L są uzyskiwane poprzez iteracyjne obliczenia, zaczynając od elementów na głównej przekątnej i stopniowo przechodząc do elementów poza przekątną.

Po uzyskaniu macierzy L i L^T , układ równań $AX = B$ można przekształcić w dwa prostsze układy:

- $LY = B$ - najpierw rozwiązywany jest układ z macierzą trójkątną dolną L i wektorem Y . Rozwiązanie pierwszego układu polega na zastosowaniu podstawiania w przód.
- $L^TX = Y$ - następnie rozwiązywany jest układ z macierzą trójkątną górną L^T i wektorem X . Rozwiązanie drugiego układu polega na zastosowaniu podstawiania wstecz.

Metoda Choleskiego jest wyjątkowo efektywna dla macierzy symetrycznych i dodatnio określonych. Jedną z głównych zalet tej metody jest jej niska złożoność obliczeniowa w

porównaniu do innych metod rozkładu, ponieważ operuje tylko na połowie macierzy, wykorzystując jej symetrię. Ponadto, metoda Choleskiego jest numerycznie stabilna i często szybsza niż metoda LU.

3. Implementacja numeryczna

3.1. Funkcje *LU* i *solveLU* (Fragment kodu 1)

Następuje deklaracja funkcji *LU*, która przyjmuje macierz A oraz wyjściowe macierze L i U . Zadeklarowana pętla zewnętrzna iteruje przez kolejne wiersze macierzy A . Pętla wewnętrzna iteruje przez kolumny U od i do $N-1$. Inicjalizowana zostaje zmienna sum , która będzie używana do obliczania sumy iloczynów elementów L i U . Następnie sumowane są iloczyny odpowiednich elementów L i U . Obliczany jest element $U[i][k]$ przez odjęcie sumy iloczynów od odpowiedniego elementu $A[i][k]$.

Następna pętla iteruje przez kolumny L od i do $N-1$. Sprawdzany jest warunek, czy jest to element diagonalny i wówczas element diagonalny L ustawiany jest na 1. Inicjalizowana jest zmienna pomocnicza sum . Pętla wewnętrzna iteruje przez wiersze L i kolumny U od 0 do $i-1$. Sumowane są odpowiednie elementy L i U . Obliczany jest element $L[k][i]$ przez odjęcie sumy iloczynów od odpowiedniego elementu $A[k][i]$, a następnie podzielony jest przez element diagonalny $U[i][i]$.

Deklarowana jest funkcja *solveLU*, która przyjmuje macierz A , wektor b oraz wyjściowy wektor x . Macierze L i U są inicjalizowane jako macierze zerowe. Wywołana zostaje funkcja *LU*, która dekomponuje macierz A na macierze L i U . Inicjalizowany zostaje wektor y jako wektor zerowy. Pętla iteruje przez elementy wektora y . Inicjalizowana jest zmienna sum dla obliczeń. Pętla wewnętrzna iteruje przez elementy wektora y do $i-1$. Zsumowane zostają iloczyny odpowiednich elementów L i y . Obliczany jest element $y[i]$ poprzez odjęcie sumy iloczynów od odpowiedniego elementu $b[i]$.

Następna pętla iteruje przez elementy wektora x od końca do początku. Ponownie inicjalizowana jest pomocnicza zmienna sum . Zsumowane zostają iloczyny odpowiednich elementów U i x . Obliczany jest element $x[i]$ przez odjęcie sumy iloczynów od odpowiedniego elementu $y[i]$, a następnie podzielenie przez element diagonalny $U[i][i]$.

```

1. void LU(double A[N][N], double L[N][N], double U[N][N]) {
2.     for (int i = 0; i < N; i++) {
3.         for (int k = i; k < N; k++) {
4.             double sum = 0;
5.             for (int j = 0; j < i; j++)
6.                 sum += (L[i][j] * U[j][k]);
7.             U[i][k] = A[i][k] - sum;
8.         }
9.
10.        for (int k = i; k < N; k++) {
11.            if (i == k)
12.                L[i][i] = 1;
13.            else {
14.                double sum = 0;
15.                for (int j = 0; j < i; j++)
16.                    sum += (L[k][j] * U[j][i]);
17.                L[k][i] = (A[k][i] - sum) / U[i][i];
18.            }
19.        }
20.    }
21. }
22.
23.
24. void solveLU(double A[N][N], double b[N], double x[N]) {
25.     double L[N][N] = {0};
26.     double U[N][N] = {0};
27.
28.     LU(A, L, U);
29.
30.     // Forward substitution to solve L*y = b
31.     double y[N] = {0};
32.     for (int i = 0; i < N; i++) {
33.         double sum = 0;
34.         for (int j = 0; j < i; j++)
35.             sum += L[i][j] * y[j];
36.         y[i] = b[i] - sum;
37.     }
38.
39.     // Backward substitution to solve U*x = y
40.     for (int i = N - 1; i >= 0; i--) {
41.         double sum = 0;
42.         for (int j = i + 1; j < N; j++)
43.             sum += U[i][j] * x[j];
44.         x[i] = (y[i] - sum) / U[i][i];
45.     }
46. }

```

Fragment kodu 1: funkcje *LU* i *solveLU*

3.2. Funkcja *cholesky* (Fragment kodu 2)

Deklarowana jest funkcja *cholesky*, która przyjmuje macierz A , wektor b oraz wyjściowy wektor x . Deklarowana jest pusta macierz trójkątna dolna L wymiaru N na N . Pętla zewnętrzna iteruje po każdym wierszu i -tego elementu macierzy L . Pętla wewnętrzna iteruje po każdym elemencie w wierszu j , aż do i -tego. Dla każdego elementu $L[i][j]$, obliczana jest suma iloczynów elementów L z odpowiednimi elementami wiersza i i j oraz elementów $A[i][i]$. Następnie obliczana jest wartość $L[i][j]$ jako pierwiastek kwadratowy z różnicy $A[i][i]$ i sumy

iloczynów. Wartość $L[i][j]$ jest aktualizowana jako iloraz różnicy $A[i][j]$ i sumy oraz elementu diagonalnego $L[j][j]$.

Tworzony jest wektor y o długości N , który będzie przechowywać rozwiązanie tymczasowe. Iteruje się po każdym elemencie wektora y . Obliczana jest suma iloczynów odpowiednich elementów macierzy L i wektora y . Elementy wektora y są aktualizowane jako iloraz różnicy $b[i]$ i sumy oraz elementu diagonalnego macierzy L .

Iteruje się w odwrotnej kolejności po każdym elemencie wektora x . Obliczana jest suma iloczynów odpowiednich elementów transponowanej macierzy L i wektora x . Elementy wektora x są aktualizowane jako iloraz różnicy $y[i]$ i sumy oraz elementu diagonalnego transponowanej macierzy L . W ten sposób funkcja przeprowadza rozkład Cholesky'ego macierzy A na iloczyn macierzy trójkątnej dolnej L i jej transpozycji, a następnie rozwiązuje układ równań liniowych $Ax = b$.

```

1. void cholesky(double A[N][N], double b[N], double x[N]) {
2.     double L[N][N] = {};
3.
4.     for (int i = 0; i < N; ++i) {
5.         for (int j = 0; j <= i; ++j) {
6.             double sum = 0.0;
7.             for (int k = 0; k < j; ++k) {
8.                 sum += L[i][k] * L[j][k];
9.             }
10.
11.             L[i][j] = sqrt(A[i][i] - sum);
12.             L[i][j] = (A[i][j] - sum) / L[j][j];
13.         }
14.     }
15.
16.
17.     // L * y = b
18.     double y[N] = {};
19.     for (int i = 0; i < N; ++i) {
20.         double sum = 0.0;
21.         for (int j = 0; j < i; ++j) {
22.             sum += L[i][j] * y[j];
23.         }
24.         y[i] = (b[i] - sum) / L[i][i];
25.     }
26.
27.     // L^T * x = y
28.     for (int i = N - 1; i >= 0; --i) {
29.         double sum = 0.0;
30.         for (int j = i + 1; j < N; ++j) {
31.             sum += L[j][i] * x[j];
32.         }
33.         x[i] = (y[i] - sum) / L[i][i];
34.     }
35. }

```

Fragment kodu 2: funkcja *cholesky*

3.3. Deklaracja N (Fragment kodu 3) i funkcja *main* (Fragment kodu 4)

Dodatkowo deklarowana jest zmienna N , która określa rozmiar macierzy i wektorów. Dla bezpieczeństwa zainicjonowana ją jako wartość `const`. Liczba N używana jest podczas deklaracji rozmiaru macierzy A i wektorów b i x w funkcji *main*. Analogicznie używana jest w funkcjach *LU* i *cholesky*, gdzie określa ona wymiary wyjściowych macierzy.

```
1. const int N = 3;
```

Fragment kodu 3: Deklaracja stałej N

W funkcji *main* inicjalizowana jest macierz A oraz wektora b z wartościami, dla których będą obliczane układy liniowe. Inicjalizowany zostaje wektor x , jako wektor zerowy. Wywołana zostaje funkcja *solveLU* w celu rozwiązania równania metodą LU. Następnie wyświetlany jest rozwiązanie dla tej metody. Aby wypisać wszystkie rozwiązania zainicjowana jest pętla iterująca przez elementy wektora x i wypisująca je na ekranie konsoli.

Analogicznie, dla tego samego zestawu danych (macierzy A , wektora b oraz nowo zainicjalizowanego wektora zerowego $x1$) wywoływana jest funkcja *cholesky*, której rozwiązania również są wypisywane za pomocą pętli iterującej po elementach wektora $x1$.

Aby porównać obie funkcje, użyto funkcji mierzenia czasu. Zadeklarowano zmienne globalne typu `clock_t` - *start* oraz *stop* oraz dwie zmienne typu `double` - *czas1* i *czas2*. Zegar uruchamiany jest przed wywołaniem funkcji i zatrzymywany jest po wypisaniu wyników na ekran konsoli. Następnie, zaraz pod wynikami wypisywany jest czas wykonania algorytmu. Dzięki temu możliwe będzie porównanie szybkości działania obu tych funkcji.

```
1. int main() {
2.     double A[N][N] = {{1, 0, 0},
3.                        {0, 1, 0},
4.                        {0, 0, 1}};
5.     double b[N] = {1, 2, 3};
6.     double x[N] = {0};
7.
8.     start = clock();
9.     solveLU(A, b, x);
10.
11.    cout << "    Solution LU:" << endl;
12.    for (int i = 0; i < N; i++)
13.        cout << " " << x[i] << " ";
14.    cout << endl;
15.    stop = clock();
16.    czas1 = (double)(stop - start) / CLOCKS_PER_SEC;
17.    cout << " time: " << czas1 << endl << endl;
18.
19.
20.    double x1[N] = {0};
21.    start = clock();
22.    solveCholesky(A, b, x1);
23.
24.    cout << "    Solution cholesky:" << endl;
```

```

25.     for (int i = 0; i < N; i++)
26.         cout << " " << x[i] << " ";
27.     cout << endl;
28.     stop = clock();
29.     czas2 = (double)(stop - start) / CLOCKS_PER_SEC;
30.     cout << " time: " << czas2 << endl << endl;
31.
32.
33.     return 0;
34. }

```

Fragment kodu 4: funkcja *main*

Sprawdzono poprawność zaimplementowanych funkcji korzystając z dwóch przykładów podanych na zajęciach.

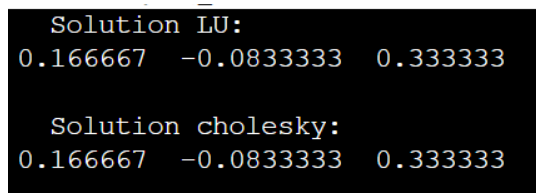
- Pierwszy test dla macierzy:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 8 & 10 \\ 3 & 10 & 22 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 7 \end{bmatrix}$$

Spodziewany wynik:

$$\begin{cases} x_1 = 0,16666667 \\ x_2 = -0,08333333 \\ x_3 = 0,33333333 \end{cases}$$

Otrzymany wynik (*Rys. 1*):



```

Solution LU:
0.166667 -0.0833333 0.333333

Solution cholesky:
0.166667 -0.0833333 0.333333

```

Rys. 1: Wydruk z konsoli dla pierwszego testu przeprowadzonego na zajęciach.

- Drugi test dla wykonano dla poniższej macierzy. Test wymagał zmiany wartości stałej N na 4:

$$\begin{bmatrix} 3 & -4 & 4 & -4 \\ 1,5 & -1 & 2 & -2 \\ 1,5 & -0,5 & 0 & -3 \\ 4,5 & -5,5 & 4 & -9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -9 \\ -3,5 \\ -2 \\ -14 \end{bmatrix}$$

Spodziewany wynik dla funkcji *LU*. W przypadku funkcji *cholesky* spodziewane są wyniki *-nan*, ponieważ jest to macierz niesymetryczna:

$$\begin{cases} x_1 = 1 \\ x_2 = 1 \\ x_3 = -1 \\ x_4 = 1 \end{cases}$$

Otrzymany wynik (Rys.2):

```
Solution LU:
-27  1  6 -13
time: 0.000103

Solution cholesky:
-nan -nan -nan -nan
time: 2.6e-05
```

Rys. 2: Wydruk z konsoli dla drugiego testu przeprowadzonego na zajęciach.

4. Testy jednostkowe

4.1. Macierz jednostkowa

Badana macierz:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Spodziewany wynik (sprawdzono w programie MatrixCalc):

$$\begin{cases} x_1 = 1 \\ x_2 = 2 \\ x_3 = 3 \end{cases}$$

Uzyskany wydruk z konsoli z wynikami oraz czasem wykonania algorytmu (Rys. 3):

```
Solution LU:
1  2  3
time: 6.6e-05

Solution cholesky:
1  2  3
time: 1.3e-05
```

Rys. 3: Wydruk z konsoli dla testu jednostkowego dla macierzy jednostkowej

4.2. Macierz 6x6

Badana macierz (Rys. 4a):

```
double A[N][N] = {
    {-4, 2, -1, 3, 0, -4},
    {1, 0, 5, -1, 0, 3},
    {-3, -2, -3, -2, 4, 5},
    {3, 3, 3, 1, -2, 0},
    {-2, 1, 0, -4, -4, -2},
    {4, -3, 4, -4, 4, -2},
};
double b[N] = {3, 0, 4, 5, 4, -5};
```

Rys. 4a: Fragment kodu przedstawiający badaną macierz 6x6

Spodziewany wynik (sprawdzono w programie MatrixCalc):

$$\begin{cases} x_1 = -0,258311 \\ x_2 = 3,50026 \\ x_3 = -0,45752 \\ x_4 = -1,25805 \\ x_5 = 0,04763 \\ x_6 = 0,429288 \end{cases}$$

Uzyskany wydruk z konsoli z wynikami oraz czasem wykonania algorytmu (Rys. 4b):

```
Solution LU:
-0.258311  3.50026  -0.45752  -1.25805  1.04763  0.429288
time: 9e-05

Solution cholesky:
-0.258311  3.50026  -0.45752  -1.25805  1.04763  0.429288
time: 2.9e-05
```

Rys. 4b: Wydruk z konsoli dla testu jednostkowego dla macierzy 6x6

4.3. Macierz 10x10

Badana macierz (Rys. 5a):

```
double A[N][N] = {
    {10, 1, 2, 3, 4, 5, 6, 7, 8, 9},
    {1, 20, 1, 2, 3, 4, 5, 6, 7, 8},
    {2, 1, 30, 1, 2, 3, 4, 5, 6, 7},
    {3, 2, 1, 40, 1, 2, 3, 4, 5, 6},
    {4, 3, 2, 1, 50, 1, 2, 3, 4, 5},
    {5, 4, 3, 2, 1, 60, 1, 2, 3, 4},
    {6, 5, 4, 3, 2, 1, 70, 1, 2, 3},
    {7, 6, 5, 4, 3, 2, 1, 80, 1, 2},
    {8, 7, 6, 5, 4, 3, 2, 1, 90, 1},
    {9, 8, 7, 6, 5, 4, 3, 2, 1, 100}
};
double b[N] = {55, 56, 57, 58, 59, 60, 61, 62, 63, 64};
```

Rys. 5a: Fragment kodu przedstawiający badaną macierz 10x10

Spodziewany wynik (sprawdzono w programie MatrixCalc):

$$\begin{cases} x_1 = 4,2437 \\ x_2 = 2,24666 \\ x_3 = 1.43121 \\ x_4 = 0,951921 \\ x_5 = 0,626271 \\ x_6 = 0,3883926 \\ x_7 = 0,207219 \\ x_8 = 0,066231 \\ x_9 = -0,044891 \\ x_{10} = -0,132903 \end{cases}$$

Uzyskany wydruk z konsoli z wynikami oraz czasem wykonania algorytmu (Rys. 5b):

```
Solution LU:
4.2437 2.24666 1.43121 0.951921 0.626271 0.388296 0.207219 0.066231 -0.0448981 -0.132903
time: 0.000103

Solution cholesky:
4.2437 2.24666 1.43121 0.951921 0.626271 0.388296 0.207219 0.066231 -0.0448981 -0.132903
time: 2.7e-05
```

Rys. 5b: Wydruk z konsoli dla testu jednostkowego dla macierzy 10x10

4.4. Macierz z zerem na głównej przekątnej

Dla tego testu zastosowano macierz z poprzedniego testu o wymiarach 10 na 10 i jedynie zamieniono liczbę 50 znajdującą się na głównej przekątnej macierzy a na liczbę 0.

Spodziewany wynik (sprawdzono w programie MatrixCalc):

$$\begin{cases} x_1 = 10.5829 \\ x_2 = 4.42418 \\ x_3 = 2.07633 \\ x_4 = 0.798588 \\ x_5 = -14.6062 \\ x_6 = -0.0455195 \\ x_7 = -0.076733 \\ x_8 = -0.0982247 \\ x_9 = -0.112758 \\ x_{10} = -0.122129 \end{cases}$$

Uzyskany wydruk z konsoli z wynikami oraz czasem wykonania algorytmu (Rys. 6):

```
Solution LU:
10.5829  4.42418  2.07633  0.798588  -14.6062  -0.0455195  -0.076733  -0.0982247  -0.112758  -0.122129
time: 6.3e-05

Solution cholesky:
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
time: 4.4e-05
```

Rys. 6: Wydruk z konsoli dla testu jednostkowego dla macierzy jednostkowej

5. Opracowanie wyników

5.1. Test dla macierzy jednostkowej

Wyniki prezentują dokładność dwóch różnych metod rozwiązywania układów równań liniowych dla trzech niewiadomych (*Tabela 1*). Oba podejścia - metoda LU oraz metoda Cholesky'ego - generują wyniki praktycznie identyczne z oczekiwanymi wartościami, co świadczy o ich skuteczności. Dodatkowo, metoda Cholesky'ego wydaje się być minimalnie bardziej efektywna czasowo, osiągając rezultaty w krótszym czasie w porównaniu z metodą LU.

Tabela 1: Zestawienie wyników dla testu dla macierzy jednostkowej

Niewiadoma	Oczekiwany wynik	Metoda LU	Metoda Choleskyego	
x_1	1	1	1	
x_2	2	2	2	
x_3	3	3	3	
		6,6e-05	1,3e-05	Czas

5.2. Test dla macierzy 6x6

Wyniki analizują układ równań liniowych dla sześciu niewiadomych (*Tabela 2*). Zarówno metoda LU, jak i metoda Cholesky'ego przynoszą rezultaty bliskie oczekiwanym wartościom dla każdej niewiadomej, co potwierdza ich dokładność. Metoda Cholesky'ego ponownie wydaje się być znacznie bardziej efektywna czasowo, osiągając wyniki w krótszym czasie w porównaniu z metodą LU, co może być istotne w praktycznych zastosowaniach. Te wyniki podkreślają skuteczność obu metod w rozwiązywaniu układów równań liniowych oraz sugerują, że metoda Cholesky'ego może być preferowana ze względu na swoją wydajność czasową.

Tabela 2: Zestawienie wyników dla testu dla macierzy 6x6

Niewiadoma	Oczekiwany wynik	Metoda LU	Metoda Choleskyego	
x_1	-0,258311	-0,258311	-0,258311	
x_2	3,50026	3,50026	3,50026	
x_3	-0,45752	-0,45752	-0,45752	
x_4	-1,25805	-1,25805	-1,25805	
x_5	0,04763	0,04763	0,04763	
x_6	0,429288	0,429288	0,429288	
		9.0e-05	2,9e-05	Czas

5.3. Test dla macierzy 10x10

Analiza układu równań liniowych dla dziesięciu niewiadomych (*Tabela 3*) pokazuje, że obie metody, LU oraz Cholesky'ego, generują wyniki bardzo zbliżone do oczekiwanych wartości dla każdej niewiadomej. Różnice między wynikami oczekiwanymi a uzyskanymi są minimalne, co potwierdza dokładność obu metod. Dodatkowo, metoda Cholesky'ego ponownie osiąga te wyniki w krótszym czasie niż metoda LU.

Tabela 3: Zestawienie wyników dla macierzy 10x10

Niewiadoma	Oczekiwany wynik	Metoda LU	Metoda Cholesky'ego	
x_1	4,2437	4,2437	4,2437	
x_2	2,24666	2,24666	2,24666	
x_3	1.43121	1.43121	1.43121	
x_4	0,951921	0,951921	0,951921	
x_5	0,626271	0,626271	0,626271	
x_6	0,3883926	0,3883926	0,3883926	
x_7	0,207219	0,207219	0,207219	
x_8	0,066231	0,066231	0,066231	
x_9	-0,044891	-0,044891	-0,044891	
x_{10}	-0,132903	-0,132903	-0,132903	
		0,000113	2,7e-05	Czas

5.4. Test dla macierzy 10x10 z zerem na głównej przekątnej

Analiza układu równań liniowych dla dziesięciu niewiadomych z zerem na głównej przekątnej (*Tabela 4*) wykazuje, że metoda LU generuje wyniki dla wszystkich niewiadomych, ale wartości są zgodne z oczekiwanymi. Natomiast metoda Cholesky'ego nie zwraca wyników dla tego zestawu danych, co sugeruje jej niestabilność w tym przypadku. Czas obliczeń dla obu metod jest zbliżony, jednakże należy podjąć dodatkowe kroki w celu zbadania stabilności i dokładności wyników uzyskanych za pomocą tych metod.

Tabela 4: Zestawienie wyników dla macierzy 10x10 z zerem na głównej przekątnej

Niewiadoma	Oczekiwany wynik	Metoda LU	Metoda Cholesky'ego	
x_1	10,5829	10,5829	-nan	
x_2	4,42418	4,42418	-nan	
x_3	2,07633	2,07633	-nan	
x_4	0,798588	0,798588	-nan	
x_5	-14,6062	-14,6062	-nan	
x_6	-0,0455195	-0,0455195	-nan	
x_7	-0,076733	-0,076733	-nan	
x_8	-0,0982247	-0,0982247	-nan	
x_9	-0,112758	-0,112758	-nan	
x_{10}	-0,122129	-0,122129	-nan	
		6,3e-05	4.4e-05	Czas

6. Wnioski

Porównując metody rozwiązywania układów równań liniowych - metodę LU i metodę Cholesky'ego - można zauważyć ich różnice i podobieństwa. Metoda Cholesky'ego wydaje się być bardziej efektywna czasowo, osiągając szybsze wyniki w porównaniu z metodą LU. Jest to istotne zwłaszcza w przypadku dużych i złożonych układów równań, gdzie czas obliczeń ma kluczowe znaczenie.

Jednakże, warto podkreślić, że metoda Cholesky'ego może być niestabilna dla niektórych układów równań, zwłaszcza tych niesymetrycznych, ujemnie określonych lub o zerowym wyznaczniku głównym. W takich przypadkach metoda LU może być bardziej niezawodna, chociaż czasem mniej efektywna. Dlatego też, wybór odpowiedniej metody powinien być uzależniony od konkretnych cech układu równań, takich jak symetria i właściwości macierzy, oraz od wymagań co do czasu obliczeń.

W praktyce, zastosowanie metody Cholesky'ego może być preferowane dla odpowiednio spójnych i symetrycznych układów równań, gdzie szybkość obliczeń jest kluczowa, podczas gdy dla bardziej złożonych lub niestandardowych przypadków metoda LU może być bardziej

uniwersalna i stabilna. W związku z tym, decyzja o wyborze metody powinna być dokładnie przemyślana, uwzględniając specyfikę problemu oraz wymagania dotyczące precyzji i wydajności obliczeń.

7. Źródła

Wykłady z Metod Numerycznych autorstwa dr hab. Danuty Szeligi

Prezentacja Rozwiązywanie układów równań liniowych - metoda LU, Choleskiego autorstwa dr. Hab. Inż. Marcina Hojnego.

Wikipedia – artykuły o układach równań liniowych, metodzie LI i metodzie Choleskiego

Program MatrixCalc: <https://matrixcalc.org/pl/slu.html>