Projekt 2

Mikołaj Piórczyński, grupa 3

14 stycznia 2022

Spis treści

1 Temat i treść zadania

Rozwiązywanie układu równań $Ax = b$ blokową metodą Cholesky'ego-Banachiewicza. Zakładamy, że $A(n \times n)$ jest macierzą symetryczną i dodatnio określoną postaci

$$A = \begin{pmatrix} A_{11} & A_{12} & 0 \\ A_{12}^T & A_{22} & A_{23} \\ 0 & A_{23}^T & A_{33} \end{pmatrix},$$

gdzie $A_{ij}(p \times p)$ i $n = 3p$.

2 Opis metody

2.1 Rozkład Cholesky'ego-Banachiewicza

Wiemy, że jeśli $A \in \mathbb{R}^{n \times n}$ jest macierzą symetryczną i dodatnio określoną, to istnieje dokładnie jedna macierz trójkątna dolna L z dodatnimi elementami na głównej przekątnej, taka, że $A = LL^T$. Rozkład ten nazywamy rozkładem Cholesky'ego-Banachiewicza macierzy A, a jego algorytm przedstawia się następująco:

---

for k $= 1, 2, \ldots, n$ **do**
  $l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2}$
  for $i = k+1, k+2, \ldots, n$ **do**
    $l_{ik} = (a_{ik} - \sum_{j=1}^{k-1} l_{ij} l_{kj})/l_{kk}$
  end for
end for

2.2 Blokowy rozkład Cholesky'ego-Banachiewicza

Niech macierz $A \in \mathbb{R}^{n \times n}$ będzie postaci:

$$A = \begin{pmatrix} A_{11} & A_{12} & 0 \\ A_{12}^T & A_{22} & A_{23} \\ 0 & A_{23}^T & A_{33} \end{pmatrix},$$

gdzie $A_{ij} \in \mathbb{R}^{p\times p}$ i $n = 3p$. Szukając blokowego rozkładu Cholesky'ego-Banachiewicza macierzy blokowej A szukamy takiej macierzy blokowo trójkątnej dolnej L, że $A = LL^T$. Algorytm wyznaczania rozkładu Cholesky'ego-Banachiewicza otrzymujemy na podstawie równania $A = LL^T$.

$$\begin{pmatrix} A_{11} & A_{12} & 0 \\ A_{12}^T & A_{22} & A_{23} \\ 0 & A_{23}^T & A_{33} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T & L_{31}^T \\ 0 & L_{22}^T & L_{32}^T \\ 0 & 0 & L_{33}^T \end{pmatrix}$$

Po rozpisaniu iloczynu znajdującego się po prawej stronie powyższej równości możemy porównać odpowiadające sobie bloki macierzy A i LL^T. Otrzymujemy wówczas przedstawione poniżej zależności, z których wyznaczymy bloki macierzy L.

$$\begin{aligned}
&L_{11}L_{11}^T = A_{11} \\
&L_{21}L_{11}^T = A_{12}^T \rightarrow L_{11}L_{21}^T = A_{12} \\
&L_{21}L_{21}^T + L_{22}L_{22}^T = A_{22} \rightarrow L_{22}L_{22}^T = A_{22} - L_{21}L_{21}^T \\
&L_{31}L_{11}^T = 0 \rightarrow L_{11}L_{31}^T = 0 \\
&L_{31}L_{21}^T + L_{32}L_{22}^T = A_{23}^T \rightarrow L_{22}L_{32}^T = A_{23} - L_{21}L_{31}^T \\
&L_{31}L_{31}^T + L_{32}L_{32}^T + L_{33}L_{33}^T = A_{33} \rightarrow L_{33}L_{33}^T = A_{33} - L_{31}L_{31}^T - L_{32}L_{32}^T
\end{aligned}$$

Macierz L_{11} wyznaczamy dokonując „zwykłego" (nieblokowego) rozkładu Cholesky'ego macierzy A_{11}. Następnie macierz L_{21} wyznaczamy rozwiązując równanie macierzowe z obliczoną macierzą L_{11}:

$$L_{11}L_{21}^T = A_{12}$$

Zauważmy, że macierz L_{11} jest dolna trójkątna, zatem możemy łatwo rozwiązać to równanie rozwiązując p układów równań liniowych z macierzą L_{11} metodą podstawienia w przód.
W analogiczny sposób macierze L_{31} i L_{32} otrzymujemy rozwiązując równania macierzowe, natomiast macierze L_{22} i L_{33} dokonując rozkładu Cholesky'ego odpowiednich macierzy.

2.3 Rozwiązywanie układów równań $Ax = b$

Podstawiając $A = LL^T$ otrzymujemy:

$$L\underbrace{L^T x}_{\text{y}} = b$$

Rozwiązanie tego układu znajdujemy rozwiązując 2 układy z macierzami blokowo trójkątnymi:

$$Ly = b \quad \text{oraz} \quad L^T x = y$$

1) $Ly = b$

$$\begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$\begin{cases} L_{11}y_1 = b_1 \\ L_{21}y_1 + L_{22}y_2 = b_2 \\ L_{31}y_1 + L_{32}y_2 + L_{33}y_3 = b_3 \end{cases}$$

$$\begin{cases} L_{11}y_1 = b_1 \\ L_{22}y_2 = b_2 - L_{21}y_1 \\ L_{33}y_3 = b_2 - L_{31}y_1 - L_{32}y_2 \end{cases}$$

2) $L^T y = x$

$$\begin{pmatrix} L_{11}^T & L_{21}^T & L_{31}^T \\ 0 & L_{22}^T & L_{32}^T \\ 0 & 0 & L_{33}^T \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

$$\begin{cases} L_{11}^T x_1 + L_{21}^T x_2 + L_{31}^T x_3 = y_1 \\ L_{22}^T x_2 + L_{32}^T x_3 = y_2 \\ L_{33}^T x_3 = y_3 \end{cases}$$

$$\begin{cases} L_{11}^T x_1 = y_1 - L_{21}^T x_2 - L_{31}^T x_3 \\ L_{22}^T x_2 = y_2 - L_{32}^T x_3 \\ L_{33}^T x_3 = y_3 \end{cases}$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Zauważmy, że macierze L_{ij} są dolne trójkątne, zatem rozwiązywanie z nimi układów równań jest proste poprzez podstawienie w przód. Analogicznie równań z macierzami L_{ij}^T są łatwo rozwiązywalne metodą podstawienie w tył.

3 Opis programu obliczeniowego

Program obliczeniowy składa się z funkcji opisanych poniżej oraz funkcji pomocniczych i skryptu z przykładami, które zostały szczegółowo opisane w następnym rozdziale.

3.1 cholesky(A)

Funkcja numerycznie oblicza rozkład Cholesky'ego-Banachiewicza symetrycznej dodatnio określonej macierzy A.

Funkcja przyjmuje:

- A - wejściowa macierz, matrix

Funkcja zwraca:

- L - macierz dolna trójkątna, taka że $A = LL^T$, matrix

Przykład:

```
>> L = cholesky([1 0 1; 0 2 0; 1 0 3])

L =

    1.0000         0         0
         0    1.4142         0
    1.0000         0    1.4142
```

3.2 forwardSubstitution(A, B)

Funkcja numerycznie rozwiązuje układ równań liniowych AX = B z dolną trójkątną macierzą A wykorzystując metodę podstawienia w przód.

Funkcja przyjmuje:

- A - macierz współczynników, matrix
- B - wektor lub macierz wyrazów wolnych, vector | matrix

Funkcja zwraca:

- X - wektor lub macierz rozwiązań układu, vector | matrix

Przykład:

```
>> A = [1 0 0; 1 1 0; 1 1 1];
b = ones(3, 1);
x = forwardSubstitution(A, b)

x =

     1
     0
     0
```

3.3 backwardSubstitution(A, B)

Funkcja numerycznie rozwiązuje układ równań liniowych AX = B z górną trójkątną macierzą A wykorzystując metodę podstawienia w tył.

Funkcja przyjmuje:

- A - macierz współczynników, matrix
- B - wektor lub macierz wyrazów wolnych, vector | matrix

Funkcja zwraca:

- X - wektor lub macierz rozwiązań układu, vector | matrix

Przykład:

```
A = [1 -2 1; 0 1 6; 0 0 1];
b = [4; -1; 2];
x = backwardSubstitution(A, b)

x =

   -24
   -13
     2
```

3.4 blockCholesky($A_{11}, A_{12}, A_{22}, A_{23}, A_{33}$)

Funkcja numerycznie oblicza blokowy rozkład Cholesky'ego-Banachiewicza symetrycznej dodatnio określonej blokowej macierzy A postaci takiej jak w treści zadania.

Funkcja przyjmuje:

- $A_{11}, A_{12}, A_{22}, A_{23}, A_{33}$ - bloki macierzy A, matrices

Funkcja zwraca:

- $L_{11}, L_{21}, L_{22}, L_{31}, L_{32}, L_{33}$ - bloki blokowo dolnej trójkątnej macierzy L, takiej że $A = LL^T$, matrices

Przykład:

```
>> [L_11, L_21, L_22, L_31, L_32, L_33] = blockCholesky(
[3.2663], [2.7407], [6.2922], [2.0382], [3.4665])

L_11 =

    1.8073


L_21 =

    1.5165
```

```
L_22 =

    1.9981


L_31 =

     0


L_32 =

    1.0201


L_33 =

    1.5576
```

3.5 solveBlockCholesky(A, b)

Funkcja numerycznie rozwiązuje układ równań liniowych Ax = b z symetryczną dodatnio określoną macierzą blokową A w postaci takiej jak w treści zadania wykorzystując blokową metodę Cholesky'ego-Banachiewicza.

Funkcja przyjmuje:

- A - macierz współczynników, matrix
- B - wektor lub macierz wyrazów wolnych, vector | matrix

Funkcja zwraca:

- X - wektor lub macierz rozwiązań układu, vector | matrix

Przykład:

```
>> A = [0.6788    0.1536   -0.5871   -0.1328         0         0;
        0.1536    0.5322   -0.1328   -0.4603         0         0;
```

```
       -0.5871   -0.1328    2.2502    0.5091   -0.4155   -0.0940;
       -0.1328   -0.4603    0.5091    1.7641   -0.0940   -0.3257;
             0         0   -0.4155   -0.0940    3.9045    0.8835;
             0         0   -0.0940   -0.3257    0.8835    3.0612];
b = ones(6, 1);
x = solveBlockCholesky(A, b)

x =

    1.8852
    2.6154
    0.8834
    1.2257
    0.2889
    0.4008
```

Złożoność obliczeniowa algorytmu jest $O(n^3)$

4 Przykłady obliczeniowe

4.1 Funkcje pomocnicze

W celu przetestowania algorytmu zostały przygotowane funkcje *generateSPD*, *generateTriSPD*, *generateBlockSPD* oraz *generateBlockSPD2* mające za zadanie generowanie przykładowych macierzy spełniających warunki zadania. Ponieważ nie było to częścią zadania, szczegółowy opis metod zastosowanych w implementacji tych funkcji nie zostanie tu przedstawiony. Dodatkowo także zostały zaimplementowane funkcje *abserror* i *relerror* liczące odpowiednio bład bezwzględny rozwiązania i bład względny.

4.1.1 generateSPD(n)

Funkcja generuje losową symetryczną dodatnio określoną macierz $n \times n$.

Funkcja przyjmuje:

- n - rozmiar generowanej macierzy, scalar

Funkcja zwraca:

- A - wygenerowana macierz, matrix

Przykład:

```
>> generateSPD(3)

ans =

    3.4818    0.3061    0.4790
    0.3061    3.6048    0.5601
    0.4790    0.5601    3.9857
```

4.1.2 generateTriSPD(n)

Funkcja generuje losową trójdiagonalną symetryczną dodatnio określoną macierz $n \times n$.

Funkcja przyjmuje:

- n - rozmiar generowanej macierzy, scalar

Funkcja zwraca:

- A - wygenerowana macierz, matrix

Przykład:

```
>> generateTriSPD(3)

ans =

    1.3757    0.7949         0
    0.7949    3.2882    1.2378
         0    1.2378    2.7034
```

4.1.3 generateBlockSPD(p)

Funkcja generuje losową symetryczną dodatnio określoną macierz blokową $3p \times 3p$ postaci takiej jak w treści zadania.

Funkcja przyjmuje:

- p - rozmiar bloków generowanej macierzy, scalar

Funkcja zwraca:

- A - wygenerowana macierz, matrix

Przykład:

```
>> generateBlockSPD(2)

ans =

    5.5571    1.7419   -3.7727   -1.1826         0         0
    1.7419    5.2508   -1.1826   -3.5648         0         0
   -3.7727   -1.1826   12.0690    3.7832    5.0937    1.5967
   -1.1826   -3.5648    3.7832   11.4038    1.5967    4.8129
         0         0    5.0937    1.5967    5.5007    1.7243
         0         0    1.5967    4.8129    1.7243    5.1975
```

4.1.4 generateBlockSPD2(A)

Funkcja generuje częściowo losową symetryczną dodatnio określoną macierz blokową $3p \times 3p$ postaci takiej jak w treści zadania wykorzystując w konstrukcji macierz kwadratową $p \times p$ A.

Funkcja przyjmuje:

- A - wykorzystywana macierz, matrix

Funkcja zwraca:

- X - wygenerowana macierz, matrix

Przykład:

```
>> generateBlockSPD2(gallery('minij', 2))

ans =

    2.7965    2.7965    0.5377    0.5377         0         0
    2.7965    5.5930    0.5377    1.0753         0         0
    0.5377    0.5377    3.2337    3.2337    1.8339    1.8339
    0.5377    1.0753    3.2337    6.4675    1.8339    3.6678
         0         0    1.8339    1.8339    2.1527    2.1527
         0         0    1.8339    3.6678    2.1527    4.3053
```

4.1.5 abserror(x, x_true)

Funkcja obliczająca błąd bezwzględny rozwiązania wykorzystując normę euklidesową.

Funkcja przyjmuje:

- x - obliczona wartość, vector
- x_true - dokładna wartość, vector

Funkcja zwraca:

- abserror - błąd bezwzględny rozwiązania, scalar

Przykład:

```
>> x_true = [1; 1; 1];
x = [ 1.9450; 1.6537; 1.1034];
>> abserror(x, x_true)

ans =

    1.1537
```

4.1.6 relerror(x, x_true)

Funkcja obliczająca błąd względny rozwiązania wykorzystując normę euklidesową.

Funkcja przyjmuje:

- x - obliczona wartość, vector
- x_true - dokładna wartość, vector

Funkcja zwraca:

- relerror - błąd względny rozwiązania, scalar

Przykład:

```
>> x_true = [1; 1; 1];
x = [1.9450; 1.6537; 1.1034];
>> relerror(x, x_true)

ans =

    0.6661
```

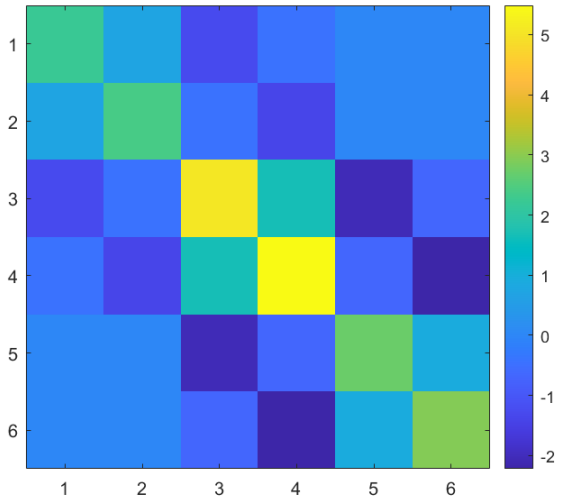
W każdym z poniższych przykładów obliczeniowych policzono rozwiązanie równania $Ax = b$ zaimplementowaną metodą, porównano tak otrzymane rozwiązanie z rozwiązaniem „dokładnym" otrzymanym przy pomocy wbudowanej w MATLABA funkcji *linsolve*, obliczono błąd względny oraz bezwzględny otrzymanego metodą rozwiązania, policzono współczynnik uwarunkowania macierzy układu oraz zwizulizowano macierz układu za pomocą mapy ciepła.

4.2 Przykład nr. 1

```
>> p = 2;
A = generateBlockSPD(p)
b = ones(3*p, 1)
x = solveBlockCholesky(A, b)
x_true = linsolve(A, b)
cond_coeff = cond(A)
abs_error = abserror(x, x_true)
rel_error = relerror(x, x_true)

A =

    2.2162    0.7286   -1.2826   -0.4217         0         0
    0.7286    2.3870   -0.4217   -1.3815         0         0
   -1.2826   -0.4217    5.0931    1.6745   -2.0605   -0.6774
   -0.4217   -1.3815    1.6745    5.4857   -0.6774   -2.2193
         0         0   -2.0605   -0.6774    2.7445    0.9023
         0         0   -0.6774   -2.2193    0.9023    2.9560


b =
```

```
     1
     1
     1
     1
     1
     1


x =

    0.7198
    0.6457
    0.6416
    0.5755
    0.7631
    0.6845


x_true =

    0.7198
    0.6457
    0.6416
    0.5755
    0.7631
    0.6845


cond_coeff =

   11.7484


abs_error =

   3.6822e-16
```

```
rel_error =

   2.2291e-16
```



Rysunek 1: Przykład nr. 1

4.3 Przykład nr. 2

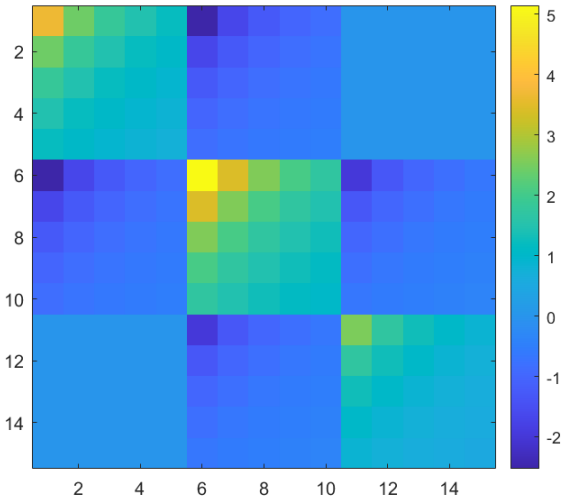
```
>> p = 5;
A = generateBlockSPD2(gallery('wilk', 5))
b = ones(3*p, 1)
x = solveBlockCholesky(A, b)
x_true = linsolve(A, b)
cond_coeff = cond(A)
abs_error = abserror(x, x_true)
rel_error = relerror(x, x_true)

A =
```

```
Columns 1 through 6

  3.6550    2.4367    1.8275    1.4620    1.2183   -2.5313
  2.4367    1.8275    1.4620    1.2183    1.0443   -1.6875
  1.8275    1.4620    1.2183    1.0443    0.9138   -1.2656
  1.4620    1.2183    1.0443    0.9138    0.8122   -1.0125
  1.2183    1.0443    0.9138    0.8122    0.7310   -0.8438
 -2.5313   -1.6875   -1.2656   -1.0125   -0.8438    5.1480
 -1.6875   -1.2656   -1.0125   -0.8438   -0.7232    3.4320
 -1.2656   -1.0125   -0.8438   -0.7232   -0.6328    2.5740
 -1.0125   -0.8438   -0.7232   -0.6328   -0.5625    2.0592
 -0.8438   -0.7232   -0.6328   -0.5625   -0.5063    1.7160
       0         0         0         0         0   -1.9880
       0         0         0         0         0   -1.3253
       0         0         0         0         0   -0.9940
       0         0         0         0         0   -0.7952
       0         0         0         0         0   -0.6627

Columns 7 through 12

 -1.6875   -1.2656   -1.0125   -0.8438         0         0
 -1.2656   -1.0125   -0.8438   -0.7232         0         0
 -1.0125   -0.8438   -0.7232   -0.6328         0         0
 -0.8438   -0.7232   -0.6328   -0.5625         0         0
 -0.7232   -0.6328   -0.5625   -0.5063         0         0
  3.4320    2.5740    2.0592    1.7160   -1.9880   -1.3253
  2.5740    2.0592    1.7160    1.4709   -1.3253   -0.9940
  2.0592    1.7160    1.4709    1.2870   -0.9940   -0.7952
  1.7160    1.4709    1.2870    1.1440   -0.7952   -0.6627
  1.4709    1.2870    1.1440    1.0296   -0.6627   -0.5680
 -1.3253   -0.9940   -0.7952   -0.6627    2.5585    1.7057
 -0.9940   -0.7952   -0.6627   -0.5680    1.7057    1.2793
 -0.7952   -0.6627   -0.5680   -0.4970    1.2793    1.0234
 -0.6627   -0.5680   -0.4970   -0.4418    1.0234    0.8528
 -0.5680   -0.4970   -0.4418   -0.3976    0.8528    0.7310

Columns 13 through 15
```

```
         0         0         0
         0         0         0
         0         0         0
         0         0         0
         0         0         0
   -0.9940   -0.7952   -0.6627
   -0.7952   -0.6627   -0.5680
   -0.6627   -0.5680   -0.4970
   -0.5680   -0.4970   -0.4418
   -0.4970   -0.4418   -0.3976
    1.2793    1.0234    0.8528
    1.0234    0.8528    0.7310
    0.8528    0.7310    0.6396
    0.7310    0.6396    0.5686
    0.6396    0.5686    0.5117

b =

     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1

x =
```

```
   1.0e+03 *

    0.0180
   -0.2516
    1.0062
   -1.5093
    0.7547
    0.0200
   -0.2803
    1.1211
   -1.6816
    0.8408
    0.0214
   -0.2998
    1.1994
   -1.7991
    0.8995


x_true =

   1.0e+03 *

    0.0180
   -0.2516
    1.0062
   -1.5093
    0.7547
    0.0200
   -0.2803
    1.1211
   -1.6816
    0.8408
    0.0214
   -0.2998
    1.1994
   -1.7991
    0.8995
```

```
cond_coeff =

   1.5448e+07


abs_error =

   1.2395e-07


rel_error =

   3.2699e-11
```



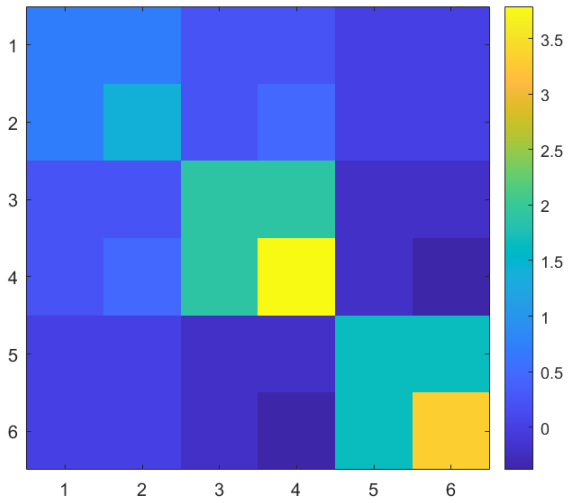
Rysunek 2: Przykład nr. 2

4.4 Przykład nr. 3

```
>> p = 2;
```

```
A = generateBlockSPD2(gallery('minij', p))
b = ones(3*p, 1)
x = solveBlockCholesky(A, b)
x_true = linsolve(A, b)
cond_coeff = cond(A)
abs_error = abserror(x, x_true)
rel_error = relerror(x, x_true)

A =

    0.7012    0.7012    0.2319    0.2319         0         0
    0.7012    1.4025    0.2319    0.4637         0         0
    0.2319    0.2319    1.8951    1.8951   -0.1921   -0.1921
    0.2319    0.4637    1.8951    3.7901   -0.1921   -0.3843
         0         0   -0.1921   -0.1921    1.6575    1.6575
         0         0   -0.1921   -0.3843    1.6575    3.3150


b =

     1
     1
     1
     1
     1
     1


x =

    1.2815
    0.0000
    0.4372
   -0.0000
    0.6540
   -0.0000
```

```
x_true =

    1.2815
   -0.0000
    0.4372
    0.0000
    0.6540
         0


cond_coeff =

   21.2311


abs_error =

   6.1230e-16


rel_error =

   4.0720e-16
```



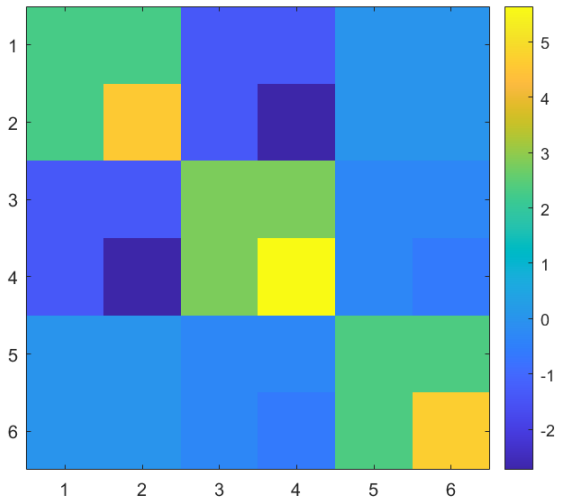
Rysunek 3: Przykład nr. 3

4.5 Przykład nr. 4

```
 >> p = 2;
A = generateBlockSPD2(gallery('gcdmat', p))
b = ones(3*p, 1)
x = solveBlockCholesky(A, b)
x_true = linsolve(A, b)
cond_coeff = cond(A)
abs_error = abserror(x, x_true)
rel_error = relerror(x, x_true)

A =

    2.2855    2.2855   -1.3639   -1.3639         0         0
    2.2855    4.5709   -1.3639   -2.7277         0         0
   -1.3639   -1.3639    2.8192    2.8192   -0.3158   -0.3158
   -1.3639   -2.7277    2.8192    5.6385   -0.3158   -0.6316
         0         0   -0.3158   -0.3158    2.3408    2.3408
         0         0   -0.3158   -0.6316    2.3408    4.6816
```

```
b =

     1
     1
     1
     1
     1
     1


x =

    0.9640
    0.0000
    0.8823
    0.0000
    0.5462
    0.0000


x_true =

    0.9640
         0
    0.8823
         0
    0.5462
         0


cond_coeff =

   24.1589


abs_error =
```

```
   4.3323e-16


rel_error =

   3.0587e-16
```



Rysunek 4: Przykład nr. 4

4.6 Przykład nr. 5

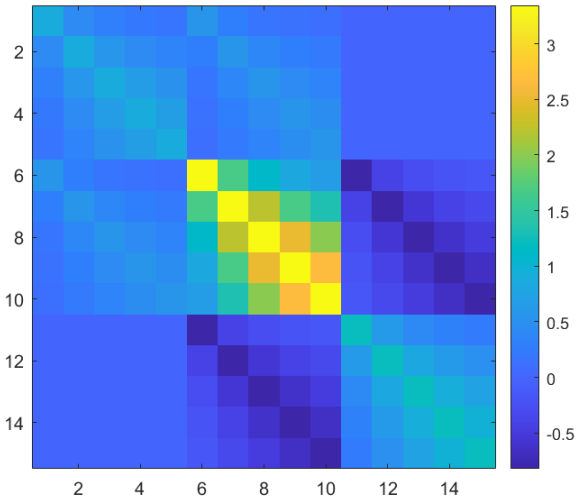
```
    >> p = 5;
A = generateBlockSPD2(gallery('lehmer', p))
b = ones(3*p, 1)
x = solveBlockCholesky(A, b)
x_true = linsolve(A, b)
cond_coeff = cond(A)
abs_error = abserror(x, x_true)
rel_error = relerror(x, x_true)

A =
```

```
Columns 1 through 6

  0.8602    0.4301    0.2867    0.2150    0.1720    0.5600
  0.4301    0.8602    0.5734    0.4301    0.3441    0.2800
  0.2867    0.5734    0.8602    0.6451    0.5161    0.1867
  0.2150    0.4301    0.6451    0.8602    0.6881    0.1400
  0.1720    0.3441    0.5161    0.6881    0.8602    0.1120
  0.5600    0.2800    0.1867    0.1400    0.1120    3.3449
  0.2800    0.5600    0.3733    0.2800    0.2240    1.6725
  0.1867    0.3733    0.5600    0.4200    0.3360    1.1150
  0.1400    0.2800    0.4200    0.5600    0.4480    0.8362
  0.1120    0.2240    0.3360    0.4480    0.5600    0.6690
       0         0         0         0         0   -0.8227
       0         0         0         0         0   -0.4113
       0         0         0         0         0   -0.2742
       0         0         0         0         0   -0.2057
       0         0         0         0         0   -0.1645

Columns 7 through 12

  0.2800    0.1867    0.1400    0.1120         0         0
  0.5600    0.3733    0.2800    0.2240         0         0
  0.3733    0.5600    0.4200    0.3360         0         0
  0.2800    0.4200    0.5600    0.4480         0         0
  0.2240    0.3360    0.4480    0.5600         0         0
  1.6725    1.1150    0.8362    0.6690   -0.8227   -0.4113
  3.3449    2.2299    1.6725    1.3380   -0.4113   -0.8227
  2.2299    3.3449    2.5087    2.0069   -0.2742   -0.5485
  1.6725    2.5087    3.3449    2.6759   -0.2057   -0.4113
  1.3380    2.0069    2.6759    3.3449   -0.1645   -0.3291
 -0.4113   -0.2742   -0.2057   -0.1645    1.2205    0.6103
 -0.8227   -0.5485   -0.4113   -0.3291    0.6103    1.2205
 -0.5485   -0.8227   -0.6170   -0.4936    0.4068    0.8137
 -0.4113   -0.6170   -0.8227   -0.6581    0.3051    0.6103
 -0.3291   -0.4936   -0.6581   -0.8227    0.2441    0.4882

Columns 13 through 15
```

```
         0         0         0
         0         0         0
         0         0         0
         0         0         0
         0         0         0
   -0.2742   -0.2057   -0.1645
   -0.5485   -0.4113   -0.3291
   -0.8227   -0.6170   -0.4936
   -0.6170   -0.8227   -0.6581
   -0.4936   -0.6581   -0.8227
    0.4068    0.3051    0.2441
    0.8137    0.6103    0.4882
    1.2205    0.9154    0.7323
    0.9154    1.2205    0.9764
    0.7323    0.9764    1.2205

b =

     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1

x =
```

```
    0.5920
    0.2368
    0.1522
    0.1128
    0.4933
    0.2812
    0.1125
    0.0723
    0.0536
    0.2343
    0.7357
    0.2943
    0.1892
    0.1401
    0.6131


x_true =

    0.5920
    0.2368
    0.1522
    0.1128
    0.4933
    0.2812
    0.1125
    0.0723
    0.0536
    0.2343
    0.7357
    0.2943
    0.1892
    0.1401
    0.6131


cond_coeff =
```

```
  112.3492

abs_error =

   2.5238e-15

rel_error =

   1.8311e-15
```



Rysunek 5: Przykład nr. 5

4.7 Przykład nr. 6

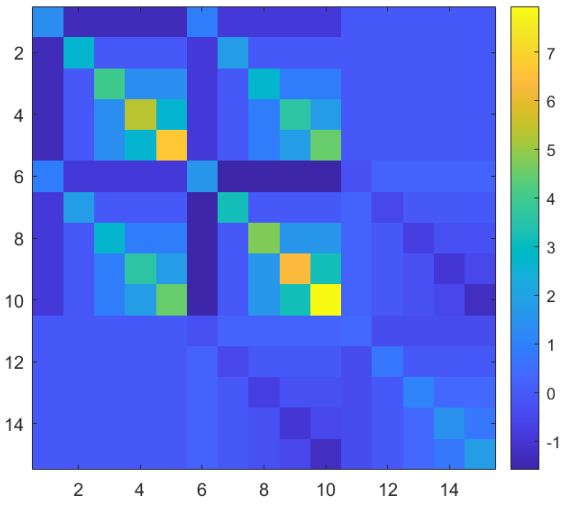
```
    >> p = 5;
A = generateBlockSPD2(gallery('moler', p))
b = ones(3*p, 1)
x = solveBlockCholesky(A, b)
```

```
x_true = linsolve(A, b)
cond_coeff = cond(A)
abs_error = abserror(x, x_true)
rel_error = relerror(x, x_true)

A =

  Columns 1 through 6

    1.3375   -1.3375   -1.3375   -1.3375   -1.3375    0.8995
   -1.3375    2.6750         0         0         0   -0.8995
   -1.3375         0    4.0125    1.3375    1.3375   -0.8995
   -1.3375         0    1.3375    5.3500    2.6750   -0.8995
   -1.3375         0    1.3375    2.6750    6.6875   -0.8995
    0.8995   -0.8995   -0.8995   -0.8995   -0.8995    1.5871
   -0.8995    1.7989         0         0         0   -1.5871
   -0.8995         0    2.6984    0.8995    0.8995   -1.5871
   -0.8995         0    0.8995    3.5978    1.7989   -1.5871
   -0.8995         0    0.8995    1.7989    4.4973   -1.5871
         0         0         0         0         0   -0.2458
         0         0         0         0         0    0.2458
         0         0         0         0         0    0.2458
         0         0         0         0         0    0.2458
         0         0         0         0         0    0.2458

  Columns 7 through 12

   -0.8995   -0.8995   -0.8995   -0.8995         0         0
    1.7989         0         0         0         0         0
         0    2.6984    0.8995    0.8995         0         0
         0    0.8995    3.5978    1.7989         0         0
         0    0.8995    1.7989    4.4973         0         0
   -1.5871   -1.5871   -1.5871   -1.5871   -0.2458    0.2458
    3.1743         0         0         0    0.2458   -0.4915
         0    4.7614    1.5871    1.5871    0.2458         0
         0    1.5871    6.3485    3.1743    0.2458         0
         0    1.5871    3.1743    7.9356    0.2458         0
    0.2458    0.2458    0.2458    0.2458    0.3612   -0.3612
```

```
   -0.4915         0         0         0   -0.3612    0.7224
         0   -0.7373   -0.2458   -0.2458   -0.3612         0
         0   -0.2458   -0.9831   -0.4915   -0.3612         0
         0   -0.2458   -0.4915   -1.2288   -0.3612         0

  Columns 13 through 15

         0         0         0
         0         0         0
         0         0         0
         0         0         0
         0         0         0
    0.2458    0.2458    0.2458
         0         0         0
   -0.7373   -0.2458   -0.2458
   -0.2458   -0.9831   -0.4915
   -0.2458   -0.4915   -1.2288
   -0.3612   -0.3612   -0.3612
         0         0         0
    1.0836    0.3612    0.3612
    0.3612    1.4448    0.7224
    0.3612    0.7224    1.8060


b =

     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
```

```
     1
     1
     1


x =

  -14.3650
   -7.2245
   -3.6963
   -2.0161
   -1.3441
  211.4759
  106.3563
   54.4148
   29.6808
   19.7872
  617.3265
  310.4683
  158.8442
   86.6423
   57.7615


x_true =

  -14.3650
   -7.2245
   -3.6963
   -2.0161
   -1.3441
  211.4759
  106.3563
   54.4148
   29.6808
   19.7872
  617.3265
  310.4683
```

```
  158.8442
   86.6423
   57.7615


cond_coeff =

   8.0375e+03


abs_error =

   3.4164e-12


rel_error =

   4.5089e-15
```

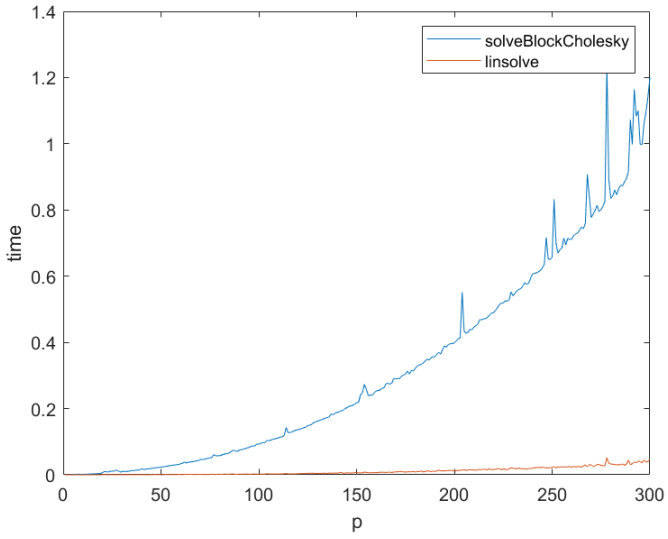


Rysunek 6: Przykład nr. 6

5 Analiza wyników

Metoda zdaje się działać poprawnie w większości przypadków. Otrzymane nią rozwiązania nie odbiegają od rozwiązań „dokładnych" o wartości większego rzędu niż wartość epsilona maszynowego. Na uwagę mogą zwrócić jedynie przykłady nr. 2 i nr. 6, dla których wartość błędu bezwzględnego rozwiązania wyniosła odpowiednio 1.2395e−07 i 3.4164e−12, natomiast błędu względnego 3.2699e−11 i 4.5089e−15. Może to być spowodowane wysokimi wskaźnikami uwarunkowania macierzy w tych przykładach rzędu 1e7 i 1e3, znacznie wyższymi niż w pozostałych przypadkach.

Dodatkowo wykonano porównanie czasów wykonania algorytmu z funkcją *linsolve* wbudowaną w MATLABA dla macierzy różnych rozmiarów generowanych za pomocą funkcji *generateBlockSPD*. Otrzymane wyniki przedstawiono na rys. 7. Z wykresu możemy łatwo odczytać, że zaimplementowany algorytm jest znacznie wolniejszy niż wbudowana w MATLABA funkcja *linsolve*, a jego czas wykonania rośnie proporcjonalnie do rozmiaru macierzy rozwiązywanego układu.



Rysunek 7: Porównanie czasów wykonania algorytmów

6 Literatura

[1] Gene H. Golub, Charles F.Van Loan, Matrix Computations, The Johns Hopkins University Press, 2013.

[2] http://pages.mini.pw.edu.pl/~wrobeli/MN_zima_2021-22/ [dostęp chroniony hasłem: 2022-01-14]