Metody numerycze - ćwiczenie 2.

Sprawozdanie

Dawid Chmielewski 311188

Rozwiązywanie układów równań liniowych.

```
clc
clear
close all
format compact

A = [4, 5, 6, 2, 3;
    1, 2, 3, 4, 5;
    7, 2, 6, 1, 2;
    2, 3, 5, 6, 7;
    3, 4, 6, 3, 2];

B = [3;
    4;
    5;
    6;
    7];

Bstart = B;
Astart = A;

Adims = size(A);
N = Adims(1);
```

Ćw 1. Eliminacja Gaussa do rozwiązywania układu równań liniowych.

Jest to algorytm zaimplementowany przeze mnie w ramach zajęć- tutaj do sprawozdania umieściłem go w niemal niezmienionej formie jako wyodrębniona funkcja na końcu skryptu.

```
Xgauss = GaussElimination(A, B, Adims)
```

```
Xgauss = 5×1
  -12.0000
  -10.6667
   18.3333
  -12.3333
    6.3333
```

```
errGauss = abs(B - A * Xgauss)
```

```
errGauss = 5×1
10^-13 ×
    0.0711
```

```
    0.0355
    0.1599
    0.2132
    0.2132
```

Wartości błędów w wektorze errGauss są bardzo małe- rzędu 10^(-13).

Ćw 2. algorytm rozwiązujący układ równań liniowych w oparciu o rozkład macierzy A na iloczyn macierzy LU. Ostatni argument funkcji LUsolving służy do włączenia poprawności rozkładu LU- 1, jeśli ma zostać obliczona (wyłączam ją do ćwiczenia 4, w którym porównuję szybkości algorytmów).

```
A = Astart;

Xlu = LUsolving(A, B, N, 1)
```

```
LUcorrect = 5×5
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
Xlu = 5×1
  -12.0000
  -10.6667
   18.3333
  -12.3333
    6.3333
```

```

errLU = abs(B - A*Xlu)
```

```
errLU = 5×1
10^-13 ×
    0.1066
         0
    0.0355
    0.0711
    0.2309
```

Wartości błędów w wektorze errLU są bardzo małe- rzędu 10^(-13), czyli takiego samego rzędu jak dla eliminacji Gaussa.

Ćw. 3. algorytm odwracający macierz za pomocą metody Gaussa-Jordana.

Zaimplementowany przeze mnie algorytm porównałem do wbudowanej funkcji inv. Jak widać, w obu przypadkach wynik jest identyczny.

```
A = Astart;

ATgaujord = GaussJordan(A, N)
```

```
ATgaujord = 5×5
    2.0000  -14.5000   -0.7500   10.5000   -2.7500
    2.0000  -11.8333   -0.9167    8.5000   -2.2500
   -3.0000   20.6667    1.3333  -15.0000    4.0000
    2.0000  -17.1667   -1.0833   12.5000   -2.7500
   -1.0000    9.1667    0.5833   -6.5000    1.2500
```

```
ATinv = inv(A)
```

```
ATinv = 5×5
    2.0000  -14.5000   -0.7500   10.5000   -2.7500
    2.0000  -11.8333   -0.9167    8.5000   -2.2500
   -3.0000   20.6667    1.3333  -15.0000    4.0000
    2.0000  -17.1667   -1.0833   12.5000   -2.7500
   -1.0000    9.1667    0.5833   -6.5000    1.2500
```

Porównując macierz odwróconą za pomocą mojego algorytmu do macierzy odwróconej za pomocą funkcji wbudowanej w Matlaba, możemy zauważyć, iż mają one takie same wartości - dowodzi to poprawnosci wykonanej przeze mnie implementacji.

Ćw. 4. porównanie szybkości zaimplementowanych algorytmów rozwiązywania układów liniowych.

```
MatrixDimNumbers = [5,10,20,50,100,150,200,300,400,500,750,1000];
nmbrs = 12;

LUtimes = zeros(1, nmbrs);
Gausstimes = zeros(1, nmbrs);
for i = 1:nmbrs
    N = MatrixDimNumbers(i);
    A = randi([1, 100], [N, N]);
    B = randi([1, 100], [N, 1]);

    %lu solving...
    tic;
    X = LUsolving(A, B, N, 0);
    LUtimes(i) = toc;

    %gauss solving...
    Adimss = size(A);
    tic;
    X = GaussElimination(A, B, Adimss);
    Gausstimes(i) = toc;
end

LUtimes
```

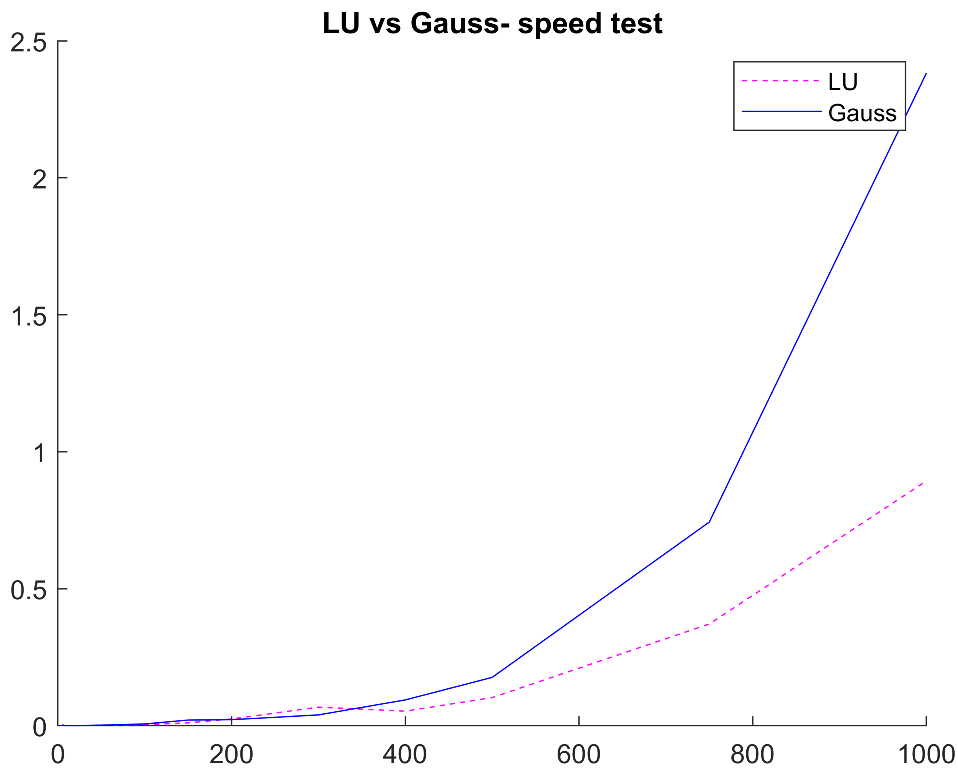
```
LUtimes = 1×12
    0.0023    0.0002    0.0004    0.0025    0.0038    0.0106    0.0246    0.0678 ⋯
```

```
Gausstimes
```

```
Gausstimes = 1×12
    0.0026    0.0002    0.0002    0.0021    0.0065    0.0206    0.0221    0.0395 ⋯
```

```
hold on;
plot(MatrixDimNumbers, LUtimes, '--', color='magenta');
plot(MatrixDimNumbers, Gausstimes, '-', color='blue');
title('LU vs Gauss- speed test');
```

```
legend('LU', 'Gauss');
hold off;
```



Wnioski: możemy zaobserwować na wykresie, iż dla większych macierzy metoda rozkładu LU staje się o wiele szybsza od metody eliminacji Gaussa. Im większymi układami równań dysponujemy, tym większa jest ta różnica. Dla małych układów natomiast eliminacja Gaussa jest szybsza- widać to chociażby z bezpośredniego porównania pierwszych wartości wektorów LUtimes i Gausstimes.

Wszystkie zadania zostały zrealizowane w oparciu o następujące zaimplementowane przeze mnie funkcje:

```
function AT = GaussJordan(A, N)
Diag = zeros(N);
for i = 1:N
    Diag(i, i) = 1;
end

A = [A, Diag];

for i = 1:N
    for j = 1:N
        if (i ~= j)
            R = A(j, i) / A(i, i);
            for k = 1:2*N
                A(j, k) = A(j, k) - R * A(i, k);
            end
        end
```

```
    end
end

for i = 1:N
    for j = N+1:2*N
        A(i, j) = A(i, j) / A(i, i);
    end
end

AT = A(:,N+1:2*N);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function X = LUsolving(A, B, N, enableLUcorr)
L = zeros(N);
U = zeros(N);
for i = 1:N
    L(i, i) = 1;
end

for j = 1:N
    for i = 1:j
        s = 0;
        %do macierzy u i,j
        for k = 1:i-1
            s = s + L(i, k) * U(k, j);
        end
        U(i, j) = A(i, j) - s;
    end
    for i = j+1:N
        s = 0;
        for k = 1:j
            s = s + L(i, k) * U(k, j);
        end
        L(i, j) = (A(i, j) - s) / U(j, j);
    end
end

if(enableLUcorr == 1)
LUcorrect = abs(A - L * U)
end
aLU = L + U;

for i = 1:N
    aLU(i, i) = aLU(i, i) - 1;
end

X = zeros(N, 1);
X(1) = B(1);
for i = 2:N
    s = 0;
```

```
    for j = 1:i-1
        s = s + aLU(i, j) * X(j);
    end
    X(i) = B(i) - s;
end

X(i) = X(N) / aLU(N, N);

for i = N-1:-1:1
    s = 0;
    for j = i+1:N
        s = s+ aLU(i, j) * X(j);
    end
    X(i) = (X(i) - s) / aLU(i, i);
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function X = GaussElimination(A, B, Adims)
X = zeros(Adims(1), 1);

for row = 1 : Adims(1)-1

    for column = Adims(1) : -1 : row+1

        %wybor wspolczynnika
        currentConf = A(column, row) / A(row, row);

        %mnozenie wiersza przez ten wspolczynnik
        for k = 1:Adims(2)

            A(column,k) = A(column, k) - currentConf * A(row,k);

        end

        B(column) = B(column) - currentConf * B(row);

    end

end

%ostatni element x
X(Adims(1)) = B(Adims(1)) / A(Adims(1), Adims(2));

%iteracja po x od dolu...
for column = Adims(1)-1 : -1 : 1

    currentSum = 0;
    for row = Adims(1) : -1 : column+1
        currentSum = currentSum + A(column,row) * X(row);
    end
```

```
    X(column) = (B(column) - currentSum) / A(column,column);

end
end
```