Metody numeryczne
Projekt nr 2

Malwina Wojewoda

6 stycznia 2022

1 Treść zadania

Rozwiązywanie równania macierzowego $AX = B$ w dziedzinie zespolonej metodą Cholesky'ego-Banachiewicza (rozkład LDL^H).
Zakładamy, że macierz A jest hermitowska dodatnio określona i jest macierzą trójdiagonalną. W pamięci komputera należy przechowywać wyłącznie trzy przekątne macierzy A.

2 Opis metody

2.1 rozkład LDL^H

Rozkład LDL^* jest wariantem klasycznego rozkładu Cholesky'ego-Banachiewicza, w którym L to macierz trójkątna dolna, a D diagonalna. Oznacza to, że wymaga się, aby elementy na diagonali macierzy L wynosiły 1 kosztem wprowadzenia dodatkowej macierzy diagonalnej D w rozkładzie. Główną zaletą tego rozkładu jest to, że może być obliczany i używany za pomocą zasadniczo tych samych algorytmów, ale unika się wyciągania pierwiastków kwadratowych.

Rozkład LDL^* jest związany z klasycznym rozkładem Cholesky'ego-Banachiewicza (LL*) w następujący sposób:
$A = LDL^* = LD^{1/2}(D^{1/2})^*L^* = LD^{1/2}(LD^{1/2})^*$

I odwrotnie, biorąc pod uwagę klasyczny rozkład Cholesky'ego $A = CC^*$ macierzy dodatnio określonej, jeśli X jest macierzą diagonalną, która zawiera główną przekątną macierzy C, to a A można rozłożyć jako LDL^*, gdzie:
$L = CX^{-1}$ (to przeskalowuje każdą kolumnę, aby elementy na przekątnej miały wartość 1)
$D = X^2$

Jeżeli A jest dodatnio określona, to wszystkie elementy diagonalne D są dodatnie.

Weźmy:

$$L(i,j) = \begin{cases} 1, & i = j \\ l_{ij}, & j < i \\ 0, & j > i \end{cases}$$

$$D(i,j) = \begin{cases} d_{i,i}, & i = j \\ 0, & i \neq j \end{cases}$$

Wtedy:

$$L^*(i,j) = L(j,i) = \begin{cases} 1, & i = j \\ l_{ij}, & i < j \\ 0, & i > j \end{cases}$$

Mamy też:

$$(LD)(i,j) = \begin{cases} d_{j,j}, & i = j \\ d_{j,j}l_{ij}, & j < i \\ 0, & j > i \end{cases}$$

I dalej:

$$A(i,j) = (LDL^*)(i,j) = \begin{cases} d_{i,i} + \sum\limits_{k=1}^{j-1} d_{k,k} l_{ik}^2, & i = j \\ d_{j,j} l_{ij} + \sum\limits_{k=1}^{j-1} d_{k,k} l_{ik} l_{jk}, & i < j \\ (LDL^*)(j,i), & j < i \end{cases}$$

Z tego idąc pierwszą kolumną otrzymujemy:
$a_{1,1} = d_{1,1}$
$a_{2,1} = d_{1,1} l_{2,1} \implies l_{2,1} = \frac{a_{2,1}}{d_{1,1}}$
$a_{3,1} = d_{1,1} l_{3,1} \implies l_{3,1} = \frac{a_{3,1}}{d_{1,1}}$, przy czym w tym momencie zauważam, że element $a_{3,1} = 0$, oraz kolejne w tej kolumnie również będą zerami, ponieważ rozważana macierz A jest macierzą trójdiagonalną.

Spójrzmy teraz na drugą kolumnę. Nie ma sensu jednak rozważać wyrazów ponad główną przekątną ponieważ macierz jest hermitowska. Mam więc:
$a_{2,2} = d_{2,2} + d_1 l_{2,1}^2 \implies d_{2,2} = a_{2,2} - d_1 l_{2,1}^2$
$a_{3,2} = d_{2,2} l_{3,2} + l_{3,1} d_1 l_{2,1} \implies l_{3,2} = \frac{a_{3,2} - l_{3,1} d_1 l_{2,1}}{d_{2,2}}$, i tu zauważam, że $l_{3,1} = 0$, czyli $l_{3,2} = \frac{a_{3,2}}{d_2}$
$a_{4,2} = d_{2,2} l_{4,2} + l_{4,1} d_1 l_{2,1}$ i w tym momencie zauważam, że $l_{4,1} = 0$, więc $a_{4,2} = d_{2,2} l_{4,2} \implies l_{4,2} = \frac{a_{4,2}}{d_{2,}} = 0$ ponieważ $a_{4,2} = 0$. Analogicznie dla kolejnych elementów tej kolumny oraz kolejnych kolumn.

W takim razie wzory na wyrazy macierzy L oraz D, które są składowymi rozkładu LDL^* macierzy trójdiagonalnej A są następujące:

$$D(i,j) = \begin{cases} a_{i,i} - d_{i-1,i-1} l_{i,i-1}, & i = j \\ 0, & i \neq j \end{cases}$$

$$L(i,j) = \begin{cases} 1, & i = j \\ \frac{a_{i,j}}{d_{j,j}}, & j < i \\ 0, & j > i \end{cases}$$

Macierz A jest dodatnio określona, więc $a_{i,i} > 0$, więc w algorytmie nie będzie problemu z dzieleniem przez zero.

2.2 Rozwiązywanie równania macierzowego z wykorzystaniem rozkładu LDL^*

$AX = B$ można zapisać jako $LDL^*X = B$, czyli $L(D(L^*X)) = B$
Podstawiając $y = L^*X$ mamy $L(Dy) = B$
następnie podstawiamy $z = Dy$ i otrzymujemy $Lz = B$.

Zatem rozwiązywanie $AX = B$ jest procesem 3-etapowym:
Najpierw musimy obliczyć $Lz = B$,
następnie $Dy = z$
i na końcu $L^*X = y$.

Aby obliczyć $Lz = B$ zastosuję metodę podstawiania w przód. Zauważmy, że to równanie może być zapisane jako:

$$\begin{array}{ccccccccc} \ell_{1,1} z_1 & & & & & & & = & b_1 \\ \ell_{2,1} z_1 & + & \ell_{2,2} z_2 & & & & & = & b_2 \\ \vdots & & \vdots & & \ddots & & & & \vdots \\ \ell_{m,1} z_1 & + & \ell_{m,2} z_2 & + & \cdots & + & \ell_{m,m} z_m & = & b_m \end{array}$$

i z tego otrzymuję:

$$z_1 = \frac{b_1}{\ell_{1,1}},$$

$$z_2 = \frac{b_2 - \ell_{2,1} z_1}{\ell_{2,2}},$$

$$\vdots$$

$$z_m = \frac{b_m - \sum_{i=1}^{m-1} \ell_{m,i} z_i}{\ell_{m,m}}.$$

Pamiętając, że rozważam macierz trójdiagonalną zauważam, że pewne elementy odejmowane w liczniku będą zerami i wtedy:
$z_i = \frac{b_i - \ell_{i,i-1} z_{i-1}}{\ell_{m,m}}$
Zauważmy, też że wyrazy na diagonali macierzy L są jedynkami, więc:
$z_i = b_i - \ell_{i,i-1} z_{i-1}$

Teraz musimy obliczyć $Dy = z$.
D jest macierzą diagonalną więc widać bezpośrednio:
$y_i = \frac{z_i}{d_{i,i}}$

Pozostało tylko obliczyć $L^* X = y$
Zastosuje tu podstawianie wstecz, ponieważ L^* to macierz trójkątna górna. Można zapisać to rówananie jako:

$$\begin{array}{ccccccc}
\ell^*_{1,1}x_1 + \ell^*_{1,2}x_2 & + & \cdots & + & \ell^*_{1,m}x_m & = & y_m \\
 & \ddots & & & \vdots & & \vdots \\
 & & \ell^*_{m-1,m-1}x_{m-1} & + & \ell^*_{m-1,m}x_m & = & y_m \\
 & & & & \ell^*_{m,m}x_m & = & y_m
\end{array}$$

i z tego otrzymuję, zauważając już elementy które się wyzerują i to, że na diagonali L są jedynki:

$$\begin{aligned}
x_m &= \frac{y_m}{\ell^*_{m,m}}, \\
x_{m-1} &= y_{m-1} - \ell^*_{m-1,m}x_m \\
&\vdots \\
x_1 &= y_1 - \ell^*_{1,2}x_2
\end{aligned}$$

Otrzymany w ten sposób wektor X jest rozwiązaniem równania.

3 Opis programu obliczeniowego

Mój program obliczeniowy składa się z następujących funkcji:

3.1 myLDLHsolve

Funkcja ta przyjmuje jako argumenty:

- *diagA* - jest to diagonala macierzy A, dla której rozwiązujemy równanie $AX = B$ (zapisana jako wektor poziomy)
- *dolnaDiagA* - jest to -1-wsza diagonala macierzy A, czyli te wyrazy macierzy, które znajdują się bezpośrednio pod główną przekątną macierzy A, dla której rozwiązujemy równanie $AX = B$ (zapisana jako wektor poziomy)
- B - wektor B, dla którego rozwiązujemy równanie $AX = B$ (zapisany jako wektor poziomy)

Funkcja ta zwraca wektor X, czyli szukane rozwiązanie równania macierzowego $AX = B$.

```
function x = myLDLHsolve(diagA, dolnaDiagA, B)

    n = length(diagA);
    if (length(dolnaDiagA)+1 ~= n) % dolna diagonala musi byc o 1 krotsza niz glowna
        error("Podano zle wymiary macierzy A")
    end
    if (length(B) ~= n) % wymiar wektora B musi byc zgodny z wymiarem macierzy A
        error("Nieprawidlowa d ugosc wektora B.")
    end

    L = complex(zeros(n,n)); %tworze macierz L odpowiednich wymiarow, wypelniona zerami
```

```
    for i = 1:n
        L(i,i)=1; %wstawiam jedynki na diagonale macierzy L
    end

    D = complex(zeros(n,1)); %tworze wektor D wypelniony zerami (kt ry jest diagonala macierzy D)
    z = complex(zeros(n,1)); %wektory z i y sa pomocnicze przy rozwiazywaniu r wnania
    y = complex(zeros(n,1));
    x = complex(zeros(n,1));

    D(1) = diagA(1); %wstawiam wartosci do macierzy D zgodnie z algorytmem
    z(1) = B(1); % obliczanie Lz = B
    y(1) = z(1)/D(1);  % Dy = z

    if length(diagA) > 1 %macierze wieksze ni  1x1:
    L(2, 1) = dolnaDiagA(1)/D(1); %wstawiam wartosci do macierzy D zgodnie z algorytmem
        for i=2:n
              D(i) = diagA(i)-diagA(i-1)*L(i,i-1)*L(i,i-1);
              z(i) = B(i)-L(i, i-1)*z(i-1); % obliczanie Lz = B
              y(i) = z(i)/D(i); % obliczanie Dy = z
            if i+1 <= n
                j=i+1;
                    L(j,i)=dolnaDiagA(i)/D(i);
            end
        end
    end
    LH = transpose(conj(L)); %macierz L* to macierz L sprz  ona i transponowana
    x(n) = y(n);
    for i=n-1:-1:1
        x(i) = y(i)-LH(i, i+1)*x(i+1); %backward substitution czyli obliczanie L*x=y
    end
```

Pozostałe funkcje służą głównie do analizy wyników.

3.2 czyDodatnioOkreslona

Funkcja ta przyjmuje jako argumenty:

- *diagA* - jest to diagonala macierzy A, zapisana jako wektor poziomy
- *dolnaDiagA* - jest to -1-wsza diagonala macierzy A, czyli te wyrazy macierzy, które znajdują się bezpośrednio pod główną przekątną macierzy A

Funkcja ta zwraca wartość logiczną, która mówi czy macierz trójdiagonalna A jest macierzą dodatnio określoną.

```
function isDodatnioOkreslona = czyDodatnioOkreslona(diagA, dolnaDiagA)
    gornaDiagA = conj(dolnaDiagA);
    A = diag(dolnaDiagA, -1) + diag(diagA, 0) + diag(gornaDiagA, 1);
    d = eig(A);
    if all(d > 0)
        isDodatnioOkreslona = true;
    else
        isDodatnioOkreslona = false;
    end
```

3.3 wbudowanyLDL

Funkcja ta przyjmuje jako argumenty:

- *diagA* - jest to diagonala macierzy A, zapisana jako wektor poziomy
- *dolnaDiagA* - jest to -1-wsza diagonala macierzy A, czyli te wyrazy macierzy, które znajdują się bezpośrednio pod główną przekątną macierzy A, zapisana jako wektor poziomy

Funkcja ta zwraca macierze L oraz D, które są rozkładem LDL^* macierzy A. Obliczone zostały z wykorzystaniem wbudowanej funkcji *ldl*().

```
function [L, D] = wbudowanyLDL(diagA, dolnaDiagA)
    gornaDiagA = conj(dolnaDiagA);
    A = diag(dolnaDiagA, -1) + diag(diagA, 0) + diag(gornaDiagA, 1);
    [L, D]  = ldl(A);
```

3.4 myLDL

Funkcja ta przyjmuje jako argumenty:

- *diagA* - jest to diagonala macierzy A, zapisana jako wektor poziomy
- *dolnaDiagA* - jest to -1-wsza diagonala macierzy A, czyli te wyrazy macierzy, które znajdują się bezpośrednio pod główną przekątną macierzy A, zapisana jako wektor poziomy

Funkcja ta zwraca macierze L oraz D, które są rozkładem LDL^* macierzy A. Funkcja została zaimplementowana przeze mnie.

```
function [L, D] = myLDL(diagA, dolnaDiagA)
    if (length(dolnaDiagA)+1 ~= length(diagA))
        error("Podano zle wymiary macierzy A")
    end
    n = length(diagA);
    L = complex(zeros(n,n)); %tworze macierz L odpowiednich wymiarow, wypelniona zerami
    for i = 1:n
        L(i,i)=1; %wstawiam jedynki na diagonale macierzy L
    end

    D = complex(zeros(n,1)); %tworze wektor D wypelniony zerami (kt ry jest diagonala macierzy D)

    D(1) = diagA(1); %wstawiam wartosci do macierzy D zgodnie z algorytmem
    if length(diagA) > 1 %macierze wieksze n i  1x1:
    L(2, 1) = dolnaDiagA(1)/D(1); %wstawiam wartosci do macierzy D zgodnie z algorytmem
        for i=2:n
            D(i) = diagA(i)-diagA(i-1)*L(i,i-1)*L(i,i-1);
            if i+1 <= n
                j=i+1;
                    L(j,i)=dolnaDiagA(i)/D(i);
            end
        end
    end
    D = diag(D);
```

3.5 wbudowanySolve

Funkcja ta przyjmuje jako argumenty:

- L - macierz, która składa się na rozkład LDL^* rozważanej macierzy A, dla której rozwiązuję równanie macierzowe $AX = B$
- D - macierz diagonalna, która składa się na rozkład LDL^* rozważanej macierzy A, dla której rozwiązuję równanie macierzowe $AX = B$
- B wektor B, dla którego rozwiązuję równanie macierzowe $AX = B$ zapisany jako wektor poziomy

Funkcja ta zwraca X, czyli rozwiązanie równania $AX = B$. Do obliczeń zostaje wykorzystana wbudowana funkcja *linsolve*().

```
function X = wbudowanySolve(L, D, B)
    n = length(D);
    if (length(B) ~= n) % wymiar wektora B musi by  zgodny z wymiarem macierzy A
        error("Nieprawid owa d ugo   wektora B.")
    end
    LH = transpose(conj(L));
    Y = linsolve(L, transpose(B));
    Z = linsolve(D, Y);
    X = linsolve(LH, Z);
```

3.6 mySolve

Funkcja ta przyjmuje jako argumenty:

- L - macierz, która składa się na rozkład LDL^* rozważanej macierzy A, dla której rozwiązuję równanie macierzowe $AX = B$
- D - macierz diagonalna, która składa się na rozkład LDL^* rozważanej macierzy A, dla której rozwiązuję równanie macierzowe $AX = B$
- B wektor B, dla którego rozwiązuję równanie macierzowe $AX = B$ zapisany jako wektor poziomy

Funkcja ta zwraca X, czyli rozwiązanie równania $AX = B$ poprzez zaimplementowany przeze mnie algorytm wykorzystujący podstawiania do przodu i wstecz.

```
function x = mySolve(L, D, B)
    n = length(L);
    if (length(B) ~= n) % wymiar wektora B musi by  zgodny z wymiarem macierzy A
        error("Nieprawid owa d ugo   wektora B.")
    end
    z = complex(zeros(n,1)); %wektory z i y s  pomocnicze przy rozwi zywaniu r wnania
    y = complex(zeros(n,1));
    x = complex(zeros(n,1));

    D = diag(D);
    z(1) = B(1); % obliczanie Lz = B
    y(1) = z(1)/D(1);  % Dy = z
    for i=2:n
            z(i) = B(i)-L(i, i-1)*z(i-1); % obliczanie Lz = B
            y(i) = z(i)/D(i); % obliczanie Dy = z
    end
    LH = transpose(conj(L)); %macierz L* to macierz L sprz  ona i transponowana
    x(n) = y(n);
    for i=n-1:-1:1
        x(i) = y(i)-LH(i, i+1)*x(i+1); %backward substitution czyli obliczanie L*x=y
    end
```

4 Przykłady

4.1 Przykład 1: $A \in \mathbb{R}^{2x2}$

$A = \begin{bmatrix} 4 & 1 \\ 1 & 9 \end{bmatrix}$, $B = \begin{bmatrix} 8 \\ 12 \end{bmatrix}$

Na początku sprawdzam czy ta macierz jest dodatnio określona.

```
>> czyDodatnioOkreslona([4 9], [1])
ans =
    logical
    1
```

Sprawdzę teraz czy zaimplementowany przeze mnie algorytm rozkładu LDL^* daje ten sam wynik co wbudowana funkcja $ldl()$.

```
>> [L1, D1] = wbudowanyLDL([4 9], [1]);
[myL1, myD1] = myLDL([4 9], [1]);
[L1, D1] == [myL1, myD1]
ans =
  2×4 logical array
   1   1   1   1
   1   1   1   1
```

Wynik jest zgodny, zatem w tym przypadku mój algorytm działa prawidłowo.

Sprawdzę teraz czy zaimplementowany przeze mnie algorytm rozwiązywania równania daje ten sam wynik co rozwiązywanie go z wykorzystaniem wbudowanej funkcji $linsolve()$.

```
>> X1 = wbudowanySolve(L1, D1, [8 12]);
myX1 = wbudowanySolve(L1, D1, [8 12]);
X1 == myX1
ans =
  2×1 logical array
   1
   1
```

Sprawdzę jeszcze czy funkcja rozwiązująca równanie $myLDLHsolve$ daje poprawny wynik.

```
>> mojX1 = myLDLHsolve([4 9], [1], [8 12]);
X1 == mojX1
ans =
  2×1 logical array
   1
   1
```

Widać, że w tym przypadku zaimplementowana przeze mnie funkcja działa prawidłowo.

4.2 Przykład 2: $A \in \mathbb{C}^{2x2}$

$A = \begin{bmatrix} 13 & -i \\ i & 2 \end{bmatrix}$, $B = \begin{bmatrix} 10 \\ 3 \end{bmatrix}$

Na początku sprawdzam czy ta macierz jest dodatnio określona.

```
>> czyDodatnioOkreslona([13 2], [i])
ans =
  logical
   1
```

Sprawdzę teraz czy zaimplementowany przeze mnie algorytm rozkładu LDL^* daje ten sam wynik co wbudowana funkcja $ldl()$.

```
>> [L2, D2] = wbudowanyLDL([13 2], [1i])
L2 =
   1.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0769i   1.0000 + 0.0000i
D2 =
  13.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   1.9231 + 0.0000i

>> [myL2, myD2] = myLDL([13 2], [1i])
myL2 =
   1.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0769i   1.0000 + 0.0000i
myD2 =
   13.0000         0
         0    2.0769
```

Wynik jest zgodny w przypadku macierzy L, jednak drugi element na diagonali macierzy $myD2$ obliczony jest niedokładnie.

Sprawdzę teraz czy zaimplementowany przeze mnie algorytm rozwiązywania równania daje ten sam wynik co rozwiązywanie go z wykorzystaniem wbudowanej funkcji $linsolve()$. Ze względu na różnice jako macierze L i D wezmę te wyznaczone przez wbudowaną funkcję ldl.

```
>> X2 = wbudowanySolve(L2, D2, [10 3])
myX2 = mySolve(L2, D2, [10 3])
X2 =
   0.8000 + 0.1200i
   1.5600 - 0.4000i
myX2 =
   0.8000 + 0.1200i
   1.5600 - 0.4000i

>> X2 == myX2
ans =
  2×1 logical array
   0
   0
```

Wynik mimo, że wygląda tak samo przy porównaniu okazuje się nie identyczny, co wskazuje na niedokładność obliczeń.

Sprawdzę jeszcze działanie funkcji $myLDLHsolve$.

```
mojX2 = myLDLHsolve([13 2], [1i], [10 3])
mojX2 =
   0.7977 + 0.1111i
   1.4444 - 0.3704i

>> X2 = wbudowanySolve(myL2, myD2, [10 3])
X2 =
   0.7977 + 0.1111i
   1.4444 - 0.3704i
```

Widać, że funkcja ta daje inny wynik niż obliczany wcześniej z wykorzystaniem macierzy $L2$ i $D1$, ponieważ jest ona tak zaimplementowana, że korzysta z macierzy $myL2$ i $myD1$. Jeśli porównamy wyniki dla funkcji $wbudowanySolve()$ z tymi argumantami, to widać, że wyniki są zgodne. W takim razie problem z niedokładnością obliczeń jest w przypadku samego rozkładu LDL.

4.3 Przykład 3: $A \in \mathbb{R}^{3x3}$

$A = \begin{bmatrix} 4 & 1 & 0 \\ 1 & 15 & 8 \\ 0 & 8 & 12 \end{bmatrix}$, $B = \begin{bmatrix} 6 \\ 3 \\ 20 \end{bmatrix}$

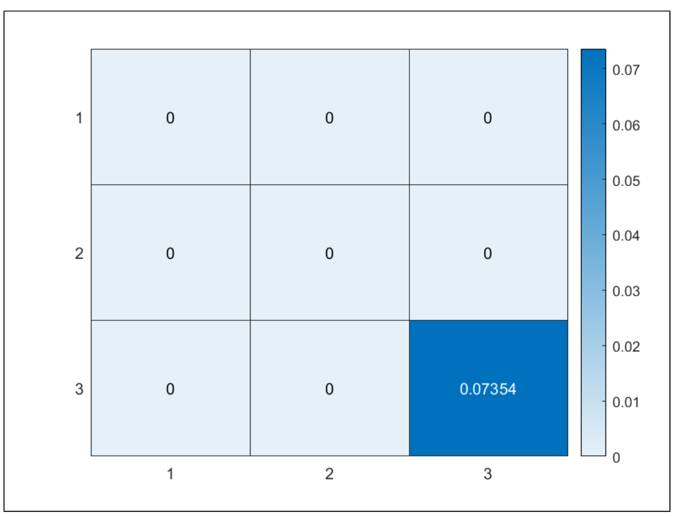
Na początku sprawdzam czy ta macierz jest dodatnio określona.

```
>> czyDodatnioOkreslona([4 15 12], [1 8])
ans =
  logical
   1
```

Sprawdzę teraz czy zaimplementowany przeze mnie algorytm rozkładu LDL^* daje ten sam wynik co wbudowana funkcja $ldl()$.

```
>> [L3, D3] = wbudowanyLDL([4 15 12], [1 8])
L3 =
    1.0000         0         0
    0.2500    1.0000         0
         0    0.5424    1.0000
D3 =
    4.0000         0         0
         0   14.7500         0
         0         0    7.6610

>> [myL3, myD3] = myLDL([4 15 12], [1 8])
myL3 =
    1.0000         0         0
    0.2500    1.0000         0
         0    0.5424    1.0000
myD3 =
    4.0000         0         0
         0   14.7500         0
         0         0    7.5875
```

Wynik jest zgodny w przypadku macierzy L, jednak trzeci element na diagonali macierzy $myD3$ obliczony jest niedokładnie.
Tę niedokładność została zwizualizowana na Rysunku 1:



Rysunek 1: wizualizacja niedokładności obliczeń macierzy D3

Sprawdzę jeszcze działanie funkcji $myLDLHsolve$.

```
>> X3 = wbudowanySolve(myL3, myD3, [6 3 20]);
```

```
>> mojX3 = myLDLHsolve([4 15 12], [1 8], [6 3 20]);
X3 == mojX3
ans =
  3×1 logical array
   1
   1
   1
```

Widać, że jeśli porównujemy z rozwiązywaniem równania przy pomocy funkcji *linsolve* dla macierzy, które są wynikiem mojego algorytmu LDL funkcja daje te same wyniki, więc niedokładność obliczeń znowu napotykana jest przy samym algorytmie LDL^*.

4.4 Przykład 4: $A \in \mathbb{C}^{3x3}$

$$A = \begin{bmatrix} 25 & -5i & 0 \\ 5i & 2 & 2i \\ 0 & -2i & 11 \end{bmatrix}, B = \begin{bmatrix} 50 \\ 20 \\ 4 \end{bmatrix}$$

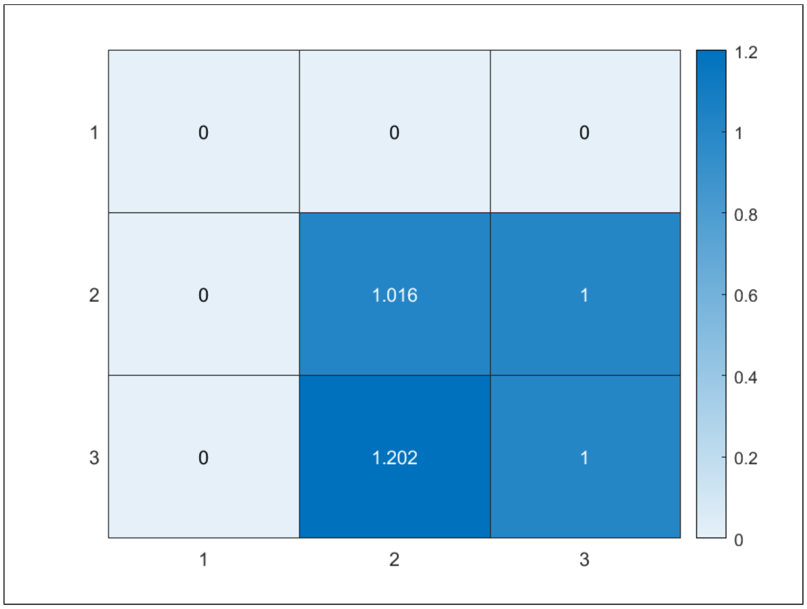
Na początku sprawdzam czy ta macierz jest dodatnio określona.

```
>> czyDodatnioOkreslona([25 2 11], [5i -2i])
ans =
  logical
   1
```

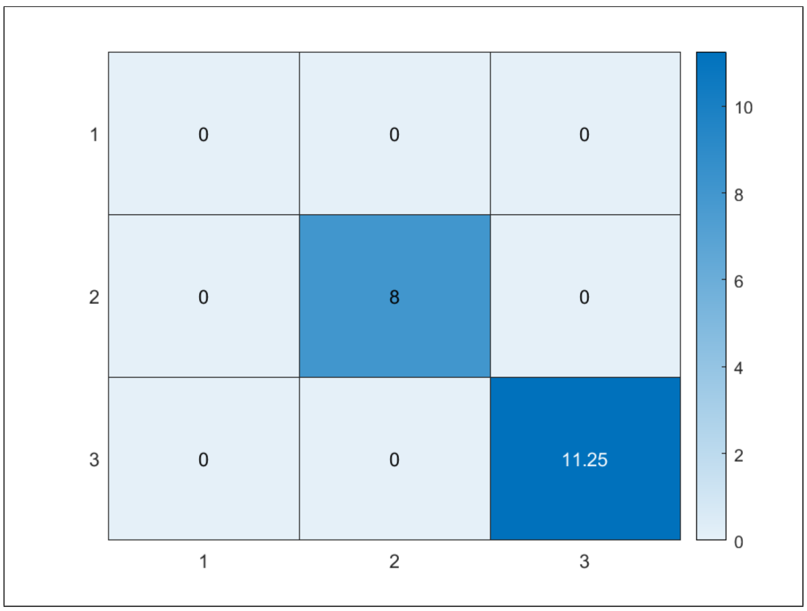
Sprawdzę teraz czy zaimplementowany przeze mnie algorytm rozkładu LDL^* daje ten sam wynik co wbudowana funkcja *ldl*().

```
>> [L4, D4] = wbudowanyLDL([25 2 11], [5i -2i])
L4 =
   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.2000i   0.0000 + 0.1818i   1.0000 + 0.0000i
   0.0000 + 0.0000i   1.0000 + 0.0000i   0.0000 + 0.0000i
D4 =
  25.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  11.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.6364 + 0.0000i

>> [myL4, myD4] = myLDL([25 2 11], [5i -2i])
myL4 =
   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.2000i   1.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 - 0.6667i   1.0000 + 0.0000i
myD4 =
   25.0000         0         0
         0    3.0000         0
         0         0   11.8889
```

Widać, że wyniki te są różne, przy czym wbudowany algorytm ldl() daje macierz L, na której diagonali nie są same jedynki, więc choćby z tego powodu wyniki mogą się tak różnić. Różnice te zostały zwizualizowane na Rysunku 3 i ??:



Rysunek 2: wizualizacja niedokładności obliczeń macierzy L4



Rysunek 3: wizualizacja niedokładności obliczeń macierzy D4

W takim wypadku sprawdzę czy mnożąc otrzymane przeze mnie macierze: LDL^* otrzymam macierz A.

```
>> myL4*myD4*transpose(conj(myL4))
ans =
  25.0000 + 0.0000i   0.0000 - 5.0000i   0.0000 + 0.0000i
   0.0000 + 5.0000i   4.0000 + 0.0000i   0.0000 + 2.0000i
   0.0000 + 0.0000i   0.0000 - 2.0000i  13.2222 + 0.0000i
```

Nie zgadza się tu tylko trzecie miejsce na diagonali, co wynika najprawdopodobniej z niedokładności obliczeń. Sprawdzę jeszcze działanie funkcji $myLDLHsolve$.

```
>> mojX4 = myLDLHsolve([25 2 11], [5i -2i], [50 20 4])
mojX4 =
   2.7863 + 1.4829i
   7.4143 - 3.9315i
   0.8972 + 1.1215i
```

```
>> X4 = wbudowanySolve(myL4, myD4, [50 20 4]);
>> X4== mojX4
ans =
  3×1 logical array
   1
   1
   1
```

Widać, że jeśli porównujemy z rozwiązywaniem równania przy pomocy funkcji *linsolve* dla macierzy, które są wynikiem mojego algorytmu LDL funkcja daje te same wyniki, więc niedokładność obliczeń znowu napotykana jest przy samym algorytmie LDL^*.

4.5 Przykład 5: $A \in \mathbb{R}^{3x3}$

$A = \begin{bmatrix} 0.82 & 0.05 & 0 \\ 0.05 & 0.43 & 2i \\ 0 & 0.1 & 0.12 \end{bmatrix}$, $B = \begin{bmatrix} 50 \\ 20 \\ 4 \end{bmatrix}$

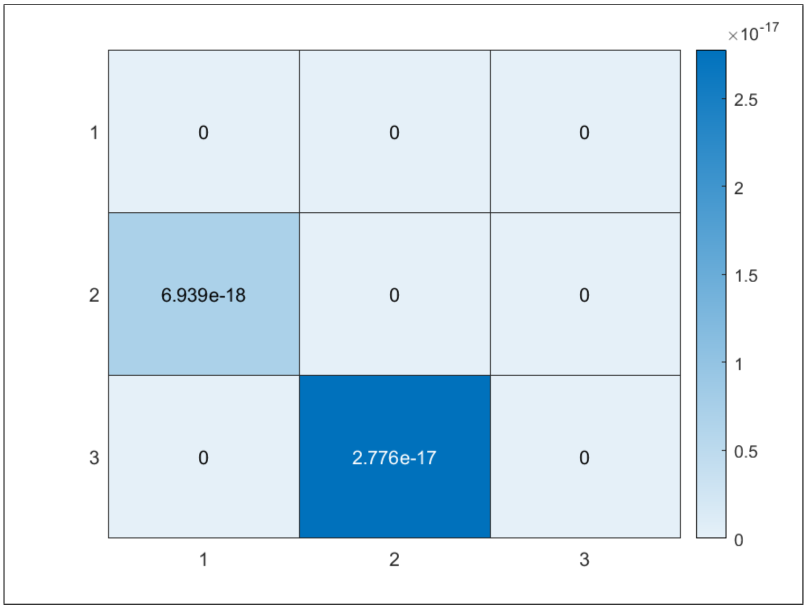
Na początku sprawdzam czy ta macierz jest dodatnio określona.

```
>> czyDodatnioOkreslona([0.82 0.43 0.12], [0.05 0.1])
ans =
  logical
   1
```

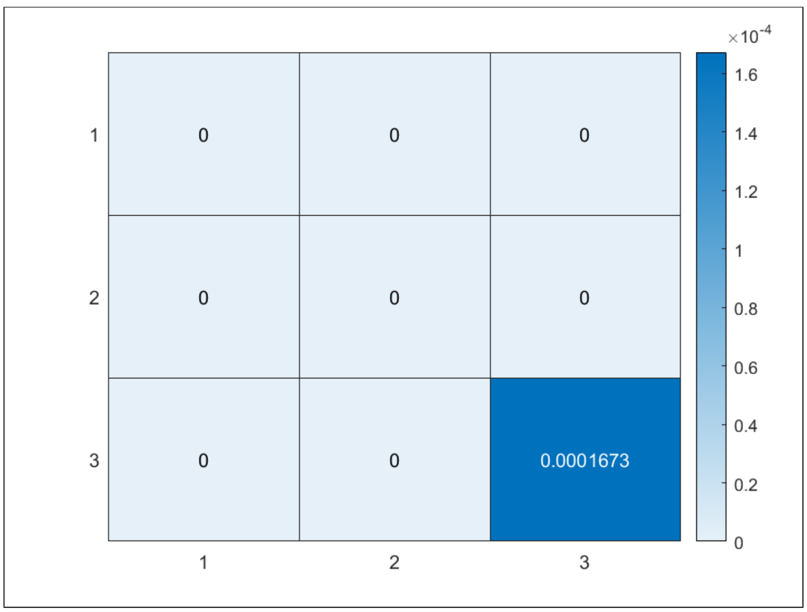
Sprawdzę teraz czy zaimplementowany przeze mnie algorytm rozkładu LDL^* daje ten sam wynik co wbudowana funkcja *ldl*().

```
>> [L5, D5] = wbudowanyLDL([0.82 0.43 0.12], [0.05 0.1])
L5 =
    1.0000         0         0
    0.0610    1.0000         0
         0    0.2342    1.0000
D5 =
    0.8200         0         0
         0    0.4270         0
         0         0    0.0966

>> [myL5, myD5] = myLDL([0.82 0.43 0.12], [0.05 0.1])
myL5 =
    1.0000         0         0
    0.0610    1.0000         0
         0    0.2342    1.0000
myD5 =
    0.8200         0         0
         0    0.4270         0
         0         0    0.0964

>> [L5, D5] == [myL5, myD5]
ans =
  3×6 logical array
   1   1   1   1   1   1
   0   1   1   1   1   1
   1   0   1   1   1   0
```

Widać, że wyniki się niemal pokrywają, przy czym, wartości na dolnej diagonali L są obliczone z pewną niedokładnością, której nie widać na pierwszy rzut oka. Trzecia wartość na diagonali macierzy D również odrobinę się różni. Różnice te zostały zwizualizowane na Rysunku 4 i 5:



Rysunek 4: wizualizacja niedokładności obliczeń macierzy L5



Rysunek 5: wizualizacja niedokładności obliczeń macierzy D5

Sprawdzę jeszcze działanie funkcji $myLDLHsolve$.

```
>> X5 = wbudowanySolve(myL5, myD5, [50 20 4])
X5 =
   58.5591
   39.6308
    0.3081

>> mojX5 = myLDLHsolve([0.82 0.43 0.12], [0.05 0.1], [50 20 4])
mojX5 =
   58.5591
   39.6308
    0.3081

>> X5 == mojX5
```

```
ans =
  3×1 logical array
   1
   1
   0
```

Mimo, że wyniki wyglądają na pierwszy rzut oka na takie same, przy porównaniu ponownie ujawnia się pewna niedokładność w przypadku $X(3)$.

4.6 Przykład 6: $A \in \mathbb{R}^{4x4}$

$$A = \begin{bmatrix} 135 & 8.4 & 0 & 0 \\ 8.4 & 10 & 0.3 & 0 \\ 0 & 0.3 & 0.12 & 1 \\ 0 & 0 & 1 & 34 \end{bmatrix}, B = \begin{bmatrix} 0.34 \\ 76 \\ 9 \\ 30 \end{bmatrix}$$

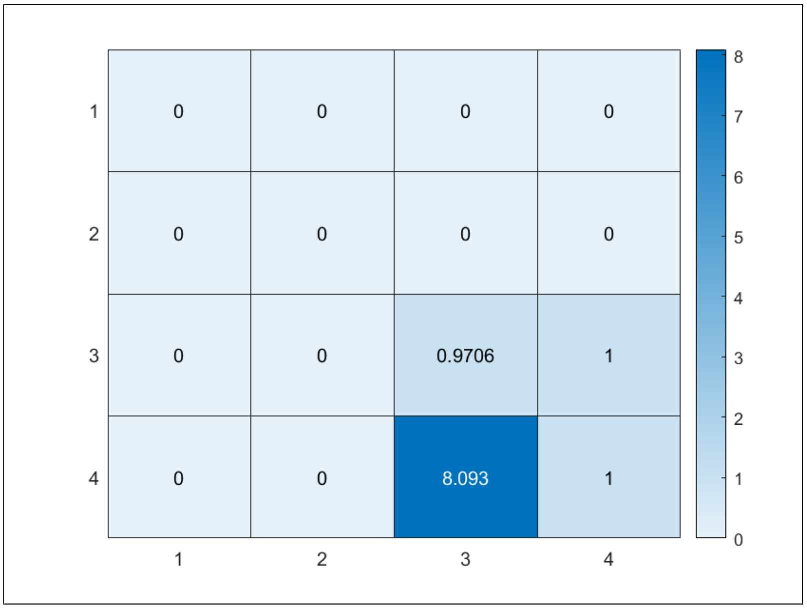
Na początku sprawdzam czy ta macierz jest dodatnio określona.

```
>> czyDodatnioOkreslona([135 10 0.12 34], [8.4 0.3 1])
ans =
  logical
   1
```

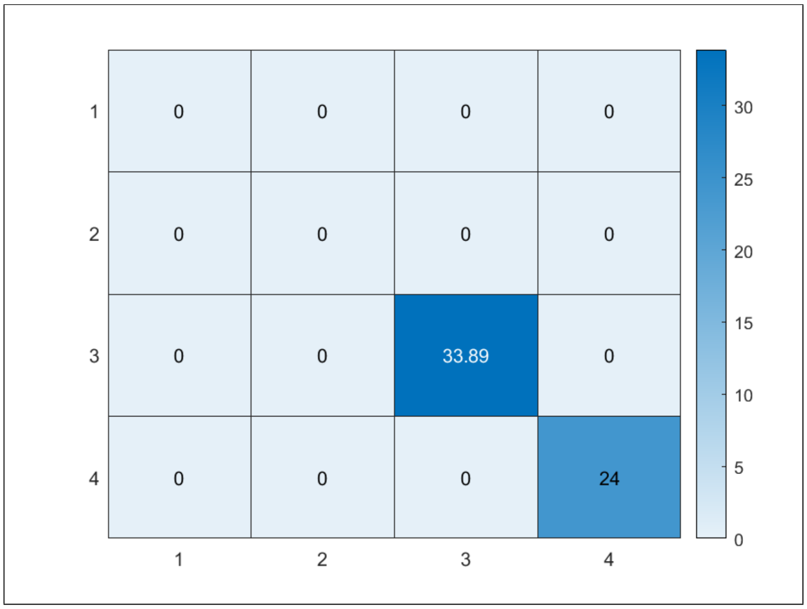
Sprawdzę teraz czy zaimplementowany przeze mnie algorytm rozkładu LDL^* daje ten sam wynik co wbudowana funkcja $ldl()$.

```
>> [L6, D6] = wbudowanyLDL([135 10 0.12 34], [8.4 0.3 1])
L6 =
    1.0000         0         0         0
    0.0622    1.0000         0         0
         0    0.0317    0.0294    1.0000
         0         0    1.0000         0
D6 =
  135.0000         0         0         0
         0    9.4773         0         0
         0         0   34.0000         0
         0         0         0    0.0811

>> [myL6, myD6] = myLDL([135 10 0.12 34], [8.4 0.3 1])
myL6 =
    1.0000         0         0         0
    0.0622    1.0000         0         0
         0    0.0317    1.0000         0
         0         0    9.0926    1.0000
myD6 =
  135.0000         0         0         0
         0    9.4773         0         0
         0         0    0.1100         0
         0         0         0   24.0790

>> [L6, D6] == [myL6, myD6]
ans =
  4×8 logical array
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   0   0   1   1   0   1
   1   1   0   0   1   1   1   0
```

Widać, że wyniki te są różne, przy czym wbudowany algorytm ldl() daje macierz L, na której diagonali nie są same jedynki, więc choćby z tego powodu wyniki mogą się tak różnić. Różnice te zostały zwizualizowane na Rysunku 6 i 7:



Rysunek 6: wizualizacja niedokładności obliczeń macierzy L6



Rysunek 7: wizualizacja niedokładności obliczeń macierzy D6

W takim wypadku sprawdzę czy mnożąc otrzymane przeze mnie macierze: LDL^* otrzymam macierz A.

```
>> myL6*myD6*transpose(conj(myL6))
ans =
  135.0000    8.4000         0         0
    8.4000   10.0000    0.3000         0
         0    0.3000    0.1195    1.0000
         0         0    1.0000   33.1716
```

Widać pewną niedokładność obliczeń, ale w ogólności wartości są przybliżeniami macierzy A, więc można stwierdzić że algorytm działa poprawnie.
Sprawdzę jeszcze działanie funkcji $myLDLHsolve$.

```
>> X6 = wbudowanySolve(myL6, myD6, [0.34 76 9 30])
X6 =
   -0.3559
```

```
    5.7606
   71.2800
   -1.2444

>> mojX6 = myLDLHsolve([135 10 0.12 34], [8.4 0.3 1], [0.34 76 9 30])
mojX6 =
   -0.3559
    5.7606
   71.2800
   -1.2444

>> X6 == mojX6
ans =
  4×1 logical array
   1
   1
   0
   0
```

Mimo, że wyniki wyglądają na pierwszy rzut oka na takie same, przy porównaniu ponownie ujawnia się pewna niedokładność w przypadku $X(3)$ oraz $X(4)$.

4.7 Przykład 7: $A \in \mathbb{C}^{4x4}$

$$A = \begin{bmatrix} 135 & -8.4i & 0 & 0 \\ 8.4i & 10 & 0.3i & 0 \\ 0 & -0.3i & 0.12 & -i \\ 0 & 0 & i & 34 \end{bmatrix}, B = \begin{bmatrix} 0.34 \\ 76 \\ 9 \\ 30 \end{bmatrix}$$

Na początku sprawdzam czy ta macierz jest dodatnio określona.

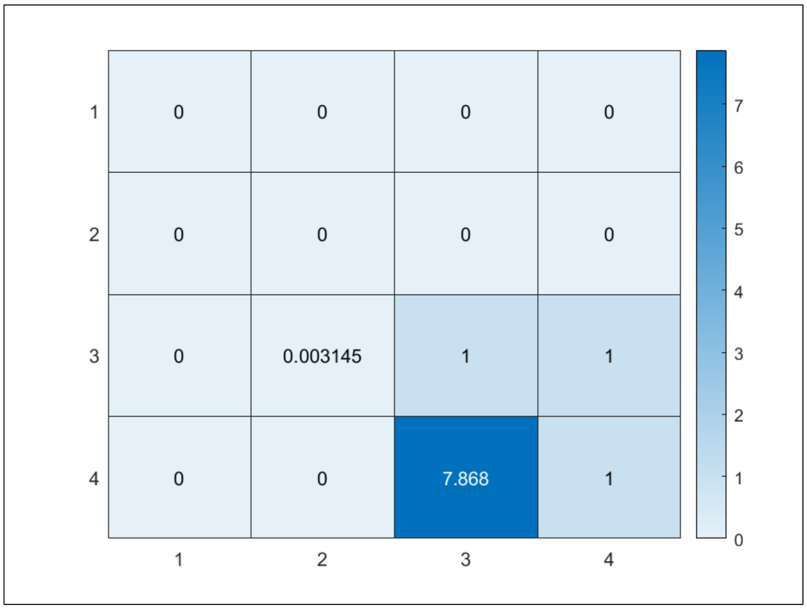
```
>> czyDodatnioOkreslona([135 10 0.12 34], [8.4i -0.3i 1i])
ans =
  logical
   1
```

Sprawdzę teraz czy zaimplementowany przeze mnie algorytm rozkładu LDL^* daje ten sam wynik co wbudowana funkcja $ldl()$.

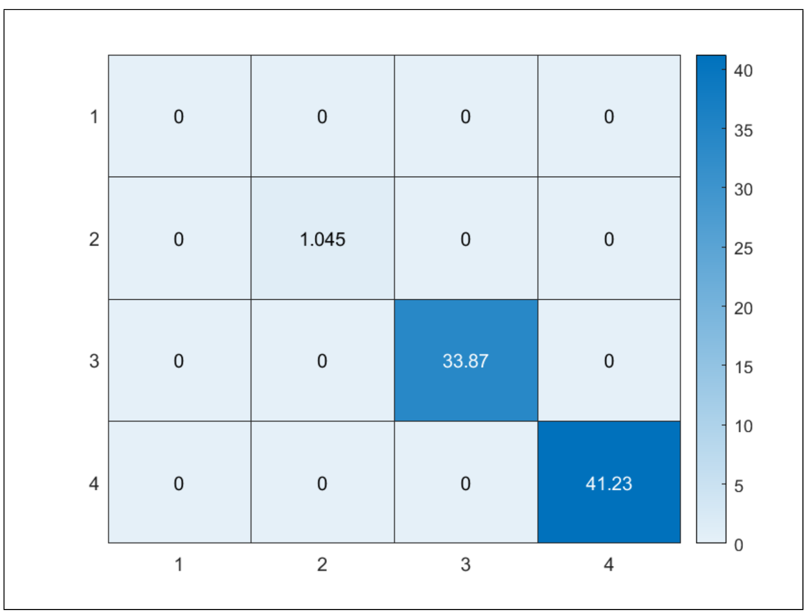
```
>> [L7, D7] = wbudowanyLDL([135 10 0.12 34], [8.4i -0.3i 1i])
L7 =
   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0622i   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 - 0.0317i   0.0000 - 0.0294i   1.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i   0.0000 + 0.0000i
D7 =
   1.0e+02 *

   1.3500 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0948 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.3400 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0008 + 0.0000i

>> [myL7, myD7] = myLDL([135 10 0.12 34],[8.4i -0.3i 1i])
myL7 =
   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0622i   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 - 0.0285i   1.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 7.8047i   1.0000 + 0.0000i
myD7 =
  135.0000         0         0         0
```

```
         0   10.5227         0         0
         0         0    0.1281         0
         0         0         0   41.3096

>> [L7, D7] == [myL7, myD7]
ans =
  4×8 logical array
   1   1   1   1   1   1   1   1
   1   1   1   1   1   0   1   1
   1   0   0   0   1   1   0   1
   1   1   0   0   1   1   1   0
```

Widać, że wyniki te są różne, przy czym wbudowany algorytm ldl() daje macierz L, na której diagonali nie są same jedynki, więc choćby z tego powodu wyniki mogą się tak różnić. Różnice te zostały zwizualizowane na Rysunku 8 i 9:



Rysunek 8: wizualizacja niedokładności obliczeń macierzy L7



Rysunek 9: wizualizacja niedokładności obliczeń macierzy D7

W takim wypadku sprawdzę czy mnożąc otrzymane przeze mnie macierze: LDL^* otrzymam macierz A.

```
>> myL7*myD7*transpose(conj(myL7))
ans =
   1.0e+02 *

   1.3500 + 0.0000i   0.0000 - 0.0840i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0840i   0.1105 + 0.0000i   0.0000 + 0.0030i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 - 0.0030i   0.0014 + 0.0000i   0.0000 - 0.0100i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0100i   0.4911 + 0.0000i
```

Widać pewną niedokładność obliczeń, ale w ogólności wartości są przybliżeniami macierzy A, więc można stwierdzić że algorytm działa poprawnie. Największą różnicę widać w przypadku $A_{4,4}$, który w wyjściowej macierzy wynosi 34, a tu 49.11. Należy jednak wziąć pod uwagę, że błąd mógł się powiększyć przy samym procesie mnożenia macierzy LDL^*.
Sprawdzę jeszcze działanie funkcji $myLDLHsolve$.

```
>> X7 = wbudowanySolve(myL7, myD7, [0.34 76 9 30])
X7 =
   0.1508 + 0.4951i
   7.9573 - 2.3831i
  83.5188 +25.7738i
   1.1356 - 1.7005i

>> mojX7 = myLDLHsolve([135 10 0.12 34], [8.4i -0.3i 1i], [0.34 76 9 30])
mojX7 =
   0.1508 + 0.4951i
   7.9573 - 2.3831i
  83.5188 +25.7738i
   1.1356 - 1.7005i

>> X7 == mojX7
ans =
  4×1 logical array
   0
   0
   0
   1
```

Wynik, mimo pewnych niedokładności w przypadku $X(1:3)$, jest bardzo zbliżony.

4.8 Przykład 8: $A \in \mathbb{R}^{5x5}$

$$A = \begin{bmatrix} 7098 & 38 & 0 & 0 & 0 \\ 38 & 1234 & 83 & 0 & 0 \\ 0 & 83 & 673 & 32 & 0 \\ 0 & 0 & 32 & 784 & 71 \\ 0 & 0 & 0 & 71 & 2034 \end{bmatrix}, B = \begin{bmatrix} 14 \\ 894 \\ 204 \\ 1054 \\ 9821 \end{bmatrix}$$

Na początku sprawdzam czy ta macierz jest dodatnio określona.

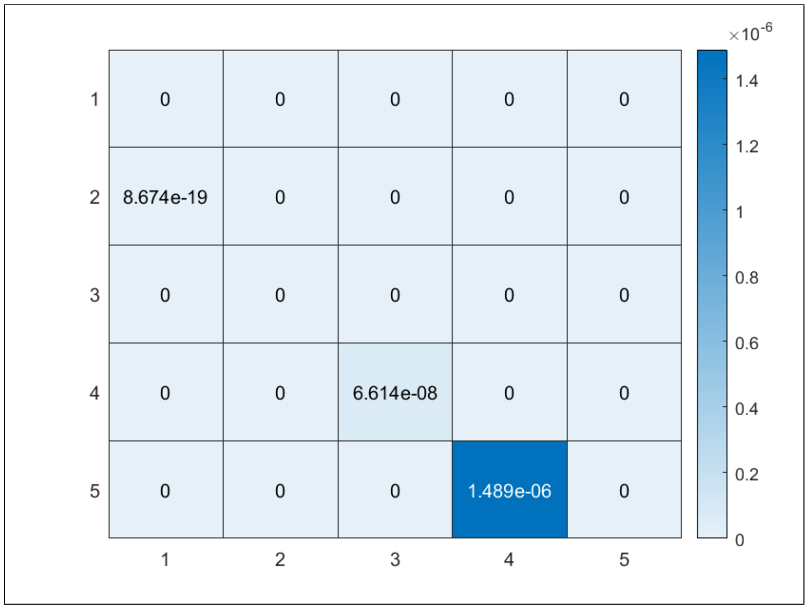
```
>> czyDodatnioOkreslona([7098 1234 673 784 2034], [38 83 32 71])
ans =
  logical
   1
```

Sprawdzę teraz czy zaimplementowany przeze mnie algorytm rozkładu LDL^* daje ten sam wynik co wbudowana funkcja $ldl()$.

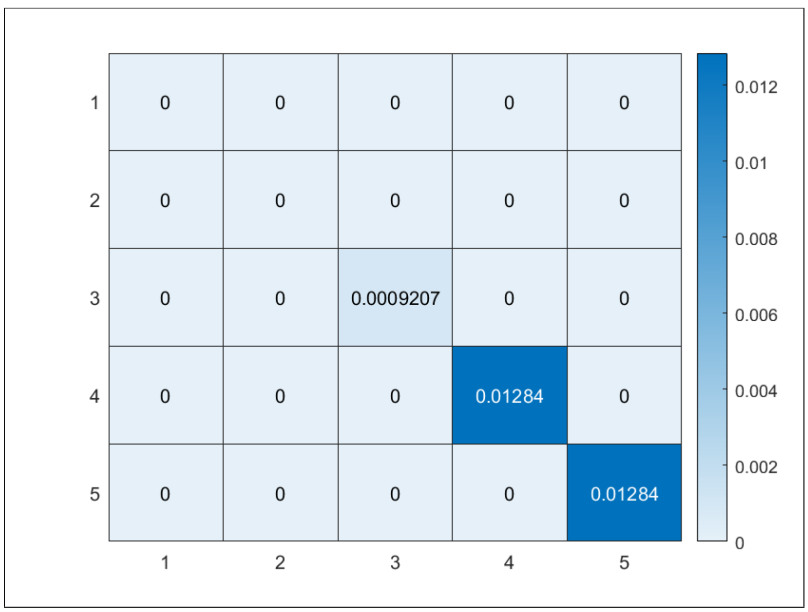
```
>> [L8, D8] = wbudowanyLDL([7098 1234 673 784 2034], [38 83 32 71])
L8 =
    1.0000         0         0         0         0
```

```
    0.0054    1.0000         0         0         0
         0    0.0673    1.0000         0         0
         0         0    0.0479    1.0000         0
         0         0         0    0.0907    1.0000
D8 =
   1.0e+03 *

    7.0980         0         0         0         0
         0    1.2338         0         0         0
         0         0    0.6674         0         0
         0         0         0    0.7825         0
         0         0         0         0    2.0276

>> [myL8, myD8] = myLDL([7098 1234 673 784 2034], [38 83 32 71])
myL8 =
    1.0000         0         0         0         0
    0.0054    1.0000         0         0         0
         0    0.0673    1.0000         0         0
         0         0    0.0479    1.0000         0
         0         0         0    0.0907    1.0000
myD8 =
   1.0e+03 *

    7.0980         0         0         0         0
         0    1.2338         0         0         0
         0         0    0.6674         0         0
         0         0         0    0.7825         0
         0         0         0         0    2.0275

>> [L8, D8] == [myL8, myD8]
ans =
  5×10 logical array
   1   1   1   1   1   1   1   1   1   1
   0   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   0   1   1
   1   1   0   1   1   1   1   1   0   1
   1   1   1   0   1   1   1   1   1   0
```

Mimo, że wyniki wyglądają na pierwszy rzut oka na takie same, przy porównaniu okazuje się, że wyniki nie są identyczne. Widać to także po zwizualizowaniu niedokładności na Rysunkach 10 i 11:



Rysunek 10: wizualizacja niedokładności obliczeń macierzy L8



Rysunek 11: wizualizacja niedokładności obliczeń macierzy D8

Sprawdzę jeszcze działanie funkcji $myLDLHsolve$.

```
>> X8 = wbudowanySolve(myL8, myD8, [14 894 204 1054 9821]);
mojX8 = myLDLHsolve([7098 1234 673 784 2034], [38 83 32 71], [14 894 204 1054 9821])
mojX8 =
   -0.0018
    0.7129
    0.1723
    0.9030
    4.7969

>> X8 == mojX8
ans =
  5×1 logical array
   1
```

```
  1
  1
  1
  1
```

Widać, że jeśli porównujemy z rozwiązywaniem równania przy pomocy funkcji *linsolve* dla macierzy, które są wynikiem mojego algorytmu LDL funkcja daje te same wyniki, więc niedokładność obliczeń znowu napotykana jest przy samym algorytmie LDL^*.

4.9 Przykład 9: $A \in \mathbb{C}^{8x8}$

$$A = \begin{bmatrix} 70 & 12-38i & 0 & 0 & 0 & 0 & 0 & 0 \\ 12+38i & 134 & 83 & 0 & 0 & 0 & 0 & 0 \\ 0 & 83 & 673 & 32i & 0 & 0 & 0 & 0 \\ 0 & 0 & -32i & 84 & 71+2i & 0 & 0 & 0 \\ 0 & 0 & 0 & 71-2i & 204 & 23 & 0 & 0 \\ 0 & 0 & 0 & 0 & 23 & 902 & 9+23i & 0 \\ 0 & 0 & 0 & 0 & 0 & 9-23i & 475 & 32-98i \\ 0 & 0 & 0 & 0 & 0 & 0 & 32+98i & 160 \end{bmatrix}, B = \begin{bmatrix} 12i \\ 32 \\ 198-i \\ 23-62i \\ 178 \\ -37i \\ 2+3i \\ 98 \end{bmatrix}$$

Na początku sprawdzam czy ta macierz jest dodatnio określona.

```
>> czyDodatnioOkreslona([70 134 673 84 204 902 475 160], [12+38i 83 -32i 71-2i 23 9-23i 32+98i])
ans =
  logical
   1
```

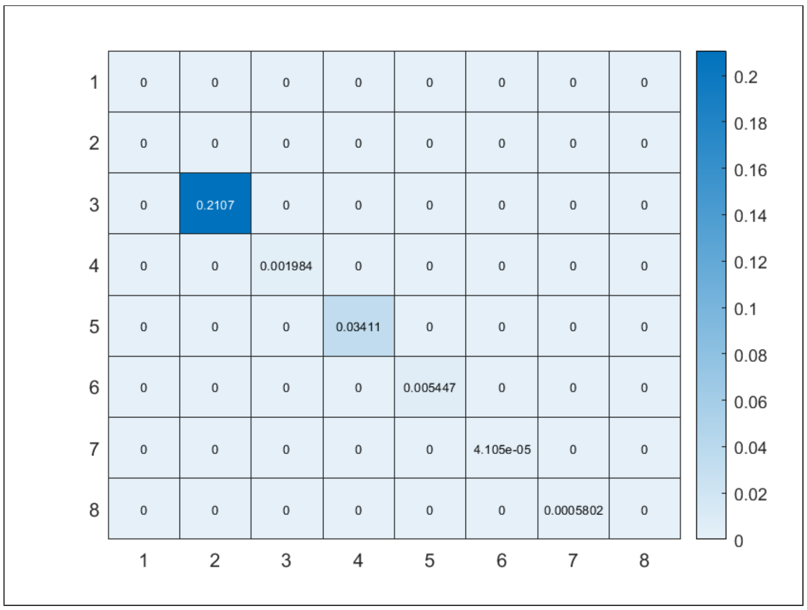
Sprawdzę teraz czy zaimplementowany przeze mnie algorytm rozkładu LDL^* daje ten sam wynik co wbudowana funkcja *ldl*().

```
>> [L9] = wbudowanyLDL([70 134 673 84 204 902 475 160], [12+38i 83 -32i 71-2i 23 9-23i 32+98i])
L9 =
  Columns 1 through 5
   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.1714 + 0.5429i   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.7456 + 0.0000i   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.0524i   1.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.8624 - 0.0243i   1.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.1612 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i

  Columns 6 through 8
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0100 - 0.0256i   1.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0675 + 0.2066i   1.0000 + 0.0000i

>> [myL9] = myLDL([70 134 673 84 204 902 475 160], [12+38i 83 -32i 71-2i 23 9-23i 32+98i])
myL9 =
  Columns 1 through 5
   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.1714 + 0.5429i   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.5401 + 0.0461i   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0005 - 0.0505i   1.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.8283 - 0.0237i   1.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.1570 - 0.0035i
```

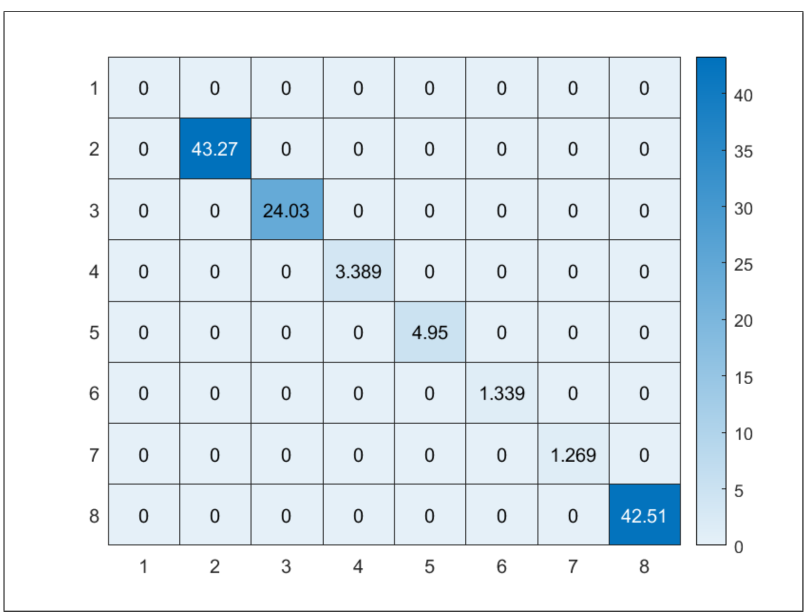
```
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i

  Columns 6 through 8
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0100 - 0.0256i   1.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0675 + 0.2060i   1.0000 + 0.0000i

>> [~, D9] = wbudowanyLDL([70 134 673 84 204 902 475 160], [12+38i 83 -32i 71-2i 23 9-23i 32+98i])
D9 =
   1.0e+02 *
  Columns 1 through 5
   0.7000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   1.1131 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   6.1111 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.8232 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   1.4272 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i

  Columns 6 through 8
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   8.9829 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   4.7432 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   1.3759 + 0.0000i

>> [~, myD9] = myLDL([70 134 673 84 204 902 475 160], [12+38i 83 -32i 71-2i 23 9-23i 32+98i])
myD9 =
   1.0e+02 *
  Columns 1 through 5
   0.7000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   1.5257 - 0.1303i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   6.3420 - 0.0668i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.8571 + 0.0004i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   1.4641 + 0.0330i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i

  Columns 6 through 8
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   8.9697 + 0.0023i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   4.7550 + 0.0046i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   1.7800 - 0.1321i
```

```
>> [L9, D9] == [myL9, myD9]
ans =
  8×16 logical array
   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   0   1   1   1   1   1   1
   1   0   1   1   1   1   1   1   1   1   0   1   1   1   1   1
   1   1   0   1   1   1   1   1   1   1   1   0   1   1   1   1
   1   1   1   0   1   1   1   1   1   1   1   1   0   1   1   1
   1   1   1   1   0   1   1   1   1   1   1   1   1   0   1   1
   1   1   1   1   1   0   1   1   1   1   1   1   1   1   0   1
   1   1   1   1   1   1   0   1   1   1   1   1   1   1   1   0
```

Widać, że wyniki obu funkcji są zbliżone, jednak wstępują niedokładności obliczeń.
Te niedokładności zostały przedstawione graficznie an Rysunkach 12 i 13:



Rysunek 12: wizualizacja niedokładności obliczeń macierzy L9



Rysunek 13: wizualizacja niedokładności obliczeń macierzy D9

Sprawdzę jeszcze działanie funkcji $myLDLHsolve$.

```
>> X9 = wbudowanySolve(myL9, myD9, [12i 32 198-1i 23-62i 178 -37i 2+3i 98])
X9 =
  -0.0217 + 0.2382i
   0.1233 + 0.0010i
   0.2350 + 0.0331i
  -0.6400 - 0.8830i
   1.1043 + 0.2869i
  -0.0249 - 0.0486i
  -0.0374 + 0.1167i
   0.5499 + 0.0359i

>> mojX9 = myLDLHsolve([70 134 673 84 204 902 475 160], [12+38i 83 -32i 71-2i 23 9-23i 32+98i], [12i 32 198
mojX9 =
  -0.0217 + 0.2382i
   0.1233 + 0.0010i
   0.2350 + 0.0331i
  -0.6400 - 0.8830i
   1.1043 + 0.2869i
  -0.0249 - 0.0486i
  -0.0374 + 0.1167i
   0.5499 + 0.0359i

>> X9 == mojX9
ans =
  8×1 logical array
   0
   0
   0
   0
   0
   0
   0
   0
```

Wynik, mimo pewnych niedokładności w przypadku $X(1:3)$, jest bardzo zbliżony.

5 Analiza wyników

Z przeanalizowanych przykładów wynika, że moja funkcja rozwiązująca równanie $AX = B$ przy użyciu rozkładu Cholesky'ego-Banachiewicza (LDL^*) działa poprawnie, przy czym zachodzą pewnie niedokładności obliczeń. Szczególnie widać je w przypadku liczb między -1 a 1.

Na niedokładność wyniku wpływa w głównej mierze sam algorytm rozkładu LDL^*. Widać to gdy rozważam osobno algorytm rozkładu i osobno obliczanie równania, mając już gotowy rozkład. W tym drugim przypadku różnice między moim algorytmem a wbudowaną funkcją do rozwiązywania równań macierzowych $linsolve()$ są mniejsze.

W większości przypadków, dla macierzy D niedokładności są większe dla elementów na diagonali o większym indeksie wierszy, ponieważ do obliczeń wykorzystują one poprzednio obliczone wartości elementów macierzy L. Jeśli te były niedokładne to z każdym kolejnym obliczeniem niedokładność się zwiększa. Można więc powiedzieć, że zarówno w przypadku macierzy L, jak i dla macierzy D niedokładność obliczania wartości macierzy jest większa im większy indeks wiersza i kolumny, w którym znajduje się ta wartość.
Inaczej jest jednak dla Przykładu 9. Wynika to najprawdopodobniej z tego, że niedokładności te składają się ze sobą w trakcie obliczeń i wzajemnie niwelują.