Raport z projektu 2

Damian Skowroński

11 stycznia 2022

1 Wprowadzenie

W projekcie miałem zaimplementować w programie Matlab wyznaczanie rozkładu Crouta macierzy $A \in \mathbb{R}^{n\times n}$, a następnie wykorzystanie tego rozkładu do rozwiązania równań macierzowych $AX = B$ oraz $XA = B$, gdzie $B \in \mathbb{R}^{n\times m}$.

1.1 Rozkład LU

Żeby opisać rozkład Crouta najpierw należy opisać czym jest rozkład LU. W rozkładzie LU macierz A zapisuje się jako iloczyn macierzy trójkątnej dolnej L oraz trójkątnej górnej U:

$$A = LU,$$

przy czym na głównej przekątnej jednej z nich znajdują się wyłącznie jedynki. Rozkład LU nie może zostać zastosowany dla każdej macierzy. Warunkiem istnienia rozkładu LU dla macierzy jest, aby każdy minor główny tej macierzy był różny niż zero:

$$\text{Niech } \Delta_i = \begin{bmatrix} a_{11} & \cdots & a_{1i} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{ii} \end{bmatrix} \text{ będzie i-tym minorem głównym macierzy } A \in \mathbb{R}^{n\times n}.$$

$$\text{Rozkład LU istnieje wtedy i tylko wtedy, gdy } \forall_{i\in\{1,\dots,n\}}\ \Delta_i \neq 0.$$

1.2 Rozkład Crouta

Rozkład Crouta jest takim rozkładem LU, że na głównej przekątnej macierzy trójkątnej górnej U znajdują się same jedynki. Rozkład ten dla $A \in \mathbb{R}^{n\times n}$ wygląda następująco:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & \cdots & u_{1n} \\ 0 & 1 & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

Stosując te oznaczenia na elementy macierzy wytłumaczę działanie zaimplementowanego przeze mnie algorytmu rozkładu Crouta.
Chcę z $A \in \mathbb{R}^{n\times n}$ otrzymać macierze $L, U \in \mathbb{R}^{n\times n}$. Wiadomo, że

$$\forall i, j = 1, \dots, n \quad a_{ij} = \begin{bmatrix} l_{i1} & \cdots & l_{in} \end{bmatrix} \begin{bmatrix} u_{1j} \\ \cdots \\ u_{nj} \end{bmatrix} = \sum_{k=1}^{n} l_{ik} u_{kj}.$$

tutaj, ponieważ L i U to macierze trójkątnymi odpowiednio dolne i górne, to:

$$\forall i, j = 1, \ldots, n \quad a_{ij} = \sum_{k=1}^{\min(i,j)} l_{ik} u_{kj}.$$

Wobec tego elementy pierwszej kolumny macierzy L są łatwe do policzenia, bo:

$$a_{i1} = l_{i1} * 1 \Rightarrow l_{i1} = a_{i1},$$

czyli pierwsza kolumna macierzy L jest równa pierwszej kolumnie macierzy A. Dalej można wyliczyć elementy pierwszego wiersza macierzy U ponieważ:

$$a_{1j} = l_{11} u_{1j} \Rightarrow u_{1j} = \frac{a_{1j}}{l_{11}}$$

Otrzymałem z tego do tej pory macierze wyglądajęce następująco:

$$L = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & 0 & \cdots & 0 \end{bmatrix} \quad U = \begin{bmatrix} 1 & \frac{a_{12}}{l_{11}} & \cdots & \frac{a_{1n}}{l_{11}} \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix},$$

następnie dopisuję brakujące elementy w kolejnych wierszach $i = 2, \ldots, n$ macierzy L i macierzy U ze wzorów:

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}, \quad \text{dla } j = 2, \ldots, n$$

$$u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}}{l_{ii}}, \quad \text{dla } j = (i+1), \ldots, n$$

1.3 Rozwiązywanie równań

Mając macierze L i U można szybko rozwiązać równanie $AX = B$, gdy za A podstawi się LU. Jest więc:

$$LUX = B$$

Żeby znaleźć rozwiązanie tego równania potrzeba rozwiązać dwa równania z macierzami trójkątnymi:

$$LY = B \quad \text{oraz} \quad UX = Y$$

.
Podobnie można rozwiązać równanie $XA = B$.

$$\begin{gathered} XA = B \\ XA = B \quad |^T \\ (XA)^T = B^T \\ A^T X^T = B^T \\ U^T L^T X^T = B^T \end{gathered}$$

Znowu są dwa równania z macierzami trójkątnymi, ale tym razem:

$$U^T Y = B^T \quad \text{oraz} \quad L^T X^T = Y$$

.
Ostatecznie z X^T otrzymuje się szukane X.

2 Implementacja w Matlabie

Stworzyłem cztery funkcje:

1. *CroutLU*
2. *LowerTriangularSolve*
3. *UpperTriangularSolve*
4. *SolveMatrixEquation*,

Dokładniejszy opis kodu funkcji znajduje się w odpowiadających im plikach. W celu pokazania działania programu stworzyłem aplikację, która jest w pliku *CroutDecompositon.mlapp*. Mimo prób, nie udało mi się jej zamieścić w internecie.

2.1 Funkcja CroutLU

W funkcji zaimplementowany jest algorym, który opisywałem w sekcji 1.2. Przyjmuje ona macierz A, którą rozkłada na macierze L i U.

```
function [L, U] = CroutLU(A)
[nrow, ncol] = size(A);
%w L pierwsza kolumna jest jak w A
L(:, 1) = A(:, 1);
%w U na diagonali sa 1, a pierszy wiersz jest latwy do policzenia
U = diag(ones(1,nrow));
U(1,2:end) = A(1,2:end)./L(1,1);
%w petli licze najpierw i-ty wiersz w L, a z tego i-ty wiersz w U
for i = 2:nrow
    for j = 2:i
        L(i, j) = A(i, j) - L(i, 1:j - 1) * U(1:j - 1, j);
    end
    for j = i + 1:nrow
        U(i, j) = (A(i, j) - L(i, 1:i - 1) * U(1:i - 1, j)) / L(i, i);
    end
end
end
```

2.2 Funkcja LowerTriangularSolve

Funkcja jest implementacją rozwiązywania równania z macierzami $Ly = B$, gdzie L jest macierzą trójkątną dolną, a y jest szukany. Przyjmuje macierze L i B zwraca wektor y.

```
function [y] = LowerTriangularSolve(L,B)
[nrowB,ncolB] = size(B);
y = zeros(nrowB,ncolB);
for i = 1:nrowB
    y(i,:) = (B(i,:) - L(i,1:i)*y(1:i,:))./L(i,i);
end
end
```

2.3 Funkcja UpperTriangularSolve

Funkcja jest implementacją rozwiązywania równania z macierzami $Ux = B$, gdzie U jest macierzą trójkątną górną, a x jest szukany. Przyjmuje macierze U i B zwraca wektor x.

```
function x = UpperTriangularSolve(U,B)
[nrowB,ncolB] = size(B);
x = zeros(nrowB,ncolB);
for i = nrowB:(-1):1
    x(i,:) = (B(i,:) - U(i,(i + 1):nrowB)*x((i + 1):nrowB,:))/U(i, i);
end
end
```

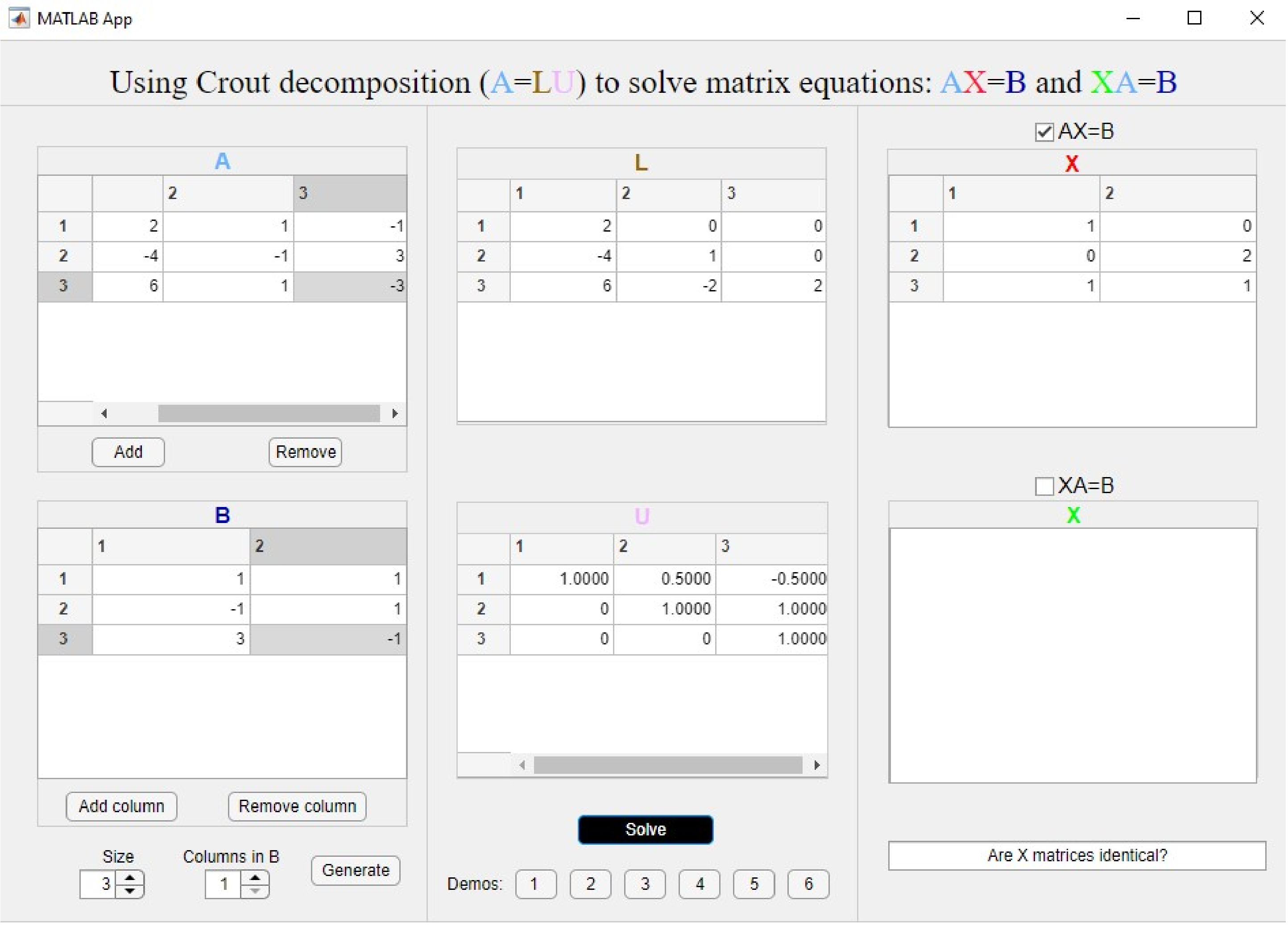
2.4 Funkcja SolveMatrixEquation

Celem funkcji jest rozwiązanie równania macierzy. Jako argumenty przyjmuje macierze A i B, a także wartość logiczną isAX, która określa, czy równanie jest postaci $AX = B$, czy $XA = B$. Zwraca szukany x. Funkcja używa poprzednio zdefiniowanych funkcji `LowerTriangularSolve` i `UpperTriangularSolve` i wykorzystuje je do policzenia równań odpowiednio $LY = B$ i $L^T X^T = Y$, oraz $UX = Y$ i $U^T Y = B^T$

```
function [x] = SolveMatrixEquation(A,B, isAX)
if nargin < 3 %jesli nie poda sie isAX to liczy AX=B
    isAX = True;
end
if isAX == false %jesli XA=B to licze dla A'X'=B' (obustronna transpozycja)
    A = A';
    B = B';
end
[L,U] = CroutLU(A);
y = LowerTriangularSolve(L,B);
x = UpperTriangularSolve(U,y);
%transpozycja x przy XA=B
if isAX==false
    x = x';
end
end
```

2.5 Aplikacja CroutDecomposition

W aplikacji można wpisać macierze $A \in \mathbb{R}^{n\times n}$ i $B \in \mathbb{R}^{n\times m}$ dla dowolnych $n, m \in \mathbb{N}$, $m \leqslant n$. Wielkość tych macierzy można zmieniać używając przycisków, które są pod nimi. Macierze przedstawione są za pomocą tabelek. Można zaznaczyć, aby aplikacja zwracała wynik dla równania $AX = B$ i/lub $XA = B$. W przypadku, gdy obie te opcje są zaznaczone otrzymuje się w polu tekstowym w prawym dolnym rogu informację o tym czy wynikowe macierze X są identyczne. Wynik otrzymuje się po naciśnięciu czarnego przycisku *Solve*. Oprócz wyniku pokazane jest również jak wyglądają macierze L i U z rozkładu Crouta. W aplikacji jest również wgrane 6 przykładów, które są opisane w następnej sekcji raportu. Znajdują się pod przyciskami *Demos*. Można wygenerować macierze $A \in \mathbb{Z}^{n\times n}$ i $B \in \mathbb{Z}^{n\times m}$. n wybiera się w spinnerze *Size*, a m w *Columns in B*. Wszystkie macierze wyświetlane w aplikacji są podpisane, a także zgadzają się kolorem tekstu z opisem na górze okna.



Przykładowe działanie aplikacji.

3 Przykłady

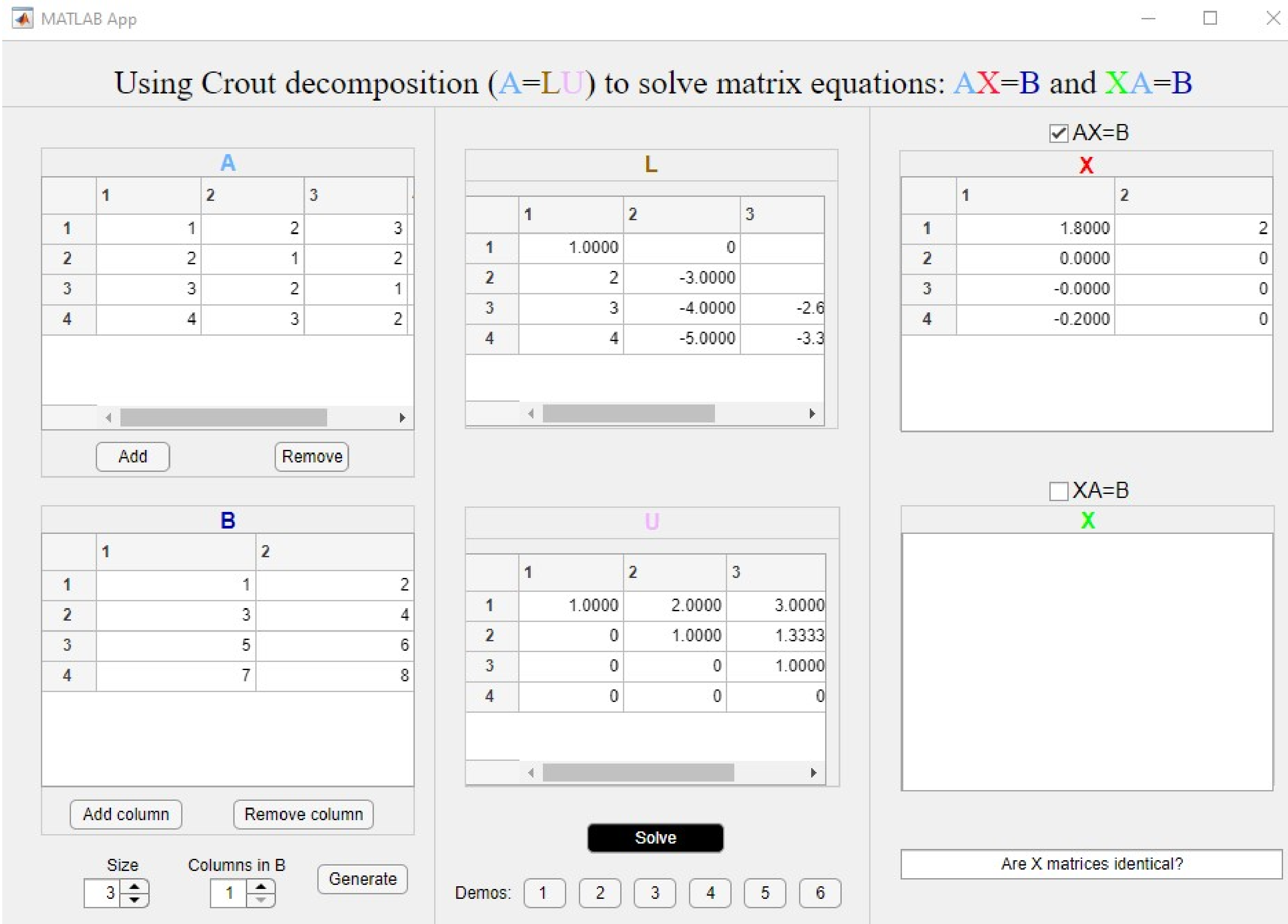
W tej sekcji pokaże działanie programu w kilku przykładach, a także porównam moje wyniki z wynikami otrzymanymi z wbudowanych w Matlabie funkcji. Oczywiście wybieram takie macierze A, które mają rozkład LU, oraz odpowiednie B.

3.1 Przykład 1

Podstawowe działanie programu. Wybieram macierze:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 2 & 3 \\ 3 & 2 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix} \quad B = \begin{bmatrix} -1 & 2 \\ 3 & -4 \\ -5 & 6 \\ 7 & -8 \end{bmatrix},$$

i liczę dla nich X w równaniu $AX = B$.



Wynik otrzymany przeze mnie to:

$$X = \begin{bmatrix} 1.8000 & 2.0000 \\ 0.0000 & 0 \\ -0.0000 & 0 \\ -0.2000 & 0 \end{bmatrix}.$$

Wynik otrzymany z funkcji `linsolve` w Matlabie jest taki sam.

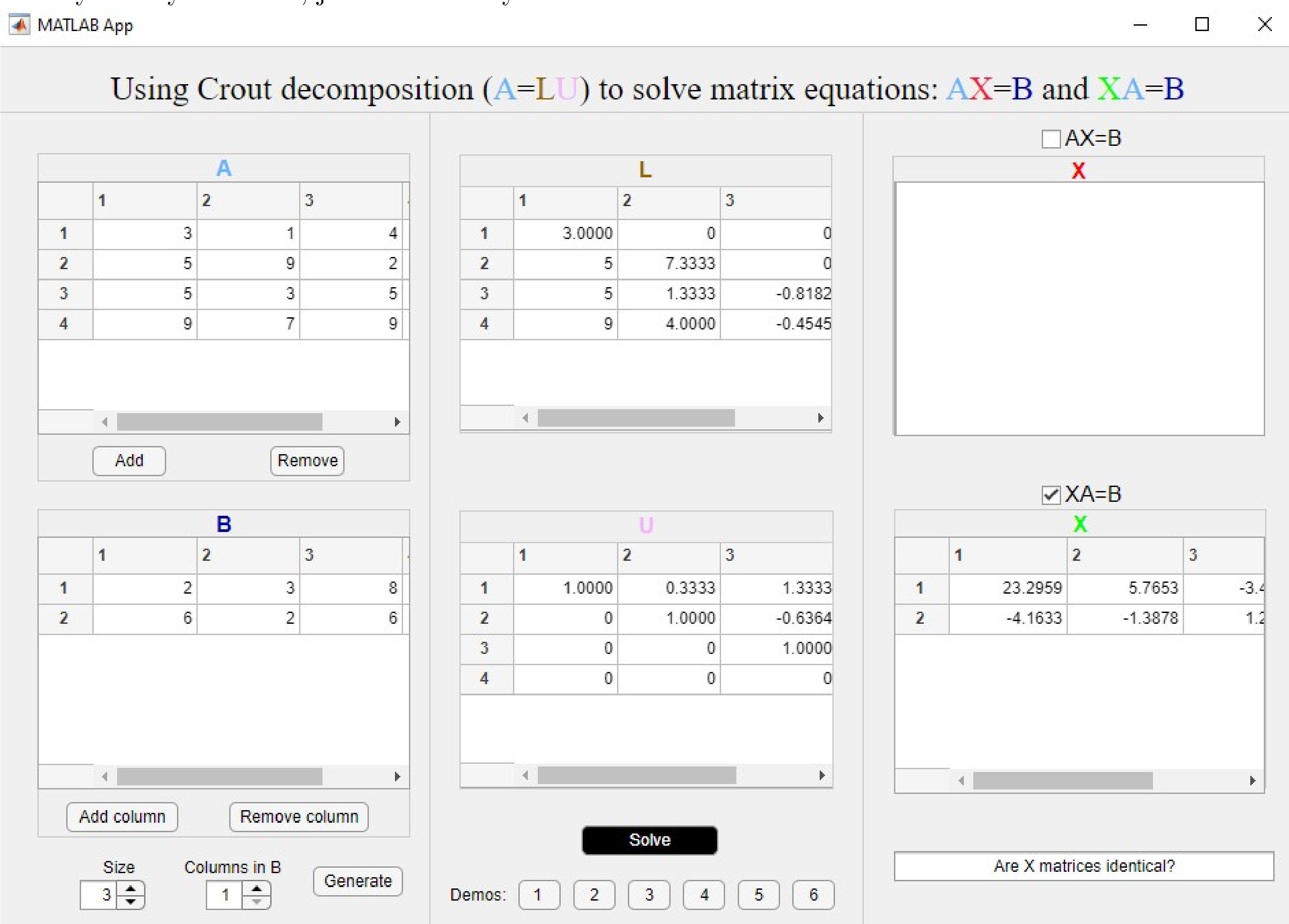
```
>> linsolve([1 2 3 4; 2 1 2 3 ; 3 2 1 2 ; 4 3 2 1],[1 2; 3 4 ; 5 6 ; 7 8])

ans =

    1.8000    2.0000
    0.0000         0
         0         0
   -0.2000         0
```

3.2 Przykład 2

Wykonuję działanie $XA = B$ dla macierzy:

$$A = \begin{bmatrix} 3. & 1 & 4 & 1 \\ 5 & 9 & 2 & 6 \\ 5 & 3 & 5 & 8 \\ 9 & 7 & 9 & 3 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 3 & 8 & 4 \\ 6 & 2 & 6 & 4 \end{bmatrix},$$

rozmiar macierzy B wykracza poza zakres zadania, ponieważ ma mniej wierszy niż macierz A. Nie stanowi to tu jednak problemu. Dla działania $XA = B$, ważne jest, aby liczba kolumn B zgadzała się z liczbą wierszy A. Wynikowe X, jest takich samych rozmiarów co B.



Wynik otrzymany przeze mnie:

$$X = \begin{bmatrix} 23.2959 & 5.7653 & -3.4184 & -8.8469 \\ -4.1633 & -1.3878 & 1.2653 & 2.1224 \end{bmatrix}.$$

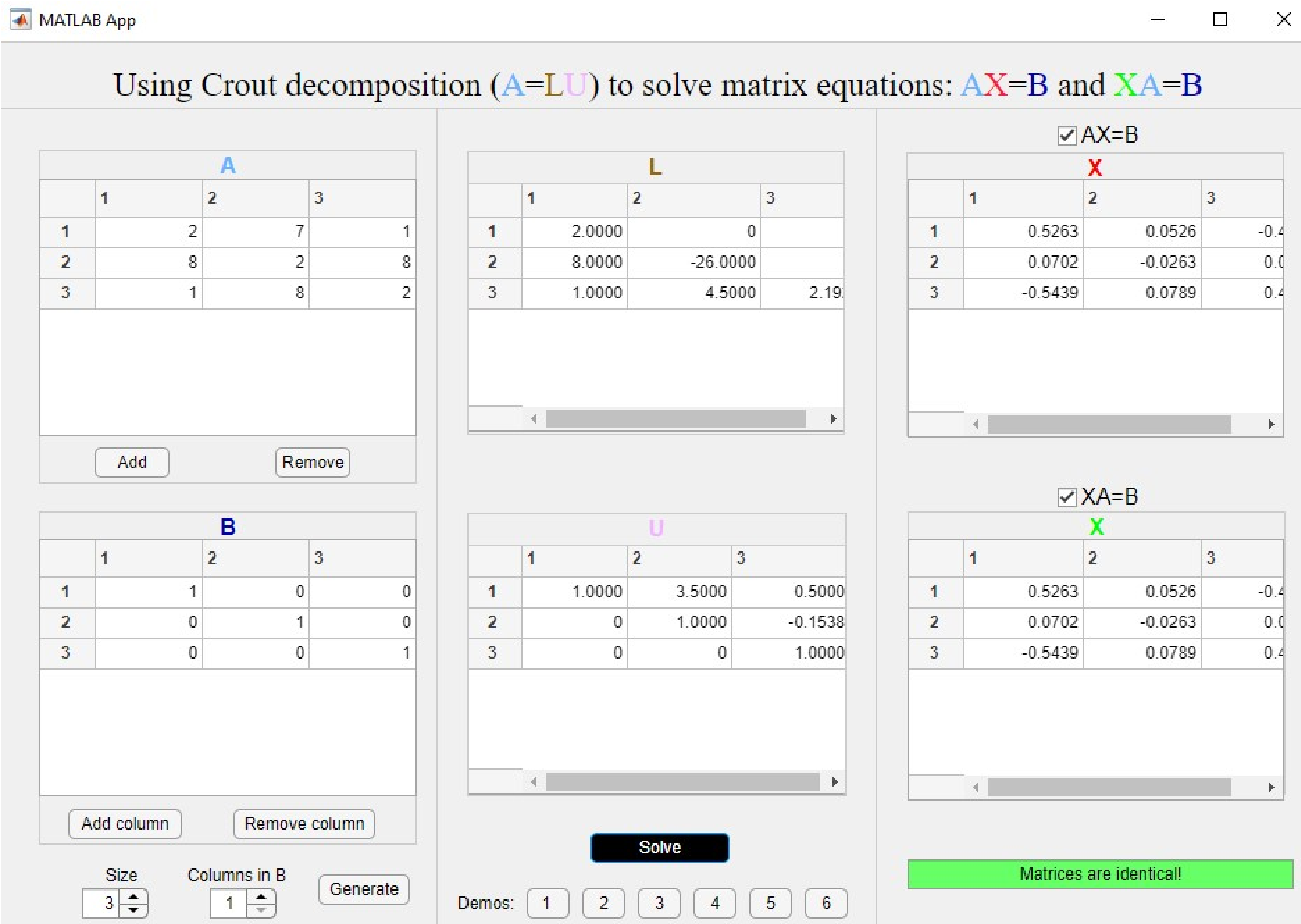
Wynik otrzymany z funkcji `mrdivide` jest taki sam:

```
>> mrdivide([2 3 8 4 ; 6 2 6 4],[3 1 4 1; 5 9 2 6 ; 5 3 5 8 ; 9 7 9 3])

ans =

   23.2959    5.7653   -3.4184   -8.8469
   -4.1633   -1.3878    1.2653    2.1224
```

3.3 Przykład 3

Biore dowolną odwracalną macierz A, która ma rozkład LU, a $B = I$.

$$A = \begin{bmatrix} 2. & 7 & 1 \\ 8 & 2 & 8 \\ 1 & 8 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$



Wynik jest spodziewany, czyli oba otzymane X są równe:

$$X = \begin{bmatrix} 0.5263 & 0.0526 & -0.4737 \\ 0.0702 & -0.0263 & 0.0702 \\ -0.5439 & 0.0789 & 0.4561 \end{bmatrix}$$

i zgadzają się z wynikami z Matlaba.

```
>> linsolve([2 7 1; 8 2 8 ; 1 8 2],diag(ones(1,3)))

ans =

    0.5263    0.0526   -0.4737
    0.0702   -0.0263    0.0702
   -0.5439    0.0789    0.4561

>> mrdivide(diag(ones(1,3)),[2 7 1; 8 2 8 ; 1 8 2])

ans =

    0.5263    0.0526   -0.4737
    0.0702   -0.0263    0.0702
   -0.5439    0.0789    0.4561
```

Sprawdziłem wyniki wystarczająco dużo razy, aby przekonać się, że program liczy rozwiązania dobrze. Nie będę zamieszczał więcej potwierdzeń z funkcji z Matlaba.

3.4 Przykład 4

Dla macierzy

$$A = \begin{bmatrix} 1 & 1 \\ -1 & 2 \end{bmatrix}$$

znajdę takie X i B, aby X w równaniach $AX = B$ i $XA = B$ były takie same. W tym celu liczę $AX = XA$. Wykonuje mnożenia:

$$AX = \begin{bmatrix} 1 & 1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} = \begin{bmatrix} x_{11} + x_{21} & x_{12} + x_{22} \\ -x_{11} + 2x_{21} & -x_{12} + 2x_{22} \end{bmatrix}$$

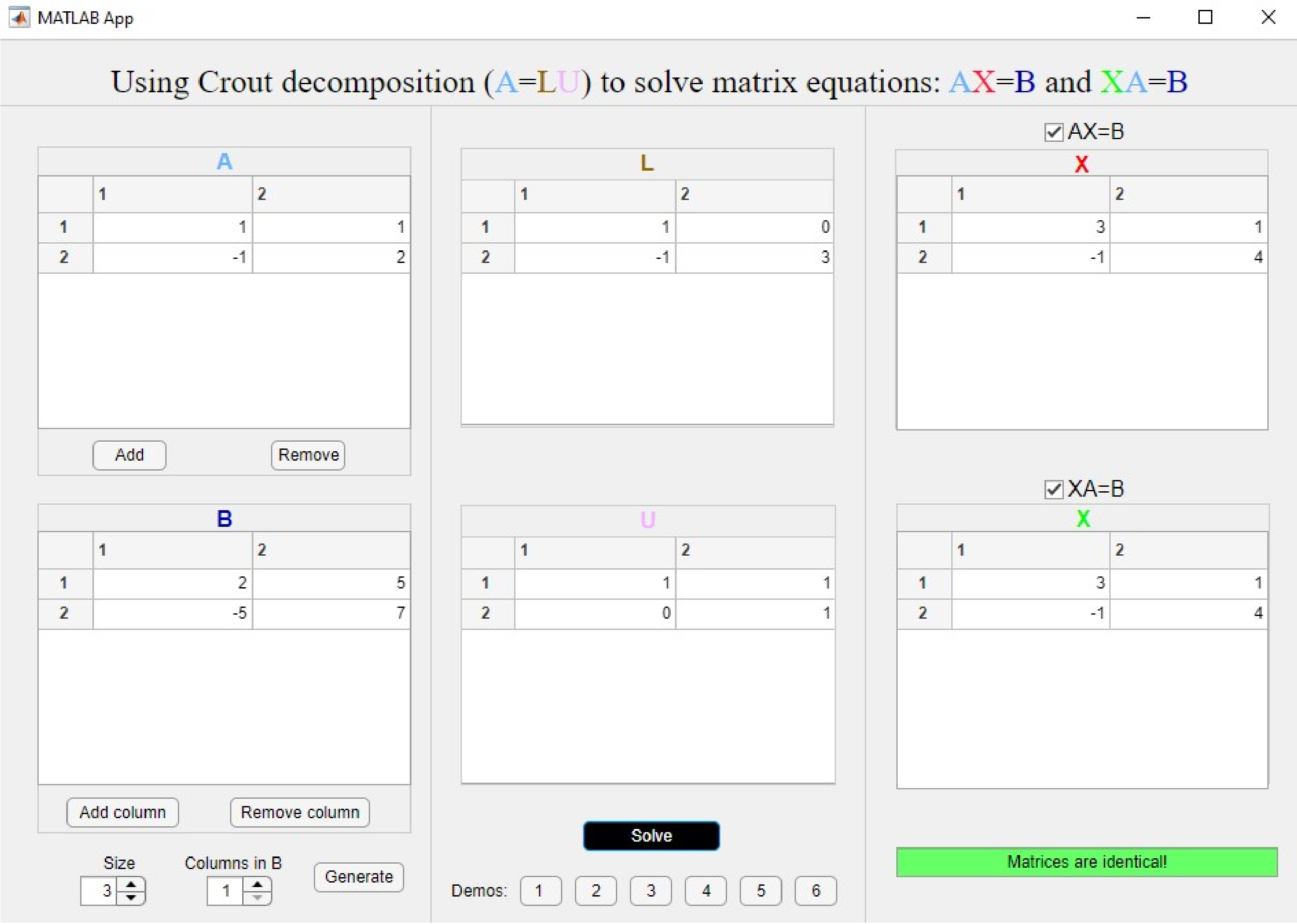
$$XA = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} x_{11} - x_{12} & x_{11} + 2x_{12} \\ x_{21} - x_{22} & x_{21} + 2x_{22} \end{bmatrix}$$

i porównując macierze otrzymuję układ równań:

$$\begin{cases} x_{11} + x_{21} = x_{11} - x_{12} \\ x_{12} + x_{22} = x_{11} + 2x_{12} \\ -x_{11} + 2x_{21} = x_{21} - x_{22} \\ -x_{12} + 2x_{22} = x_{21} + 2x_{22} \end{cases} \Rightarrow \begin{cases} x_{12} = -x_{21} \\ x_{12} = x_{22} - x_{11} \end{cases}$$

Teraz wybieram liczby spełniające te równania, czyli X może wyglądać na przykład tak:

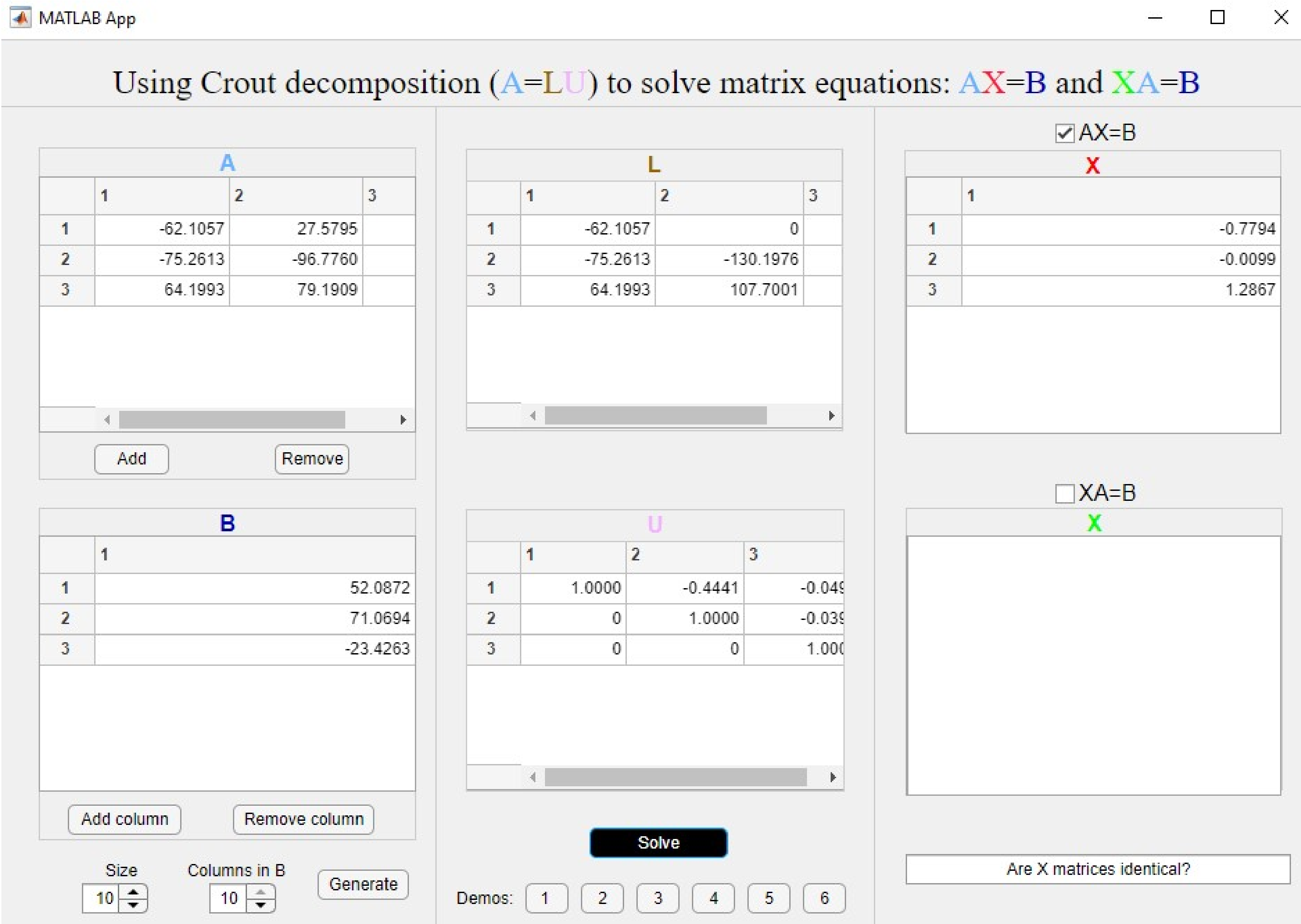
$$X = \begin{bmatrix} 3 & 1 \\ -1 & 4 \end{bmatrix}, \text{więc można policzyć, że } B = \begin{bmatrix} 2 & 5 \\ -5 & 7 \end{bmatrix}$$



3.5 Przykład 5

Ponieważ, A i B były w przykładach do tej pory zawsze macierzami o elemenatach całkowitych, teraz wybiore te elementy dowolnie z przedziału $[-100, 100]$. Wygenerowane losowo macierze to:

$$A = \begin{bmatrix} -62.1057 & 27.5795 & 3.0750 \\ -75.2613 & -96.7760 & 8.9044 \\ 64.1993 & 79.1909 & 21.2884 \end{bmatrix} \quad B = \begin{bmatrix} 52.0872 \\ 71.0694 \\ -23.4263 \end{bmatrix},$$



Wynik:

$$X = \begin{bmatrix} -0.7794 \\ -0.0099 \\ 1.2867 \end{bmatrix}$$

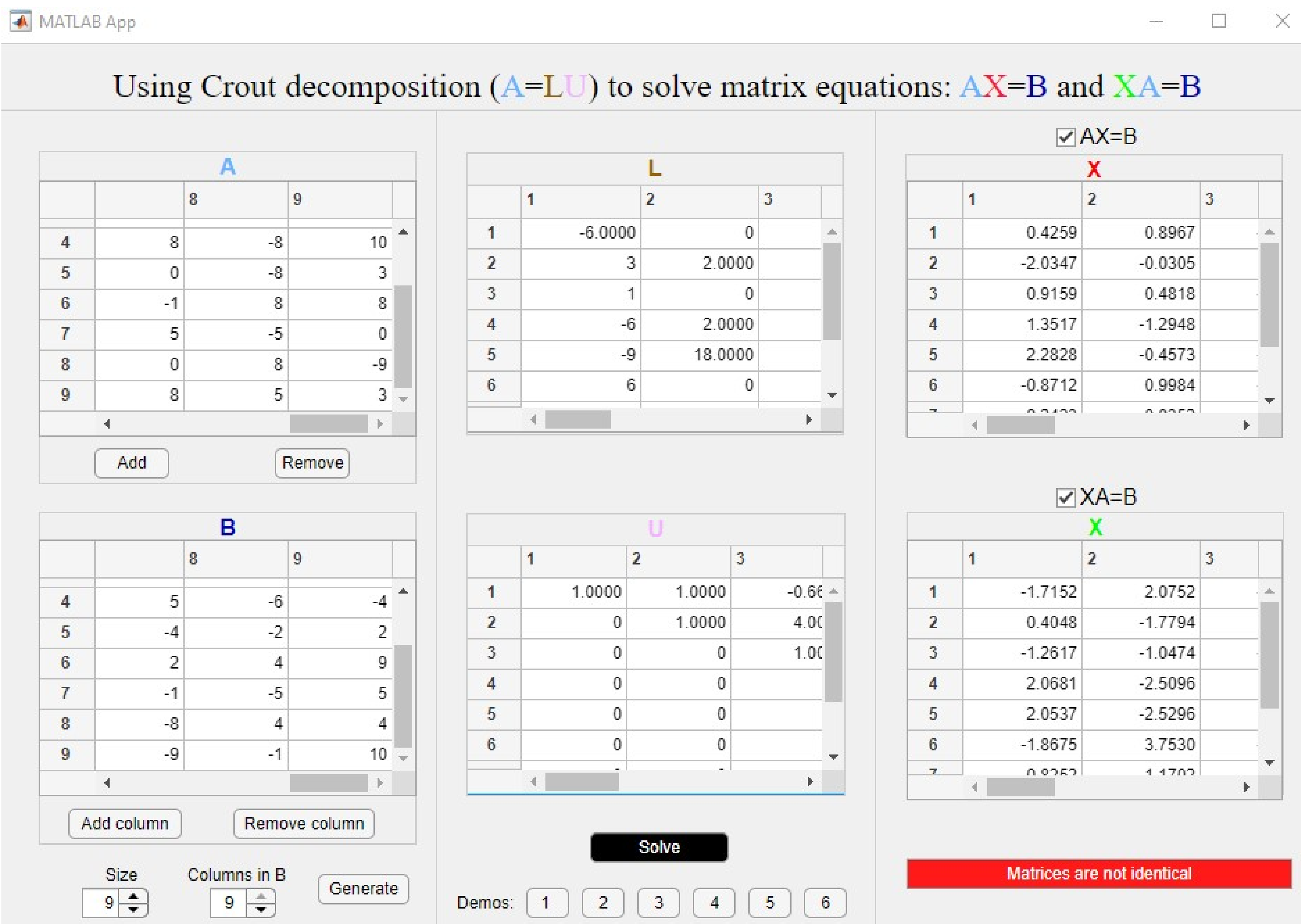
Sprawdzenie dla pewności:

```
>> linsolve([ -62.1057   27.5795    3.0750;
             -75.2613  -96.7760    8.9044;
             64.1993   79.1909   21.2884], ...
             [  52.0872;   71.0694;  -23.4263])

ans =

   -0.7794
   -0.0099
    1.2867
```

3.6 Przykład 6

Ostatni przykład jest z macierzy stworzonych przy użyciu zaimplementowanego w aplikacji generatora.

$$
A = \begin{bmatrix}
-6 & -6 & 4 & -2 & -4 & -7 & -4 & 0 & 9 \\
3 & 5 & 6 & 2 & -9 & 1 & -1 & 0 & -7 \\
1 & 1 & 1 & 8 & -1 & 8 & 2 & -5 & -2 \\
-6 & -4 & -1 & 7 & -3 & -2 & 8 & -8 & 10 \\
-9 & 9 & 3 & 8 & 1 & 4 & 0 & -8 & 3 \\
6 & 6 & 8 & 9 & 9 & 3 & -1 & 8 & 8 \\
6 & 8 & 4 & 7 & -4 & 0 & 5 & -5 & 0 \\
-1 & -8 & 1 & -9 & -3 & 0 & 0 & 8 & -9 \\
-8 & -6 & -4 & -9 & 8 & 9 & 8 & 5 & 3
\end{bmatrix}
\quad
B = \begin{bmatrix}
-8 & 2 & -8 & 3 & -4 & 1 & -3 & -4 & -6 \\
-9 & 2 & 10 & 6 & -4 & -7 & -5 & -3 & -6 \\
8 & 4 & -9 & 6 & 8 & -6 & 6 & -4 & 3 \\
-5 & 3 & 7 & -9 & -1 & 5 & 5 & -6 & -4 \\
-9 & -1 & 7 & -4 & 7 & 7 & -4 & -2 & 2 \\
7 & -7 & -9 & 4 & -8 & -9 & 2 & 4 & 9 \\
-7 & 6 & 1 & 0 & 8 & 6 & -1 & -5 & 5 \\
8 & -5 & 9 & 10 & -9 & 10 & -8 & 4 & 4 \\
-8 & 4 & -3 & 5 & 8 & -3 & -9 & -1 & 1
\end{bmatrix},
$$



Tym razewm wyniki nie zgadzają się i widać to już po elementach, które mieszczą się w okienkach X. W rzeczywistości trudno jest znaleźć takie macierze A i X większe od 2×2, aby $AX = XA$. Wynika to z obliczeń podobnych do tych w 3.4.

4 Podsumowanie

Dzięki rozkładowi Crouta, można szybko policzyć wynik dla równania macierzowego. Mając na uwadze pokazane przeze mnie przykłady widać, że rozwiązania są poprawne. Nie znalazłem błędów do conajmniej czterech cyfr znaczących.