Szymon Ociepka *Data wykonania: 15.05.2024*

Metoda LU, Choleskiego

Rozwiązywanie układów równań liniowych

Metody numeryczne

1. Wprowadzenie teoretyczne

1.1 metoda LU

Metoda LU, znana również jako dekompozycja LU, jest jednym z podstawowych algorytmów stosowanych w analizie numerycznej do rozwiązywania układów równań liniowych. Nazwa metody pochodzi od “Lower-Upper” (Dolna-Górna), ponieważ polega na rozkładzie macierzy na iloczyn dwóch macierzy: jednej **dolnotrójkątnej** (L) i jednej **górnotrójkątnej** (U).

Dekompozycja LU polega na przedstawieniu macierzy **A** jako iloczynu dwóch macierzy **L** i **U**

$$A = L \cdot U \tag{1}$$

Po zastosowaniu dekompozycji LU, pierwotny układ równań liniowych można przekształcić do postaci $A \cdot X = L \cdot U \cdot X = D$.

Dla przykładowej macierzy kwadratowej 3x3 L oraz U będą wyglądać następująco:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix}$$

Rysunek 1.1 elementy macierzy L

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Rysunek 1.2 elementy macierzy U

Obliczamy wartości poszczególnych ich elementów macierzy:

$l_{ii} = 1$

$$u_{ik} = a_{ik} - \sum_{j=1}^{i-1} l_{ij} \cdot u_{jk} \tag{2}$$

gdzie:

$k = i, i+1, \ldots, n$

$i = 1, 2, \ldots, n$

$$l_{ki} = \frac{a_{ki} - \sum_{j=1}^{i-1} l_{kj} \cdot u_{ji}}{u_{ii}} \tag{3}$$

gdzie:

$k = i+1, i+2, \ldots, n$

$i = 1, 2, \ldots, n$

Następnie, rozwiązujemy dwa układy równań: $L \cdot Y = D$ (4) dla **y**, a potem $U \cdot X = Y$ (5) dla **x**.

1.2 metoda Choleskiego

Metoda Choleskiego to efektywna metoda numeryczna stosowana do rozwiązywania układów równań liniowych. Jest ona szczególnie przydatna, gdy mamy do czynienia z macierzami symetrycznymi i dodatnio określonymi. Metoda Choleskiego polega na rozkładzie macierzy **A** na iloczyn dwóch macierzy: **L** i $\mathbf{L^T}$, gdzie **L** to macierz dolna, a $\mathbf{L^T}$ to transpozycja macierzy **L**. Rozkład ten jest możliwy dzięki temu, że macierz **A** jest symetryczna i dodatnio określona.

Rozkład Choleskiego można zapisać jako:

$$A = L \cdot L^T \tag{6}$$

Po zastosowaniu dekompozycji LU, pierwotny układ równań liniowych można przekształcić do postaci $A \cdot X = L \cdot L^T \cdot X = D$.

Dla przykładowej macierzy kwadratowej 3x3 L oraz L^T będą wyglądać następująco:

$$L = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix}$$

Rysunek 1.3 elementy macierzy L

$$L^T = \begin{bmatrix} l_{11} & l_{12} & l_{13} \\ 0 & l_{22} & l_{23} \\ 0 & 0 & l_{33} \end{bmatrix}$$

Rysunek 1.4 elementy macierzy L^T

Obliczamy wartości poszczególnych ich elementów macierzy:

$$l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2} \tag{7}$$

gdzie:

$j = 1, 2, \ldots, n$

$$l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot l_{jj}\right) \cdot \frac{1}{l_{jj}} \tag{8}$$

gdzie:

$i = 1, 2, \ldots, j-1$

Następnie, rozwiązujemy dwa układy równań: $L \cdot Y = D$ (9) dla **y**, a potem $L^T \cdot X = Y$ (10) dla **x**.

Rozwiązanie tych dwóch układów równań jest zazwyczaj szybsze i bardziej stabilne numerycznie. Ważne jest jednak, aby pamiętać, że metoda Choleskiego jest efektywna

tylko dla specyficznych macierzy (symetryczne, dodatnio określone). Dla innych typów macierzy konieczne może być zastosowanie innej metody numerycznej (np. metody LU).

2. Opis implementacji numerycznej

2.1 implementacja metody LU

```
//obliczenie L oraz U
for (int i = 0; i < n; i++)
{
    for (int j = i; j < n; j++)
    {
        U[i][j] = tab[i][j];
        for (int k = 0; k < i; k++)
            U[i][j] -= L[i][k] * U[k][j];
    }

    for (int j = i + 1; j < n; j++)
    {
        L[j][i] = tab[j][i];
        for (int k = 0; k < i; k++)
            L[j][i] -= L[j][k] * U[k][i];
        L[j][i] /= U[i][i];
    }
}
```

Fragment kodu 2.1: obliczanie macierzy dolnej trójkątnej L oraz górnej trójkątnej U

Zmienna **tab** jest to tablica dwuwymiarowa zawierająca macierz wejściową **A** oraz jej wektor wyrazów wolnych. Zmienne **L** oraz **U** są to dynamiczne tablice dwuwymiarowe, której rozmiar zależy od rozmiaru tablicy **tab**.

Pierwsza pętla **for** przechodzi przez wszystkie wiersze macierzy.

Druga pętla **for** jest odpowiedzialna za obliczanie elementów macierzy **U**. Wartość **U[i][j]** jest obliczana poprzez odejmowanie iloczynu odpowiednich elementów z macierzy **L** i **U** od elementu **tab[i][j]**.

Trzecia pętla **for** jest odpowiedzialna za obliczanie elementów macierzy **L.** Wartość **L[j][i]** jest obliczana poprzez odejmowanie iloczynu odpowiednich elementów z macierzy **L** i **U** od elementu **tab[j][i]**, a następnie dzielenie przez **U[i][i]**.

W ten sposób, macierz wejściowa tab jest rozkładana na dwie macierze: dolną trójkątną L i górną trójkątną U. Te dwie macierze mogą być następnie użyte do rozwiązania równania liniowego.

```
// obliczanie y-ów z macierzy trójkątnej dolnej
double* y = new double[n];

for (int i = 0; i < n; i++) {
    y[i] = tab[i][n];
    for (int j = 0; j < i; j++) {
        y[i] -= tab[i][j] * y[j];
    }
    y[i] /= tab[i][i];
}
```

Fragment kodu 2.2: obliczanie wartości y z macierzy trójkątnej dolnej L

Następnie po przypisaniu danych do tablicy **tab** z tablicy **L** obliczane są **wartości y** według wzoru (4), dzięki którym obliczymy z pomocą macierzy **U** nasze wartości **x**.

```
// obliczanie x-ów z macierzy trójkątnej górnej
double* x = new double[n];

for (int i = n - 1; i >= 0; i--) {
    x[i] = tab[i][n];
    for (int j = i + 1; j < n; j++) {
        x[i] -= tab[i][j] * x[j];
    }
    x[i] /= tab[i][i];
}
```

Fragment kodu 2.3: obliczanie wartości x z macierzy trójkątnej górnej U

W tym korku przypisujemy dane z tablicy **U** do tablicy **tab** oraz obliczone w poprzednim kroku wartości **y** jako jej wektor wyrazów wolnych. Finalnie obliczamy wartości **x** według wzoru (5) stosując postępowanie odwrotne. W ten sposób, **tablica x** jest wypełniana rozwiązaniami układu równań, zaczynając od ostatniego równania i idąc do góry.

2.2 implementacja metodu Choleskiego

```
//obliczenie L
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {
        double sum = 0;
        if (j == i) {
            for (int k = 0; k < j; k++)
                sum += pow(L[j][k], 2);
            L[j][j] = sqrt(tab[j][j] - sum);
        }
        else {
            for (int k = 0; k < j; k++)
                sum += L[i][k] * L[j][k];
            L[i][j] = (tab[i][j] - sum) / L[j][j];
        }
    }
}
```

Fragment kodu 2.4: obliczanie macierzy dolnej trójkątnej L

Dwie zewnętrzne pętle **for** przechodzą przez każdy element w macierzy **L**. **Pierwsza** pętla **for** przechodzi przez wiersze, a **druga** pętla **for** przechodzi przez kolumny do **i.**

Wewnątrz tych pętli, zmienna **sum** jest inicjalizowana na 0.

Następnie, jeśli **j** jest równe **i**, to jest to element na przekątnej macierzy **L**. Wtedy, dla każdego elementu w wierszu do **j**, dodajemy kwadrat elementu do **sum**. Następnie, pierwiastek kwadratowy z różnicy między elementem macierzy wejściowej **tab** na przekątnej a **sum** jest przypisywany do elementu na przekątnej macierzy **L**.

Jeśli **j** nie jest równe **i**, to jest to element poniżej przekątnej. Wtedy, dla każdego elementu w wierszu do **j**, dodajemy iloczyn odpowiednich elementów macierzy **L** do **sum**. Następnie, różnica między elementem macierzy wejściowej **tab** a **sum**, podzielona przez element na przekątnej macierzy **L**, jest przypisywana do odpowiedniego elementu macierzy **L**.

W ten sposób, ten fragment kodu wypełnia macierz **L** tak, że $L \cdot L^T$ jest równa macierzy wejściowej **A**, gdzie L^T to transpozycja macierzy **L**. Tablica **tab** zawiera tablice wejściową **A** oraz jej wektor wyrazów wolnych.

```
// obliczanie y-ów z macierzy trójkątnej dolnej L
double* y = new double[n];

for (int i = 0; i < n; i++) {
    y[i] = tab[i][n];
    for (int j = 0; j < i; j++) {
        y[i] -= tab[i][j] * y[j];
    }
    y[i] /= tab[i][i];
}
```

Fragment kodu 2.5: obliczanie wartości y z macierzy trójkątnej dolnej L

Następnie po przypisaniu danych do tablicy **tab** z tablicy **L** obliczane są **wartości y** według wzoru (9), dzięki którym obliczymy z pomocą macierzy L^T nasze wartości **x**.

```
//transponowanie macierzy L do tab
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        tab[j][i] = L[i][j];
    }
    tab[i][n] = y[i];
}
```

Fragment kodu 2.6: transpozycja macierzy L oraz przypisanie obliczonych wartości y jako wektor wyrazów wolnych do tablicy tab

Następnie transponujemy macierz ***L*** oraz od razu przypisujemy jej dane do tablicy **tab**. Również, w tym samym kroku przypisujemy nowe dane wektora wyrazów wolnych, które teraz zawierają nasze wartości **y** obliczone w poprzednim kroku.

```
// obliczanie x-ów z macierzy trójkątnej górnej
double* x = new double[n];

for (int i = n - 1; i >= 0; i--) {
    x[i] = tab[i][n];
    for (int j = i + 1; j < n; j++) {
        x[i] -= tab[i][j] * x[j];
    }
    x[i] /= tab[i][i];
}
```

Fragment kodu 2.7: obliczanie wartości x z transponowanej macierzy L

Mając macierz L^T oraz wartości **y**, obliczamy wartości **x** za pomocą wzoru (10) stosując postępowanie odwrotne. W ten sposób, **tablica x** jest wypełniana rozwiązaniami układu równań, zaczynając od ostatniego równania i idąc do góry.

3. Testy kodu numerycznego

Test I

Macierz 3x3 = {(4, -2, 2); (-2, 2, 2); (2, 2, 14)}

wektor wyrazów wolnych = {(-6); (4); (0)}

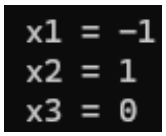
Wynik programu Matrix calculator:

$x_1 = -1$

$x_2 = 1$

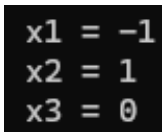
$x_3 = 0$

Wynik zaimplementowanej metody LU:



Fragment konsoli: wynik LU dla testu I

Wynik zaimplementowanej metody Choleskiego:



Fragment konsoli: wynik Choleskiego dla testu I

Test II

Macierz 4x4 = {(25, 15, 20, -15); (15, 13, 20, -15); (20, 20, 48, -4); (-15, -15, -4, 59)}

wektor wyrazów wolnych = {(-75); (-65); (-76); (137)}

Wynik programu Matrix calculator:

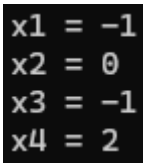
$x_1 = -1$

$x_2 = 0$

$x_3 = -1$

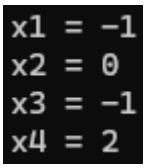
$x_4 = 2$

Wynik zaimplementowanej metody LU:



Fragment konsoli: wynik LU dla testu II

Wynik zaimplementowanej metody Choleskiego:



Fragment konsoli: wynik Choleskiego dla testu II

Test III

Macierz 3x3 = {(1, 2, 3); (2, 8, 10); (3, 10, 22)}
wektor wyrazów wolnych = {(1); (3); (7)}

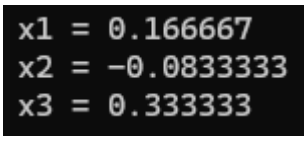
Wynik programu Matrix calculator:

$x_1 = 0.1666667$

$x_2 = -0.0833333$

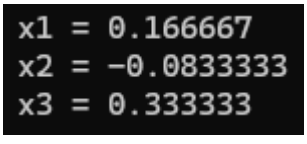
$x_3 = 0.3333333$

Wynik zaimplementowanej metody LU:



Fragment konsoli: wynik LU dla testu III

Wynik zaimplementowanej metody Choleskiego:



Fragment konsoli: wynik Choleskiego dla testu III

Test IV

Macierz 6x6 = {(4, 2, 1, 3, 0, 4); (1, 3, 5, 1, 0, 3); (3, 2, 3, 2, 4, 5); (3, 3, 3, 5, 1, 0); (2, 1, 0, 4, 20, 2); (4, 3, 4, 4, 4, 12)}

wektor wyrazów wolnych = {(3); (0); (4); (5); (4); (5)}

Wynik programu Matrix calculator:

$x_1 = 1.3997817$

$x_2 = -4.1205240$

$x_3 = 2.0043668$

$x_4 = 1.4305677$

$x_5 = -0.0037118$

$x_6 = -0.1635371$

Wynik zaimplementowanej metody LU:

```
x1 = 1.39978
x2 = -4.12052
x3 = 2.00437
x4 = 1.43057
x5 = -0.00371179
x6 = -0.163537
```

Fragment konsoli: wynik LU dla testu IV

Wynik zaimplementowanej metody Choleskiego:

```
x1 = -173.5
x2 = -133
x3 = 243
x4 = 27.5
x5 = 19.75
x6 = -5.25
```

Fragment konsoli: wynik Choleskiego dla testu IV

Test V

Macierz 6x6 = {(50.53, 4.35, 5.3, 5.5, 4.63, 3.62); (4.35, 68.23, 63.2, 6.3, 4.62, 0.35); (5.3, 63.2, 60.32, 2.63, 5.3, 6.23); (5.5, 6.3, 2.63, 41.98, 3.85, 8.5); (4.63, 4.62, 5.3, 3.85, 156.7, 9.53); (3.62, 0.35, 6.23, 8.5, 9.53, 180.3)}

wektor wyrazów wolnych = {(4.2); (4.32); (4); (8.5); (0.4); (6.4)}

Wynik programu Matrix calculator:

x_1 = 0,04886481

x_2 = −0,2340130

x_3 = 0,2971817

x_4 = 0,2102403

x_5 = −0,008134223

x_6 = 0,01521935

Wynik zaimplementowanej metody LU:

```
x1 = 0.0488648
x2 = -0.234013
x3 = 0.297182
x4 = 0.21024
x5 = -0.00813422
x6 = 0.0152193
```

Fragment konsoli: wynik LU dla testu V

Wynik zaimplementowanej metody Choleskiego:

```
x1 = 0.0488648
x2 = -0.234013
x3 = 0.297182
x4 = 0.21024
x5 = -0.00813422
x6 = 0.0152193
```

Fragment konsoli: wynik Choleskiego dla testu V

4. Analiza wyników

W celu głębszej analizy powstała tabela, w której porównane zostały wyniki zaimplementowanych metod z wynikami programu „Matrix calculator” na podstawie powyższych testów.

Tabela 1: wyniki Testu I

wartości	**Matrix Calculator**	**LU**	**Choleski**
x_1	-1	-1	-1

x_2	1	1	1
x_3	0	0	0

Tabela 2: wyniki Testu II

wartości	**Matrix Calculator**	**LU**	**Choleski**
x_1	-1	-1	-1
x_2	0	0	0
x_3	-1	-1	-1
x_4	2	2	2

Tabela 3: wyniki Testu III

wartości	**Matrix Calculator**	**LU**	**Choleski**
x_1	0.1666667	0.1666667	0.1666667
x_2	−0.0833333	−0.0833333	−0.0833333
x_3	0.3333333	0.3333333	0.3333333

Tabela 4: wyniki Testu IV

wartości	**Matrix Calculator**	**LU**	**Choleski**
x_1	1.3997817	1.39978	-173.5
x_2	−4.1205240	−4.12052	-133
x_3	2.0043668	2.00437	243
x_4	1.4305677	1.43057	27.5
x_5	−0.0037118	−0.00371179	19.75
x_6	−0.1635371	−0.163537	-5.25

Tabela 5: wyniki Testu V

wartości	**Matrix Calculator**	**LU**	**Choleski**
x_1	0,04886481	0,0488648	0,0488648

x_2	−0,2340130	−0,234013	−0,234013
x_3	0,2971817	0,297182	0,297182
x_4	0,2102403	0,21024	0,21024
x_5	−0,008134223	−0,00813422	−0,00813422
x_6	0,0152193 5	0,0152193	0,0152193

Dzięki analizie wyników można stwierdzić, że przedstawiona metoda została zaimplementowana poprawnie. Wyniki uzyskane za pomocą zaimplementowanego programu z wyjątkiem testu IV są praktycznie identyczne względem wyników uzyskanych za pomocą programu Matrix Calculator. Jednakże, występuje parę przypadków gdzie zauważono minimalne rozbieżności, które mogą wynikać z precyzji elementów - liczby zmiennoprzecinkowe o dużej liczbie miejsc po przecinku. W trakcie obliczeń mogą wystąpić błędy zaokrąglenia wynikające z ograniczonej precyzji liczb zmiennoprzecinkowych. Te błędy mogą się kumulować w trakcie obliczeń, prowadząc do zaokrąglania wyników. Niepoprawny wynik testu IV dla metody Choleskiego jest spowodowany tym iż, metoda ta ma pewne ograniczenia, działa tylko dla macierzy symetrycznych i dodatnio określonych. Jeżeli macierz nie spełnia tych warunków, metoda Choleskiego może nie działać poprawnie. Dla tego przykładu (IV) widać że macierz nie jest symetryczna, ponieważ wyrazy położone symetrycznie względem przekątnej głównej nie są równe. W przypadku macierzy w teście IV, na przykład, element w drugim wierszu, pierwszej kolumnie (1) nie jest równy elementowi w pierwszym wierszu, drugiej kolumnie (2). Zatem, ta macierz **nie jest symetryczna**.

5. Wnioski

W tym sprawozdaniu przedstawiono oraz omówiono metodę LU używaną do rozwiązywania układów równań liniowych. Jest to metoda efektywna, ponieważ pozwala na rozwiązanie różnych układów równań z tą samą macierzą, ale różnymi wektorami prawych stron. Metoda ta jest stosowana w wielu dziedzinach, takich jak inżynieria, fizyka, ekonomia, a także w naukach komputerowych. Przedstawiona została również metoda Choleskiego, która jest specjalnym przypadkiem metody LU i jest stosowana do rozwiązywania układów równań liniowych, gdzie macierz jest symetryczna i dodatnio określona. Jest to metoda efektywna i szybka, ale jej zastosowanie jest ograniczone do pewnych typów problemów. Pokazano oraz wytłumaczono prostą implementację tych metod w języku c++. Z analizy przeprowadzonych testów wynika że program został zaimplementowany poprawnie. Testy wykazały, że zastosowana implementacja jest szczególnie efektywna pod względem uzyskanych wyników, natomiast mogą wystąpić minimalne błędy zaokrąglenia wynikające z ograniczonej precyzji liczb zmiennoprzecinkowych.

6. Źródła

- prezentacja z zajęć „lab 10.pdf”

- Beata Pańczyk, Edyta Łukasik, Jan Sikora, Teresa Guziak, Metody numeryczne w przykładach, Politechnika Lubelska, 2012
- matrixcalc.org