Metody numeryczne
Projekt nr 2

Jan Skwarek
numer albumu: 305734

28 stycznia 2022

1 Treść zadania

Rozwiązywanie układu równań liniowych $AX = B$, gdzie $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times n}$, zmodyfikowaną metodą Crouta (tj. poprzez rozkład $A = UL$, gdzie U jest macierzą trójkątną górną z jedynkami na głównej przekątnej, a L macierzą trójkątną dolną). Wyznaczanie macierzy A^{-1} oraz $det(A)$ na podstawie rozkładu.
Porównać wyniki z otrzymanymi wbudowaną funkcją Matlaba `inv`.

2 Opis wykorzystywanych metod numerycznych

2.1 Zmodyfikowany rozkład Crouta

Standardowy rozkład Crouta to rozkład pewnej macierzy A na macierz trójkątną dolną L oraz trójkątną górną U taki, że $A = LU$ i macierz U ma jedynki na głównej przekątnej. W zadaniu użyłem nieco zmodyfikowanej wersji powyższego rozkładu. Rozkład, zamiast $A = LU$, ma postać $A = UL$. Algorytm ten wyznacza się następująco. Macierz A rozpisujemy jako iloczyn macierzy U i L:

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

Następnie rozpisujemy iloczyn znajdujący się po prawej stronie i porównujemy odpowiadające sobie elementy macierzy A i UL. Otrzymane zależności pozwoloną nam wyznaczyć elementy macierzy U i L. Należy pamiętać, że nie dla każdej macierzy istnieje rozkład UL. Aby rozkład ten wystąpił, macierz musi być oczywiście kwadratowa oraz żaden z jej wiodących minorów głównych, ale patrząc od prawego dolnego rogu, nie może być równy zeru.

2.2 Rozwiązywanie układów równań za pomocą rozkładu UL

Metoda ta jest analogiczna do rozwiązywania układów równań liniowych za pomocą rozkładu LU. Mamy dany układ równań liniowych $Ax = b$. Podstawiamy $A = UL$ i otrzymujemy:

$$ULx = b.$$

Następnie rozwiązanie takiego układu znajdujemy, rozwiązując dwa układy:

$$Uy = b \qquad \text{oraz} \qquad Lx = y.$$

.

2.3 Wyznaczanie macierzy odwrotnej za pomocą rozkładu UL

Metoda ta jest analogiczna do metody wyznaczania macierzy odwrotnej za pomocą rozkładu LU. Macierz odwrotna A^{-1} do macierzy A spełnia równanie:

$$AA^{-1} = I = A^{-1}A,$$

gdzie I oznacza macierz jednostkową. Jeżeli oznaczymy sobie $x = A^{-1}$, powyższe równanie możemy zapisać w następujący sposób:

$$Ax = I.$$

Podobnie jak w poprzedniej metodzie numerycznej, oznaczmy sobie $A = UL$. Otrzymamy wtedy:

$$ULx = I.$$

Następnie rozwiązanie takiego układu znajdujemy, rozwiązując dwa układy:

$$Uy = I \qquad \text{oraz} \qquad Lx = y.$$

Macierz odwrotną możemy również obliczyć w nieco inny sposób. Warto zauważyć, że oczywiście:

$$A^{-1} = (UL)^{-1} = L^{-1}U^{-1}.$$

2.4 Obliczanie wyznacznika macierzy za pomocą rozkładu UL

Zarówno macierz U, jak i macierz L, to macierze trójkątne. Obliczanie wyznacznika takiej macierzy sprowadza się do wymnożenia elementów leżących na głównej przekątnej. Z własności wyznacznika (przyjmujemy, że jakaś macierz A została rozłożona $A = UL$):

$$det(A) = det(UL) = det(U)det(L) = det(L).$$

Ostatnia równość wynika z bardzo prostego faktu. Macierz A rozkładamy metodą Crouta, co oznacza, że macierz górnotrójkątna U ma same jedynki na przekątnej, a więc jej wyznacznik jest równy 1. Oznacza to, że policzenie wyznacznika macierzy A sprowadza się do przemnożenia wszystkich elementów znajdujących się na przekątnej macierzy dolnotrójkątnej L.

3 Opis programu obliczeniowego

Mój program obliczeniowy składa się z czterech funkcji. Opiszę każdą w tej sekcji.

3.1 rozkladCroutaUL

Funkcja ta przyjmuje następujące elementy:

- A - jest to macierz, której rozkład UL chcemy uzyskać, nie dla każdej macierzy istnieje taki rozkład - program w razie czego wyrzuci błąd (w przypadku gdy macierz nie jest kwadratowa lub jeden z jej wiodących minorów głównych, ale patrząc od prawego dolnego rogu jest równy zero)

Funkcja zwraca następujące wartości:

- U - macierz górnotrójkątna z jedynkami na głównej przekątnej, która powstała w wyniku rozkładu $A = UL$
- L - macierz dolnotrójkątna, która powstała w wyniku rozkładu $A = UL$

```
% (C) Jan Skwarek
function [U, L] = rozkladCroutaUL(A)
    % sprawdzamy wymiary macierzy A
    [rows, columns] = size(A);
    % przypadek w ktorym macierz A nie jest kwadratowa - rozklad UL takiej
    % macierzy nie istnieje - wyrzucamy blad
    if rows ~= columns
        error("Macierz nie jest kwadratowa!")
    end
    % standardowa implementacja algorytmu rozkladu Crouta, zmienione sa
    % tylko nieco indeksy aby uzyskac rozklad UL zamiast rozkladu LU
    for i = rows:-1:1
        % uzupelniamy ostatni rzad macierzy L
        L(rows, i) = A(rows, i);
        % w rozkladzie Crouta macierz gornotrojkatna ma jedynki na glownej
        % przekatnej
        U(i, i) = 1;
    end
    for j = rows - 1:-1:1
        if L(rows, rows) == 0 || abs(L(rows, rows)) < 1e-5
```

```
            % wartosc w ostatnim wierszu w ostatniej kolumnie nie moze byc
            % rowna zero - dzielenie przez zero - jeden z wiodacych minorow
            % glownych ale patrzac od prawego dolnego rogu macierzy moze
            % byc rowny zero
            error("Dzielenie przez zero!" + ...
                " Jeden z wiodacych minorow glownych, ale liczonych od" + ...
                " prawego dolnego rogu macierzy wejsciowej jest rowny zero!" + ...
                " Rozklad UL nie istnieje.")
        end
        % uzupelniamy ostatnia kolumne macierzy U
        U(j, rows) = A(j, rows) / L(rows, rows);
    end
    for i = rows - 1:-1:1
        for j = rows - 1:-1:i
            % uzupelniamy kolejne kolumny macierzy L
            L(j, i) = A(j, i) - dot(L(rows:-1:j + 1, i), U(j, rows:-1:j + 1));
        end
        for j = i - 1:-1:1
            % sprawdzamy czy na glownej przekatnej macierzy L pojawilo sie
            % zero. Jezeli tak, to przerywamy program i wyrzucamy blad -
            % rozklad UL nie istnieje - jeden z wiodacych minorow glownych
            % ale patrzac od prawego dolnego rogu macierzy wejsciowej A
            % jest rowny zero
            if L(i, i) == 0 || abs(L(i, i)) < 1e-5
                error("det(L) bliski zeru. Dzielenie przez zero!" + ...
                    " Jeden z wiodacych minorow glownych, ale liczonych" + ...
                    " od prawego dolnego rogu macierzy wejsciowej jest rowny zero!" + ...
                    " Rozklad UL nie istnieje.")
            end
            % uzupelniamy kolejne kolumny macierzy U
            U(j, i) = (A(j, i) - dot(L(rows:-1:i + 1, i), U(j, rows:-1:i + 1))) / L(i, i);
        end
    end
end
```

Uwaga! Instrukcje warunkowe wyrzucające drugi oraz trzeci *error* zostały tak zaprogramowane, aby wykrywać również liczby bliskie zeru (domyślnie liczba zostaje uznana za bliską zeru, kiedy jej wartość bezwzględna jest mniejsza od $1e-5$, ale można to w każdym momencie zmienić w kodzie w mgnieniu oka).

3.2 rozwiazUkladRownanUL

Funkcja ta przyjmuje następujące elementy:

- A - jest to macierz, której rozkład UL chcemy uzyskać i potem rozwiązać układ równań typu $Ax = B$
- B - jest to pewna macierz będąca prawą stroną w naszym układzie równań: $Ax = B$

Funkcja zwraca następujące wartości:

- X - macierz będąca macierzą wynikową układu równań $Ax = B$, gdzie $x = X$

```
% (C) Jan Skwarek
function X = rozwiazUkladRownanUL(A, B)
    % korzystamy z wczesniejszej funkcji aby uzyskac rozklad UL macierzy A
    [upperA, lowerA] = rozkladCroutaUL(A);
    % rozwiazuje rownanie UY = B
    Y = upperA\B;
    % rozwiazuje rownanie LX = Y
```

```
    X = lowerA\Y;
end
```

Warto wspomnieć jeszcze o obsłudze przypadków złośliwych. Jeżeli nie istnieje rozkład macierzy $A = UL$, poinformuje nas o tym poprzednia funkcja, wyrzucając odpowiedni błąd. Jeżeli natomiast układ równań będzie błędny, np. nie będą się zgadzały wymiary macierzy wynikowej z wymiarami macierzy wejściowej, to błąd z odpowiednią informacją wyrzuci już MATLAB.

3.3 wyznaczMacierzOdwrotna

Funkcja ta przyjmuje następujące elementy:

- A - jest to macierz, której rozkład UL chcemy uzyskać i potem wyznaczyć jej macierz odwrotną A^{-1}

Funkcja zwraca następujące wartości:

- $invMatrix$ - macierz będąca macierzą odwrotną macierzy wejściowej A

```
% (C) Jan Skwarek
function invMatrix = wyznaczMacierzOdwrotna(A)
    % korzystamy z wczesniejszej funkcji aby uzyskac rozklad UL macierzy A
    [upperA, lowerA] = rozkladCroutaUL(A);
    % sprawdzamy wymiary macierzy A
    [rows, ~] = size(A);
    % tworzymy macierz jednostkowa potrzebna do wyznaczenia macierzy
    % odwrotnej
    identityMatrix = eye(rows);
    % UY = I
    Y = upperA\identityMatrix;
    % Lx = Y, gdzie x to macierz odwrotna do macierzy wejsciowej A
    invMatrix = lowerA\Y;
end
```

Jeżeli rozkład UL danej macierzy nie istnieje to funkcja $rozkladCroutaUL$ wyrzuci błąd.

3.4 obliczWyznacznik

Funkcja ta przyjmuje następujące elementy:

- A - jest to macierz, której rozkład UL chcemy uzyskać i potem policzyć jej wyznacznik

Funkcja zwraca następujące wartości:

- $wyznacznik$ - wyznacznik macierzy wejściowej A

```
% (C) Jan Skwarek
function wyznacznik = obliczWyznacznik(A)
    % korzystamy z wczesniejszej funkcji aby uzyskac rozklad UL macierzy A
    [~, lowerA] = rozkladCroutaUL(A);
    % sprawdzamy wymiary macierzy A
    [rows, ~] = size(A);
    wyznacznik = 1;
    % mnozymy kolejne wartosci znajdujace sie na przekatnej macierzy
    % dolnotrojkatnej L
    for i = 1:rows
        wyznacznik = wyznacznik * lowerA(i, i);
    end
end
```

Jeżeli rozkład UL danej macierzy nie istnieje to funkcja $rozkladCroutaUL$ wyrzuci błąd.

4 Przykłady obliczeniowe

4.1 Przykład 1 - macierz nie jest kwadratowa

Zacznijmy od trywialnego przykładu. Na wejściu dana została macierz, która nie jest macierzą kwadratową, a co za tym idzie, nie istnieje dla niej rozkład UL, na którym opiera się cały program. Wygenerujmy sobie najpierw jakąś losową macierz spełniającą warunki przykładu.

```
>> exA = randi([1 100], 2, 6)

exA =

    82    13    64    28    96    16
    91    92    10    55    97    98
```

Jak zachowają się poszczególne funkcje?

```
>> rozkladCroutaUL(exA)
Error using rozkladCroutaUL (line 8)
Macierz nie jest kwadratowa!

>> rozwiazUkladRownanUL(exA)
Error using rozkladCroutaUL (line 8)
Macierz nie jest kwadratowa!

Error in rozwiazUkladRownanUL (line 4)
    [upperA, lowerA] = rozkladCroutaUL(A);

>> wyznaczMacierzOdwrotna(exA)
Error using rozkladCroutaUL (line 8)
Macierz nie jest kwadratowa!

Error in wyznaczMacierzOdwrotna (line 4)
    [upperA, lowerA] = rozkladCroutaUL(A);

>> obliczWyznacznik(exA)
Error using rozkladCroutaUL (line 8)
Macierz nie jest kwadratowa!

Error in obliczWyznacznik (line 4)
    [~, lowerA] = rozkladCroutaUL(A);
```

Dokładnie takiego rezultatu oczekiwaliśmy. Z oczywistych powodów nie ma sensu porównywać otrzymanych wyników ze wbudowanymi funkcjami MATLABA.

4.2 Przykład 2 - jeden z wiodących minorów głównych macierzy, ale liczonych niestandardowo, bo od prawego dolnego rogu równy zero

Zobaczmy, jak program zareaguje, gdy zostanie mu dana macierz, której jeden z wiodących minorów głównych, ale liczonych od prawego dolnego rogu jest równy zero.

```
>> exB = [4 2 1 2; 6 5 4 1; 8 2 2 0; 4 3 2 1]

exB =

     4     2     1     2
     6     5     4     1
     8     2     2     0
     4     3     2     1
```

Przyjrzyjmy się na moment wiodącym minorom głównym powyższej macierzy, ale liczonym niestandardowo, bo od prawego dolnego rogu. Pierwszy minor jest oczywiście równy ostatniej wartości na diagonali, a więc 1. Kolejny minor możemy policzyć ze wzoru na wyznacznik macierzy $2x2$, a więc równy jest $2*1-2*0 = 2$. Trzeci wyznacznik policzmy metodą Sarrus'a: $1*2*5+2*2*1+3*0*4-1*2*3-4*2*1-5*0*2 = 14-14 = 0$. Trzeci wiodący minor główny liczony w tak niestandardowy sposób jest równy zero, a więc oczekujemy, że funkcja wyrzuci nam błąd - rozkład UL danej macierzy nie istnieje. Przetestujmy.

```
>> rozkladCroutaUL(exB)
Error using rozkladCroutaUL (line 45)
det(L) bliski zeru. Dzielenie przez zero! Jeden z wiodacych minorow glownych, ale liczonych od
prawego dolnego rogu macierzy wejsciowej jest rowny zero! Rozkład UL nie istnieje.
```

Jak zareagują pozostałe funkcje?

```
>> rozwiazUkladRownanUL(exB)
Error using rozkladCroutaUL (line 45)
det(L) bliski zeru. Dzielenie przez zero! Jeden z wiodacych minorow glownych, ale liczonych od
prawego dolnego rogu macierzy wejsciowej jest rowny zero! Rozkład UL nie istnieje.

Error in rozwiazUkladRownanUL (line 4)
    [upperA, lowerA] = rozkladCroutaUL(A);

>> wyznaczMacierzOdwrotna(exB)
Error using rozkladCroutaUL (line 45)
det(L) bliski zeru. Dzielenie przez zero! Jeden z wiodacych minorow glownych, ale liczonych od
prawego dolnego rogu macierzy wejsciowej jest rowny zero! Rozkład UL nie istnieje.

Error in wyznaczMacierzOdwrotna (line 4)
    [upperA, lowerA] = rozkladCroutaUL(A);

>> obliczWyznacznik(exB)
Error using rozkladCroutaUL (line 45)
det(L) bliski zeru. Dzielenie przez zero! Jeden z wiodacych minorow glownych, ale liczonych od
prawego dolnego rogu macierzy wejsciowej jest rowny zero! Rozkład UL nie istnieje.

Error in obliczWyznacznik (line 4)
    [~, lowerA] = rozkladCroutaUL(A);
```

Dokładnie takiego wyniku się spodziewaliśmy. Ponownie nie ma sensu sprawdzać otrzymanych wyników ze wbudowanymi funkcjami MATLABA.

4.3 Przykład 3 - macierz $3x3$

Wygenerujmy sobie losowo macierz $3x3$ z wyrazami od 0 do 10.

```
>> ex3 = randi([0 10], 3, 3)

ex3 =

     8    10     3
     9     6     6
     1     1    10

>> [temp1 temp2] = rozkladCroutaUL(ex3)

temp1 =

    1.0000    1.7963    0.3000
         0    1.0000    0.6000
```

```
         0         0    1.0000


temp2 =

   -7.3889         0         0
    8.4000    5.4000         0
    1.0000    1.0000   10.0000
```

Pomnóżmy teraz $temp1$ i $temp2$, aby zweryfikować czy zaproponowany przez algorytm rozkład UL jest poprawny (dopóki macierz jest mała, możemy sobie pozwolić na takie trywialne zabiegi).

```
>> temp1 * temp2

ans =

    8.0000   10.0000    3.0000
    9.0000    6.0000    6.0000
    1.0000    1.0000   10.0000
```

Wyszło dokładnie to samo! Spróbujmy rozwiązać teraz jakiś układ równań.

```
>> ex3b = randi([0 10], 3, 1)

ex3b =

    10
     1
    10

>> rozwiazUkladRownanUL(ex3, ex3b)

ans =

   -2.1629
    2.4386
    0.9724
```

Zweryfikujmy poprawność tego rozwiązania.

```
>> ex3 * ans

ans =

   10.0000
    1.0000
   10.0000
```

Wyszła nam dokładna wartość $ex3b$! Sprawdźmy dla innego układu równań.

```
>> ex3c = randi([0 10], 3, 2)

ex3c =

    10     1
     5     4
     8    10
```

```
>> rozwiazUkladRownanUL(ex3, ex3c)

ans =

   -0.9799   -0.2155
    1.5614   -0.0351
    0.7419    1.0251

>> ex3 * ans

ans =

   10.0000    1.0000
    5.0000    4.0000
    8.0000   10.0000
```

Dokładnie to samo! Zajmijmy się teraz macierzą odwrotną i wyznacznikiem.

```
>> wyznaczMacierzOdwrotna(ex3)

ans =

   -0.1353    0.2431   -0.1053
    0.2105   -0.1930    0.0526
   -0.0075   -0.0050    0.1053

>> inv(ex3)

ans =

   -0.1353    0.2431   -0.1053
    0.2105   -0.1930    0.0526
   -0.0075   -0.0050    0.1053

>> obliczWyznacznik(ex3)

ans =

  -399

>> det(ex3)

ans =

  -399
```

Wszystkie wyniki uzyskane przez algorytmy są identyczne w porównaniu z tymi, otrzymanymi przez funkcje wbudowane w MATLABa.

4.4 Przykład 4 - macierz $5x5$

Wygenerujmy losowo macierz $5x5$ z wyrazami od 0 do 100. Jest to ostatni przykład, w którym wkleję całe generowane macierze (w kolejnych przykładach ich wymiary będą już zbyt duże).

```
> ex4 = randi([0 100], 5, 5)

ex4 =

    80    94    66     4    95
```

```
    96    68    17     9     3
    66    76    71    83    44
     3    75     3    70    38
    85    39    27    32    77

>> [temp1 temp2] = rozkladCroutaUL(ex4)

temp1 =

    1.0000    1.4122    0.3819   -0.6545    1.2338
         0    1.0000    0.2566    0.1430    0.0390
         0         0    1.0000    1.1938    0.5714
         0         0         0    1.0000    0.4935
         0         0         0         0    1.0000


temp2 =

 -190.3739         0         0         0         0
   81.8535   61.8027         0         0         0
   63.9255  -12.8450   67.8972         0         0
  -38.9481   55.7532  -10.3247   54.2078         0
   85.0000   39.0000   27.0000   32.0000   77.0000

>> answear = temp1 * temp2

answear =

   80.0000   94.0000   66.0000    4.0000   95.0000
   96.0000   68.0000   17.0000    9.0000    3.0000
   66.0000   76.0000   71.0000   83.0000   44.0000
    3.0000   75.0000    3.0000   70.0000   38.0000
   85.0000   39.0000   27.0000   32.0000   77.0000

>> abs(ex4 - answear)

ans =

   1.0e-13 *

         0    0.1421         0         0         0
    0.1421         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0
```

Widzimy więc, że różnice są subtelne, rzędu $1e-13$. Możemy zatem przyjąć, że algorytm charakteryzuje się bardzo wysoką dokładnością. Policzmy teraz jakiś układ równań.

```
>> answear = rozwiazUkladRownanUL(ex4, ex4b)

answear =

    0.0573    0.4347   -0.1475    0.1745    0.1660
    0.1593    0.7505    0.4505   -0.0982    0.9731
   -0.0763   -1.0480   -0.7186    0.4955    0.4783
    0.1011   -0.0057    0.7553    0.3177    0.0967
    0.6850    0.3670    0.3143    0.2137   -0.7022
```

```
>> answear2 = ex4\ex4b

answear2 =

    0.0573    0.4347   -0.1475    0.1745    0.1660
    0.1593    0.7505    0.4505   -0.0982    0.9731
   -0.0763   -1.0480   -0.7186    0.4955    0.4783
    0.1011   -0.0057    0.7553    0.3177    0.0967
    0.6850    0.3670    0.3143    0.2137   -0.7022

>> abs(answear - answear2)

ans =

   1.0e-15 *

    0.1318    0.1110    0.0555    0.1110    0.0555
    0.1388    0.1110    0.1665    0.0139    0.3331
    0.1388    0.2220    0.2220    0.1665    0.2220
    0.0971    0.0295    0.1110         0    0.4163
         0    0.1665    0.0555         0         0
```

Ponownie możemy zaobserwować znikome wręcz różnice w obydwu wynikach (rzędu $1e-15$). Bardzo wysoka dokładność algorytmu. Policzmy teraz wyznacznik i macierz odwrotną.

```
>> answear1 = obliczWyznacznik(ex4)

answear1 =

  -3.3344e+09

>> answear2 = det(ex4)

answear2 =

  -3.3344e+09

>> abs(answear1 - answear2)

ans =

   4.7684e-07

>> answear1 = wyznaczMacierzOdwrotna(ex4)

answear1 =

   -0.0053    0.0074    0.0001   -0.0046    0.0084
    0.0070    0.0064   -0.0043    0.0088   -0.0107
    0.0063   -0.0058    0.0138   -0.0116   -0.0097
   -0.0097   -0.0023    0.0071    0.0039    0.0061
    0.0041   -0.0084   -0.0057    0.0031    0.0100

>> answear2 = inv(ex4)

answear2 =

   -0.0053    0.0074    0.0001   -0.0046    0.0084
    0.0070    0.0064   -0.0043    0.0088   -0.0107
```

```
    0.0063   -0.0058    0.0138   -0.0116   -0.0097
   -0.0097   -0.0023    0.0071    0.0039    0.0061
    0.0041   -0.0084   -0.0057    0.0031    0.0100

>> abs(answear1 - answear2)

ans =

   1.0e-17 *

    0.0867    0.2602    0.0529    0.0867         0
    0.0867    0.4337    0.0867         0    0.3469
    0.0867    0.1735    0.1735    0.1735    0.1735
         0    0.3903    0.0867    0.1735    0.3469
         0         0    0.0867    0.1735    0.1735
```

Po raz kolejny możemy potwierdzić wysoką dokładność algorytmu. Różnice rzędu $1e-7$ przy wyznaczniku i $1e-17$ przy macierzy odwrotnej.

4.5 Przykład 5 - macierz $10x10$

Dochodzimy już do nieco większych macierzy. Tym razem nie będę ich umieszczał w tym dokumencie, a jedynie sprawdzał różnice z macierzami bądź wynikami uzyskanymi przez wbudowane funkcje MATLABA.

```
>> ex5 = randi([0 100], 10, 10)

>> [temp1 temp2] = rozkladCroutaUL(ex5)

>> abs(answear1 - ex5)

ans =

   1.0e-13 * (...)
```

Ponownie bardzo dokładny wynik. Sprawdźmy układy równań.

```
>> ex5b = randi([0 100], 10, 10)

>> answear1 = rozwiazUkladRownanUL(ex5, ex5b)

>> answear2 = ex5\ex5b

>> abs(answear1 - answear2)

ans =

   1.0e-12 * (...)
```

Sprawdźmy jeszcze wyznacznik i macierz odwrotną.

```
>> answear1 = obliczWyznacznik(ex5)

answear1 =

  -7.7333e+17

>> answear2 = det(ex5)
```

```
answear2 =

  -7.7333e+17

>> abs(answear1 - answear2)

ans =

        4352
```

Różnica wyszła nieco większa niż w poprzednich przykładach, ale błąd jest wciąż niewielki. Pełna analiza w następnym rozdziale.

```
>> answear1 = wyznaczMacierzOdwrotna(ex5)

>> answear2 = inv(ex5)

>> abs(answear1 - answear2)

ans =

   1.0e-14 * (...)
```

4.6 Przykład 6 - macierz $100x100$

W kolejnym przykładzie powiększę znacznie rozważane macierze. Na tapet wezmę macierz o wymiarze $100x100$.

```
>> ex6 = randi([0 100], 100, 100)

>> [temp1 temp2] = rozkladCroutaUL(ex6)

>> answear1 = temp1 * temp2

>> abs(answear1 - ex6)

ans =

   1.0e-10 * (...)
```

Sprawdźmy jeszcze układy równań.

```
>> ex6b = randi([0 100], 100, 100)

>> answear1 = rozwiazUkladRownanUL(ex6, ex6b)

>> answear2 = ex6\ex6b

>> abs(answear1 - answear2)

ans =

   1.0e-11 * (...)
```

Zweryfikujmy jeszcze wyznacznik i macierz odwrotną.

```
>> answear1 = obliczWyznacznik(ex6)

answear1 =
```

```
 -5.1122e+226

>> answear2 = det(ex6)

answear2 =

 -5.1122e+226

>> abs(answear1 - answear2)

ans =

  2.6248e+214

>> answear1 = wyznaczMacierzOdwrotna(ex6)

>> answear2 = inv(ex6)

>> abs(answear1 - answear2)

ans =

   1.0e-13 * (...)
```

Ponownie bardzo wysoka dokładność.

4.7 Przykład 7 - macierz $1000x1000$

Ostatni przykład będzie również najbardziej ekstremalny. Sprawdźmy jak program zachowa się przy macierzach $1000x1000$.

```
>> ex7 = randi([0 100], 1000, 1000);

>> [temp1 temp2] = rozkladCroutaUL(ex7);

>> answear1 = temp1 * temp2;

>> sum(abs(answear1 - ex7))

ans =

   1.0e-06 * (...)
```

Sprawdźmy jeszcze układy równań.

```
>> ex7b = randi([0 100], 1000, 1000);

>> answear1 = rozwiazUkladRownanUL(ex7, ex7b);

>> answear2 = ex7\ex7b;

>> sum(abs(answear1 - answear2))

ans =

   1.0e-06 * (...)
```

Zweryfikujmy jeszcze wyznacznik i macierz odwrotną.

```
>> answear1 = obliczWyznacznik(ex7)

answear1 =

   Inf

>> answear2 = det(ex7)

answear2 =

   Inf

>> abs(answear1 - answear2)

ans =

   NaN

>> answear1 = wyznaczMacierzOdwrotna(ex7);

>> answear2 = inv(ex7);

>> sum(abs(answear1 - answear2))

ans =

   1.0e-09 * (...)
```

W przypadku macierzy odwrotnej - ponownie uzyskaliśmy wynik z bardzo dobrą dokładnością. Niestety przy macierzach takiego kalibru program nie wykonuje się błyskawicznie - trzeba poczekać czasem kilka sekund. Wyznacznik natomiast wyszedł tak duży, że MATLAB nie był w stanie ocenić dokładności naszego programu obliczeniowego w tym przypadku.

5 Analiza uzyskanych danych

5.1 Macierze, dla których rozkład UL nie istnieje

Podczas testowania programu wielokrotnie analizowałem macierze, dla których rozkład UL nie istnieje. Są to macierze, które nie są kwadratowe oraz takie, których jeden z wiodących minorów głównych liczonych w niestandardowy sposób - bo od prawego dolnego rogu do lewego górnego - jest równy zero. W większości przypadków moje testy były w pełni kontrolowane, aczkolwiek zdarzyło się też trafić na taką macierz przypadkiem i później zweryfikować, że faktycznie rozkład UL dla niej nie istnieje. W każdym razie 100% takich przypadków zostało wykryte przez algorytm. Można więc założyć, że jest on w pełni skuteczny w rozpoznawaniu macierzy, dla których rozkład UL nie istnieje (w rzeczywistości dużo zależy od danych i od ustalonej granicy, kiedy liczba ”jest bliska zeru”, przyjętej w instrukcji warunkowej wyrzucającej błąd). Niektóre takie przypadki można zaobserwować w poprzednim rozdziale w Przykładzie 1 oraz Przykładzie 2.

5.2 Macierze $3x3$

Na podstawie dwudziestu losowych macierzy o wymiarach $3x3$ (macierze do układów równań wybierane były również losowo) policzyłem średni błąd względny i bezwzględny mojego programu obliczeniowego. (Przykład 3)

- dla samego rozkładu UL - błąd względny jak i błąd bezwzględny były tak bliskie zeru, że możemy spokojnie założyć równość zeru, program okazał się praktycznie idealny
- dla liczenia wyznacznika - sytuacja wygląda podobnie jak w przypadku wyżej
- dla liczenia układów równań - sytuacja analogiczna
- dla liczenia macierzy odwrotnej - sytuacja analogiczna

Możemy spokojnie założyć, że w przypadku macierzy o wymiarze $3x3$ program jest niemal idealny, a wszystkie błędy są tak mikroskopijne, że pomijalne.

5.3 Macierze $5x5$

Na podstawie dwudziestu losowych macierzy (dla których istniał rozkład UL - w przeciwnym wypadku losowana była kolejna macierz) o wymiarach $5x5$ (macierze do układów równań wybierane były również losowo) policzyłem średni błąd względny i bezwzględny. (Przykład 4)

- dla samego rozkładu UL - średni błąd bezwzględny wyniósł $1.0e - 13$
- dla liczenia wyznacznika - średni błąd bezwzględny wyniósł $5.0e - 07$ natomiast średni błąd względny wyniósł $-1.5e - 16 * 100\%$
- dla liczenia układów równań - średni błąd bezwzględny wyniósł $1.0e - 15$
- dla liczenia macierzy odwrotnej - średni błąd bezwzględny wyniósł $1.0e - 17$

W przypadku macierzy $5x5$ możemy wyciągnąć podobne wnioski, co w przypadku macierzy $3x3$. Ogromna dokładność programu obliczeniowego, a błędy - raczej pomijalne.

5.4 Macierz $10x10$

Tym razem przebadałem dziesięć losowych macierzy o wymiarach $10x10$. Tak wyglądają błędy względne i bezwzględne dla liczenia poszczególnych wartości. (Przykład 5)

- dla samego rozkładu UL - średni błąd bezwzględny wyniósł $1.0e - 12$
- dla liczenia wyznacznika - średni błąd bezwzględny wyniósł 5000 natomiast średni błąd względny wyniósł $-5.0e - 15 * 100\%$
- dla liczenia układów równań - średni błąd bezwzględny wyniósł $1.0e - 12$
- dla liczenia macierzy odwrotnej - średni błąd bezwzględny wyniósł $1.0e - 14$

W przypadku macierzy $10x10$ program zachowywał się podobnie jak w poprzednich przykładach. Wszystkie te błędy są oczywiście pomijalne.

5.5 Macierz $100x100$

Tym razem przebadałem 3 losowe macierze o wymiarach $100x100$. Tak wyglądają błędy względne i bezwzględne dla liczenia poszczególnych wartości. (Przykład 6)

- dla samego rozkładu UL - średni błąd bezwzględny wyniósł $1.0e - 10$
- dla liczenia wyznacznika - średni błąd bezwzględny wyniósł $2.0e + 215$ natomiast średni błąd względny wyniósł $-5.0e - 13 * 100\%$
- dla liczenia układów równań - średni błąd bezwzględny wyniósł $1.0e - 11$
- dla liczenia macierzy odwrotnej - średni błąd bezwzględny wyniósł $1.0e - 13$

Warto zauważyć, że rozmiar macierzy zwiększył się w złożoności kwadratowej względem poprzedniego przykładu, natomiast średni błąd bezwzględny zwiększył się mniej więcej dziesięciokrotnie. Dalej jest on jednak jak najbardziej pomijalny.

5.6 Macierz $1000x1000$

Tym razem przebadałem 2 losowe macierze o wymiarach $1000x1000$. Tak wyglądają błędy względne i bezwzględne dla liczenia poszczególnych wartości. (Przykład 7)

- dla samego rozkładu UL - średni błąd bezwzględny wyniósł $1.0e - 06$
- dla liczenia wyznacznika - niestety wyznacznik za każdym razem okazywał się zbyt duży dla MATLABA, więc nie mam tutaj odpowiednich danych
- dla liczenia układów równań - średni błąd bezwzględny wyniósł $1.0e - 06$
- dla liczenia macierzy odwrotnej - średni błąd bezwzględny wyniósł $1.0e - 09$

Nawet dla tak dużych macierzy program "wypluwał" świetne wyniki z pomijalnym błędem.

6 Podsumowanie

Podsumowując, program obliczeniowy zaproponowany przeze mnie świetnie się nadaje, zarówno do liczenia układów równań, liczenia wyznacznika jak i odwracania macierzy. Nawet przy macierzach wielkości $1000x1000$ wartości liczone przez program nie odbiegają znacząco od wzorcowych wyników generowanych przez wbudowane funkcje MATLABA, a wszystkie błędy są pomijalne. Oczywiście im większa macierz, tym mniejsza dokładność, jednak macierze musiałyby mieć wymiar rzędu kilkunastu tysięcy, aby pojawił się błąd, którego nie moglibyśmy pominąć (przy rozsądnych wartościach w kolumnach).

7 Bibliografia

- D. Kincaid, W. Cheney, Analiza numeryczna, Wydawnictwa Naukowo-Techniczne, Warszawa 2006
- https://en.wikipedia.org/wiki/Crout_matrix_decomposition
- Notatki do Metod Numerycznych autorstwa dr. Iwony Wróbel
- https://wazniak.mimuw.edu.pl/index.php?title=Metody_numeryczne
- https://mycareerwise.com/programming/category/numerical-analysis/crouts-method#python
- https://www.quora.com/When-does-a-matrix-not-have-an-LU-decomposition
- https://math.stackexchange.com/questions/1372166/a-ul-factorization
- https://ocw.mit.edu/courses/mathematics/18-06sc-linear-algebra-fall-2011/ax-b-and-the-four-subspaces/ factorization-into-a-lu/
- https://www.numerade.com/ask/question/instead-of-the-lu-decomposition-we-can-also-use-ul-decompositio
- http://web.mit.edu/18.06/www/Spring09/pset2-s09-soln.pdf
- https://www.researchgate.net/publication/225599466_LU-_versus_UL-Factorization_of_Integral_ Operators_with_Semi-Separable_Kernel