Rozkład Crouta

Mateusz Kwiatkowski

1 Problem

Wyznaczanie rozkładu Crouta macierzy $A \in \Re^{n\times n}$. Wykorzystanie tego rozkładu do rozwiązania równań macierzowych $AX = B$ oraz $XA = B$, gdzie $B \in \Re^{n\times m}$. Wykonanie testów dla różnych macierzy B, m.in. dla takich, dla których $AX = XA$ (np. $B = I$). Porównanie wyników.

2 Wstęp

Celem projektu było zaimplementowanie rozkładu Crouta oraz pokazanie jego zastosowań. Pierwsza część została przeze mnie zrealizowana w funkcji Crout(A), która przyjmuje macierz kwadratową, a zwraca macierze L (trójkątna dolna) i U (trójkątna górna z samymi jedynkami na diagonali), które spełniają równość $LU = A$. W celu rozwiązywania równań napisałem dwie funkcje. Pierwsza z nich to X_lewo(A,B). Wylicza ona $X \in \Re^{m\times n}$ z równania $XA = B$ dla danych $A \in \Re^{n\times n}$ oraz $B \in \Re^{m\times n}$. Druga funkcja to X_prawo(A,B), która działa w sposób analogiczny do pierwszej i rozwiązuje $X \in \Re^{n\times m}$ z równania $AX = B$ dla danych $A \in \Re^{n\times n}$ oraz $B \in \Re^{n\times m}$. By przetestować działanie zaimplementowanych przeze mnie metod, napisałem funkcje Testy(), przeprowadzającą testy na losowych macierzach dla podanych wymiarów, oraz Crout_Blad(), która oblicza błąd rozwiązywania równwań przy pomocy rozkładu Crouta dla podanych macierzy $A \in \Re^{n\times n}$ i B.

3 Rozkład Crouta

3.1 Omówienie metody

Rozkład Crouta jest to rozkład macierzy kwadratowej $A \in \Re^{n\times n}$ na macierze $L, U \in \Re^{n\times n}$, takie że L jest macierzą dolną trójkątną, U macierzą górną trójkątną z samymi jednykami na diagonali oraz zachodzi równość $LU = A$. Znając te własności macierzy jesteśmy w stanie wyprowadzić algorytm na obliczenie L i U.

$$\begin{bmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ l_{31} & l_{32} & l_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & 1 & u_{23} & \dots & u_{2n} \\ 0 & 0 & 1 & \dots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}$$

Po przemnożeniu macierzy po lewej stronie równości otrzymujemy następującą postać:

$$\begin{bmatrix} l_{11} & l_{11}u_{12} & l_{11}u_{13} & \dots & l_{11}u_{1n} \\ l_{21} & l_{21}u_{12} + l_{22} & l_{21}u_{13} + l_{22}u_{23} & \dots & l_{21}u_{15} + l_{22}u_{25} \\ l_{31} & l_{31}u_{12} + l_{32} & l_{31}u_{13} + l_{32}u_{23} + l_{33} & \dots & l_{31}u_{15} + l_{32}u_{25} + l_{33}u_{35} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n1}u_{12} + l_{n2} & l_{n1}u_{13} + l_{n2}u_{23} + l_{n3} & \dots & \sum_{i=1}^{n-1} l_{ni}u_{in} + l_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}$$

Widzimy zatem, że zachodzi następujący wzór: $a_{ij} = \sum_{k=1}^{min(i,j)} l_{ik}u_{kj}$
To wystarcza do skonstruowania algorytmu:
$L = zeros(size(A))$;
$U = eye(size(A))$;
for $i = 1, 2, ..., n$
    for $m = 1, 2, ..., i$
        $l_{im} = a_{im} - L(i, 1 : (m-1))U(1 : (m-1), m)$;
    end
    if $l_{ii} \neq 0$
        $U(i, (i+1) : n) = (A(i, (i+1) : n) - L(i, 1 : (i-1))U(1 : (i-1), (i+1) : n))/l_{ii}$;
    end
end
Powyższy algorytm jest już sfaktoryzowany (użyłem do tego matlabowej składni). Jak widać, nie ma potrzeby obliczania osobno wartości z U, ponieważ można działać na całych wierszach. Inaczej jest w przypadku macierzy L, gdyż do obliczania kolejnych wartości są potrzebne poprzednie wartości z tego samego wiersza. Największą wadą tej metody jest jej kwadratowa złożoność, jednak nie da się jej zoptymalizować.

3.2 Przykłady

Na początek pokażę działanie funkcji na prostym przykładzie. Niech A=$\begin{bmatrix} 24 & 57 & 23 \\ 12 & 38 & 79 \\ 40 & 28 & 47 \end{bmatrix}$.
Program postępując zgodnie z powyższym algorytmem wykona następujące obliczenia:

$$l_{11} = a_{11} = 24$$
$$u_{12} = \frac{a_{12}}{l_{11}} = \frac{57}{24},\ u_{13} = \frac{a_{13}}{l_{11}} = \frac{23}{24}$$
$$l_{21} = a_{21} = 12,\ l_{22} = a_{22} - l_{21}u_{12} = 9,5$$
$$u_{23} = \frac{a_{23}}{l_{22}} \approx 7,1053$$
$$l_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23} \approx 484,7193$$

Po wywołaniu Crout(A) otrzymamy zatem następujący wynik:

$$L = \begin{bmatrix} 24.0000 & 0 & 0 \\ 12.0000 & 9.5000 & 0 \\ 40.0000 & -67.0000 & 484.7193 \end{bmatrix}, U = \begin{bmatrix} 1.0000 & 2.3750 & 0.9583 \\ 0 & 1.0000 & 7.1053 \\ 0 & 0 & 1.0000 \end{bmatrix}$$

Po sprawdzeniu otrzymujemy, że rzeczywiście zachodzi równość $LU = A$.
Teraz rozważmy macierz A=$\begin{bmatrix} 3 & 0 & 0 \\ 7 & 9 & 0 \\ 2 & 0 & 5 \end{bmatrix}$. Jest to macierz trójątna górna, zatem od razu widać, że rozwiązaniem spełniającym warunki zadania jest $L = A, U = I$. Dokładnie taki wynik otrzymamy po wywołaniu funkcji Crout(A).
Zdecydowanie ciekawszy rezultat otrzymamy dla A=$\begin{bmatrix} 65 & 79 & 20 \\ 0 & 43 & 72 \\ 0 & 0 & 2 \end{bmatrix}$, która jest macierzą trójkątną górną. Gdyby nie założenie, że U musi mieć jedynki na diagonali to rozwiązaniem byłoby $L = I, U = A$. Takie macierze spełniają warunki rozkładu Doolittle'a, jednak nie spełniają warunków rozkładu Crouta. Po wywołaniu funkcji Crout(A) dostaniemy wyniki:

$$L = \begin{bmatrix} 65 & 0 & 0 \\ 0 & 43 & 0 \\ 0 & 0 & 2 \end{bmatrix}, U = \begin{bmatrix} 1.0000 & 1.2154 & 0.3077 \\ 0 & 1.0000 & 1.6744 \\ 0 & 0 & 1.0000 \end{bmatrix}$$

Jak można zauważyć, L jest macierzą diagonalną (na tej diagonali są wartości z diagonali A), a i-ty wiersz macierzy U (dla $i \in \{1,2,3\}$) to i-ty wiersz A podzielony przez a_{ii}. Analogiczne wyniki otrzymamy dla innych macierzy A trójkątnych górnych.

3.3 Możliwe błędy

Po omówieniu działania programu trzeba jeszcze wspomnieć o przypadkach, w których zwróci on błąd. Podstawowym założeniem jest to, że macierz A musi być kwaderatowa. Jeśli jednak użytkownik wpisze macierz o innych wymiarach to wyświetlony zostanie komunikat: Macierz nie jest kwadratowa. Może również zdarzyć się tak, że macierz A nie będzie miała rozkładu LU. Dzieje się tak, gdy $a_{11} = 0$ oraz któryś wyraz w pierwszym wierszu A jest niezerowy. Przykładem takiej macierzy jest $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$. Mój program rozpoznaje takie sytuacje i zamiast obliczać rozkład, wyświetli komunikat: A nie ma rozkładu LU.

4 Rozwiązywanie równań macierzowych

4.1 Równania $AX = B$

Najbardziej powszechnym zastosowaniem rozkładu Crouta jest wykorzystanie go do rozwiązywania równań macierzowych. Niech $A \in \Re^{n\times n}$ oraz $B \in \Re^{n\times m}$. Korzystając z rozkładu Crouta $LU = A$ jesteśmy w stanie wyliczyć $X \in \Re^{n\times m}$ z równania $AX = B$. Po podstawieniu otrzymujemy $LUX = B$. To równanie rozwiążemy dwuetapowo. Na początek należy znaleźć rozwiązanie $C \in \Re^{n\times m}$ równania $LC = B$, a następnie wykorzystać tę macierz do wyliczenia $X \in \Re^{n\times m}$ z równości $UX = C$.

$$\begin{bmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ l_{31} & l_{32} & l_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{bmatrix} \begin{bmatrix} c_{11} & c_{12} & c_{13} & \dots & c_{1m} \\ c_{21} & c_{22} & c_{23} & \dots & c_{2m} \\ c_{31} & c_{32} & c_{33} & \dots & c_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & c_{n3} & \dots & c_{nm} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{1m} \\ b_{21} & b_{22} & b_{23} & \dots & b_{2m} \\ b_{31} & b_{32} & b_{33} & \dots & b_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & b_{n3} & \dots & b_{nm} \end{bmatrix}$$

Po wykonaniu mnożenia otrzymamy następującą postać:

$$\begin{bmatrix} l_{11}c_{11} & l_{11}c_{12} & l_{11}c_{13} & \dots & l_{11}c_{1m} \\ l_{21}c_{11} + l_{22}c_{21} & l_{21}c_{12} + l_{22}c_{22} & l_{21}c_{13} + l_{22}c_{23} & \dots & l_{21}c_{1m} + l_{22}c_{2m} \\ \sum_{i=1}^{3} l_{3i}c_{i1} & \sum_{i=1}^{3} l_{3i}c_{i2} & \sum_{i=1}^{3} l_{3i}c_{i3} & \dots & \sum_{i=1}^{3} l_{3i}c_{im} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} l_{ni}c_{i1} & \sum_{i=1}^{n} l_{ni}c_{i2} & \sum_{i=1}^{n} l_{ni}c_{i3} & \dots & \sum_{i=1}^{n} l_{ni}c_{im} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{1m} \\ b_{21} & b_{22} & b_{23} & \dots & b_{2m} \\ b_{31} & b_{32} & b_{33} & \dots & b_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & b_{n3} & \dots & b_{nm} \end{bmatrix}$$

Otrzymaliśmy zatem następujący wzór: $b_{ij} = \sum_{k=1}^{i} l_{ik}c_{kj}$

To po przekształceniu daje nam: $c_{ij} = \frac{b_{ij} - \sum_{k=1}^{i-1} l_{ik}c_{kj}}{l_{ii}}$, dla $l_{ii} \neq 0$

Używając tego wzoru można napisać sfaktoryzowany algorytm tworzący macierz C:

C=zeros(n,m);
for $i = 1, 2, ..., n$
    if $l_{ii} \neq 0$
        C(i,:)=(B(i,:)-L(i,1:(i-1))C(1:(i-1),:))/l_{ii};
    end
end

Po obliczeniu macierzy C możemy przejść do rozwiązywania równania $UX = C$.

$$\begin{bmatrix} 1 & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & 1 & u_{23} & \dots & u_{2n} \\ 0 & 0 & 1 & \dots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1m} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2m} \\ x_{31} & x_{32} & x_{33} & \dots & x_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{nm} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & \dots & c_{1m} \\ c_{21} & c_{22} & c_{23} & \dots & c_{2m} \\ c_{31} & c_{32} & c_{33} & \dots & c_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & c_{n3} & \dots & c_{nm} \end{bmatrix}$$

$$\begin{bmatrix} x_{11}+\sum_{k=2}^{n} u_{1k}x_{k1} & x_{12}+\sum_{k=2}^{n} u_{1k}x_{k2} & x_{13}+\sum_{k=2}^{n} u_{1k}x_{k3} & \dots & x_{1m}+\sum_{k=2}^{n} u_{1k}x_{k3} \\ x_{21}+\sum_{k=3}^{n} u_{2k}x_{k1} & x_{22}+\sum_{k=3}^{n} u_{2k}x_{k2} & x_{23}+\sum_{k=3}^{n} u_{2k}x_{k3} & \dots & x_{2m}+\sum_{k=3}^{n} u_{2k}x_{km} \\ x_{31}+\sum_{k=4}^{n} u_{3k}x_{k1} & x_{32}+\sum_{k=4}^{n} u_{3k}x_{k2} & x_{33}+\sum_{k=4}^{n} u_{3k}x_{k3} & \dots & x_{3m}+\sum_{k=4}^{n} u_{3k}x_{km} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{nm} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & \dots & c_{1m} \\ c_{21} & c_{22} & c_{23} & \dots & c_{2m} \\ c_{31} & c_{32} & c_{33} & \dots & c_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & c_{n3} & \dots & c_{nm} \end{bmatrix}$$

Z tego otrzymujemy wzór: $x_{ij} = c_{ij} - \sum_{k=i+1}^{n} u_{ik}x_{kj}$
Korzystając z tego wzoru jesteśmy w stanie sformułować algorytm obliczający kolejne wiersze macierzy X, zaczynając od wiersza n-tego, a kończąc na wierszu pierwszym:

```
X=zeros(n,m);
for i = n, n − 1, ..., 1
    X(i,:)=C(i,:)-U(i,(i+1):n)X((i+1):n,:);
end
```

Dokładnie takie operacje wykonuje mój program X_prawo(A,B) rozwiązujący równania typu $AX = B$.

4.2 Równania $XA = B$

Korzystając z rozkładu Crouta $LU = A$ możemy również rozwiązać równanie $XA = B$, gdzie $A \in \Re^{n\times n}$ oraz $B, X \in \Re^{m\times n}$. Wiele kroków będzie analogicznych do poprzedniego podpunktu, dlatego pozwolę sobie na skrótowe potraktowanie niektórych obliczeń. Na początku zajmę się rozwiązaniem równania $CU = B$, gdzie $C \in \Re^{m\times n}$.

$$\begin{bmatrix} c_{11} & c_{12}+c_{11}u_{12} & c_{13}+\sum_{k=1}^{2} c_{1k}u_{k3} & \dots & c_{1n}+\sum_{k=1}^{n-1} c_{1k}u_{kn} \\ c_{21} & c_{22}+c_{21}u_{12} & c_{23}+\sum_{k=1}^{2} c_{2k}u_{k3} & \dots & c_{2n}+\sum_{k=1}^{n-1} c_{2k}u_{kn} \\ c_{31} & c_{32}+c_{31}u_{12} & c_{33}+\sum_{k=1}^{2} c_{3k}u_{k3} & \dots & c_{3n}+\sum_{k=1}^{n-1} c_{3k}u_{kn} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2}+c_{m1}u_{12} & c_{m3}+\sum_{k=1}^{2} c_{mk}u_{k3} & \dots & c_{mn}+\sum_{k=1}^{n-1} c_{mk}u_{kn} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{1n} \\ b_{21} & b_{22} & b_{23} & \dots & b_{2n} \\ b_{31} & b_{32} & b_{33} & \dots & b_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & b_{m3} & \dots & b_{mn} \end{bmatrix}$$

Taka forma tego równania pozwala nam na stworzenie algorytmu obliczającego kolejne kolumny macierzy C:

```
C=zeros(m,n);
for i = 1, 2, ..., n
    C(:,i)=B(:,i)-C(:,1:(i-1))U(1:(i-1),i);
end
```

Mając macierz C możemy przejść do obliczenia X z równania $XL = C$.

$$\begin{bmatrix} x_{11}l_{11}+\sum_{k=2}^{n} x_{1k}l_{k1} & x_{12}l_{22}+\sum_{k=3}^{n} x_{1k}l_{k2} & x_{13}l_{33}+\sum_{k=4}^{n} x_{1k}l_{k3} & \dots & x_{1n}l_{nn} \\ x_{21}l_{11}+\sum_{k=2}^{n} x_{2k}l_{k1} & x_{22}l_{22}+\sum_{k=3}^{n} x_{2k}l_{k2} & x_{23}l_{33}+\sum_{k=4}^{n} x_{2k}l_{k3} & \dots & x_{2n}l_{nn} \\ x_{31}l_{11}+\sum_{k=2}^{n} x_{3k}l_{k1} & x_{32}l_{22}+\sum_{k=3}^{n} x_{3k}l_{k2} & x_{33}l_{33}+\sum_{k=4}^{n} x_{3k}l_{k3} & \dots & x_{3n}l_{nn} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1}l_{11}+\sum_{k=2}^{n} x_{mk}l_{k1} & x_{m2}l_{22}+\sum_{k=3}^{n} x_{mk}l_{k2} & x_{m3}l_{33}+\sum_{k=4}^{n} x_{mk}l_{k3} & \dots & x_{mn}l_{nn} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & \dots & c_{1n} \\ c_{21} & c_{22} & c_{23} & \dots & c_{2n} \\ c_{31} & c_{32} & c_{33} & \dots & c_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & c_{m3} & \dots & c_{mn} \end{bmatrix}$$

Analogicznie jak przy obliczaniu macierzy C, tworzymy kolejne kolumny macierzy X, jednak tym razem musimy zacząć od kolumny n-tej:

X=zeros(m,n);
for $i = n, n-1, ..., 1$
    if $l_{ii} \neq 0$
        X(:,i)=(C(:,i)-X(:,(i+1):n)L((i+1):n,i))/l_{ii};
    end
end
Ten algorytm został przeze mnie zaimplementowany w funkcji X_lewo(A,B).

4.3 Przykłady

Najprostszym zastosowaniem funkcji X_lewo() i X_prawo() jest wykorzystanie ich do rozwiązywania układów liniowych zapisanych w formie macierzowej. Rozwiążmy na przykład równanie AX=B, dla

$$A = \begin{bmatrix} 6 & 8 & 9 & 2 & 3 \\ 13 & 3 & 46 & 2 & 7 \\ 34 & 21 & 23 & 98 & 5 \\ 4 & 16 & 45 & 3 & 12 \\ 35 & 28 & 4 & 17 & 90 \end{bmatrix}, B = \begin{bmatrix} 3 \\ 17 \\ 8 \\ 90 \\ 5 \end{bmatrix}$$

Po wywołaniu funkcji X_prawo(A,B) otrzymujemy $X = [-4.9369, 1.7806, 1.4270, 1.0185, 1.1657]^T$. Sprawdzenie pokazuje, że rzeczywiście $AX = B$.
Teraz na przykładzie zaprezentuję działanie funkcji X_lewo(). Rozwiążę równanie $XA = B$ dla

$$A = \begin{bmatrix} 25 & 14 & 2 & 56 & 3 \\ 25 & 46 & 32 & 90 & 12 \\ 57 & 9 & 13 & 34 & 5 \\ 34 & 57 & 2 & 7 & 18 \\ 100 & 29 & 8 & 3 & 19 \end{bmatrix}, B = \begin{bmatrix} 34 & 2 & 90 & 7 & 8 \\ 13 & 68 & 29 & 78 & 55 \\ 89 & 4 & 24 & 8 & 1 \end{bmatrix}$$

Na początku program sprawdza, czy A jest kwadratowa oraz czy wymiary macierzy pozwalają na ich przemnożenie, a następnie stosuje algorytm wyznaczający rozkład Crouta:

$$L = \begin{bmatrix} 25 & 0 & 0 & 0 & 0 \\ 25 & 32 & 0 & 0 & 0 \\ 57 & -22.92 & 29.9275 & 0 & 0 \\ 34 & 37.96 & -36.3075 & -193.5994 & 0 \\ 100 & -27 & 25.3125 & -133.6757 & 4.5995 \end{bmatrix}, U = \begin{bmatrix} 1 & 0.56 & 0.08 & 2.24 & 0.12 \\ 0 & 1 & 0.9375 & 1.0625 & 0.2813 \\ 0 & 0 & 1 & -2.3165 & 0.1539 \\ 0 & 0 & 0 & 1 & -0.0456 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

W dalszej części program stosuje powyższe macierze do rozwiązania równania $XLU = B$. W pierwszej kolejności zostanie rozwiązane równanie pomocnicze $CU = B$. W wyniku przedstawionego wcześniej algorytmu na obliczanie C otrzymujemy:

$$C = \begin{bmatrix} 34 & -17.04 & 103.255 & 188.1367 & 1.4029 \\ 13 & 60.72 & -28.965 & -82.7329 & 37.0463 \\ 89 & -45.84 & 59.855 & -4 & -6.1825 \end{bmatrix}$$

Teraz wystarczy już tylko rozwiązać równanie $XL = C$. Po zastosowaniu algorytmu otrzymujemy ostateczny wynik:

$$X = \begin{bmatrix} -4.6461 & 2.3864 & 1.7577 & -1.1824 & 0.3050 \\ 2.4746 & 4.7499 & -14.0086 & -5.1340 & 8.0543 \\ -1.5092 & -0.6209 & 4.2879 & 0.9488 & -1.3441 \end{bmatrix}$$

Możemy sprawdzić, że rozwiązanie jest poprawne czyli $XA = B$.
Ciekawym przypadkiem jest równanie typu AX=I. Niech I będzie macierzą jednostkową 5x5 oraz

$$A = \begin{bmatrix} 3 & 5 & 7 & 9 & 11 \\ 12 & 7 & 20 & 9 & 15 \\ 13 & 17 & 5 & 6 & 17 \\ 1 & 2 & 6 & 4 & 3 \\ 0 & 8 & 0 & 1 & 20 \end{bmatrix}$$

Wiemy, że rozwiązaniem tego równania jest macierz odwrotna do A. Zatem przy użyciu funkcji X_prawo(A,I) orzymuję równość:

$$A^{-1} = X = \begin{bmatrix} 0.0800 & 0.0759 & 0.0161 & -0.3597 & -0.0606 \\ -0.1436 & -0.0882 & 0.0845 & 0.3911 & 0.0147 \\ -0.1248 & 0.0264 & -0.0137 & 0.2356 & 0.0251 \\ 0.2035 & -0.0426 & -0.0003 & -0.0952 & -0.0654 \\ 0.0472 & 0.0374 & -0.0338 & -0.1517 & 0.0474 \end{bmatrix}$$

Wiemy również, że $AA^{-1} = I = A^{-1}A$, a z tego wynika równość X_prawo(A,I)=A^{-1}=X_lewo(A,I).

4.4 Możliwe błędy

Najprostszym możliwym błędem w użyciu funkcji X_prawo() i X_lewo() jest podanie macierzy o niewłaściwych wymiarach (warunki sformułowłem podczas omawiania obu typów równań). Moje programy sprawdzają rozmiary macierzy, a gdy znajdą błąd wyświetlają komunikat: Wymiary macierzy się nie zgadzają.
Wiemy również, że niektóre równania macierzowe nie mają rozwiązań. Moje programy posiadają podstawowe funkcje sprawdzające, czy dane równanie nie jest sprzeczne. Na przykład w przypadku X_prawo() sprawdzam, czy istnieje wiersz macierzy A mający same zera, a odpowiadajcy mu wiersz B jest niezerowy (wtedy równanie jest sprzeczne, więc szukanie X jest bezsensowne). Przykładem takiego równania jest $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} X = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Po wpisaniu takich danych do funkcji X_prawo() zostanie zwrócony komunikat: Równanie nie ma rozwiązań.
To jednak nie wyczerpuje wszystkich możliwości. By stwierdzić rodzaj równania trzeba użyć innej metody, np. metody Gaussa.

5 Podsumowanie

Rozkład Crouta to dość prosty do zaimplementowania algorytm, który ma wiele praktycznych zastosowań. Poza rozwiązywaniem równań może być również użyty do obliczania wyznacznika i odwrotności macierzy. Wielką zaletą Matlaba jest możliwość wykonywania działań na wektorach, co jest bardzo pomocne przy obliczaniu tego rozkładu. Wyniki zwracane przez moje funkcje podawane są z bardzo dobrym przybliżeniem (co widać po zastosowaniu Crout_Blad()), a minimalne błędy rzędu 10^{-10} wynikają z zaokrągleń. Przy rozwiązywaniu równań należy pamiętać o sprawdzeniu wyniku. Algorytm ma złożoność kwadratową, taką jak większość algorytmów obliczających rozkłady macierzy, np. rozkład Doolittle'a.