[MNUM] projekt 1 – Marcin Dziedzic

Zadanie 1.4

1. Program wyznaczający dokładność maszynową komputera.

Dokładność maszynowa komputera jest to maksymalny względny błąd reprezentacji zmiennoprzecinkowej. Od strony programu jest zależny od typu danych jakiego używamy przy obliczeniach, czyli od tego czy rozważaną liczbą jest typ pojedynczej precyzji czy też podwójnej. Od strony sprzętowej zaś zależy od liczby rejestrów w procesorze komputera, gdyż to właśnie w nich przechowywane są tymczasowe wyniki obliczeń.
Dokładność maszynowa oznaczana przez *eps* jest z punktu widzenia operacji arytmetycznych bardzo ważną wielkością. Od niej zależy dokładność wykonywanych obliczeń. Przykładowo dla liczby 1/3, która jest okresowa 100*(1/3) dla liczby pojedynczej precyzji na moim komputerze nie wynosi 333.3... lecz 33.333351. Błędy podczas operacji arytmetycznych kumulują się. Znajomość eps pozwala na wyliczenie maksymalnego względnego błędu tych operacji.
Epsilon jest wielkością, która wprost wynika z liczby bitów mantysy dla danej precyzji.

1) *$Eps=2^{-t}$ - gdzie t oznacza ilość bitów mantysy.*

Dla liczby pojedynczej precyzji (32 bity) w zgodzie ze standardem IEEE 754 mantysa ma 23 bity,
dla liczby podwójnej precyzji (64 bity) mantysa ma 52 bity.

Zatem wprost z równania 1) mamy dla liczby pojedynczej precyzji

2) *eps=2-23=1.1921e-07*

dla podwójnej precyzji:

3) *eps=2-52=2,2204e-16*

Algorytm do wyliczenia dokładności maszynowej:

Epsilon jest również definiowane jako najmniejsza liczba, która po dodaniu do liczby 1 jest różny od 1.

4) *1+eps~=1*

zatem wystarczy napisać pętlę, która będzie dzieliła eps początkowo równy 1 przez 2 do momentu, gdy wynik zsumowany z 1 będzie równy 1, potem wystarczy pomnożyć eps przez 2 i dokładność maszynowa jest już wyznaczona.
Można by zadać pytanie: skoro eps jest najmniejszą liczbą dla której 1+eps!=1 to przecież lepiej jest dzielić eps przez 1.5 albo jeszcze lepiej przez 1.1. Otóż nie gdyż procesor operuje na liczbach binarnych i definicja eps podana jako równanie 1) jest inna.

```
% Marcin Dziedzic
% projekt 1
% zadanie 1.4.1

tic()
eps=1;

disp('duble precision: ')


while eps+1~=1 % eps jest najmniejsza wartoscia taka ze po dodaniu do 1
wynik jest rozny od 1
eps=eps/2; % dzielimy przez 2 gdyz liczby sa zapisane w rejestrach binarnie
end

eps=eps*2; % mnoze przez 2 gdyz wychodzi z petli gdy 1+eps=1 a eps jest
liczba taka ze 1+eps~=1
eps

toc()
disp('single precision: ')


tic()
eps=1;
eps=single(eps);%na single precion

while eps+1~=1
eps=eps/2;
end

eps=eps*2;
eps
toc()
```

```
>> MarcinDziedzic_14_1
duble precision:

eps =

   2.2204e-16

Elapsed time is 0.005760 seconds.
single precision:

eps =

  single

  1.1921e-07

Elapsed time is 0.014058 seconds.
```

2. Metoda eliminacji Gaussa-Jordana

Wstęp

Dzielimy pierwszy wiersz przez element centralny $a_{11}^{(1)}$, a następnie zerujemy pierwsza kolumnę z wyjątkiem elementu w pierwszym wierszu.

Dzielimy drugi wiersz przez element centralny $a_{11}^{(1)}$, następnie, postępując analogicznie jak w eliminacji Gaussa, zerujemy cala druga kolumnę oprócz elementu w drugim wierszu – tzn. Zerujemy tez elementy nad diagonala (w taki sam sposób).

Nakład obliczeń jest rzędu $D = O\left(\frac{1}{2}n^3\right)$, $M = O(\frac{1}{2}n^3)$. Metodę można stosować przy jednokrotnym rozwiązaniu układu równina liniowych (nie dostajemy żadnego rozkładu macierz), a szczególnie przy rozwiązaniu tzw. Obciętego układu równań.

Implementacja

Rozpocząłem od implementacji funkcji tworzącej poszczególne macierze a), b) i c)

```
function [y] = md_prepare_data_a (n)

    A = zeros(n, n+1);

    for w = 1:n
        for k = 1:n
            if w == k
                A(w,k) = 10;
            elseif (w == k-1) || (w == k+1) % todo
                A(w,k) = 3;
            else
                A(w,k) = 0;
            end
        end
    end

    for w = 1:n
        for k = n+1
            A(w,k) = 0.5 + (2.5 * w);
        end
    end

    y = A;

end
```

```
function [y] = md_prepare_data_b (n)

    A = zeros(n, n+1);

    for w = 1:n
        for k = 1:n
            if w == k
                A(w,k) = 1/4;
            else
                A(w,k) = 3 * (w-k) + 4;
            end
        end
    end
```

```
    for w = 1:n
        for k = n+1
            A(w,k) = 3.5 - 0.4*w;
        end
    end

    y = A;

end
```

```
function [y] = md_prepare_data_c (n)

    A = zeros(n, n+1);

    for w = 1:n
        for k = 1:n
            A(w,k) = 1 / (2 * (w+k+1));
        end
    end

    for w = 1:n
        for k = n+1
            A(w,k) = 5 / (3*w);
        end
    end

    y = A;

end
```

Oto funkcja zawierajaca algorytm Gaussa-Jordana:

```
% k iterator kolumny
% w iterator wiersza
% A aktualny wiersz ktorym zeruje
% x(wiersz,kolumna)

function [y] = md_gauss_jordan (x)
    for k = 1:(length(x)-1) % iterujemy po kazdej kolumnie
        A = x(k,:);
        A = A/A(k); % dziele aby uzyskac 1 na przekontnej
        x(k,:) = A;

        for w = 1:(length(x)-1) % iterujemy po kazdym wierszu w danej
kolumnie
            if k ~= w
               x(w,:) = A * x(w,k) * -1 + x(w,:); % zerowanie kolumny
oprocz przekatnej
            end
        end
    end
%       y = x;
     y = x(:,length(x));

end
```

Oraz funkcje pozwalające wyliczyć blad obliczen

```
function [R1] = md_blad_residuum (A, x)

    w = size(A,1); % ilosc wierszy
    R = zeros(w);
    for i = 1:w
        R(:,i) = A(:,i);
    end

    b = A(:,i+1);

    R1 = R * x - b;
End
```

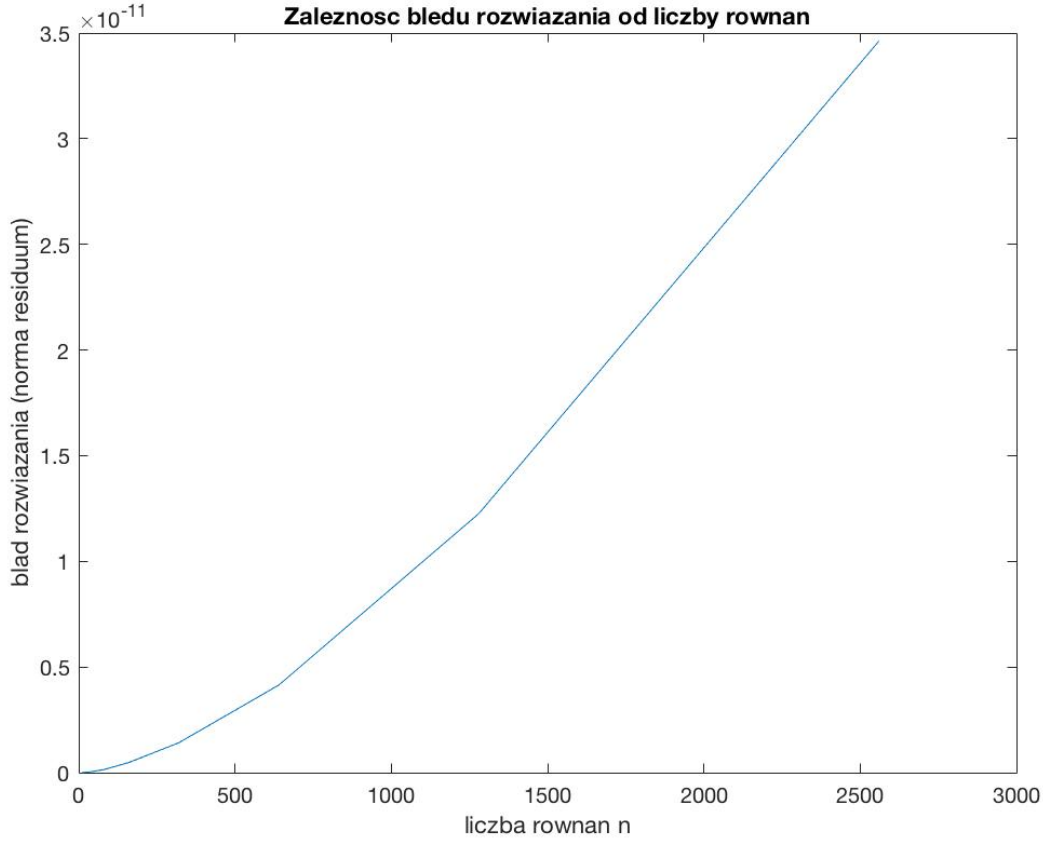
```
function [y] = md_norma_residuum (x)
    w = size(x,1);
    sum = 0;
    for i = 1:w
       sum = sum + (x(i,1) * x(i,1));
    end
    y = sqrt(sum);

end
```

W końcu funkcja „main” pozwalajaca przeprowadzić niezbędne obliczenia oraz stowrztc wykresy zaleznosci bledu rozwiazan od liczby rownan:

```
function md_exec (max_time)

    n = 10;
    N = [];
    E = [];
    while true
        N = [N ; n];
        tic()
        A = md_prepare_data_c(n);
        n
        B = md_gauss_jordan(A);
        C = md_blad_residuum(A,B);
        D = md_norma_residuum(C);
        E = [E ; D];
        if max_time < toc()
            break
        end
        n = n * 2;
    end
    N
    E
    plot(N,E);
end
```

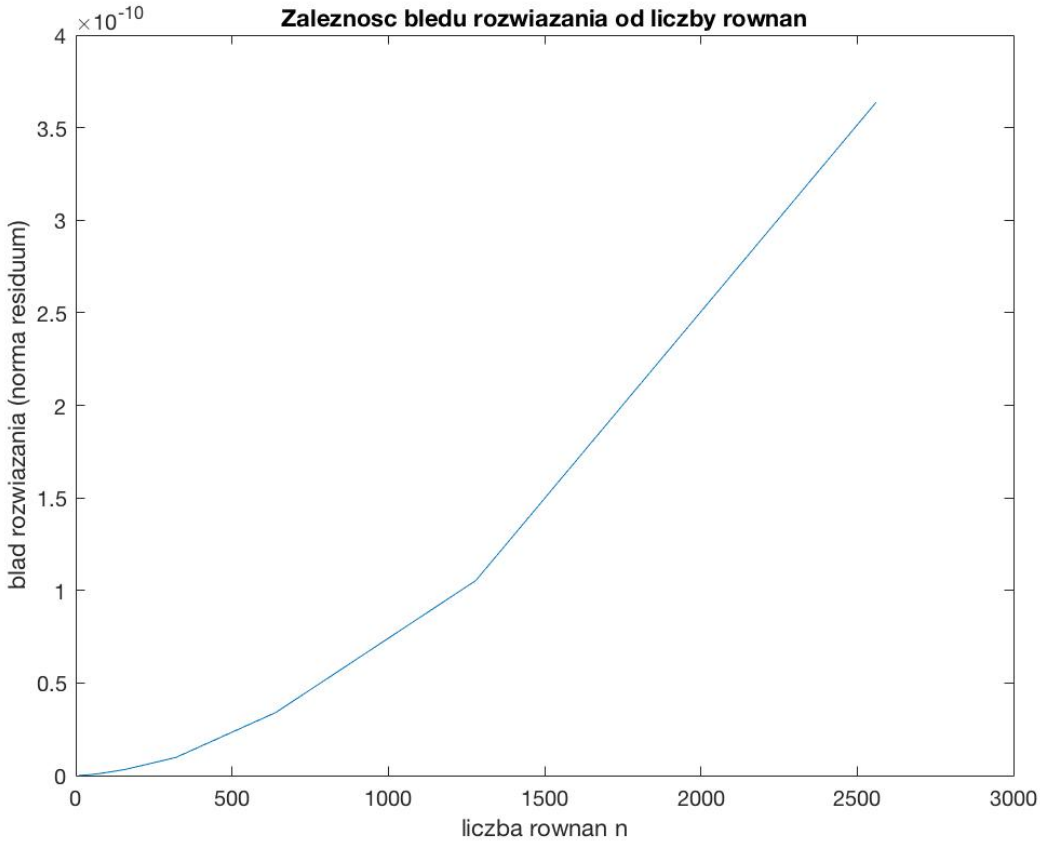
Wyniki obliczeń:

a)



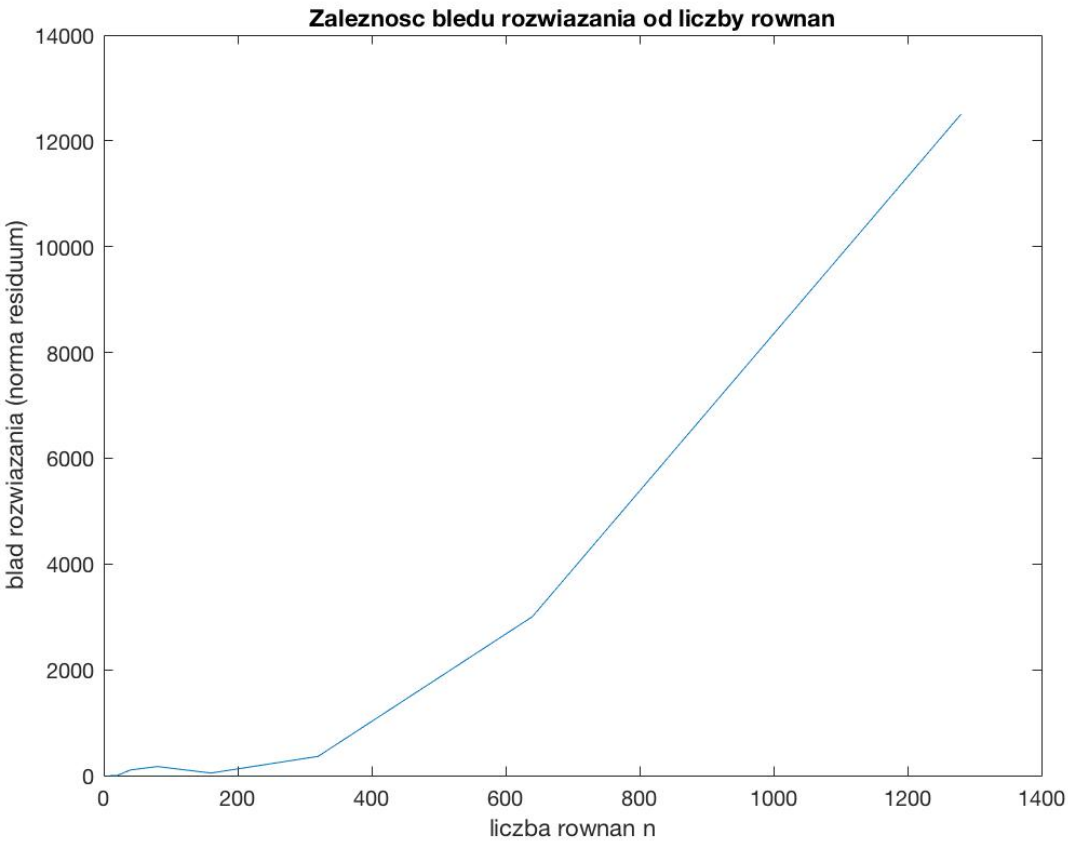
Z wykresu widać, że błąd wyniku rośnie nieliniowo względem ilości równań. Natomiast błędy te są rzędu e^(-11). Możemy z tego wywnioskować, że rozwiązanie podpunktu a) tą metodą daje dokładne rezultaty.

b)



Z wykresu widać, że błąd wyniku rośnie nieliniowo względem ilości równań. Natomiast błędy są o jeden rząd większe niż w poprzednim przypadku e^(-10), również wzrost ich jest szybszy niż w podpunkcie a). Mimo to, możemy stwierdzić, że ta metoda również jest dokładna dla podpunktu b).

c)



Badana metoda zupełnie nie sprawdzała się dla większej ilości równań. O ile błędy rozwiązania dla układów równań mniejszych od 200 mogłaby być akceptowalna przy niektórych wyliczeniach, to dla powyżej 200 równań wyniki ciężko nazwać rozwiązaniami – są to wartości zupełnie odmienne.

3. Metoda iteracyjna Gaussa-Seidela

Rozwiązywanie układu n równań liniowych A×x = b metodą iteracyjną Gaussa-Seidela.
Aby móc przeprowadzić metodę iteracyjną Gaussa-Seidela, na początku musimy sprawdzić, czy macierz A spełnia warunek dostateczny zbieżności, czyli silnej dominacji diagonalnej (oznacza to, że suma wszystkich elementów w wierszu poza diagonalnym, nie może być większa od elementu diagonalnego).
Gdy warunek ten jest spełniony możemy przejść do rozkładu macierzy A na macierze: L – macierz poddiagonalną, U – naddiagonalną i D – diagonalną. W następujący sposób:

A = L + D + U

Przykład:

$$\underset{\mathbf{A}}{\begin{vmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{vmatrix}} = \underset{\mathbf{L}}{\begin{vmatrix}0 & 0 & 0\\ 4 & 0 & 0\\ 7 & 8 & 0\end{vmatrix}} + \underset{\mathbf{D}}{\begin{vmatrix}1 & 0 & 0\\ 0 & 5 & 0\\ 0 & 0 & 9\end{vmatrix}} + \underset{\mathbf{U}}{\begin{vmatrix}0 & 2 & 3\\ 0 & 0 & 6\\ 0 & 0 & 0\end{vmatrix}}$$

Pojedynczą iterację możemy zapisać w postaci:

$$D * x^{(i+1)} = -L * x^{(i+1)} - U * x^{(i)} + b;\ \ i = 0, 1, 2, \dots$$

Otrzymujemy stąd znowu układ n równań skalarnych. Składowe nowego wektora wyznaczamy kolejno, poczynając od pierwszej:
gdzie $w^{(i)} = U * x^{(i)} - b$.

$$x_1{}^{(i+1)} = -\frac{w_1^{(i)}}{d_{11}},$$

$$x_2{}^{(i+1)} = -\frac{-l_{21} * x_1^{(i+1)} - w_2^{(i)}}{d_{22}},$$

$$x_3{}^{(i+1)} = -\frac{-l_{31} * x_1^{(i+1)} - l_{32} * x_2^{(i+1)} - w_3^{(i)}}{d_{33}},$$

itd.

Dla tej metody ważne jest ustalenie warunku, który gdy zostanie spełniony, ma spowodować przerwanie iterowania i wypisanie wyniku. W tym przypadku będzie to osiągnięcie założonej dokładności.
Kod źródłowy programu

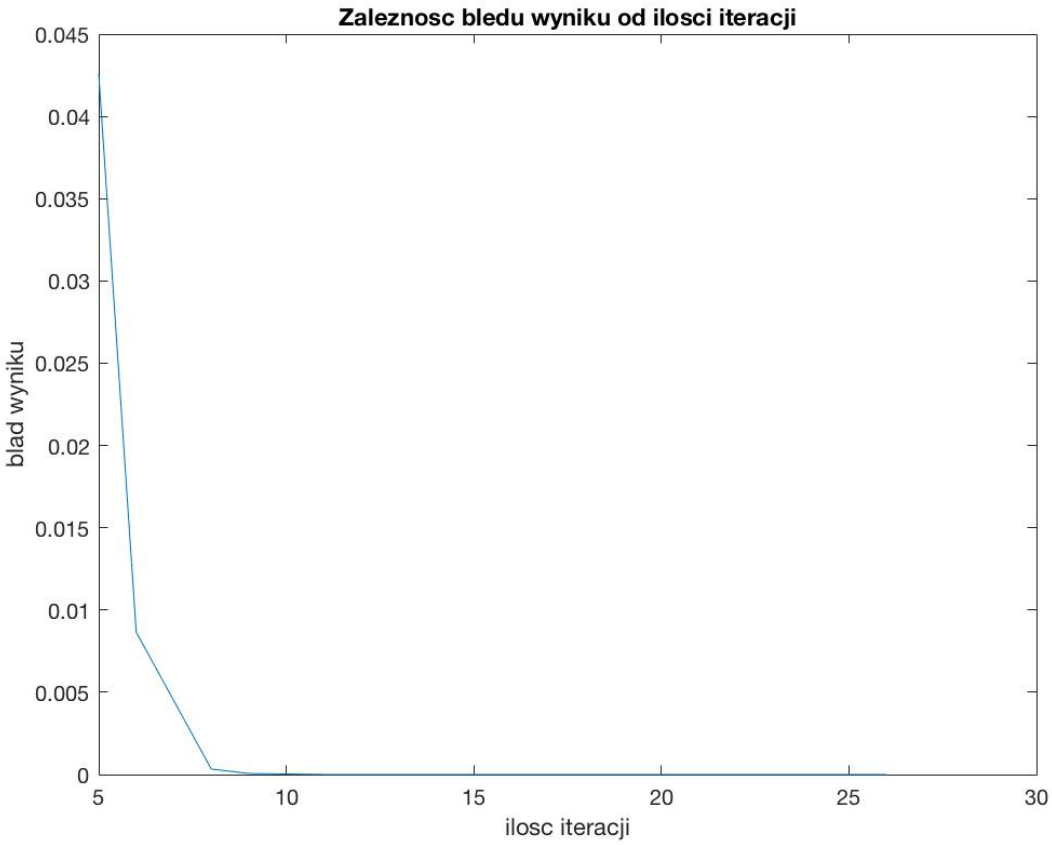
```
function [y] = md_gauss_seidel (A, e)

    % sekcja inicjalizacyjna
    w = size(A,1);
    k = size(A,2);
    r = 1;
    x = zeros(w,1);
    b = A(:,k);
    A = A(:, 1:k-1);

    %Sprawdzenie warunku dostatecznego zbieznosci - silnej dominacji
    %diagonalnej macierzy A
    for i = 1:w
        %Sumujemy wszystkie elementy w wierszu, poza diagonalnym
        sum = 0;
        for j = 1:w
            if i~=j
                sum = sum + abs(A(i,j));
            end
        end
        %Jesli suma jest wieksza od elementu to warunek nie jest spelniony
```

```
        if sum > abs(A(i,i))
            disp('Warunek silnej dominacji diagonalnej nie jest
spelniony');
        return
        end
    end

    % petla glowna
    while(r>e)
        z = x; % zachowanie wyniku poprzedniej iteracji
        for i = 1:w
            % wyliczenie nowych wartosci dla wektora x
            x(i,1) =  (1/A(i, i)) * (b(i) - A(i,:) * x + A(i, i) * x(i));
        end
        % liczymy blad z normy euklidesowej
        r = x-z;
        r = norm(r);
    end
    y = x;
end
```

---

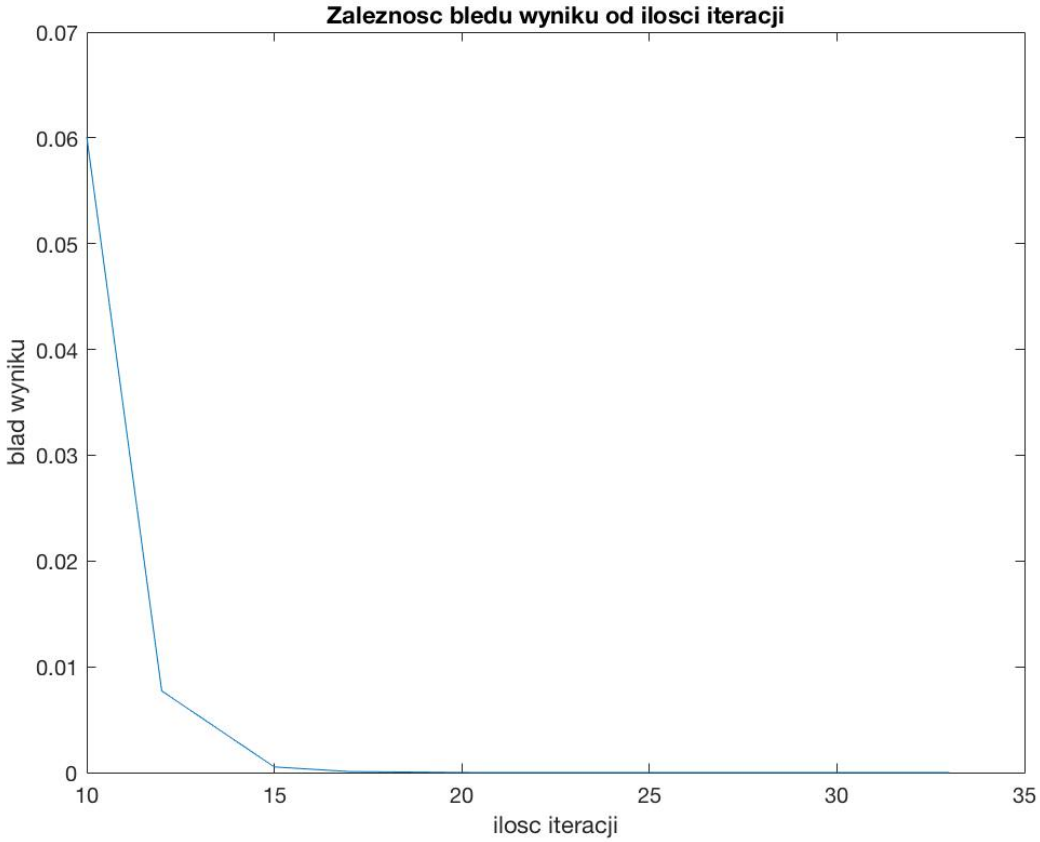
wyniki:

```
>> md_gauss_seidel(md_prepare_data_d, 0.00001)

ans =

    0.1122
    1.4927
    0.5882
    1.7046
```



Jak możemy zauważyć na wykresie dla małej liczby iteracji błąd rozwiązania jest duży, ale bardzo szybko maleje wraz ze wzrostem liczby iteracji. Dla ilości iteracji większej od 10 błąd osiąga wartość bardzo bliską zeru. Można więc przyjąć, że metoda Gaussa-Seidela daje w tym przypadku dokładne rezultaty.

Dane z zadania 2 z podpunktu a) (ilość równań = 1280):



Jak widać na wykresie błąd wyniku bardzo szybko zbiega do 0. Po 20 iteracji jest on już równy prawie 0. Wnioskujemy z tego, że dla tych danych metoda Gaussa- Seidela daje dobre rezultaty. Ponadto w tym przypadku jest ona szybsza niż metoda Gaussa-Jordana z zadania 2.
Dane z zadania 2 z podpunktu b) i c):
Dla tych danych metoda Gaussa-Seidela nie działa, ponieważ nie spełniają one warunku dostatecznego dla tej metody, czyli silnej dominacji diagonalnej. Po uruchomieniu procesu dla tych danych od razu dostajemy informacje 'Warunek silnej dominacji diagonalnej nie jest spelniony' i proces zostaje przerwany. Dzieje się tak, ponieważ w tych przypadkach ciągi $x^{(i)}$ były rozbieżne.

Podsumowanie

Porównując oba algorytmy rozwiązywania układów n równań liniowych, możemy zauważyć, że metoda Gaussa-Seidela daje lepsze rezultaty dla odpowiedniej liczby iteracji, ponadto jest ona znacznie szybsza. Natomiast nie jest ona tak uniwersalna jak metoda Gaussa-Jordana, dzięki której możemy rozwiązać dużo większą liczbę układów równań, ale dla liczby równań przewyższającej 1280, staje się ona bardzo nieefektywna, ponieważ czas obliczenia wektora wynikowego dla takiej liczby równań przekracza 5 minut, ponadto nie rośnie liniowo, lecz wykładniczo.

W związku z tym metoda Gaussa-Seidela powinna być stosowana dla macierzy, które spełniają odpowiednie warunki (warunek dostateczny dla tej metody: silna dominacja diagonalna), natomiast dla innych metoda Gaussa-Jordana.