ŁUKASZ STANISZEWSKI, NR INDEKSU: 304098

METODY NUMERYCZNE

PROJEKT 1 – ZADANIA 1.11

1 ZADANIE 1 – DOKŁADNOŚĆ MASZYNOWA KOMPUTERA

Celem tego zadania było napisanie programu wyznaczającego dokładność maszynową komputera i uruchomienie go na swoim komputerze.

W arytmetyce zmiennopozycyjnej występują dane wejściowe obarczone błędami reprezentacji maszynowej jak i elementarne operacje oraz funkcje, które generują błędy numeryczne. Standard IEEE754 narzucił komputerom wymóg zorganizowania jednostki arytmetycznej tak, aby wartość bezwzględna z błędu zaokrąglenia ($\boldsymbol{\varepsilon}$) wyniku działań elementarnych była nie większa od dokładności maszynowej komputera (eps), co można zapisać (wzór 1.5):

$$|\boldsymbol{\varepsilon}| \leq \boldsymbol{eps}.$$

W ogólnym ujęciu dokładność maszynową $\boldsymbol{eps}$ można określić jako taką najmniejszą dodatnią liczbę maszynową $\boldsymbol{g}$, że wynik działania zmiennopozycyjnego $\boldsymbol{fl}$ polegającego na dodaniu tej liczby do $\mathbf{1}$ będzie wciąż liczbą większą od $\mathbf{1}$, co można zapisać (wzór 1.6):

$$\boldsymbol{eps} = \mathbf{min}\{\boldsymbol{g} \in \boldsymbol{M}: \boldsymbol{fl}(\mathbf{1} + \boldsymbol{g}) > \mathbf{1},\ \boldsymbol{g} > \mathbf{0}\},$$

gdzie $\boldsymbol{M}$ oznacza zbiór liczb maszynowych oraz $\boldsymbol{M} \subset \boldsymbol{R}$.

Przy założeniu, że podstawą w reprezentacji zmiennoprzecinkowej liczb jest 2, zaczynając od $\boldsymbol{g} = \mathbf{2^0} = \mathbf{1}$, w pętli dzieliłem liczbę $\boldsymbol{g}$ przez $\mathbf{2}$ do momentu, aż kolejny taki iloraz zsumowany z liczbą $\mathbf{1}$ przestanie dawać liczbę większą od $\mathbf{1}$, co jest realizowane w funkcji $\boldsymbol{precision.m}$, której kod wygląda następująco:

```
function [eps] = precision()
% REPREZENTACJA Oblicza dokładność maszynową (eps) komputera.
    % deklaracja + inicjalizacja zmiennych
    eps = 1; % eps reprezentuje g przed podzieleniem
    % (najbardziej aktualne g, ale spełniające warunek)
    g = 1; % g jest w kolejnych iteracjach pętli dzielone przez 2
    fl = 1.0 + g; % wynik zsumowania aktualnej wersji g z 1
    while fl > 1.0
        % wynik powinien zwracać takie g, które spełnia dalej warunek dlatego
        % konieczne jest aby eps zapamiętywało g przed podzieleniem
        eps = g;
        g = g / 2.0;
        fl = 1.0 + g;
    end
end
```

A wynikiem działania programu jest:

```
>> eps = precision()
eps =
   2.2204e-16
```

Wynik ten oznacza, że dokładność maszynowa $\boldsymbol{eps}$ mojego komputera wynosi $\mathbf{2.2204} * \mathbf{10^{-16}} = \mathbf{2^{-52}}$, co jest zgodne ze standardem **IEEE754** dla liczb w podwójnej precyzji (gdzie mantysa zapisywana jest właśnie na **52 bitach**).

2 ZADANIE 2 – ELIMINACJA GAUSSA Z CZĘŚCIOWYM WYBOREM ELEMENTU PODSTAWOWEGO

Metoda eliminacji Gaussa jest skończoną metodą rozwiązywania układów równań liniowych postaci $\boldsymbol{Ax} = \boldsymbol{b}$. Metoda ta realizowana jest w dwóch etapach:

1. Eliminacji zmiennych wykonywanym w $\boldsymbol{n}$ krokach, gdzie w wyniku odpowiednich przekształceń zarówno na macierzy $\boldsymbol{A}$ jak i wektorze $\boldsymbol{b}$, otrzymamy równoważny układ równań z macierzą trójkątną górną w miejscu macierzy $\boldsymbol{A}$.
2. Zastosowania algorytmu rozwiązania układu równań z macierzą trójkątną górną.

Jej wersja realizowana w tym zadaniu – z częściowym wyborem elementu podstawowego – wprowadza do etapu eliminacji zmiennych w każdym kroku $\boldsymbol{k}$ konieczność wybrania ze zbioru elementów macierzy $\boldsymbol{A}$ z kolumny $\boldsymbol{k}$ z wierszy o indeksie niemniejszym niż $\boldsymbol{k}$, element o największym module (wzór 2.28):

$$\left|\boldsymbol{a}_{\boldsymbol{ik}}^{(\boldsymbol{k})}\right| = \max_{\boldsymbol{j}}\left\{\left|\boldsymbol{a}_{\boldsymbol{jk}}^{(\boldsymbol{k})}\right|, \boldsymbol{j} = \boldsymbol{k}, \boldsymbol{k}+\boldsymbol{1}, \dots, \boldsymbol{n}\right\}.$$

i zamianę w macierzy rozszerzonej miejscami wiersza $\boldsymbol{i}$ wraz z wierszem $\boldsymbol{k}$.

Działanie te nie dość, że prowadzi do mniejszych błędów numerycznych, to również eliminuje sytuację, gdy w $\boldsymbol{k}$-tym kroku $\boldsymbol{a}_{\boldsymbol{kk}}^{(\boldsymbol{k})} = \boldsymbol{0}$. Tak więc realizację całego algorytmu można opisać w punktach:

1. Połączenie macierzy A oraz wektora b w macierz rozszerzoną $\boldsymbol{Ab}$:
$$\left[\begin{array}{ccc|c} \boldsymbol{a_{11}} & \cdots & \boldsymbol{a_{1n}} & \boldsymbol{b_1} \\ \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{a_{n1}} & \cdots & \boldsymbol{a_{nn}} & \boldsymbol{b_n} \end{array}\right]$$
2. W $\boldsymbol{n}-\boldsymbol{1}$ krokach wykonanie etapu eliminacji zmiennych, gdzie **$\boldsymbol{k}$-ty** krok tego działania to zamiana **$\boldsymbol{k}$-tego** wiersza macierzy rozszerzonej z wierszem $\boldsymbol{i}$, w którym znajduje się wcześniej opisany element o największym module (wybierany częściowo), a następnie wykonanie działania:
$\boldsymbol{w_i} = \boldsymbol{w_i} - \frac{\boldsymbol{a}_{\boldsymbol{ik}}^{(\boldsymbol{k})}}{\boldsymbol{a}_{\boldsymbol{kk}}^{(\boldsymbol{k})}} * \boldsymbol{w_k}$, gdzie $i = \boldsymbol{k}+\boldsymbol{1}, \dots, \boldsymbol{n}$ oraz $\boldsymbol{w_i}$ to **$\boldsymbol{i}$-ty** wiersz macierzy rozszerzonej.
3. W wyniku powyższego etapu, po $\boldsymbol{n}-\boldsymbol{1}$ krokach, uzyskany układ równań to $\boldsymbol{A}^{(\boldsymbol{n})}\boldsymbol{x} = \boldsymbol{b}^{(\boldsymbol{n})}$, gdzie $\boldsymbol{A}^{(\boldsymbol{n})}$ to macierz trójkątna górna.
4. Na końcu, w celu zdobycia wektora $\boldsymbol{x}$, koniecznym jest rozwiązanie układu $\boldsymbol{A}^{(\boldsymbol{n})}\boldsymbol{x} = \boldsymbol{b}^{(\boldsymbol{n})}$, stosując klasyczny algorytm rozwiązania układu równań z macierzą trójkątną polegający na przejściu od wiersza ostatniego w górę i wykonaniu działania (wzór 2.17):
$$\boldsymbol{x_k} = \frac{(\boldsymbol{b_k} - \sum_{\boldsymbol{j}=\boldsymbol{k}+\boldsymbol{1}}^{\boldsymbol{n}} \boldsymbol{a_{kj}}\boldsymbol{x_j})}{\boldsymbol{a_{kk}}} \text{ dla } \boldsymbol{k} = \boldsymbol{n}, \boldsymbol{n}-\boldsymbol{1}, \boldsymbol{n}-\boldsymbol{2}, \dots, \boldsymbol{1}.$$

Uwaga: w swojej realizacji algorytmu postanowiłem działania z punktów 2 oraz 4 wykonać bezpośrednio na macierzy rozszerzonej, a nie na macierzy A i wektorze b oddzielnie, aby uprościć kod.

Powyżej przedstawiony schemat działań w celu obliczenia rozwiązania układu równań metodą eliminacji Gaussa z wyborem elementu podstawowego został zrealizowany w funkcji *solveGaussPartial.m*.

```
function [x] = solveGaussPartial(A, b)
%SOLVEGAUSSPARTIAL Wyznacza rozwiazanie ukladu rownan liniowych Ax=b.
    n = size(A, 1);         % liczba rownan
    Ab = [A b];   % dla ulatwienia laczymy poziomo A z b (macierz rozszerzona)
    for k=1:n               % k to krok eliminacji gaussa
        % wybor elementu glownego
        main_el = abs(Ab(k,k));
        main_el_ind = k;
        for j = k+1:n
            if abs(Ab(j, k)) > main_el
                main_el = abs(Ab(j, k));
                main_el_ind = j;
            end
        end
        % zamiana wierszy
        temp_row = Ab(main_el_ind,:);
        Ab(main_el_ind,:) = Ab(k, :);
        Ab(k, :) = temp_row;

        % eliminacja Gaussa - liczymy l_ik i odejmujemy pomnozone przez
        % wiersz k od wiersza i macierzy rozszerzonej
        for i=k+1:n
            l = Ab(i,k) / Ab(k,k);
            Ab(i,:)=Ab(i,:) - l * Ab(k, :);
        end
    end
    % teraz posiadamy już macierz trojkatna oraz przeksztalcony wektor
    % prawych stron b i rozwiazujemy rownanie od dolu
    x = zeros(n,1);
    for k = n:-1:1
        sum_x = 0;
        for j = k+1:n
            sum_x = sum_x + x(j)*Ab(k,j);
        end
        x(k) = (Ab(k,n+1) - sum_x)/Ab(k,k);
    end
 end
```

Na początku koniecznym było przetestowanie funkcji dla zadania wymiaru $\boldsymbol{n = 5}$ z gęstą macierzą $\boldsymbol{A = GG^T}$ i wektora $\boldsymbol{b}$ wygenerowanych losowo:

```
>> [A, b] = prepareTest();
>> x = solveGaussPartial(A, b)

x =

    0.0060
    0.8827
   -0.5848
   -0.2053
    0.3134

>> norm(A*x-b,2)

ans =

   1.9860e-14
```

Wynik ten oznacza, że dla zadania o małym wymiarze i gęstej macierzy, zaimplementowany przeze mnie algorytm sprawdza się dobrze.

Następnie, koniecznym było zastosowanie algorytmu do rozwiązywania układów równań o rozmiarze n dla $\boldsymbol{n} = \mathbf{5}, \mathbf{10}, \mathbf{50}, \mathbf{100}, \mathbf{200}$.

W pierwszym kroku, dla macierzy $\boldsymbol{A}$ oraz wektora $\boldsymbol{b}$ z podpunktu (a) obliczyłem błąd rozwiązania $\boldsymbol{\varepsilon_1} = ||\boldsymbol{Ax} - \boldsymbol{b}||$ dla każdego $\boldsymbol{n}$ przy użyciu skryptu:

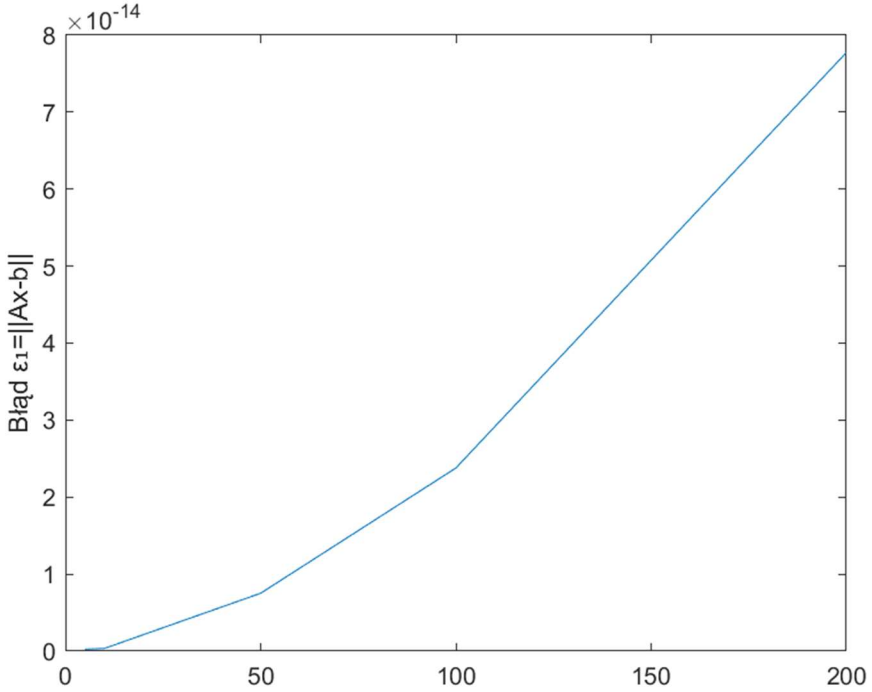
```
n_of_equations = [5, 10, 50, 100, 200];          % mozliwe n
errors_1 = zeros(size(n_of_equations,2), 1);     % wektor bledow
% wyznaczanie bledu dla kazdego n
for n_idx = 1:size(n_of_equations,2)
    n = n_of_equations(n_idx);
    [A, b] = prepareMatricesA(n); % zdobycie macierzy A i wektora b
    x = solveGaussPartial(A, b);
    errors_1(n_idx,1) = norm(A*x-b,2);
end
% wyswietlenie wektora bledow
disp(errors_1);
% zobrazowanie zaleznosci w postaci wykresu
plot(n_of_equations, errors_1);
xlabel("Liczba równań");
ylabel("Błąd ε₁=||Ax-b||");
```

W wyniku czego otrzymałem zestawienie liczby układów równań do błędu rozwiązania $\boldsymbol{\varepsilon_1}$:

$\boldsymbol{n}$	5	10	50	100	200
$\boldsymbol{\varepsilon_1} = ||\boldsymbol{Ax} - \boldsymbol{b}||$	$0.0025 * 10^{-13}$	$0.0035 * 10^{-13}$	$0.0752 * 10^{-13}$	$0.2378 * 10^{-13}$	$0.7774 * 10^{-13}$

Tabela 1 – zestawienie liczby układów równań n do błędu rozwiązania ε_1 dla macierzy i wektora z podpunktu (a) dla metody eliminacji Gaussa z częściowym wyborem el. podstawowego

oraz rysunek zależności $\boldsymbol{\varepsilon_1} = ||\boldsymbol{Ax} - \boldsymbol{b}||$ do liczby równań $\boldsymbol{n}$:



Wykres 1 – zależność błędu rozwiązania ε_1 do liczby układów równań n dla macierzy i wektora z podpunktu (a) dla metody eliminacji Gaussa z częściowym wyborem el. podstawowego

Następnie, w sposób analogiczny do podpunktu (a), dla macierzy $\boldsymbol{A}$ oraz wektora $\boldsymbol{b}$ z podpunktu (b) przeprowadziłem obliczenia, zamieniając w skrypcie *exercise2.m* linię kodu nr 6:

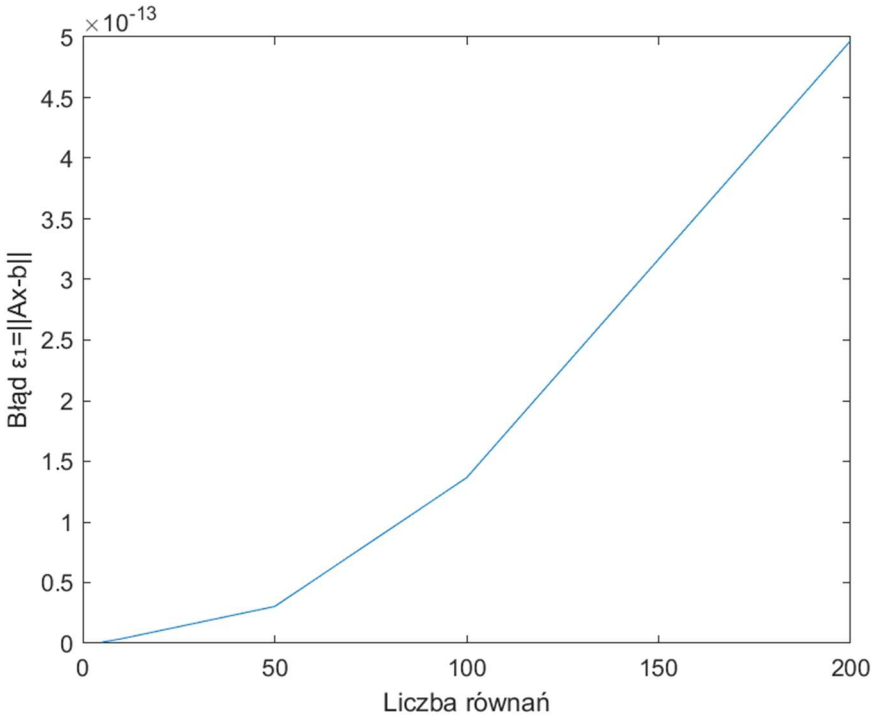
```
…
n = n_of_equations(n_idx);
[A, b] = prepareMatricesB(n); % zdobycie macierzy A i wektora b z (b)
x = solveGaussPartial(A, b);
…
```

W wyniku uruchomienia skryptu otrzymałem zestawienie tabelaryczne liczby układów równań do błędu rozwiązania $\boldsymbol{\varepsilon_1}$:

$\boldsymbol{n}$	5	10	50	100	200
$\boldsymbol{\varepsilon_1 = \|\|Ax - b\|\|}$	$0.0009 * 10^{-12}$	$0.0036 * 10^{-12}$	$0.0303 * 10^{-12}$	$0.1365 * 10^{-12}$	$0.4969 * 10^{-12}$

Tabela 2 – zestawienie liczby układów równań n do błędu rozwiązania ε_1 dla macierzy i wektora z podpunktu (b) dla metody eliminacji Gaussa z częściowym wyborem el. podstawowego

a także rysunek zależności $\boldsymbol{\varepsilon_1 = ||Ax - b||}$ do liczby równań $\boldsymbol{n}$:



Wykres 2 – zależność błędu rozwiązania ε_1 do liczby układów równań n dla macierzy i wektora z podpunktu (b) dla metody eliminacji Gaussa z częściowym wyborem el. podstawowego

Na podstawie wyżej przeprowadzonych symulacji można stwierdzić, że błąd rozwiązania układu równań metodą eliminacji Gaussa z częściowym wyborem elementu podstawowego rośnie wielomianowo wraz ze zwiększaniem się liczby równań, które za jego pomocą trzeba rozwiązać. Dla punktu **(a)** i **(b)** największym uzyskanym błędem rozwiązania jest wynik $\boldsymbol{\varepsilon_1 \cong 7.7 * 10^{-14}}$, co oznacza, że metoda ta z *całkiem* dużą dokładnością daje poprawne rozwiązanie.

3 ZADANIE 3 – METODA ITERACYJNA JACOBIEGO

Przy rozwiązywaniu układów równań liniowych można spotkać się z sytuacją, gdy problem będzie wielkowymiarowy, a macierze – rzadkie. W takich przypadkach przydatne okazać się może skorzystanie z iteracyjnych metod rozwiązywania układów równań liniowych, gdzie zaczynając z przybliżenia początkowego rozwiązania, w kolejnych krokach poprawiamy je, aby zbiegało do rozwiązania.

Jedną z takich metod jest metoda Jacobiego, polegająca na:

1. Zdekomponowaniu macierzy $\boldsymbol{A}$ na sumę macierzy poddiagonalnej $\boldsymbol{L}$, macierzy diagonalnej $\boldsymbol{D}$ oraz macierzy naddiagonalnej $\boldsymbol{U}$: $\boldsymbol{A} = \boldsymbol{L} + \boldsymbol{D} + \boldsymbol{U}$.
2. Dopóki nie będzie spełniony warunek stopu (w tym przypadku dopóki norma euklidesowa różnicy między kolejnymi przybliżeniami rozwiązania nie będzie mniejsza niż podane $\boldsymbol{\varepsilon_2}$), wykonywaniu przybliżenia wektora $\boldsymbol{x}$ (wzór 2.59):

$$\boldsymbol{x_j^{(i+1)}} = -\frac{\boldsymbol{1}}{\boldsymbol{d_{jj}}} * \left(\sum_{\boldsymbol{k=1}}^{\boldsymbol{n}} \left(\boldsymbol{l_{jk}} + \boldsymbol{u_{jk}} \right) \boldsymbol{x_k^{(i)}} - \boldsymbol{b_j} \right), \qquad \boldsymbol{j = 1, 2, \dots, n}$$

Dla metody Jacobiego istotne jest zwrócenie uwagi na to, że żeby ona zadziała, macierz $\boldsymbol{A}$ musi spełniać dwa istotne założenia:

1. Macierz diagonalna $\boldsymbol{D}$, na którą m.in. dekomponuje się macierz $\boldsymbol{A}$, jest macierzą nieosobliwą (wartości na diagonali $\boldsymbol{A}$ są niezerowe).
2. Aby metoda Jacobiego była zbieżna, macierz $\boldsymbol{A}$ musi charakteryzować się silną diagonalną dominacją wierszową lub kolumnową, tzn. musi spełniać jeden z poniższych warunków:
   a. $|\boldsymbol{a_{ii}}| > \sum_{\boldsymbol{j=1, j \neq i}}^{\boldsymbol{n}} |\boldsymbol{a_{ij}}|$, dla każdego $\boldsymbol{i = 1, 2, \dots, n}$ – **dominacja wierszowa**,
   b. $|\boldsymbol{a_{jj}}| > \sum_{\boldsymbol{i=1, j \neq i}}^{\boldsymbol{n}} |\boldsymbol{a_{ij}}|$, dla każdego $\boldsymbol{j = 1, 2, \dots, n}$ – **dominacja kolumnowa**.

Warunki te sprawdzane są z użyciem funkcji *checkDominance.m*:

```
function [is_dominant] = checkDominance(A)
%CHECKDOMINANCE Sprawdza czy macierz A jest silnie diagonalnie dominująca.
    is_dominant_row = true;
    is_dominant_col = true;
    for i=1:size(A, 1)
        sum_row = 0;
        sum_col = 0;
        for j=1:size(A, 1)
            if i ~= j
                sum_row = sum_row + abs(A(i,j));
                sum_col = sum_col + abs(A(j,i));
            end
        end
        if abs(A(i,i)) <= sum_row
            is_dominant_row = false;
        end
        if abs(A(i, i)) <= sum_col
            is_dominant_col = false;
        end
    end
    if is_dominant_row || is_dominant_col
        is_dominant = true;
    else
        is_dominant = false;
    end
end
```

W wyniku sprawdzenia dominacji macierzy $\boldsymbol{A}$ z punktu (a):

```
>> b = checkDominance(A)

b =

  logical

   1
```

co oznacza, że macierz $\boldsymbol{A}$ posiada silną diagonalną dominację wierszową lub kolumnową, natomiast dla macierzy $\boldsymbol{A}$ z punktu (b):

```
>> b = checkDominance(A)

b =

  logical

   0
```

co oznacza, że macierz $\boldsymbol{A}$ nie posiada tu zarówno silnej diagonalnej dominacji wierszowej jak i kolumnowej i wtedy iteracyjnie wyliczany wektor $\boldsymbol{x}$ nie zbiega do rozwiązania.

Funkcja, rozwiązująca układ $\boldsymbol{n}$ równań liniowych metodą Jacobiego z podanym błędem granicznym $\boldsymbol{\varepsilon_2}$ liczonym jako norma euklidesowa różnicy kolejnych przybliżeń $\boldsymbol{x}$ realizowana jest w *solveJacobi.m*:

```
function [x] = solveJacobi(A, b, e2)
    %SOLVEJACOBI Oblicza układ równań metodą Jacobiego
    %   sprawdzenie dominacji diagonalnej
    if(checkDominance(A) == false)
       disp("Macierz bez silnej dominacji diagonalnej!");
       return;
    end
    %   Stworzenie macierzy L, D i U
    n = size(A, 1);   % n - liczba równań
    L = zeros(n);
    D = zeros(n);
    U = zeros(n);
    x = zeros(n, 1);   % x to wektor wynikowy
    for i=1:n
        D(i,i) = A(i, i);
        for j=1:n
            if i>j
                L(i,j) = A(i,j);
            elseif i<j
                U(i, j) = A(i, j);
            end
        end
    end
    if(det(D)==0)
        disp("Macierz D nieosobliwa!");
        return;
    end
```

```
    lim_error = e2;
    % dopoki nie jest przekroczony błąd graniczny
    while lim_error >= e2
        x_before = x;
        % tworzymy wektor x dla kolejnego kroku przyblizenia
        for j=1:n
            sum_k = 0;
            for k=1:n
                sum_k = sum_k + (L(j,k) + U(j,k))*x_before(k);
            end
            x(j) = -1 / D(j,j) * (sum_k - b(j));
        end
        % na koncu roznica w postaci normy z różnicy kolejnych przyblizen
        lim_error = norm(x - x_before, 2);
    end
end
```

Z powodu, że macierz $\boldsymbol{A}$ z podpunktu (b) nie posiada silnej dominacji, nie liczyłem błędu $\boldsymbol{\varepsilon_1}$ dla macierzy $\boldsymbol{A}$ i wektora $\boldsymbol{b}$ z (b), tylko skupiłem się na wyliczaniu dokładności rozwiązania przykładu (a) w zależności od liczby układu równań $\boldsymbol{n}$ dla kilku możliwych wartości błędu granicznego $\boldsymbol{\varepsilon_2}$. Skrypt realizujący to znajduje się w pliku *exercise3.m*.

a) Dla błędu granicznego $\boldsymbol{\varepsilon_2 = 0.00000001}$:

n	$\varepsilon_1 = \|\|Ax - b\|\|$
5	$0.7167 * 10^{-7}$
10	$0.7687 * 10^{-7}$
50	$0.8318 * 10^{-7}$
100	$0.8292 * 10^{-7}$
200	$0.8215 * 10^{-7}$

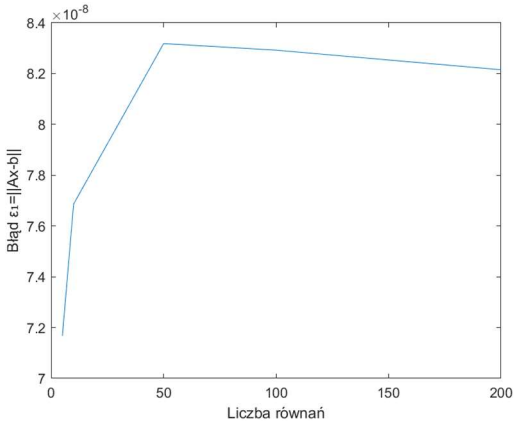


Tabela 3 i Wykres 3 – reprezentują zależnośc błędu rozwiąania ε_1 do liczby równań liniowych n do rozwiązania układu równań liniowych (a) dla metody Jacobiego dla błędu granicznego $\varepsilon_2 = 0.00000001$.

b) Dla błędu granicznego $\boldsymbol{\varepsilon_2 = 0.0001}$:

n	$\varepsilon_1 = \|\|Ax - b\|\|$
5	$0.5564 * 10^{-3}$
10	$0.7585 * 10^{-3}$
50	$0.7950 * 10^{-3}$
100	$0.8926 * 10^{-3}$
200	$0.8580 * 10^{-3}$

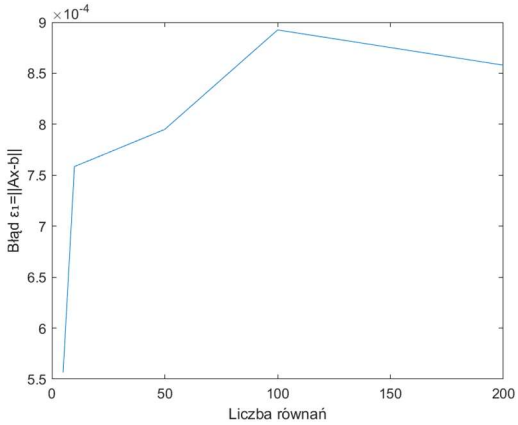


Tabela 4 i Wykres 4 – reprezentują zależność błędu rozwiąania ε_1 do liczby równań liniowych n do rozwiązania układu równań liniowych (a) dla metody Jacobiego i błędu granicznego $\varepsilon_2 = 0.0001$.

c) Dla błędu granicznego $\boldsymbol{\varepsilon_2 = 0.01}$:

n	$\varepsilon_1 = \|\|Ax - b\|\|$
5	0.0692
10	0.0783
50	0.0875
100	0.0761
200	0.0879

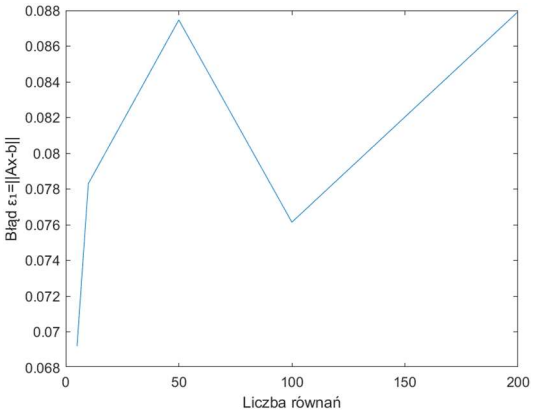


Tabela 5 i Wykres 5 – reprezentują zależność błędu rozwiąania ε_1 do liczby równań liniowych n do rozwiązania układu równań liniowych (a) dla metody Jacobiego i błędu granicznego $\varepsilon_2 = 0.01$.

W tym przypadku nie widać zależności wartości błędu rozwiązania $\boldsymbol{\varepsilon_1}$ od liczby równań liniowych do rozwiązania $\boldsymbol{n}$, co oznacza, że błąd wyniku będzie w przypadku metody Jacobiego nieprzewidywalny, jednak dla nisko dobranego parametru błędu granicznego (nie większego niż $\boldsymbol{0.001}$), metoda daje poprawne wyniki z, w miarę, dużą dokładnością – można zauważyć, że otrzymana dokładność błędu rozwiązania $\boldsymbol{\varepsilon_1}$ zależy tu od dokładności samego błędu granicznego $\boldsymbol{\varepsilon_2}$ i w zależności od tego, jaki błąd rozwiązania jest dla nas akceptowalny, tak możemy sterować parametrem błędu granicznego.