MNUM Projekt 1.

Autor:	Numer projektu:	Prowadzący:	Liczba punktów:
Mariusz Słapek	1.42	dr inż. Adam Krzemianowski	

Kody do poszczególnych programów zostały umieszczone na końcu sprawozdania.

Zadanie 1

1. Treść:

Napisać program wyznaczający dokładność maszynową komputera i wyznaczyć ją na swoim komputerze.

2. Ogólny zarys:

Ludzie obliczając dane zadania *„na kartce”* liczą je w sposób analityczny (model idealny – wszystkie obliczenia wykonujemy dokładnie) - inaczej wygląda sytuacja licząc coś na komputerze (nie potrafią one wykonywać obliczeń analitycznych, liczą numerycznie). Metody numeryczne – niezależnie od metody – obarczone są niepewnością wynikającą z dokładności maszynowej.

3. Dokładność maszynowa:

a)definicja:

Wykonując elementarne obliczenia na komputerze możemy zauważyć, iż często nie są to dokładne wyniki. Wynik elementarnych działań (tj. dodawanie, odejmowania itd.) możemy zapisać w postaci:

$$fl(x \pm y) = (x \pm y)*(1+\varepsilon),$$

gdzie:
$\varepsilon \leq |eps|$ - przyjmujemy, że błąd elementarnych operacji arytmetycznych to błąd zaokrąglenia wyniku operacji (standard IEEE 754)
(analogicznie dla innych działań tj. mnożenia, dzielenie, pierwiastkowania)

Dokładność maszynową *eps* możemy określić jako najmniejszą dodatnią liczbę maszynową g taką, że zachodzi relacja *fl(1+g)* >1.
Im mniejszy jest *eps*, tym większa jest precyzja wykonywanych obliczeń.

b)zależność:

Dokładność maszynowa zależy w dużej mierze od dwóch czynników – typów danych oraz rozmiaru rejestrów.

4. Wyniki:

Stworzyłem program liczący dokładność maszynową dokładnie z definicją przytoczoną powyżej. Dokładność w programie oznaczę przez zmienną i zainicjuje wartością 1, aby osiągnąć wartość nierozróżnialną przez komputer korzystam z pętli *while*. Program wyjdzie z pętli gdy zostanie spełniony warunek zgodny z definicją. Oznacza to, że komputer nie rozróżnia tej liczby z liczbą 1.
Z programu wyszło mi, iż dokładność mojego komputera jest równa: 2.2204e-16.
Otrzymany wynik jest w przybliżeniu równy wzorcowej wartości dokładności maszynowej dla

systemu 64-bitowego wg standardu *IEEE 754*.

Porównuję działanie mojego algorytmu z działaniem wbudowanej funkcji programu “eps”. Wynik jest taki sam jak wynik powstały w wyniku działania mojego algorytmu.

Zadanie 2

1. Treść:

Proszę napisać program rozwiązujący układ n równań liniowych $Ax = b$ wykorzystując eliminację Gaussa z częściowym wyborem elementu podstawowego. Proszę zastosować program do rozwiązania podanych niżej układów równań dla rosnącej liczby równań $n = 10, 20, 40, 80, \ldots$. Liczbę tych równań proszę zwiększać aż do momentu gdy czas potrzebny na rozwiązanie układu staje się zbyt duży (lub metoda zawodzi).
Dla każdego rozwiązania proszę obliczyć błąd rozwiązania (liczony jako norma residuum) i dla każdego układu równań wykonać rysunek zależności tego błędu od równań n.

2. Ogólny zarys:

Mamy wiele metod rozwiązywania układów równań liniowych (graficzne, podstawiania itd.). W metodach numerycznych stosujemy inne metody, tj. metody eliminacyjne, do których należą m.in.: *metoda eliminacji Gaussa* oraz *metoda Gaussa-Jordana*.

W moim zadaniu będę układ równań będę rozwiązywał metodą *eliminacji Gaussa* z częściowym wyborem elementu podstawowego.

3. Metoda eliminacji Gaussa:

Metoda eliminacji Gaussa zaliczana jest do metod skończonych - wynik otrzymujemy po skończonej (ściśle określonej) liczbie przekształceń zależnej od wymiarowości zadania.

a) algorytm metody:

Algorytm eliminacji Gaussa możemy podzielić na dwa etapy:
1. Eliminacja zmiennych – w wyniku przekształcenia macierzy A i wektora b (jako macierz rozszerzona);
2. Postępowanie odwrotne – stosujemy algorytm rozwiązania układu z macierzą trójkątną.

Algorytm do etapu eliminacji zmiennych:
1. wyzerowanie elementów kolumny pierwszej, oprócz elementu w wierszu pierwszym (krok 1);
2. wyzerowanie elementów kolumny drugiej, z wyjątkiem elementów w wierszu 1 i 2, w sposób analogiczny do poprzedniego kroku (krok 2);
3. itd. aż po n -1 krokach uzyskujemy układ równań $\boldsymbol{A\,x = b}$, gdzie **A** to macierz trójkątna górna;

W metodzie *częściowej eliminacji Gaussa z wyborem elementu głównego* jako element główny w *k-tym* kroku wybieramy, spośród elementów kolumny, element o największym module (w wierszu *i-tym*). Następnie zamieniamy wiersz *i-ty* z *k-tym* i stosujemy standardowy algorytm eliminacji Gaussa dla *k-tego* kroku. Wykonując algorytm *eliminacji Gaussa* czasami spotykamy sytuację, że jakiś element na diagonali jest równy 0. Wówczas możemy tego uniknąć stosując algorytm *eliminacji Gaussa z wyborem elementu głównego*. Wybór ten może być częściowy lub pełny. Ponadto stosowanie w metodzie *eliminacji Gaussa* z wyborem elementu głównego prowadzi do najmniejszych błędów numerycznych.

b) norma residuum – błąd rozwiązania:

W programie do wyliczenia błędu rozwiązania posłużyłem się najpierw policzeniem wektora residuum (przedstawionego poniżej) wg następującego wzoru:

$$\boldsymbol{residuum = A * X - B}$$

gdzie:
A – macierz współczynników układu;
X – macierz obliczonych wartości zmiennych
B – wektor wyników.

Następnie policzyłem normę euklidesową wektora **residuum**.

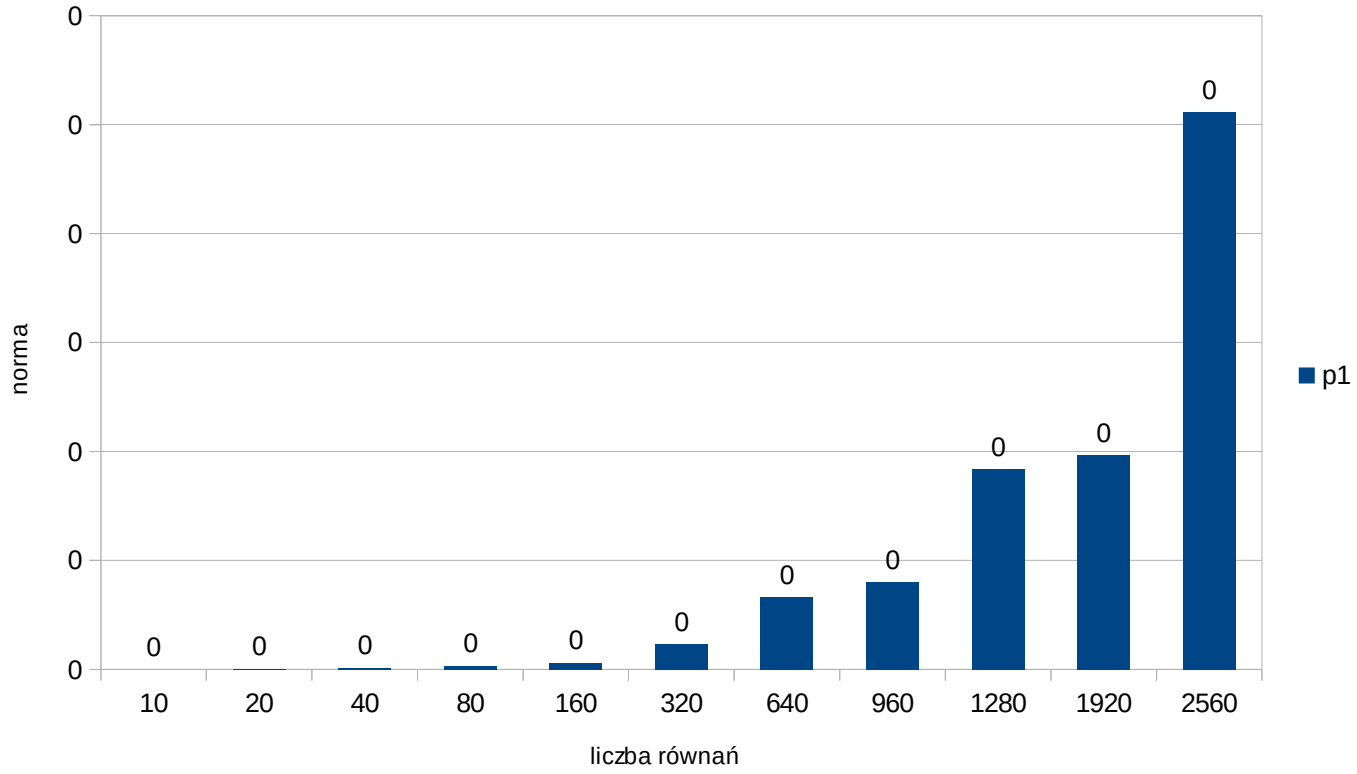
4. Prezentacja otrzymanych wyników w postaci tabeli i wykresów:

Uwaga: Dla liczby równań *n = 2560* czas potrzebny na rozwiązanie układu staje się zbyt duży, z tego powodu wykresy powstały do tej liczby.

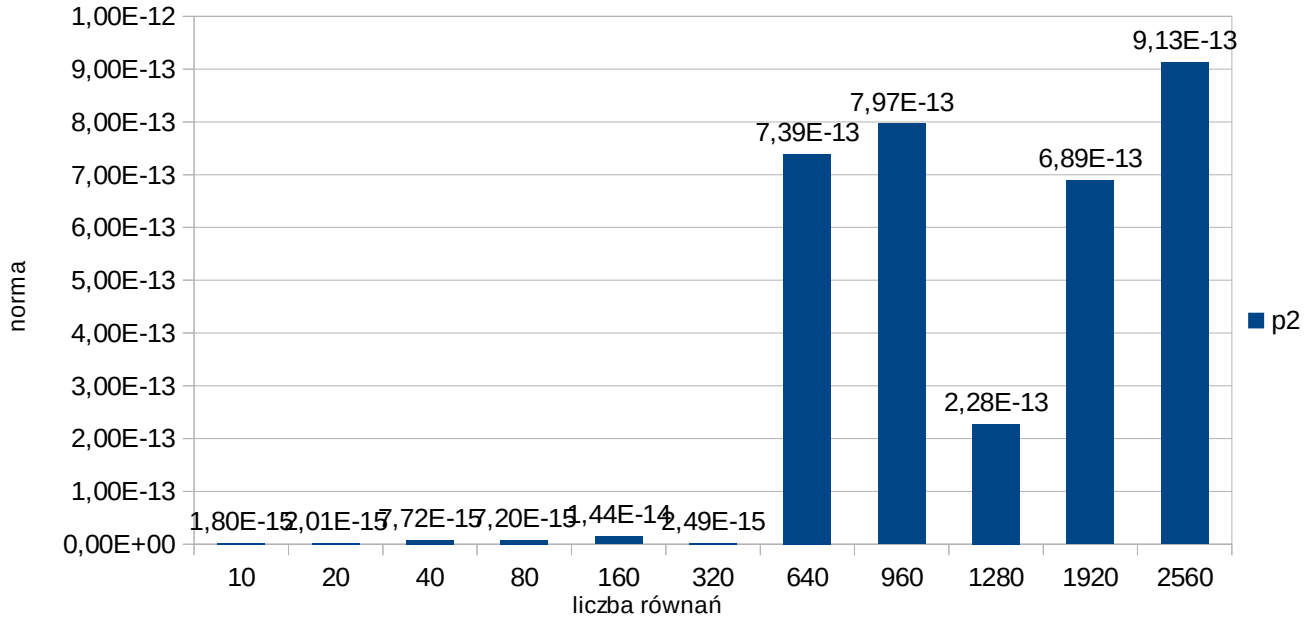
Tabela wyników (dokładniejsze liczby i wykresy w pliku *wykresy.ods*):

ilość iteracji	10	20	40	80	160	320	640	960	1280	1920	2560
p1	0	8,88E-16	3,20E-15	6,40E-15	1,19E-14	4,59E-14	1,32E-13	1,60E-13	3,68E-13	3,93E-13	1,02E-12
p2	1,80E-15	2,01E-15	7,72E-15	7,20E-15	1,44E-14	2,49E-15	7,39E-13	7,97E-13	2,28E-13	6,89E-13	9,13E-13
p3	1,22E-04	0,0468	0,4156	1,1467	0,3434	0,7723	0,1638	3,0964	4,2653	0,1273	0,4135

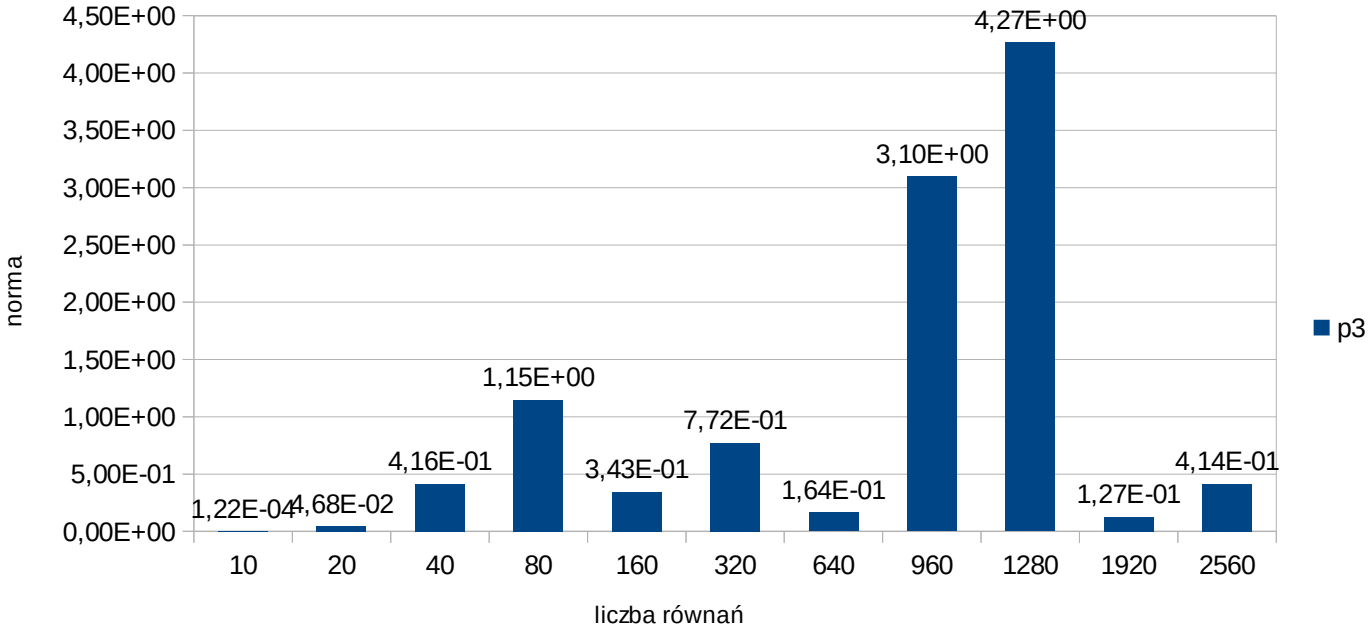
Dla przykładu z zadania 2 podpunktu 1 powstał taki wykres (niestety małe liczby *Microsoft Office* prze konwertował na 0) :



Dla przykładu z zadania 2 podpunktu 2 powstał taki wykres:



Dla przykładu z zadania 2 podpunktu 3 powstał taki wykres (dla tzw. macierzy **Hilberta**):



Wyniki
Czas wykonania dla operacji n = 640 równań:

```
>> metoda_eleminacji_Gaussa (pA, 640)
Elapsed time is 1.732659 seconds.

ans =

    0.1715
```

```
>> metoda_eleminacji_Gaussa (pA, 160)
Elapsed time is 0.008670 seconds.
```

5. Komentarz do otrzymanych wyników oraz wnioski z eksperymentów:

W zadaniu 2 podpunkcie 1 (czasami określony jako *pA*)widać, że dla niewielkich *n* (mniejszych od 20) wartość normy w skali tabeli są niezauważalne. Natomiast dla większych *n* charakterystyka jest wyraźnie liniowo rosnąca, a więc im większa ilość równań tym większy błąd rozwiązania. Błędy rozwiązania są rzędu *E-16* do *E-13* co jest stosunkowo małym błędem.

W zadaniu 2 podpunkcie 2 (czasami określonym jako *pB*) zauważamy, iż poniżej 320 równań błąd wyniku jest bardzo mały (rzędu *E-15 – E-14*). Gdy liczba równań przekracza 320 widzimy gwałtowny wzrost błędy wyniku, który nie zmienia się diametralnie dla liczby równań w przedziale od 640 do 2560. Patrząc na tą macierz możemy zauważyć, iż elementy na diagonali są zawsze najmniejsze i rosną przyjmując największą wartość dla elementów *pB(1,n)* oraz *pB(n,1).* Przy dużej liczbie n rozbieżności dla elementów na diagonali a dla tych wartość w pobliżu *pB(1, n)* i *pB(n,1)* są bardzo duże. Badając wskaźnik uwarunkowania macierzy, który pozwala na oszacowanie, z jaką (maksymalnie) dokładnością (do ilu miejsc po przecinku) możemy podać wynik widzimy, iż jest on dość duży. Może to powodować nagły wzrost błędów reprezentacji liczb. Mimo wszystko norma dla podpunktu 2. jest mała. Jest ona trochę gorzej uwarunkowana niż z podpunktu 1, lecz norma jest względnie mała.

W zadaniu 2 podpunkcie 3 (czasami określonym jako *pC*) wyraźne widać, iż mamy tu do czynienia z **macierzą Hilberta**, która jest tylko pomnożona przez współczynnik *0.4*.
Macierz Hilberta jest podręcznikowym przykładem macierzy, która jest źle uwarunkowana. Nawet dla niewielkiej liczby równań numeryczne rozwiązanie układów równań jest praktycznie niemożliwe. Wskaźnik uwarunkowania dla macierzy Hilberta stopnia N rośnie eksponencjalnie. Dlatego wyniki, które nam wyszły są takie niedokładne, nie należy się nimi sugerować.

Licząc uwarunkowania macierzy, za pomocą specjalnej funkcji w *Matlabie – cond,* przekunujemy się, iż nasze wykresy zgadzają się z wynikami uwarunkowania.

Oto wyniki programu z *Matlaba* dla *n = 10* (po lewej stronie) i *n =100* (prawa strona):

```
>> cond(A1)

ans =

    4.5503

>> cond(A2)

ans =

  175.5685

>> cond(A3)

ans =

   2.4242e+14
```

```
>> cond(A1)

ans =

    4.9942

>> cond(A2)

ans =

   1.7152e+04

>> cond(A3)

ans =

   1.1318e+19
```

Zadanie 3

1. Treść zadania:

Proszę napisać program rozwiązujący układ n równań liniowych *Ax = b* wykorzystując metodę Jacobiego i użyć go do rozwiązania danego układu równań liniowych.
Proszę sprawdzić dokładność rozwiązania oraz spróbować zastosować zaprogramowaną metodę do rozwiązania układów równań z zadania 2.

2. Ogólny zarys:

W przypadku, gdy mamy do rozwiązania problem wielkowymiarowy oraz macierze są rzadkie do rozwiązania układów równań ***Ax = B*** wykorzystujemy iteracyjne metody rozwiązywania układów równań. Zaletą stacjonarnych metod iteracyjnych jest ich prostota powodująca, że są one wyjątkowo wdzięczne do szybkiego zaprogramowania. Możemy wyróżnić kilka metod iteracyjnych tj.: metoda *Jacobiego, Gaussa-Seidela, SOR*. W metodach iteracyjnych nieznana jest liczba kroków, zależna od wymaganej dokładności zadania. Wszystkie bazują na podziale macierzy na ***A*** na trzy części: na diagonalną (***D***), ściśle trójkątną dolną (***L***), ściśle trójkątną górną (***U***).

3. Metoda Jacobiego:
   a) opis metody:

Jak już wspomniałem metoda Jacobiego bazuje na rozkładzie macierzy ***A***. Zakładając, że wartości na diagonali macierzy A są różne od 0 (inaczej: macierz **D** jest nieosobliwa) możemy zaproponować metodę (we wzorze b – wektor prawej strony układu równań – wektor rozwiązań):

$$x_{k+1} = D^{-1}(b - (L + U)x_k),$$

która często jest podawana w postaci:

$$X(i+1) = (-1) * D^{-1}(L + U) * X(i) + D^{-1} * b$$

Startując z przybliżenia początkowego rozwiązania (podanego przez użytkownika), w kolejnych krokach poprawiamy owe przybliżenie, zbiegając do rozwiązania. Iterację przerywamy, gdy osiągniemy założoną dokładność. Liczba kroków algorytmu zależy od dokładności z jaką chcemy je obliczyć. Metoda ta jest równoległa tzn.: każdą współrzędną nowego przybliżenia możemy wyznaczyć niezależnie od pozostałych.

   b) warunek dostateczny zbieżności metody:

Warunkiem dostatecznym zbieżności jest silna diagonalna dominacja macierzy A – wierszowa lub kolumnowa, tzn.:

$$\left|a_{ii}\right| > \sum_{j=1, j\neq i}^{n} \left|a_{ij}\right|, \; i = 1, 2, ..., n - dominacja \quad wierszowa$$

$$\left|a_{jj}\right| > \sum_{i=1, i\neq j}^{n} \left|a_{ij}\right|, \; j = 1, 2, ..., n - dominacja \quad kolumnowa$$

Układ z zadania 3 spełnia te warunki.

4. Prezentacja otrzymanych wyników oraz komentarz do otrzymanych wyników:

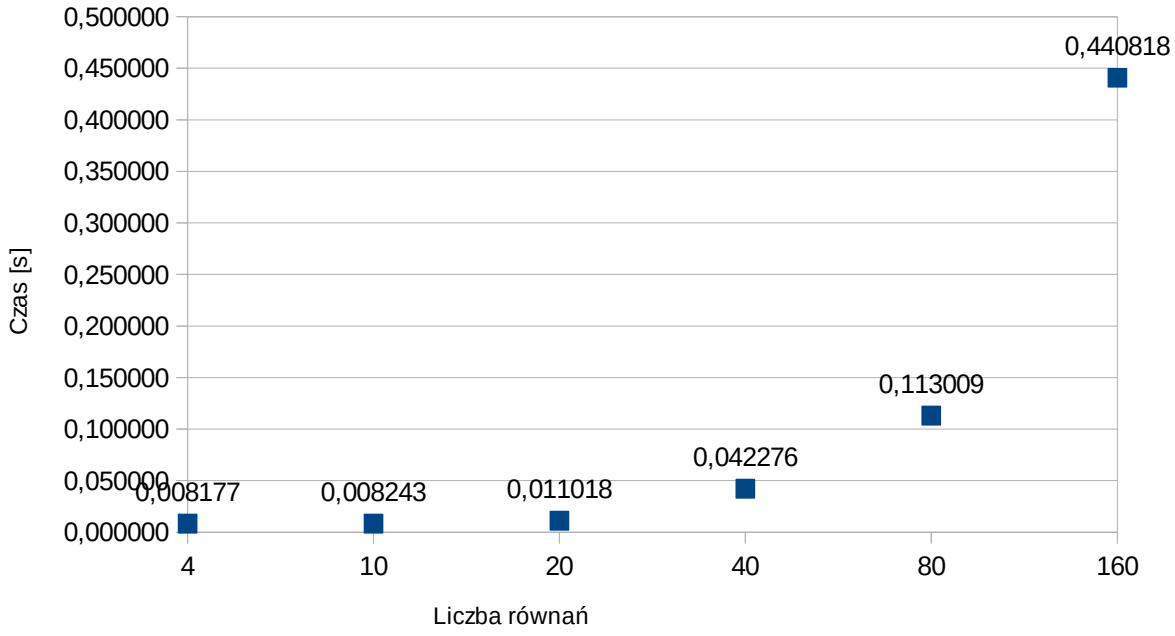
UWAGA: Podpunkty B i C z zadania 2. nie spełniają warunku silnej diagonalnej dominacji. Dla podpunktu B i C zadziałał warunek przerywania algorytmu dla macierzy nie posiadającej silnej diagonalnej dominacji.

Iterację przerywamy, gdy uda nam się uzyskać założoną dokładność – w moim zadaniu przyjąłem *10^(-10)*. Dokładność obliczam jaką normę euklidesową – różnica wektorów dwóch ostatnich rozwiązań.
Konkretnie w moim wypadku program zakończył się gdy norma wynosiła: *6,89767520018795E-11.*

A dla poszczególnych rozwiązań błędy wynosiły:
-2,06026307125740e-11
2,15141238157912e-11
4,39395186901947e-11
4,40429914760898e-11

Układ równań z:	**Liczba niewiadomych:**	**Czas wykonania (w sekundach):**
zad 3	4	0.008177
zad 2A	10	0.008243
zad 2A	20	0.011018
zad 2A	40	0.042276
zad 2A	80	0.113009
zad 2A	160	0.440818

Wykres czasu od liczby równań:



Wynik działania Matlaba:

```
>> metoda_jakobiego(A, B, 4)
Elapsed time is 0.008177 seconds.

ans =

    0.4225
    0.2217
    0.4609
    0.9758
```

```
>> metoda_jakobiego(A1, b1, 10)
Elapsed time is 0.015603 seconds.
```

```
>> metoda_jakobiego(A3, b3, 10)
Nie jest spełniony warunek silnej diagonalizacji
>> metoda_jakobiego(A2, b2, 10)
Nie jest spełniony warunek silnej diagonalizacji
```

W przypadku podpunktu A z zadania drugiego *metoda Eliminacji Gaussa* zadziałała szybciej *metoda Jacobiego* niż z wyborem elementu podstawowego (np. w przypadku układu o wymiarze *n = 10* czas zamiast *0.000083* sekund wyniósł *0.008243* sekund). Czas wykonania metody iteracyjnej zależy od dokładności z jaką chcemy obliczyć niewiadome.

W ćwiczeniu mogliśmy przekonać się, iż czas wykonania metody iteracyjnej (m.in. *metoda Jacobiego*) zależy od dokładności z jaką chcemy obliczyć dane niewiadomą, lecz nie nadają się do rozwiązywania wielu układów (muszę one spełniać ściśle określone warunki). Metody eliminacyjne są za to bardziej ogólne (ich zakres stosowania jest znacznie szerszy – więcej układów równań możemy policzyć tą metodą).
Podczas wykonywania równań widzimy, iż czas wykonania jest zależny od dokładności z jaką chcemy obliczyć (metody iteracyjne) oraz liczby równań. Często dla równań iteracyjnych musimy wybrać priorytet: czas lub dokładność. Zaletą stacjonarnych metod iteracyjnych jest ich prostota powodująca, że są one wdzięczne do szybkiego zaimplementowania.

Wzory oraz metody wykorzystywane podczas tworzenia algorytmów były na podstawie książki *Metody numeryczne* autorstwa prof. dr hab. inż. Piotra Tatjewskiego.

Kod programów:

1)

```
function [eps] = machine_eps()
    g = 1; %%inicjalizuje pewna zmienna

    while 1+g ~= 1  %%warunek pętli: wykonuj się, aż nie będziesz widziała
różnicy pomiędzy 1 + eps a 1, wtedy poznamy naszą dokładność maszynową
        eps = g;
        g = g/2; %%zmiejszamy liczbę g do takiego momentu, aż komputer nie
rozróżni g a liczby 1 (dzielimy przez dwa gdyż komputer pracuje w systemie
binarnym
    end

    disp(eps); %%wyświetlenie dokładności maszynowej
end
```

Tworzenie macierzy z zadania 2.

```
n = 10; (ew. 20, 30, itd. w zależności od długości macierzy)

%zainicjowanie macierzy
A1 = zeros(n, n);
A2 = zeros(n, n);
A3 = zeros(n, n);
b1 = zeros(n, 1);
b2 = zeros(n, 1);
b3 = zeros(n, 1);

%tworzenie macierzy z pierwszego przykładu
for i = 1:n
    for j = 1:n
        if i == j
            A1(i, j) = 9;
        elseif i == j-1 || i == j+1
            A1(i, j) = 3;
        else
            A1(i, j) = 0;
        end
    end
end

for i = 1:n
    b1(i) = 1.5 + 0.5 * i;
end
%tworzenie macierzy z drugiego przykładu
for i = 1:n
    for j = 1:n
        if i == j
            A2(i, j) = 1/7;
        else
            A2(i, j) = 11*(i-j) +2;
        end
    end
end

 for i = 1:n
     b2(i) =1 + 0.4*i;
 end

 %tworzenie macierzy z drugiego przykładu
```

```
for i = 1:n
    for j = 1:n
       A3(i,j) = 2/(5*(i+j+1));
    end
end

 for i = 1:n
     if mod(i, 2) == 0
         b3(i) = 8/(7*i);
     else
         b3(i) = 0;
     end
 end

 %%rozszerzone macierze dla przykładu A, B, C z zadania 2.
 pA = [A1, b1];
 pB = [A2, b2];
 pC = [A3, b3];
```

2)

Algorytm:

```
function X = metoda_eleminacji_Gaussa (A, n)
tic;
X = zeros(n,1); %%alokacja pamięci do wektora rozwiązań
for i = 1:n
    pom = abs(A(i, i)); %%zmienna pomocnicza przechowująca aktualnie największą
wartość w kolumnie
    poz = i; %%zmienna pomocnicza przechwoująca aktualnie miejsce największej
wartości w kolumnie
    %%szukam wiersza, ktory będzie miał największy moduł w kolumnie
    for j = (i+1):n
        if(abs(A(j, i)) > pom)
            pom = abs(A(j, i));
            poz = j;
        end
    end %%jeśli cały wiersz ma wartości tylko 0, wowczas detA = 0 i rownanie
jest zalezne od parametrów
    if(pom == 0)
        fprintf(1, "Równanie zależne od paramentrów, detA = 0", n);
        return;
    end
    %%zamiana miejscami i-tego wiersza o największą aktualnie wartością z w i-
tej
    %%kolumnie z j-tym wierszem.
    zamiana = A(i,:);
    A(i, :) = A(poz, :);
    A(poz, :) = zamiana;

    %obliczanie współczynników
   for j=(i+1):n
   l = A(j,i)/A(i,i);
   A(j,:) = A(j,:) - l*A(i,:); %%macierz trójkątna
   end
end
%%jesli macierz trojkatna gorna ma w elemencie (n, n) wartość 0, oraz
%%element (n, n+1) oznacza to, że det(A) = 0, elementy są zależne
    if A(n,n) == 0
        if A(n,(n+1)) == 0
            fprintf(1, 'Det(A) = 0!', n);
```

```
            return
        else %%w innym wypadku oznacza to, że macierz jest sprzeczna
            fprintf(1, 'Macierz sprzeczna!\n', n);
            return
        end
    end
   %%rozwiązywania układu równań zgodnie ze wzorem z książki prof.
   %%Tatjewskiego
   for i = n:-1:1
        licznik = 0;
        for s=n:-1:i
            licznik = licznik+(A(i,s)*X(s));
        end
        X(i,1) = (A(i,(n+1))-licznik)/A(i,i) ;  %%rozwiązujemy układu
   end
   blad = 0;
   blad = A(:,1:n)*X-A(:,n+1); %% zmienna określająca błąd mojego rozwiązania
toc;
end
```

3)

Macierze do trzeciego zadania:

```
A = [2, -0.5, 1, -0.2; 1, -3, 0.5, -0.5;1,1,4,-0.5;1,0.5,-1,3];

L = [0, 0, 0, 0; 1, 0, 0, 0;1,1,0,0;1,0.5,-1,0];

D = [2, 0, 0, 0;0, -3, 0, 0;0, 0, 4, 0;0, 0, 0, 3];

U = [0,-0.5,1, -0.2; 0, 0, 0.5, -0.5;0, 0, 0, -0.5;0,0,0,0];

LU = L + U; %tak nazwana zmienna, będzie ona równa sumie macierzy L i U

B = [1; -0.5; 2; 3];
```

Algorytm:

```
function Y_nas = metoda_jakobiego (A,B,n)
    A1 = A; %%przypisuje tą macierz do innej macierzy aby funkcja nie zmieniała
wartości elementów macierzy A
    D = zeros(n, n);
    tic;

    %%sprawdzenie czy macierz jest silnie diagonalna
    for i = 1:n
        wiersze = 0;
        kolumny = 0;
        for j = 1:n
            wiersze = wiersze + abs(A1(i, j));
            kolumny = kolumny + abs(A1(j, i));
        end
        wiersze = wiersze - abs(A1(i, i));
        kolumny = kolumny - abs(A1(i, i));


    end
    if wiersze > abs(A1(i, i)) && kolumny > abs(A1(i, i))
        disp("Nie jest spełniony warunek silnej diagonalizacji"); %%sprawdzam
czy jest spełniony warunek silnej diagonalizacji
        %%return;
```

```
    end
    %inicjalizacja danych macierzy danych w zadaniu
    for i = 1:n
        D(i, i) = A1(i, i); %%macierz D będzie macierzą diagonali D
        A1(i, i) = 0; %%macierz A będzie sumą macierzy L + U
    end
    LU = A1; %%tworzenie macierzy LU = L + U
    Y_pop = zeros(n,1);
    Y_nas = zeros(n,1);
    for i = 1:n
        Y_pop(i, 1) = 0; %%tworzymy macierz do przechwywania wyników
poprzedniego
        Y_nas(i, 1) = 0; %%wynik następny
    end

    norma_euklidesowa = 1;
    Y_pop = B;
    %implementacja algorytmu
    while(norma_euklidesowa > 10^(-10))

        Y_nas = (-1) * D^(-1)* LU * Y_pop + D^(-1)*B; %%wzór wprost z książki
prof. Tatjewskiego
        norma_euklidesowa = norm(A*Y_nas-B);  %%obliczanie normy euklidesowej -
warunek stopu
        Y_pop = Y_nas; %%przypisanie wartością poprzednim wartości obliczonej w
danej interacji
    %implementacja błędu - norma euklidesa
    end
    blad = A*Y_nas-B; %%macierz, która będzie zawierała błąd od poprawnego
wyniku
    toc;
end
```