[MNUM] projekt 1 – Marcin Dziedzic

Zadanie 1.4

# 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

1. *eps=2-23=1.1921e-07*

dla podwójnej precyzji:

1. *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.

1. *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.

# Metoda eliminacji Gaussa-Jordana

Dzielimy pierwszy wiersz przez element centralny , a następnie zerujemy pierwsza kolumnę z wyjątkiem elementu w pierwszym wierszu.

Dzielimy drugi wiersz przez element centralny , 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 , . Metodę można stosować przy jednokrotnym rozwiązaniu układu równina liniowych (nie dostajemy żadnego rozkładu macierz), a szczególnie przy rozwiązaniu tzw. Obciętego układu równań.

% 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

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

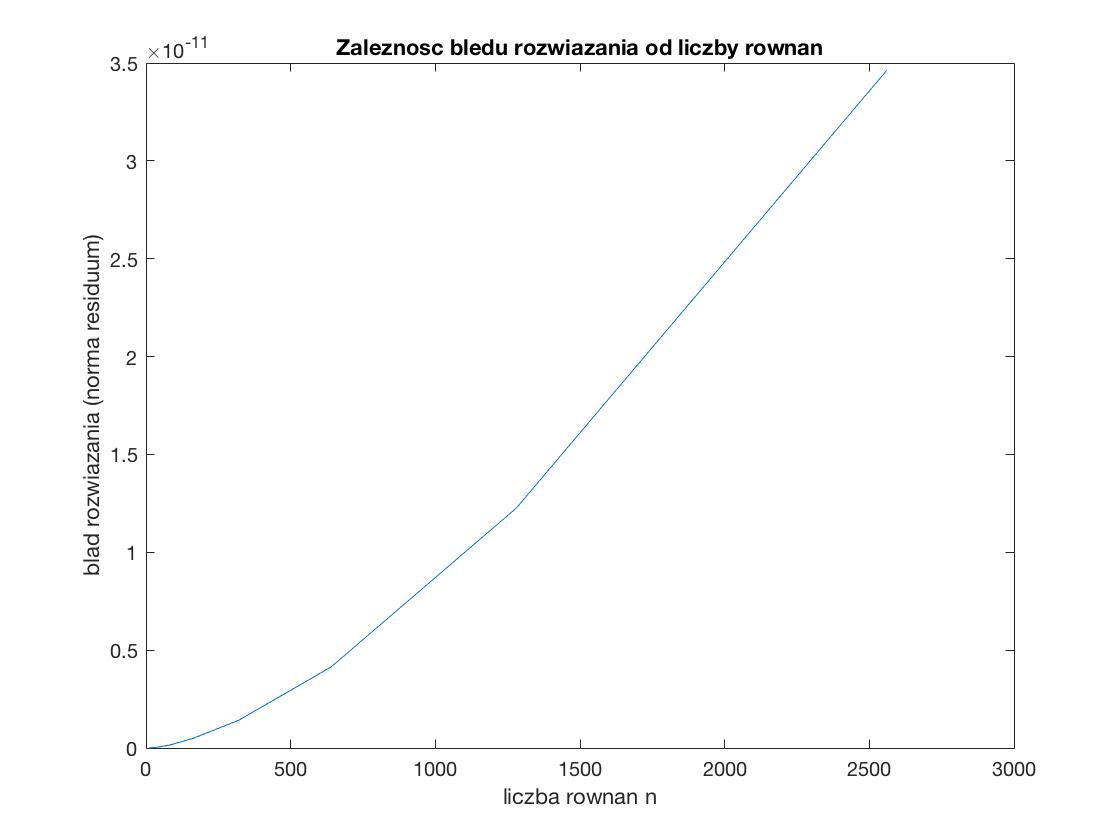
N

E

plot(N,E);

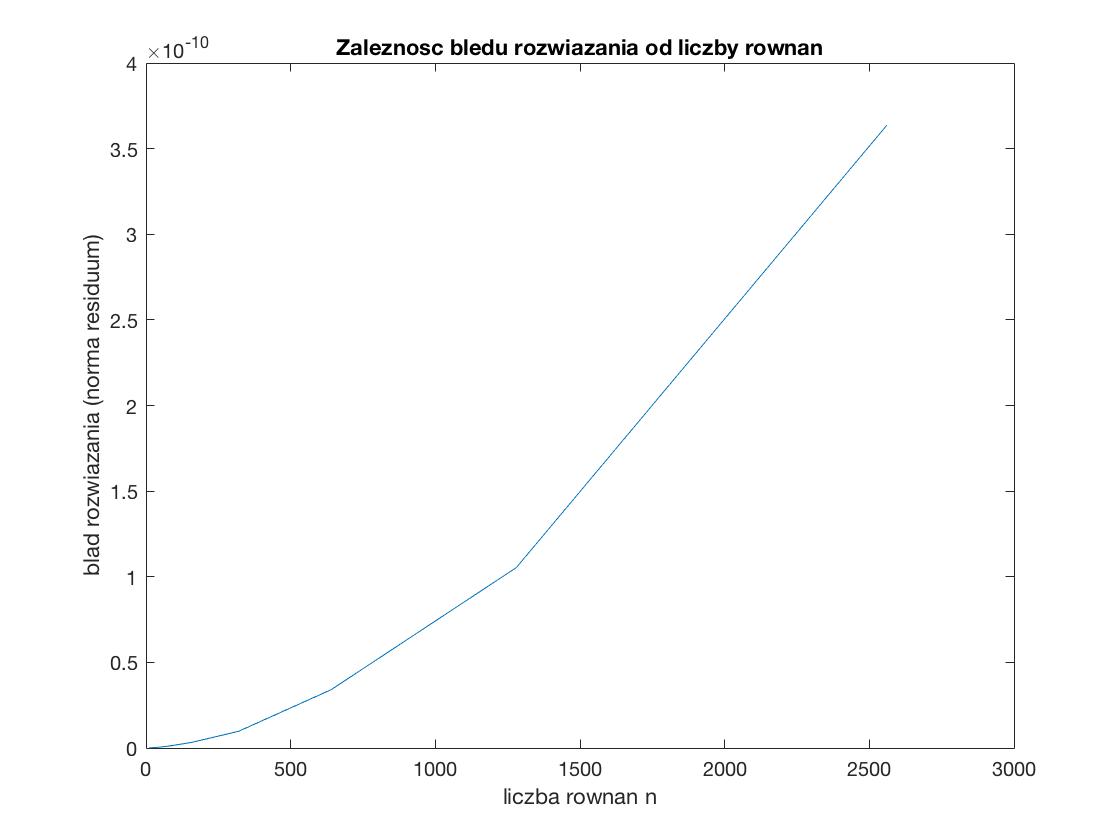
end

a)



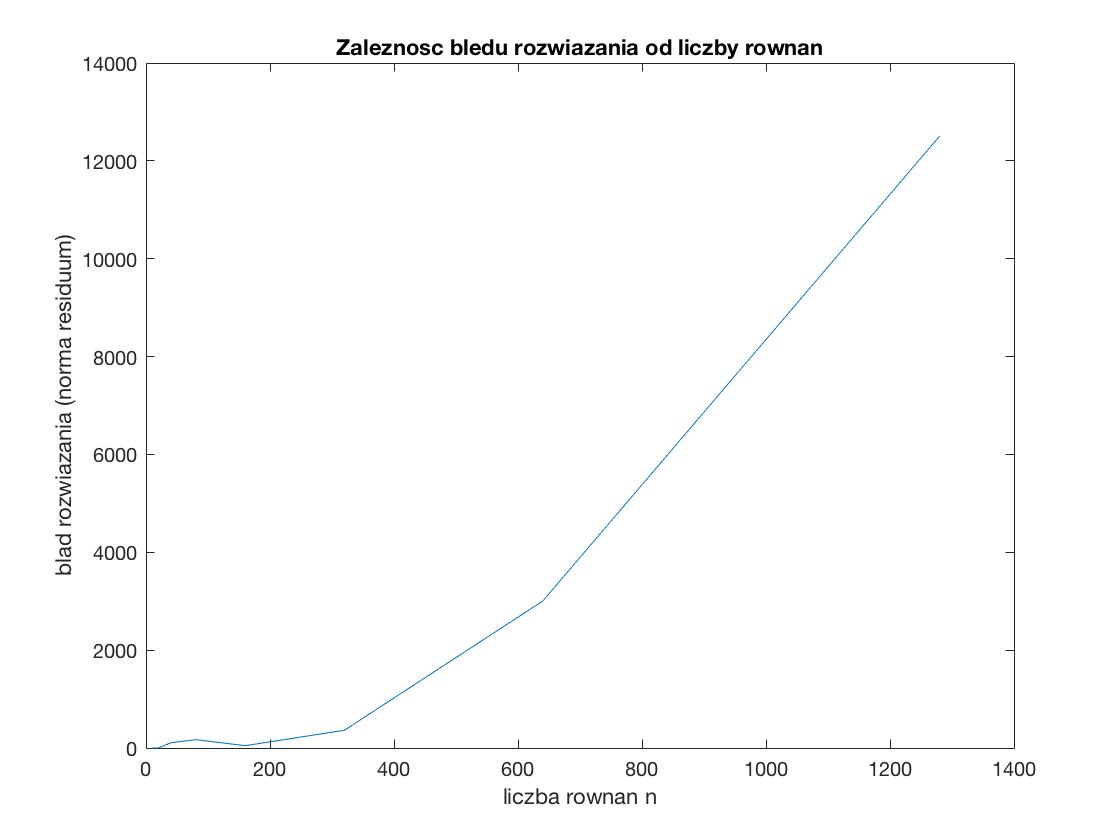
Z wykresu widać, że błąd wyniku rośnie nieliniowo względem ilości równań. Natomiast błędy te są rzędu e^(-12). Możemy z tego wywnioskować, że rozwiązanie podpunktu a) tą metodą daje dokładne rezultaty.

b)



Z wykresu widać, że błąd wyniku rośnie nieliniowo względem ilości równań. Natomiast błędy są większe niż w poprzednim przypadku bo są rzędu e^(-5), również wzrost ich jest szybszy niż w podpunkcie a). Mimo to, możemy stwierdzić, że ta metoda również jest dokładna dla podpunktu b).

c)



Z wykresu widać, że błąd wyniku nie rośnie wraz ze wzrostem ilości równań. Dla małej ilości równań bardzo szybko rośnie, potem maleje, ale znowu w okolicach 1500 równań zaczyna rosnąć, a przy 2500 znowu maleć. Z wykresu wnioskujemy, że i ta metoda jest w miarę dokładna, bo rząd błędu wynosi około e^(-5). Natomiast nie możemy przewidzieć, mniej więcej wielkości błędu, jak to miało miejsce w poprzednich podpunktach.

# 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:

**A L D U**

Pojedynczą iterację możemy zapisać w postaci:

Otrzymujemy stąd znowu układ n równań skalarnych. Składowe nowego wektora wyznaczamy kolejno, poczynając od pierwszej:

gdzie

,

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);

% 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)

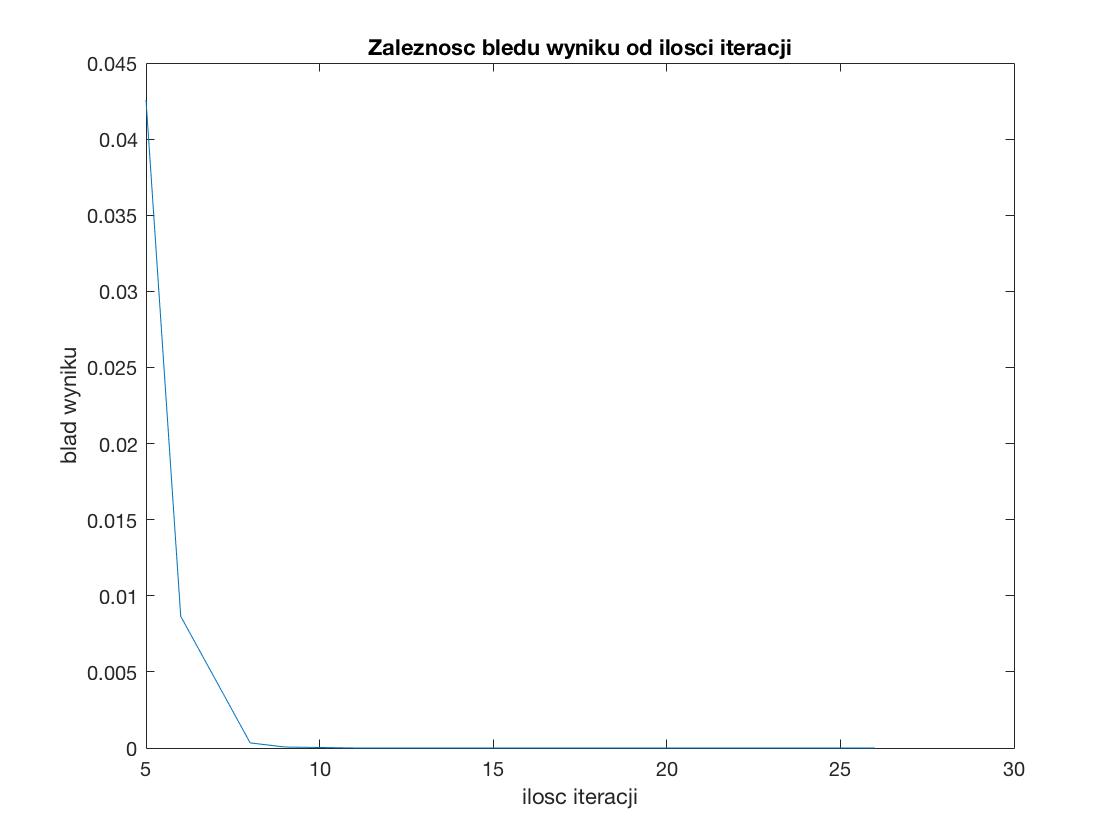
ans =

0.1122

1.4927

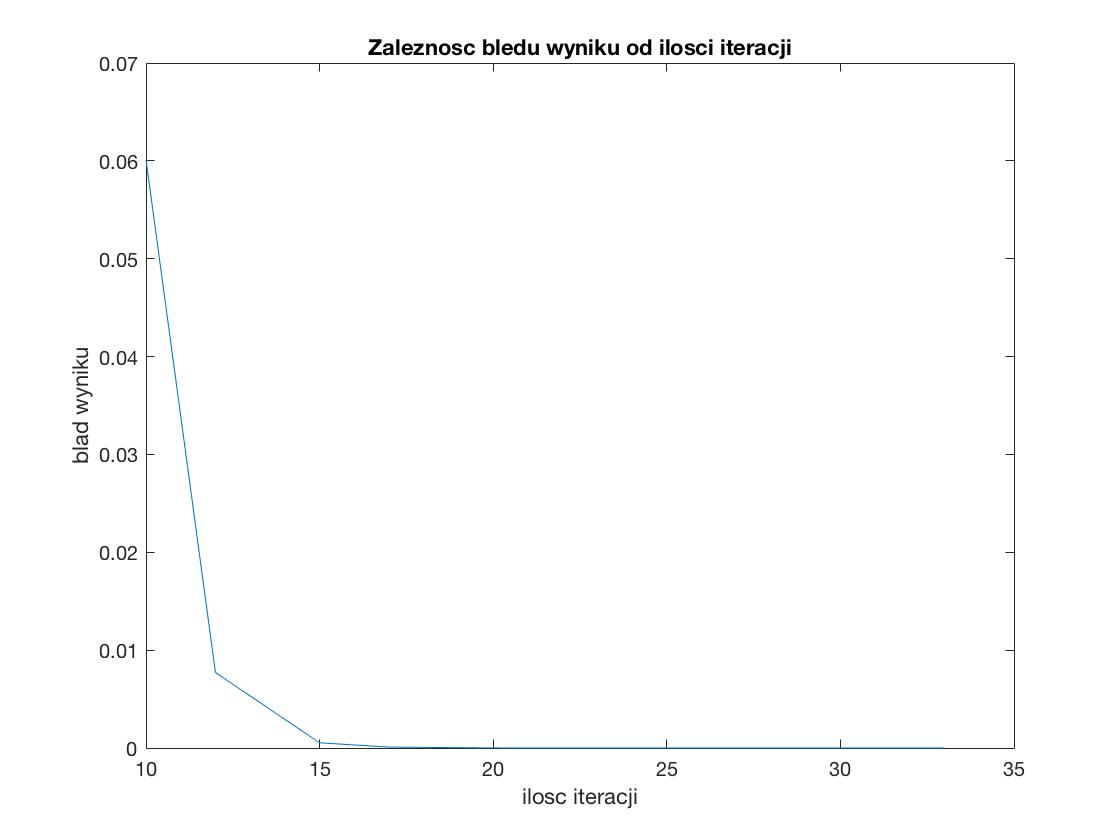
0.5882

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 = 2560):



Jak widać na wykresie błąd wyniku bardzo szybko zbiega do 0. Po 30 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 rozkładu LU 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 były rozbieżne.