Kacper Marchlewicz

MNUM – Projekt 1, zadanie 1.22

1.Napisać program wyznaczający dokładność maszynową komputera i uruchomić go na swoim komputerze.

Maksymalny błąd względny reprezentacji zmiennoprzecinkowej zależy jedynie od liczby bitów mantysy i nazywany jest dokładnością maszynową (precyzją maszynową – machine precision), i oznaczany jest tradycyjnie prze eps. Jest to najmniejsza liczba nieujemna, której dodanie do jedynki daje wynik nierówny 1, tzn. jest to najmniejszy ε, dla którego:

$$\varepsilon + 1 \neq 1$$

Definicja na podstawie książki Metody numeryczne prof. P. Tatjewskiego; OWPW, Warszawa 2013

W standardzie IEE754 w formacie 64 bitowym mantysa jest 53 bitowa (lecz 1 bit jest domyślny, zawsze równy 1) co umożliwia zakodowanie najmniejszej liczby równej 2^(-52). Według tej teorii procesor zbudowany w architekturze 64-bitowej potrafiłby rozróżnić 1 od 1+2^(-52), lecz nie potrafiłby rozróżnić 1 od 1+2^(-53).

Stworzyłem skrypt dodający do 1 coraz to mniejsze potęgi liczby 2. Jeśli suma będzie równa 1 (czyli dla komputera ε będzie przybliżony do 0) program zwróci poprzednią potęgę – tą dla której suma była różna od 1.

Program:

```
w = 0;
s = 2;
while s>1 % funkcja sprawdza czy suma będzie większa od 1
    w=w-1;
    s=1+2^w;
end
display(w+1); % najmniejsza potęga dla której suma jest > 1
display(2^(w+1)); % jest to szukana dokładność
```

Wyniki działania:

```
 -52

  2.2204e-16
```

Otrzymany wynik pokrywa się ze wzorcową wartością epsilonu dla liczb podwójnej precyzji w systemie 64- bitowym dla standardu IEEE 754-2008, tzn.

ε ≈ 2.22e – 16 ≈ 2^(-52)

Stąd wynik należy uznać za poprawny. Przeprowadzone doświadczenie wskazało, że komputer faktycznie nie rozróżnia liczby 1 od sumy 1 i liczby 2 razy mniejszej od uzyskanej (czyli 2^(-53)), co potwierdza otrzymany wynik.

2. Napisać uniwersalną procedurę (solwer) rozwiązującą układ n równań linowych Ax=B, wykorzystując podaną metodę – eliminacja Gaussa z częściowym wyborem elementu podstawowego.

Metoda eliminacji Gaussa zaliczana jest do metod skończonych, tzn. takich, w których wynik otrzymujemy po skończonej, ściśle określonej liczbie przekształceń zależnej od wymiarowości zadania. Algorytm eliminacji Gaussa dzieli się na dwa etapy:

1) Eliminacja zmiennych – w wyniku przekształceń macierzy A i wektora B otrzymamy równoważny układ równań z macierzą trójkątną górną.

2) Postępowanie odwrotne (back-substitution) – stosujemy algorytm rozwiązania układu z macierzą trójkątną.

Definicja z książki Metody numeryczne prof. P. Tatjewskiego; OWPW, Warszawa 2013

Stosowanie w metodzie eliminacji Gaussa wyboru częściowego elementu podstawowego prowadzi do mniejszych błędów numerycznych. Szukamy w niej największego co do modułu elementu w danej kolumnie i umieszczamy na przekątnej macierzy.

Implementacja metody eliminacji Gaussa z częściowym wyborem elementu podstawowego:

```
%imx - indeks liczby największej
%lmx - liczba największa
%dA/dB - wiersze które będą zamieniane

function [X] = GaussPodstCz_solver(A,B)
n=length(B);    %ilość równań
X=zeros(n,1);   %utworzenie macierzy z rozwiązaniami
for j=1:n
    imx=j;
    lmx=abs(A(j,j));  %wybranie początkowej największej wartości w kolumnie
    for i=j+1:n
        if abs(A(i,j)) > lmx    %sprawdzamy czy jest największa w kolumnie,
jeśli nie podmieniamy indeks
            lmx=abs(A(i,j));
            imx=i;
        end
```

```
        if imx ~= j    %zamieniamy wiersze tak aby z największą wartością
(indeksem) znajdował się na przekątnej
            dA=A(j,:);
            dB=B(j,:);
            A(j,:)=A(imx,:);
            B(j,:)=B(imx,:);
            A(imx,:)=dA;
           B(imx,:)=dB;
        end
    end
    for i2=j+1:n    %zerujemy wartości pod przekątną (odejmowanie wierszy)
        l = A(i2,j)/A(j,j);
        A(i2,:)=A(i2,:)-l*A(j,:);
        B(i2,:)=B(i2,:)-l*B(j,:);
    end
end
X(n,1)=B(n,1)/A(n,n);   %wyliczamy ostatni x
for m=n-1:-1:1  %rozwiązujemy układ z macierzą trójkątną (postępowanie
odwrotne)
    X(m,1)=B(m,1);
    for q=1:n-m
        X(m,1)=X(m,1)-A(m,(n+1-q))*X((n+1-q),1);
    end
    X(m,1) = X(m,1)/A(m,(n-q));
end
end
```

Kody określające dane w macierzach podpunktów zadania

Testowanie solwera:

```
function [A,B] = zad2_generacja(n)
G = 10*rand(n);
B = 30*rand(n,1);
A = G*(G');
end
```

Podpunkt A:

```
function [A,B] = zad2a_generacja(n)
for i=1:n
    for j=1:n
        if i==j
        A(i,j)=12;
        elseif i==j-1 | i==j+1
        A(i,j) = 3.8;
        else
        A(i,j)=0;
        end
    end
end
for i=1:n
B(i,1)=4.5-0.5*i;
end
end
```

Podpunkt B:

```
function [A,B] = zad2b_generacja(n)
for i=1:n
    for j=1:n
```

```
        if i==j
        A(i,j)=1/3;
        else
        A(i,j)=3*(i-j)+3;
        end
    end
end
for i=1:n
B(i,1)=0.5-0.6*i;
end
end
```

W celu utworzenia wykresu zadanie rozwiązałem w formie funkcji for (tutaj na przykładzie podpunktu B):

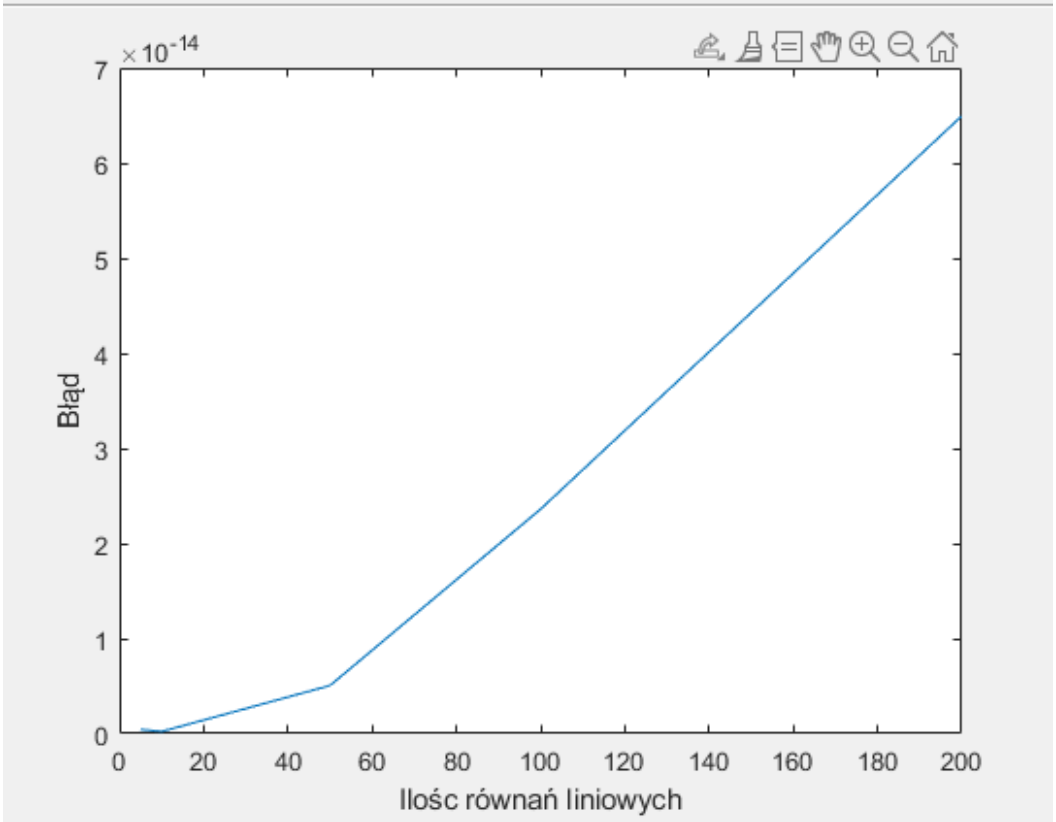
```
% ilość równań - z
%ilość punktów - h
%kolumna pierwsza - k1
%kolumna druga - k2

z=[5,10,50,100,200];     %kolejno liczba równań które będziemy rozwiązywać
h = length(z);
k1=zeros(1,h);
k2=zeros(1,h);
for n=z(1:end)
    [A,B] = zad2b_generacja(n);     %tworzenie macierzy A i B
    X = GaussPodstCz_solver(A,B);          %obliczenie X
    E=norm(A*X-B);                  %obliczenie normy
    k1(1,h)=n;    %kolumny zawierają ilość równań
    k2(1,h)=E;    % a także błędy
    h = h-1;
end
plot(k1,k2);           % wykres pokazuje zależność ilości równań od błędu
xlabel('Ilosc równań liniowych');
ylabel('Błąd');
function [A,B] = zad2a_generacja(n)
for i=1:n
    for j=1:n
        if i==j
        A(i,j)=1/3;
        else
        A(i,j)=3*(i-j)+3;
        end
    end
end
for i=1:n
B(i,1)=0.5-0.6*i;
end
end
```

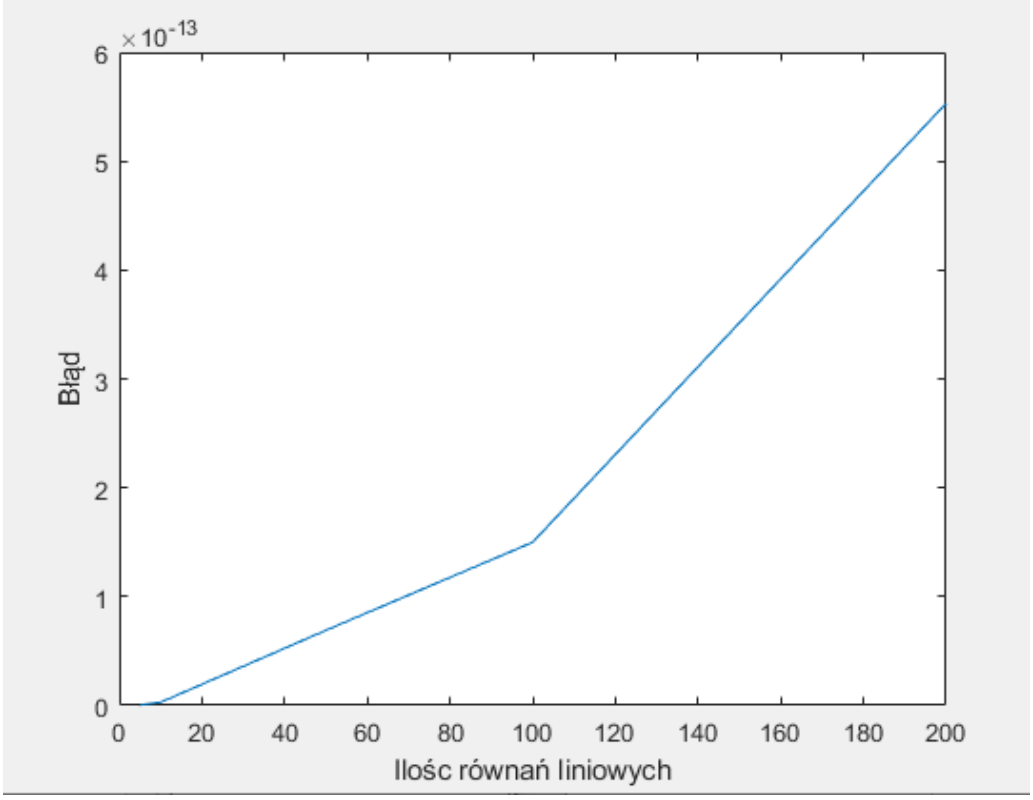
Ilość rozwiązań oraz obliczony błąd są zapisywane w odpowiednich wektorach. Po przejściu przez wszystkie n równań tworzony jest wykres.

Wykresy prezentowały się następująco:

Podpunkt A:



Podpunkt B:



Z powyższych wykresów widać, że dla podpunktów A i B występuje wyraźny wzrost błędu obliczeniowego w przypadku większych układów równań, co wynika z liczby potrzebnych operacji i kumulacji błędów. Dokładne wyniki zamieściłem w pliku excel.

3. Napisać program rozwiązujący n równań liniowych Ax=b wykorzystując metodę Jacobiego

Metoda Jacobiego jest metodą iteracyjną w odróżnieniu od metody eliminacji Gaussa. Metoda opiera się na kilku krokach: Najpierw rozkładana jest macierz A na 3 macierze (L- poddiagonalną, D- diagonalną, U- naddiagonalną). Zsumowane tworzą macierz A. Następnie możliwe jest iteracyjne wyliczenie macierzy rozwiązań ze wzoru:

$X(i+1) = -D-1(L+U)\ X(i)+D-1B$

$X(początkowy) = 0,\ i = 0,1,2,\ldots$

Warunkiem dostatecznym zbieżności metody Jacobiego jest silna diagonalna dominacja macierzy A, wierszowa lub kolumnowa.

Szczegóły w książce Metody numeryczne prof. P. Tatjewskiego; OWPW, Warszawa 2013

Z uwagi na iteracyjne rozwiązanie, potrzebne wykonywanie testu stopu, polegającego na sprawdzaniu czy uzyskana w danym kroku dokładność jest dla nas wystarczająca.

Kod metody Jacobiego:

```
function [X] = Jacobi_solver(A,B,E2)
n=length(B);
L=zeros(n);
D=zeros(n);
U=zeros(n);
error=E2+1; %na wejściu ustawione byleby był większy od E2, do którego
będziemy dążyć
X=zeros(n,1);
for i=1:n % tworzymy macierze L,D,U
    for j=1:n
        if i == j
            D(i,j)=A(i,j);
        end
        if j > i
            U(i,j)=A(i,j);
        end
        if i > j
            L(i,j)=A(i,j);
        end
    end
end
while E2<=error %do tego będzie dążył nasz błąd
    pX=X;
    X=-D^(-1)*(L+U)*X+D^(-1)*B; %obliczamy X
    error=norm(X-pX);
end
```

```
end
```

Następnie wykorzystałem utworzony solwer do podpunktów a i b z zadania 2. Potrzebne było ustalenie błędu granicznego. Uznałem liczbę 1e-14 za odpowiednią. Jest to liczba niewielka, ale nie na tyle, żeby znacznie wydłużyć działanie komputera.

Podpunkt A:

```
E2=1e-14;
n=5;
[A,B] = zad2a_generacja(n);
X = Jacobi_solver(A,B,E2);
E1=norm(A*X-B);

function [A,B] = zad2a_generacja(n)
for i=1:n
    for j=1:n
        if i==j
        A(i,j)=12;
        elseif i==j-1 | i==j+1
        A(i,j) = 3.8;
        else
        A(i,j)=0;
        end
    end
end
for i=1:n
B(i,1)=4.5-0.5*i;
end
end
```

Metoda zadziałała prawidłowo, układy zostały rozwiązane. Błędy wyszły większe niż w metodzie Gaussa, co wynika z wyznaczonego błędu granicznego. Dla błędów granicznych mniejszych rzędów wynik byłby dokładniejszy, lecz czas pracy wydłuża się zauważalnie.

Podpunkt B:

```
E2=1e-14;
n=5;
[A,B] = zad2b_generacja(n);
X = Jacobi_solver(A,B,E2);
E1=norm(A*X-B);

function [A,B] = zad2b_generacja(n)
for i=1:n
    for j=1:n
        if i==j
        A(i,j)=1/3;
        else
        A(i,j)=3*(i-j)+3;
        end
    end
end
for i=1:n
B(i,1)=0.5-0.6*i;
```

```
end
end
```

Układy równań podpunktu B nie wykonały się prawidłowo – wynika to z tego, że macierze nie spełniają warunku dostatecznej zbieżności metody Jacobiego, którym jest silna diagonalna dominacja macierz.

Literatura:

Metody numeryczne prof. P. Tatjewskiego; OWPW, Warszawa 2013