27 grudnia 2022 r.

Jakub Grzywaczewski
Grupa nr 2

Rozwiązywanie układów równania macierzowego $XA = B$ przy użyciu rozkładu Cholesky'ego-Banachiewicza w liczbach zespolonych

Projekt nr 19

1 Opis metody

1.1 Układ równań macierzowych

Praca skupią się na znalezieniu efektywnego sposobu rozwiązywania układów równań macierzowych używając rozkładu Cholesky'ego-Banachiewicza. Układ równań macierzowych jest to uogólnienie macierzowej metody rozwiązywania równań typy:

$$XA = b$$

,gdzie A jest macierzą współczynników układu, B wektorem wyrazów wolnych.

Przykładowo układ równań:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_1 + \cdots + a_{1,n}x_1 = b_1 \\ a_{2,1}x_2 + a_{2,2}x_2 + \cdots + a_{2,n}x_2 = b_2 \\ \vdots \\ a_{m,1}x_m + a_{m,2}x_m + \cdots + a_{m,n}x_m = b_m \end{cases}$$

możemy zapisać jako równanie macierzowe:

$$\begin{bmatrix} x_1 & x_2 & \cdots & x_m \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{1,2} & \cdots & a_{1,n} \\ \vdots & & & \\ a_{m,1} & a_{1,2} & \cdots & a_{1,n} \end{bmatrix} = \begin{bmatrix} b_1 & b_2 & \cdots & b_m \end{bmatrix}$$

Uogólnienie pozwala nam na obliczanie wartości x dla kilku wektorów b bez wykonywania dużej ilości nadmiernych obliczeń, poprzez zastąpienie wektora b macierzą B, która jest pionowo nałożonymi na siebie wektorami b dla każdego przypadku.

Zatem kiedy piszemy

$$XA = B$$

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ \vdots & & & \\ x_{k,1} & x_{k,2} & \cdots & x_{k,m} \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{1,2} & \cdots & a_{1,n} \\ \vdots & & & \\ a_{m,1} & a_{1,2} & \cdots & a_{1,n} \end{bmatrix} = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,m} \\ \vdots & & & \\ b_{k,1} & b_{k,2} & \cdots & b_{k,m} \end{bmatrix}$$

,pokrywamy jednocześnie przypadek B będącego wektorem jak i macierzą.

1.2 Układy proste do rozwiązania

W rozkładach macierzy na iloczyny często chcemy, aby owe macierze składowe były w formie macierz dolno (bądz górno) trójkątnych. Dzieje się to z powodów, iż macierz trójkątne (oznaczone tutaj L) mają istotne własności m.in.

1. Wyznacznik to iloczyn elementów na diagonali $\det L = \prod_{k=1}^{n} l_{kk}$
2. Układy równań $XL = B$ rozwiązywane w $O(n^2)$. Intuicyjny algorytm.
3. Łatwe obliczenie macierzy odwrotnej.

1.3 Rozkład Cholesky'ego-Banachiewicza

Rozkład Cholesky'ego-Banachiewicza jest jednym z wielu stosowanych w praktyce rozkładów macierzy na czynniki. Rozkład ten jest stosowany praktycznie wyłącznie dla macierz symetrycznych dodatnio określonych, ponieważ wtedy jest on jednoznacznie określony. Metoda polega na rozbiciu macierz wejściowej A na iloczyn macierzy dolno trójkątnej L oraz jej sprzężenia L^*.

$$A = LL^*$$

.

Ze względu na fakt, iż w praktyce często spotykamy się z macierzami symetrycznymi (bądź Hermanowskimi) oraz dodatnio określonymi rozkład ten jest niezwykle ważny w świecie matematyki obliczeniowej.

Implementacja algorytmu wyznaczania rozkładu Cholesky'ego-Banachiewicza w pseudokodzie:

Algorithm 1 Rozkład Cholesky'ego-Banachiewicza

Require: A - macierz nxn symetryczna dodatnio określona
  for $k = 1, 2, \cdots, n$ **do**
    $l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2}$ ▷ Element diagonali
    for $i = k+1, k+2, \cdots, n$ **do** ▷ Pozostałe elementy w kolumnie
      $l_{ik} = \left(a_{ik} - \sum_{j=1}^{k-1} l_{ij} l_{kj}\right) / l_{kk}$
    end for
  end for

Wykorzystując wektoryzację oraz syntaks Matlab'a jesteśmy w stanie pozbyć się sum oraz wyeliminować wewnętrzną pętlę.

Algorithm 2 Rozkład Cholesky'ego-Banachiewicza Zwektoryzowany

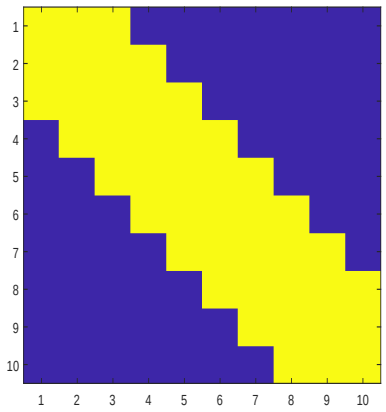
Require: A - macierz nxn symetryczna dodatnio określona
Ensure: $L \leftarrow zeros(n)$
  for $k = 1, 2, \cdots, n$ **do**
    $r \leftarrow L(k, 1 : (k-1))$
    $L(k,k) \leftarrow \sqrt{A(k,k) - r * (r.')}$ ▷ Element diagonali
    $sektor \leftarrow L((k+1) : n, 1 : (k-1))$
    $L((k+1) : n, k) \leftarrow (A((k+1) : n, k) - sektor * (r.'))/L(k,k)$ ▷ Pozostałe elementy w kolumnie
  end for

1.4 Specjalny przypadek

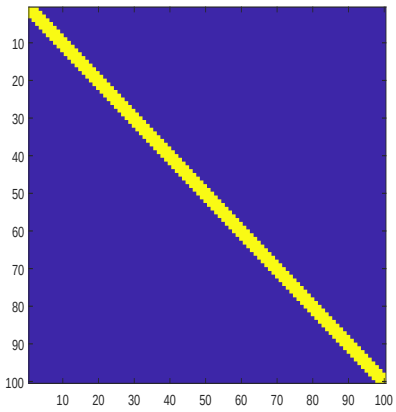
W owej pracy skupimy się jednak na specjalnym przypadkiem rozkładu Cholesky'ego-Banachewicza. Dokładniej spojrzymy jak można wykorzystać owy rozkład do rozwiązywania układów równań macierzowych $XA = B$, kiedy macierz A jest macierzą hermitowską dodatnio określoną oraz jest ona macierzą pięciodiagonalną.

Ograniczając możliwości form wejściowych macierzy, jesteśmy w stanie znacząco przyśpieszyć wykonywanie rozkładu Cholesky'ego-Banachiewicza wykorzystując fakt, iż przy wystarczająco dużych wymiarach macierze 5 diagonalne składają się w większości z zer.

Przedstawiam przykłady macierzy 5-diagonalnych. Kolor fioletowy indykuje zerowe miejsca w macierzy.

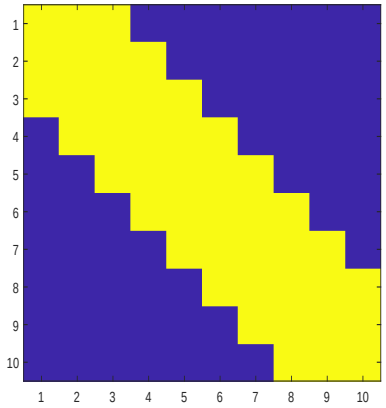


Rysunek 1: Macierz 10x10

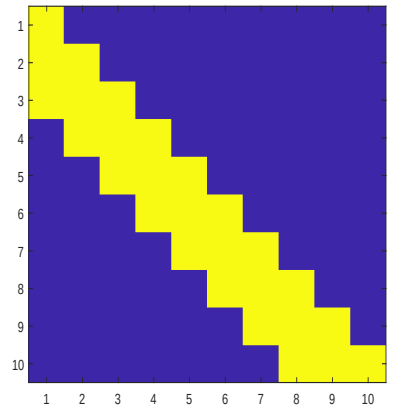


Rysunek 2: Macierz 100x100

Widzimy zatem dlaczego przy większych macierzach możliwość operacji tylko na niezerowych elementach ma sens optymalizacyjny.



Rysunek 3: Macierz 10x10



Rysunek 4: Macierz powstała z macierzy 10x10 poprzez rozkład

Rozkład Cholesy'ego macierzy m-diagonalnej tworzy macierz L, która jest macierzą "pół m-diagonalną"to znaczy macierzą, która jest dolno-, bądź górno- trójkątna, posiada główną diagonale oraz $\frac{\lfloor m \rfloor}{2}$ diagonali poniżej (lub powyżej) głównej.

Pozwala nam to przy rozwiązywaniu równań macierzowych $XL = B$ wykonać potrzebne obliczenia w czasie $O(m * n)$, jeżeli m jest stała dla danej rodziny macierzy daje nam to praktyczne rozwiązanie w czasie linowym.

2 Opis funkcjonalności implementacji metody w Matlabie.

Program obliczeniowy składa się z 10 funkcji oraz skryptów.

2.1 Główne funkcje / cholDecomp oraz cholDecompDiag

Owe funkcje są implementacją algorytmu rozkładu Cholesky'ego-Banachiewicza. Pierwsza z nich cholDecomp() jest funkcją, ogólną działającą, dla każdej macierzy symetrycznej dodatnio określonej. Natomiast druga z nich cholDecompDiag() jest funkcją napisaną specjalnie do rozkładu macierz m-diagonalnych. Funkcja ta wykorzystuje dodatkowe optymalizacje, aby znacząco przyśpieszyć wykonywanie obliczeń dla tego rodzaju macierzy. Szczególnie widoczna jest różnica, przy dużych i tym samych rzadkich macierzach.

Ogólna: cholDecomp

```
1  function [L] = cholDecomp(A)
2  % Funckja obliczajaca rozklad Cholesky'ego-Banachewicza macierzy (A).
3  % Macierz A musi byc macierza kwadratowa pozytywnie polokreslona
4  % w przypadku podania macierzy o innej specyfikacji otrzymamy blad
5  % Funkcja zwraca macierz L dolno-trojkatna o tym samych wymiarach …
       jak A
6  % spelniajaca rownanie A = LL*. (rozklad Cholesky'ego)
7
8  if ¬ismatrix(A)
9      error("Nie podano arugmentu A, badz argument nie jest macierza")
10 end
11
12 if ¬issymmetric(A)
13     error("Macierz A nie jest symetryczna")
14 end
15
16 % Sprawdzamy czy macierz A jest dodatnio polokreslona z pewna …
       niepewnoscia
17 eigenVals = eig(A);
18 tol = length(eigenVals) * eps(max(eigenVals));
19 if ¬all(eigenVals > -tol)
20     error("Macierz A nie jest pozytywnie polokreslona")
21 end
22
23 n = length(A);
24 L = zeros(n);
25
26 % Specjalny przypadek pierwszej kolumny
27 L(1,1) = sqrt(A(1,1));
28 L(2:n, 1) = A(2:n, 1) ./ L(1,1);
29
30 % Pozostale kolumny
31 for k = 2:n
```

```
32      r = L(k, 1:(k-1)); % Wektor poziomy
33      L(k, k) = sqrt(A(k,k) - r * (r'));
34      sektor = L( (k+1):n, 1:(k-1) );
35      L( (k+1):n, k ) = (A( (k+1):n, k ) - sektor * (r')) / L(k,k);
36  end
```

Zoptymalizowana pod symetryczne m-diagonalne: cholDecompDiag

```
1   function [L] = cholDecompDiag(A, m)
2   % Funckja obliczajaca rozklad Cholesky'ego-Banachewicza macierzy …
        m-diagonalnej (A).
3   % Macierz A musi byc macierza kwadratowa pozytywnie polokreslona
4   % w przypadku podania macierzy o innej specyfikacji otrzymamy blad
5   % Funkcja zwraca macierz L dolno-trojkatna o tym samych wymiarach …
        jak A
6   % spelniajaca rownanie A = LL*. (rozklad Cholesky'ego)
7
8   if ¬ismatrix(A)
9       error("Nie podano arugmentu A, badz argument nie jest macierza")
10  end
11
12  if ¬issymmetric(A)
13      error("Macierz A nie jest symetyczna")
14  end
15
16  % Sprawdzamy czy macierz A jest dodatnio polokreslona z pewna …
        niepewnoscia
17  eigenVals = eig(A);
18  tol = length(eigenVals) * eps(max(eigenVals));
19  if ¬all(eigenVals > -tol)
20      error("Macierz A nie jest pozytywnie polokreslona")
21  end
22
23  n = length(A);
24  L = zeros(n);
25
26  % Liczba niezerowych elementow w dol pierwszej kolumnie od diagonali.
27  m_k = idivide(m, int32(2));
28
29  % Specjalny przypadek pierwszej kolumny
30  L(1,1) = sqrt(A(1,1));
31  L(2:(m_k+1), 1) = A(2:(m_k+1), 1) ./ L(1,1);
32
33  % Pozostale kolumny
34  for k = 2:n
35      % Tworzymy wektor indeksow niezerowych elementow dla wierza
36      slWinX = max(1, k-m_k):(k-1);
37
38      r = L(k, slWinX);
39
40      % Diagonalny element
41      L(k, k) = sqrt(A(k,k) - r * (r'));
42
43      % Tworzymy wektor indeksow niezerowych elementow dla kolumny
```

```
44      slWinY = (k+1):min(k+m_k, n);
45
46      % Reszta kolumny
47      sektor = L( slWinY, slWinX );
48      L( slWinY, k ) = (A( slWinY, k ) - sektor * (r')) / L(k,k);
49  end
```

2.2 Funkcje solve*

Funkcje solve* rozwiązują układ równań macierzowy $XA = B$ dla danej macierzy A oraz B.

1. solveLower - Rozwiązuje równanie macierzowe $XL = B$, z macierzą L dolnotrójkątną.
2. solveLowerDiag - Zoptymalizowana wersjca solveLower pod macierze dolnotrójkąte z macierzy m-diagonalnych.
3. solveUpper - Rozwiązuje równanie macierzowe $XL = B$, z macierzą L górnotrójkątną.
4. solveUpperDiag - Zoptymalizowana wersjca solveUpper pod macierze górnotrójkąte z macierzy m-diagonalnych.
5. solveByChol - Połączenie solveUpperDiag, solverLowerDiag oraz cholDecompDiag w celu rozwiązywania równania $XA = B$ dla macierz A będącej symetryczną dodatio określoną m-diagonalną.

```
1  function [X] = solveLower(L, B)
2  % Funkcja rozwizzujzca uklad rowan macierzowych XL = B,
3  % gdzie macierz L jest macierza dolno-trojkatna (odwracalna).
4  % Argument B moze byc albo macierza, badz wektorem poziomym
5
6  if ¬ismatrix(L)
7      error("Argument L nie podany, badz nie jest macierza");
8  end
9
10 if ¬ismatrix(B)
11     error("Argument B nie podany, badz nie jest macierza");
12 end
13
14 if ¬istril(L)
15     error("Macierz L nie jest macierza dolno-trojkatna");
16 end
17
18
19 [nRowsB, nColsB] = size(B);
20 [nRowsL, nColsL] = size(L);
21
```

```
22 if nColsB ≠ nColsL
23     error("Macierze nie sa odpowiednych wymiarow")
24 end
25
26 % Dla ulatwienie notacji
27 n = nRowsL; % Liczba wierszy
28 k = nRowsB; % Liczba kolumn
29 X = zeros(k, n);
30
31 X(1:k, n) = B(1:k, n) ./ L(n,n);
32 for i = (n-1):-1:1
33     rest = X(1:k, (i+1):n) * L((i+1):n, i);
34     X(1:k, i) = (B(1:k, i) - rest) ./ L(i,i);
35 end
```

```
1  function [X] = solveUpper(U, B)
2  % Funkcja rozwiazujaca uklad rowan macierzowych XU = B,
3  % gdzie macierz U jest macierza gorno-trojkatna (odwracalna).
4  % Argument B moze byc albo macierza, badz wektorem poziomym
5
6  if ¬ismatrix(U)
7      error("Argument U nie podany, badz nie jest macierza");
8  end
9
10 if ¬ismatrix(B)
11     error("Argument B nie podany, badz nie jest macierza");
12 end
13
14 if ¬istriu(U)
15     error("Macierz U nie jest macierza gorno-trojkatna");
16 end
17
18 [nRowsB, nColsB] = size(B);
19 [nRowsU, nColsU] = size(U);
20
21 if nColsB ≠ nColsU
22     error("Macierze nie sa odpowiednych wymiarow")
23 end
24
25 % Dla ulatwienie notacji
26 n = nRowsU; % Liczba wierszy
27 k = nRowsB; % Liczba kolumn
28 X = zeros(k, n);
29
30 X(1:k, 1) = B(1:k, 1) ./ U(1,1);
31 for i = 2:n % Index kolumny, ktora obliczamy
32     rest = X(1:k, 1:(i-1)) * U(1:(i-1), i);
33     X(1:k, i) = (B(1:k, i) - rest) ./ U(i,i);
34 end
```

```
1  function [X] = solveByDiagChol(A, B, m)
```

```
% Funckja przyjmuje macierz A, ktora jest macierza symetryczna ... dodatnio
% okreslona m-diagonalna oraz macierzy B i uzywajac rozkladu
% Cholesky'ego-Bancheiwicza zwraca rozwiazanie rownania macierzowego
% XA = B.

% if ¬ismatrix(A)
%     error("Nie podano arugmentu A, badz argument nie jest ... macierza")
% end
%
% if ¬issymmetric(A)
%     error("Macierz A nie jest symetryczna")
% end
%
% % Sprawdzamy czy macierz A jest dodatnio polokreslona z pewna ... niepewnoscia
% eigenVals = eig(A);
% tol = length(eigenVals) * eps(max(eigenVals));
% if ¬all(eigenVals > -tol)
%     error("Macierz A nie jest pozytywnie polokreslona")
% end

L = cholDecompDiag(A, m);
Y = solveUpperDiag(L', B, m);
X = solveLowerDiag(L, Y, m);

end
```

2.3 Funkcje generujące macierze

Do obliczeń potrzebne nam są macierze z specyficznej rodziny. Mianowicie wspomniane wcześniej macierze symetryczne dodatnio określone m-diagonalne. W celu generowania losowych macierz w tej formie powstały 3 funkcje:

1. randKdiag - Funkcja generuje losową macierz symetryczną dodatnio określona z elementami w liczbach rzeczywistych używając funkcji rand() jako podstawy.
2. randKdiagC - To samo co w funkcji randKdiag, lecz macierz posiada element z liczy zespolonych
3. onesKdiag - Generuje macierz m-diagonalna wypełnioną na tych diagonalach jedynkami (do wizualizacji) [nie jest ona jednak pozytywnie dodatnio określona]

```
function [A] = randKdiag(n, m)
% Funkcja tworzy losowa dodatnio okreslona macierz m-diagonalna o
% wymiarach nxn z elementami w liczbach rzeczywistych.

k = idivide(m, int32(2)) + 1;
if n < k
```

```
7       error("Nie mozna utworzyc macierzy m-diagonalej o takich …
            wymiarach");
8   end
9
10  M = rand(n);
11  % Upewniany sie, ze macierz bedzie macierza spanujaca cala …
        przestrzen.
12  while det(M) == 0
13      M = rand(n);
14  end
15
16  Z = zeros(n);
17
18  for i = 0:(k - 1)
19      Z = Z + diag(diag(M, i), i);
20  end
21
22  A = 0.5 * (Z + Z') + eye(n);
23  end
```

```
1   function [Z] = randKdiagC(n, m)
2   % Funkcja tworzy losowa dodatnio okreslona macierz m-diagonalna o
3   % wymiarach nxn z elementami w liczbach zespolonych.
4
5   k = idivide(m, int32(2)) + 1;
6   if n < k
7       error("Nie mozna utworzyc macierzy m-diagonalej o takich …
            wymiarach");
8   end
9
10  M = complex(rand(n, n), rand(n,n));
11  % Upewniany sie, ze macierz bedzie macierza spanujaca cala …
        przestrzen.
12  while det(M) == 0
13      M = complex(rand(n, n), rand(n,n));
14  end
15
16  A = M*(M') + n*eye(n);
17  Z = zeros(n);
18
19  Z = Z + diag(diag(A, 0), 0);
20  for i = (-k+1):-1
21      Z = Z + diag(diag(A, i), i);
22  end
23  for i = 1:(k-1)
24      Z = Z + diag(diag(A, i)', i);
25  end
26  end
```

```
1    function [Z] = onesKdiag(n, m)
2   % Funkcja tworzy macierz m-diagonalna wypelniona jedynkami.
```

```
3  % wymiarach nxn z elementami w liczbach rzeczywistych.
4
5  k = idivide(m, int32(2)) + 1;
6  if n < k
7      error("Nie mozna utworzyc macierzy m-diagonalej o takich ...
           wymiarach");
8  end
9
10 M = ones(n);
11 Z = zeros(n);
12
13 for i = (-k + 1):(k - 1)
14     Z = Z + diag(diag(M, i), i);
15 end
16
17 end
```

3 Przykłady obliczeniowe

Jako przykłady użycia i zastosowania tej specyficznej metody rozkładu Cholesky'ego-Banachewicza postanowiłem sprawdzić/zbadać porównać następujące aspekty:

- Obliczania rozkładu dla macierzy m-diagonalnych $\in \mathbb{R}^{n\times n}$.
- Obliczania rozkładu dla macierzy m-diagonalnych $\in \mathbb{C}^{n\times n}$.
- Rozwiązywanie układów równań wektor-macierz $\in \mathbb{R}^{n\times n}$.
- Rozwiązywanie układów równań wektor-macierz $\in \mathbb{C}^{n\times n}$.
- Wyznaczenie kiedy solveByChol() się opłaca?
- Wyliczanie macierzy odwrotnej.

Do wyliczania czasów obliczeń została funkcji w matlab'ie nazwana "timeit". A w wyznaczania błędów rachunkowych wyliczany został błąd względnym z użyciem podstawowej normy macierzowej i porównywane były zaimplementowane w tej pracy metody z wbudowanymi z Matlab'a.

Oznaczmy:

- A_{bld} - macierz(bądź wektor) powstała przez użycie metody wbudowanej.
- A_{imp} - macierz(bądź wektor) powstała przez użycie metody zaimplementowanej.

Wtedy błąd względny definiujemy jako:

$$error = \frac{\|A_{bld} - A_{imp}\|}{\|A_{bld}\|}$$

3.1 Obliczania rozkładu dla macierzy m-diagonalnych $\in \mathbb{R}^{n\times n}$.

Skrypty: cholSpeed.m | plotCreator.m

W tym przykładzie tworzyłem losowe macierze m-diagonalne $\in \mathbb{R}^{n\times n}$ używając funkcji randKdiag(), a następnie rozkładałem je używając funkcji:

- Wbudowanej chol()
- Zaimplementowanej cholDecompDiag()
- Zaimplementowanej cholDecomp()

Błędy obliczałem względem implementacji chol() dla macierzy zwracanych przez cholDecompDiag() oraz cholDecomp().

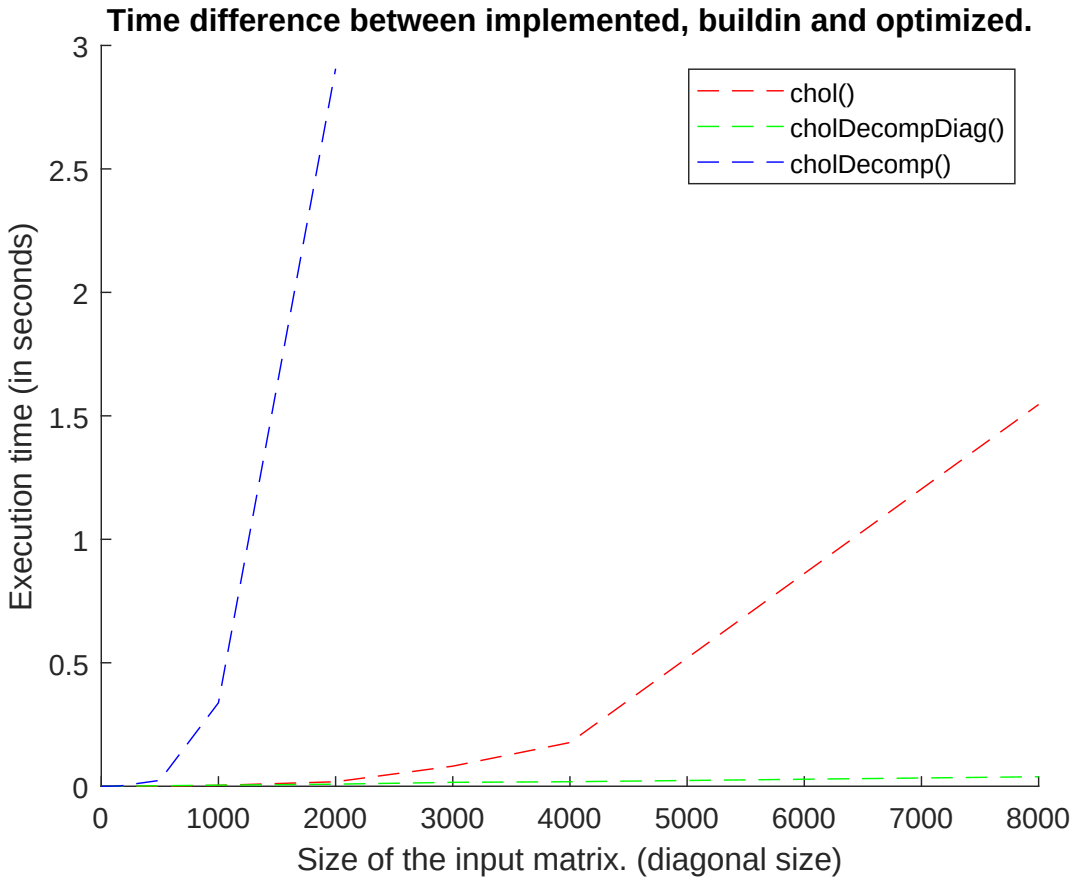
Wyniki prezentują się następująco:

Tabela 1: Względne błędy funkcji (norma macierzowa) w zależności od wielkości macierzy ($\times 10^{-15}$)

n	cholDecompDiag()	cholDecomp()
5	0.0103	0.0103
10	0.0528	0.0528
20	0.0171	0.0171
40	0.1344	0.1344
80	0.1437	0.1437
160	0.1301	0.1301
500	0.1694	0.1694
1000	0.1645	0.1645
2000	0.1592	0.1592
3000	0.1672	0.1672
4000	0.1487	0.1487
8000	0.1994	0.1994

Tabela 2: Czas wykonania funkcji w sekundach w zależności od wielkości macierzy

n	chol()	cholDecompDiag()	cholDecomp()
10	0.0000	0.0000	0.0000
20	0.0000	0.0001	0.0001
40	0.0000	0.0001	0.0002
80	0.0000	0.0003	0.0004
160	0.0001	0.0005	0.0012
500	0.0006	0.0019	0.0238
1000	0.0036	0.0044	0.3381
2000	0.0182	0.0084	2.9059
3000	0.0814	0.0158	
4000	0.1770	0.0183	
8000	1.5460	0.0385	



Rysunek 5: Wykres porównujący czasy wykonania

3.2 Obliczania rozkładu dla macierzy m-diagonalnych $\in \mathbb{C}^{n\times n}$.

Skrypty: cholSpeed.m | plotCreator.m

W tym przykładzie tworzyłem losowe macierze m-diagonalne $\in \mathbb{R}^{n\times n}$ używając funkcji randKdiagC(), a następnie rozkładałem je używając funkcji:

- Wbudowanej chol()
- Zaimplementowanej cholDecompDiag()
- Zaimplementowanej cholDecomp()

Błędy obliczałem względem implementacji chol() dla macierzy zwracanych przez cholDecompDiag() oraz cholDecomp().

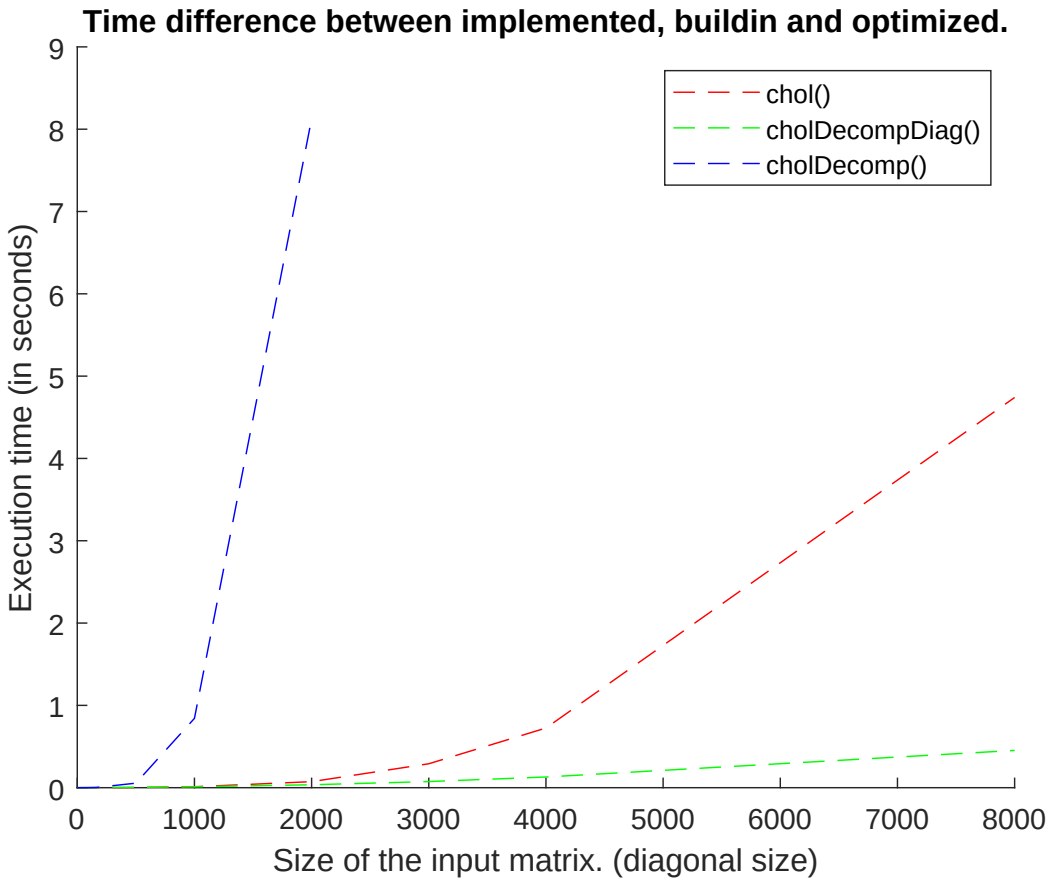
Wyniki prezentują się następująco:

Tabela 3: Względne błędy funkcji (norma macierzowa) w zależności od wielkości macierzy ($\times 10^{-15}$)

n	cholDecompDiag()	cholDecomp()
5	0	0
10	0.0568	0.0568
20	0.1133	0.1133
40	0.1297	0.1297
80	0.1356	0.1356
160	0.1327	0.1327
500	0.1232	0.1303
1000	0.1778	0.1964
2000	0.1448	0.1424
3000	0.1961	0.2019
4000	0.1668	0.1736
8000	0.1529	0.1537

Tabela 4: Czas wykonania funkcji w sekundach w zależności od wielkości macierzy

n	chol()	cholDecompDiag()	cholDecomp()
5	0.0000	0.0000	0.0000
10	0.0000	0.0001	0.0001
20	0.0000	0.0001	0.0001
40	0.0000	0.0002	0.0002
80	0.0001	0.0004	0.0006
160	0.0001	0.0012	0.0025
500	0.0020	0.0041	0.0549
1000	0.0115	0.0114	0.8421
2000	0.0732	0.0356	8.1259
3000	0.2907	0.0743	
4000	0.7249	0.1298	
8000	4.7405	0.4529	



Rysunek 6: Wykres porównujący czasy wykonania

3.3 Rozwiązywanie układów równań wektor-macierz $\in \mathbb{R}^{n\times n}$.

Skrypty: cholSolveSpeed.m | plotCreatorSolve.m

W tym przykładzie tworzyłem losowo macierz m-diagonalną $\in \mathbb{R}^{n\times n}$ używając funkcji randKdiagC(), a dla wylosowanego wektora X wyliczałem B i porównywałem X z wektorem zwracanym przez:

- (linsolve(A.', B.')).'
- B / A
- B * inv(A)
- solveByChol(A, B, 5)

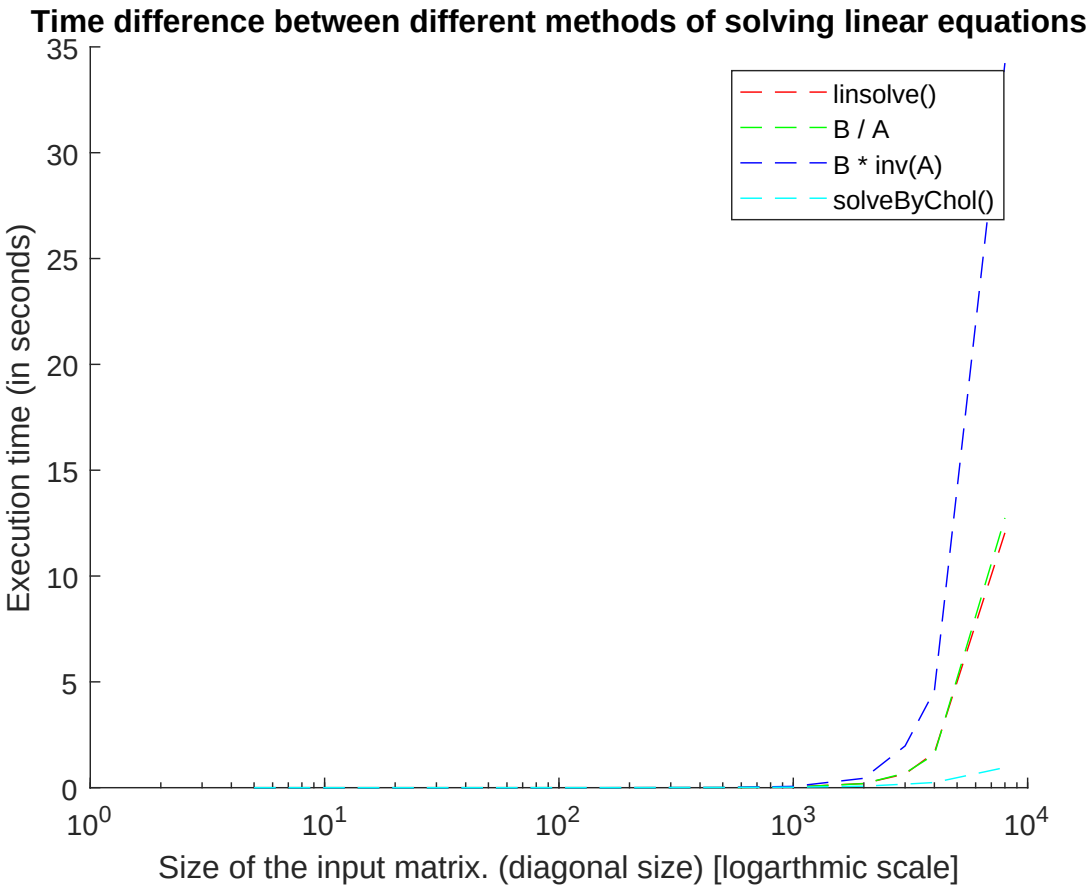
Wyniki prezentują się następująco:

Tabela 5: Względne błędy funkcji (norma wektorowa) w zależności od wielkości macierzy ($\times 10^{-15}$)

n	linsolve()	B / A	B * inv(A)	solveByChol()
5	0.2804	0.3625	0.3134	0.3369
10	0.2781	0.2086	0.3669	0.1820
20	0.1673	0.3749	0.1916	0.3281
40	0.2640	0.2767	0.4103	0.2990
80	0.2244	0.2971	0.2904	0.2632
160	0.2208	0.2327	0.5409	0.2477
500	0.2173	0.2391	0.4446	0.2309
1000	0.2352	0.2719	0.5057	0.2565
2000	0.2199	0.2659	0.5152	0.2583
3000	0.2158	0.2604	0.5205	0.2472
4000	0.2175	0.2578	0.5201	0.2473
8000	0.2196	0.2542	0.5438	0.2469

Tabela 6: Czas wykonania funkcji w sekundach w zależności od wielkości macierzy

n	linsolve()	B / A	B * inv(A)	solveByChol()
5	0.0000	0.0000	0.0000	0.0001
10	0.0000	0.0000	0.0000	0.0001
20	0.0000	0.0000	0.0000	0.0002
40	0.0000	0.0000	0.0000	0.0005
80	0.0001	0.0000	0.0000	0.0009
160	0.0002	0.0001	0.0004	0.0018
500	0.0029	0.0023	0.0042	0.0058
1000	0.0100	0.0066	0.0300	0.0134
2000	0.0612	0.0349	0.1566	0.0341
3000	0.2233	0.1259	0.3980	0.0622
4000	0.4283	0.2447	0.9779	0.0987
8000	3.3183	1.7504	8.1858	0.3294



Rysunek 7: Wykres porównujący czasy wykonania

3.4 Rozwiązywanie układów równań wektor-macierz $\in \mathbb{C}^{n\times n}$.

Skrypty: cholSolveSpeed.m | plotCreatorSolve.m

W tym przykładzie tworzyłem losowo macierz m-diagonalną $\in \mathbb{C}^{n\times n}$ używając funkcji randKdiagC(), a dla wylosowanego wektora X wyliczałem B i porównywałem X z wektorem zwracanym przez:

- (linsolve(A.', B.')).'
- B / A
- B * inv(A)
- solveByChol(A, B, 5)

Wyniki prezentują się następująco:

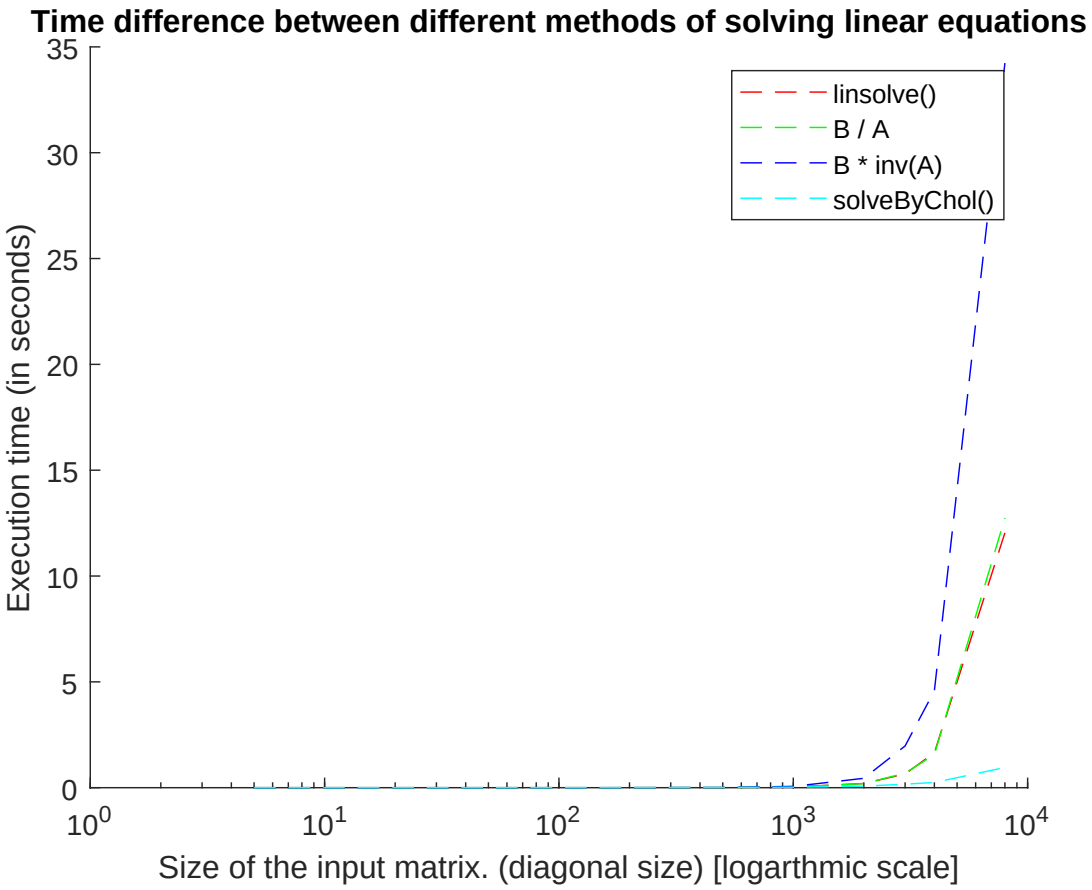
Tabela 7: Względne błędy funkcji (norma wektorowa) w zależności od wielkości macierzy

n	linsolve()	B / A	B * inv(A)	solveByChol()
5	0.0000	0.0000	0.0000	0.3708
10	0.0000	0.0000	0.0000	0.2380
20	0.0000	0.0000	0.0000	0.2694
40	0.0000	0.0000	0.0000	0.1809
80	0.0000	0.0000	0.0000	0.1055
160	0.0000	0.0000	0.0000	0.0784
500	0.0000	0.0000	0.0000	0.0569
1000	0.0000	0.0000	0.0000	0.0342
2000	0.0000	0.0000	0.0000	0.0243
3000	0.0000	0.0000	0.0000	0.0209
4000	0.0000	0.0000	0.0000	0.0184
8000	0.0000	0.0000	0.0000	0.0129

Zauważmy, iż tutaj błędy nie są pomnożone przez (10^{-15}), zatem owe błędy nie są już niestety pomijalne.

Tabela 8: Czas wykonania funkcji w sekundach w zależności od wielkości macierzy

n	linsolve()	B / A	B * inv(A)	solveByChol()
5	0.0000	0.0000	0.0000	0.0002
10	0.0000	0.0000	0.0000	0.0001
20	0.0000	0.0000	0.0000	0.0003
40	0.0000	0.0000	0.0000	0.0009
80	0.0001	0.0001	0.0001	0.0010
160	0.0003	0.0003	0.0006	0.0023
500	0.0053	0.0047	0.0074	0.0111
1000	0.0347	0.0302	0.0617	0.0244
2000	0.1978	0.1919	0.4435	0.0709
3000	0.6487	0.6869	1.9723	0.1678
4000	1.6514	1.5652	4.5792	0.2453
8000	12.0418	12.7452	34.2354	0.9531



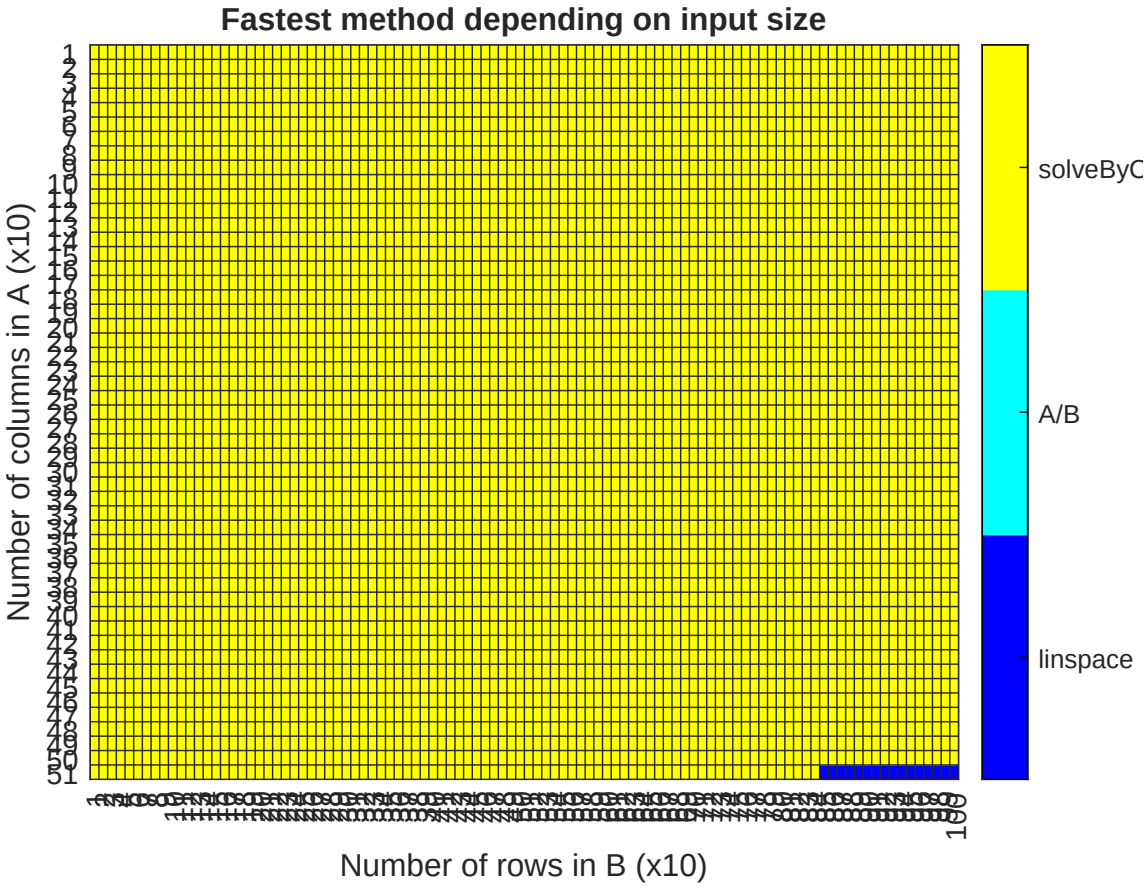
Rysunek 8: Wykres porównujący czasy wykonania

3.5 Wyznaczenie kiedy solveByChol() się opłaca?

Skrypty: bestPickCalc.m | plotBestPick.m

Aby sprawdzić dokładnie dla jakich macierzy wykonywanie rozwiązywania układów $XA = B$ przy pomocy rozkładu Cholesky'ego-Banachiewicza przeprowadziłem obliczenia dla 50 różnych macierzy 5-diagonalnych w $\mathbb{C}^{\kappa\times\kappa}$ A (od 10×10 do 500×500), a dla każdej z nich sprawdziłem tempo obliczania X dla 100 różnych wektorów od $(10 \times n$ do $1000 \times n)$.

Wnyki okazały się zaskakujące jako, iż nie we wszystkich przypadkach opłacało się użyć rozkładu Cholesky'ego-Banachiewicza. Inne metody zaczęły wygrywać pod względem czasu wykonania kiedy macierz B i A były duże i była duża różnica miedzy ich wymiarami.



Rysunek 9: Wykres najszybszej metody z zależności od wejścia

3.6 Wyliczanie macierzy odwrotnej.

Patrząc na poprzedni przykład jedną z nachodzących myśli jest: Co byłoby gdyby macierz B była jednostkowa. Matlab ma dobrą i szybką implementacje.

Zobaczmy jak w porównaniu do wbudowanej metody szybkie jest obliczanie odwrotności macierzy przy pomocy rozwiązywania układu równań XA = I.

Wyniki prezentują się następująco:

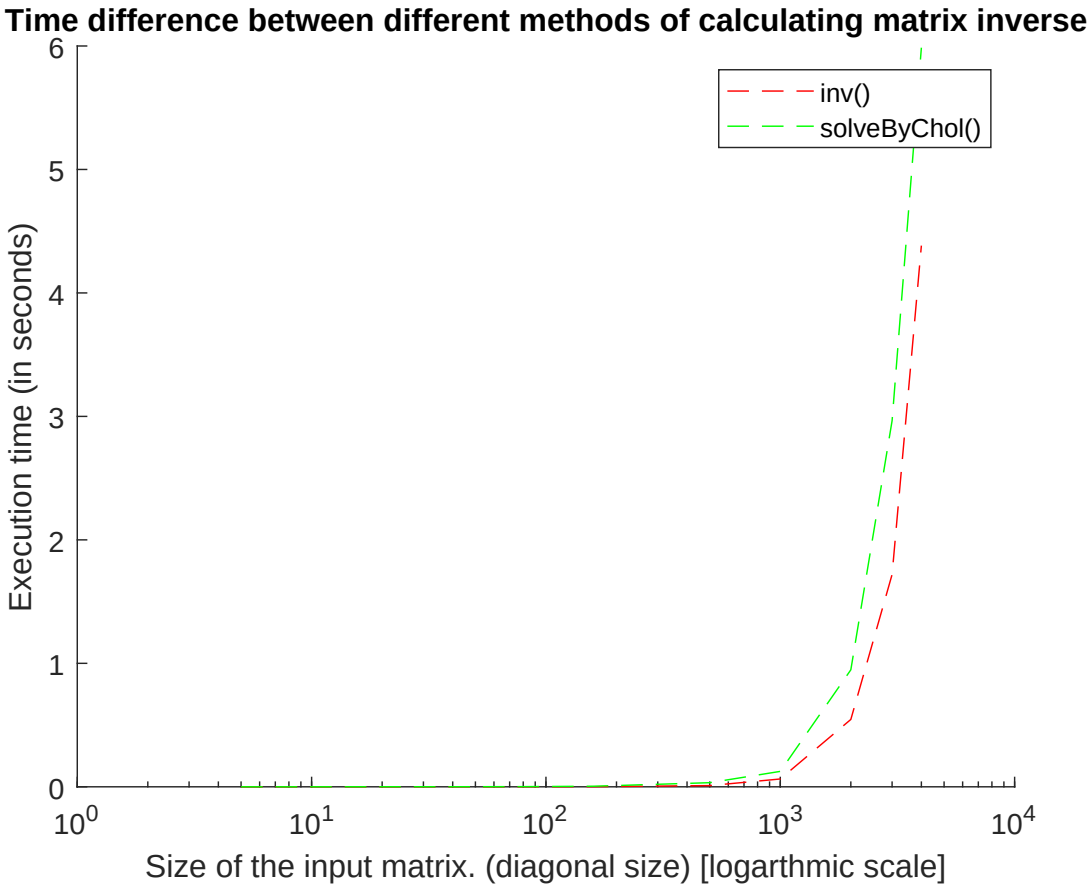
Tabela 9: Względne błędy funkcji (norma macierzowa) w zależności od wielkości macierzy

n	solveByChol
5	0.3447
10	0.1541
20	0.1201
40	0.1291
80	0.1008
160	0.0786
500	0.0581
1000	0.0389
2000	0.0257
3000	0.0243
4000	0.0211

Zauważmy, iż tutaj błędy nie są pomnożone przez (10^{-15}), zatem owe błędy nie są tak bliskie zeru.

Tabela 10: Czas wykonania funkcji w sekundach w zależności od wielkości macierzy

n	inv()	solveByChol()
5	0.0000	0.0001
10	0.0000	0.0001
20	0.0000	0.0005
40	0.0001	0.0006
80	0.0002	0.0014
160	0.0008	0.0047
500	0.0099	0.0333
1000	0.0634	0.1249
2000	0.5455	0.9475
3000	1.7196	2.9650
4000	4.3831	5.9861



Rysunek 10: Wykres najszybszej metody z zależności od wejścia

4 Analiza wyników

Przyglądając się bliżej wynikom przedstawionym powyżej można dojść do następujących konkluzji:

1. Optymalizacja algorytmu pod dane wejściowe ma znaczenie. Kiedy pracujemy przy dużych liczbach lub wykonujemy operację dużą liczbę razy dla danej rodziny wejściowej, dobrym rozwiązaniem jest zoptymalizowanie algorytmu, aby wykorzystywał wiadome własności badanej rodziny. W tym przypadku badaną rodziną była rodzina macierzy symetrycznych dodatnio określonych i 5cio-diagonalnych. Poprzez zoptymalizowanie początkowego algorytmu byliśmy w stanie

Po przejściu przez funkcje, których współgranie z metodą prostokątów widzimy powyżej oraz więcej, jednym z wniosków jaki jesteśmy w stanie zaobserwować jest fakt, iż błąd dla węzła środkowego jest prawie zawsze najmniejszy, a kiedy nie jest najmniejszy nie odbiega on w dużym stopniu od najlepszego.

2. Wyznaczanie macierzy odwrotnej używając rozkładu Cholesky'ego powoduje błędy rachunkowe. Prawdopodobnym powodem tej zmiany są błędy rachunkowe, które powstają przy dzieleniu liczb zespolonych.

3. Nawet kiedy pracujemy przy macierzach m-diagonlanych. Algorytmy specjalnie przystosowany do całej rodziny m-diagonalnych macierzy nie zawsze jest najefektywniejszym algorytmem dla pewnej macierzy z tej rodziny. Jak zaobserwowaliśmy w przykładzie 5. metoda solveByChol była pokonana przez linsolve pod względem czasu wykonania.

4. Rozkład Cholesky'ego ma duże zastosowania przy macierzach rzadkich. Kiedy jesteśmy w stanie przewidzieć części macierzy, które będą miały ustalone wartości to warto wykorzystać ten części, aby uwzględnić je w implementacji, aby otrzymywać lepsze wyniki dla owej rodziny macierzy.

Literatura

[1] Notatki do Metod Numerycznych autorstwa dr. Iwony Wróbel