MNUM – PROJEKT, zadanie 1.21

Wojciech Rokicki gr. AR2

1. Do wyznaczenia **maszynowej dokładności komputera** został zastosowany algorym znajdujący najmniejszą liczbę, która po dodaniu do jedności zmienia wynik.

$$eps \stackrel{\text{def}}{=} \min\{g \in M : fl(1+g) > 1, g > 0\}$$

Wyznaczenie dokladnosci maszynowej komputera

```
x=0;
e=1;
while x~=1
    x=1;
    e=e/2;
    x=x+e;
end
%mnożę e przez 2
disp(e*2);
```

Wynik: Eps = 2.2204e-16
Wniosek: Komputer wykonuje obliczenia z dokładnością eps=2.2204e-16.
Zgodność ze standardem IEEE 754 dla liczb zmiennoprzecinkowych o podwójnej precyzji.
Liczba o 16 miejscach znaczących (dziesiętnie) -> mantysa 53 bitowa.

2. **Metoda eliminacji Gaussa z częściowym wyborem elementu podstawowego** została zrealizowana następującym algorytmem (opisuje k-ty krok):
Ozn. (k-ty krok(idziemy po kolumnach np. dla zerowania pierwszej kolumny k=1)),
(i-ty wiersz, >k, <=n)

$$a_{11}^{(1)}x_1 + a_{12}^{(1)}x_1 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)}$$
$$a_{21}^{(1)}x_1 + a_{22}^{(1)}x_2 + \cdots + a_{2n}^{(1)}x_n = b_2^{(1)}$$
$$\cdots\cdots$$
$$a_{n1}^{(1)}x_1 + a_{n2}^{(1)}x_1 + \cdots + a_{nn}^{(1)}x_n = b_n^{(1)}$$

-Wybieramy za każdym razem w k-tym kroku jako element główny, element spośród elementów $a_{jk}^{(k)}(k \le j \le n)$ element o największym module (oznaczony jako (ik)-ty) tzn.

$$\left|a_{ik}^{(k)}\right| = \max_j\left\{\left|a_{jk}^{(k)}\right|, \qquad j = k, k+1, \dots, n\right\}$$

Gdy już znajdziemy szukany element to zamieniamy wiersze i-ty z k-tym.

-Następnie wyznaczamy współczynniki i umieszczamy je w macierzy L

$$l_{ik} \stackrel{\text{def}}{=} \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$$

, wykonujemy operacje na wierszach:

$$w_i = w_i - l_{ik}w_k$$

-Po n-1 krokach uzyskujemy macierz trójkątną dolną L i układ równań

$$A^{(n)}x = b^{(n)}$$

```
U=A;         %gornej macierzy przypisujemy macierz poczatkowa
L=eye(n);    %dolna inicjalizujemy jedynkami na diagonali
pom=2;       %zmienna pomocnicza
for j=1:n    %przejscie kolejno po kazdej kolumnie
    max=0; %zerujemy wartość maksymalną
    poz=j; %ustawiamy poczatkowa pozycje (1. wiersz) na k-ty (tutaj j-ty) krok
    for x=j:n %przechodzimy po wszystkich wspolczynnikach ponizej pierwszego w j-tym kroku wlacznie
        if abs(U(x,j))>abs(max) %porownujemy moduly zapamietanej max wartosci z bierzacym wspolczynnikiem
           max=U(x,j); %jesli element byl wiekszy to zapamietujemy jego wartosc...
           poz=x; %... i pozycje
        end
    end
    U([j poz],:)=U([poz j],:); %zamieniamy wiersze (1. z tym gdzie wystapila najwieksza wartosc bezwzgledna)
    b([j poz],1)=b([poz j],1);
    for i=pom:n             %obliczamy ilorazy, modyfikujemy wiersze
        L(i,j)=U(i,j)/U(j,j);
        U(i,:)=U(i,:)-(L(i,j)*U(j,:));
        b(i,1)=b(i,1)-(L(i,j)*b(j,1));
    end
    pom=pom+1;
end
```

Wynik: Po wykonaniu powyższego algorytmu na zadanych macierzach otrzymujemy macierz dolną L ze współczynnikami i macierz górną U, które po pomnożeniu prze siebie dają macierz wyjściową.

Przykład:

A =

10	3	0	0	0
3	10	3	0	0
0	3	10	3	0
0	0	3	10	3
0	0	0	3	10

U =

10.0000	3.0000	0	0	0
0	9.1000	3.0000	0	0
0	0	9.0110	3.0000	0
0	0	0	9.0012	3.0000
0	0	0	0	9.0001

L =

1.0000	0	0	0	0
0.3000	1.0000	0	0	0
0	0.3297	1.0000	0	0
0	0	0.3329	1.0000	0
0	0	0	0.3333	1.0000

L*U =

10	3	0	0	0
3	10	3	0	0
0	3	10	3	0
0	0	3	10	3
0	0	0	3	10

Obliczenie wyniku – wektor x-ów

Po otrzymaniu macierzy:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1,n-1}x_{n-1} + a_{1n}x_n &= b_1 \\
a_{22}x_2 + \cdots + a_{2,n-1}x_{n-1} + a_{2n}x_n &= b_2 \\
&\cdots\cdots \\
a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n &= b_{n-1} \\
a_{nn}x_n &= b_n
\end{aligned}
$$

Do obliczenia wyniku wykorzystano następujące wzory:

$$x_n = \frac{b_n}{a_{nn}}$$

$$x_{n-1} = \frac{\left(b_{n-1} - a_{n-1,n}x_n\right)}{a_{n-1,n-1}}$$

$$x_k = \frac{\left(b_k - \sum_{j=k+1}^{n} a_{kj}x_j\right)}{a_{kk}}$$

```
%Obliczanie wyniku
x(n,1)=b(n,1)/U(n,n); %Obliczenie ostatniego wyniku

for z=n-1:-1:1
    x(z,1)=(b(z,1)-U(z,z+1:n)*x(z+1:n,1))/U(z,z);
end

disp(x);
```

Przykład:

A =

10	3	0	0	0
3	10	3	0	0
0	3	10	3	0
0	0	3	10	3
0	0	0	3	10

x =

0.2015
0.3285
0.5370
0.5482
1.1355

U*x =

3.0000
4.6000
6.4835
8.3415
10.2199

b =

3.0000
4.6000
6.4835
8.3415
10.2199

Norma residuum

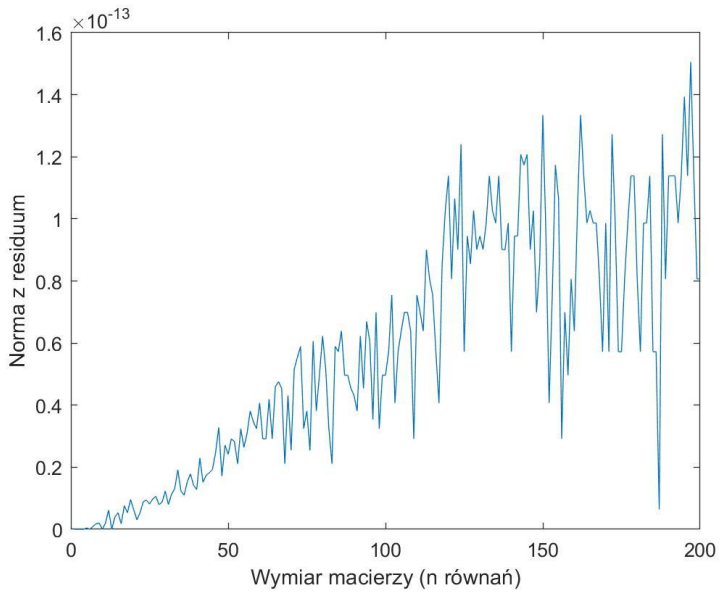
Obliczana (residuum) według następującego wzoru:

$$r^{(1)} \stackrel{\text{def}}{=} Ax^{(1)} - b$$

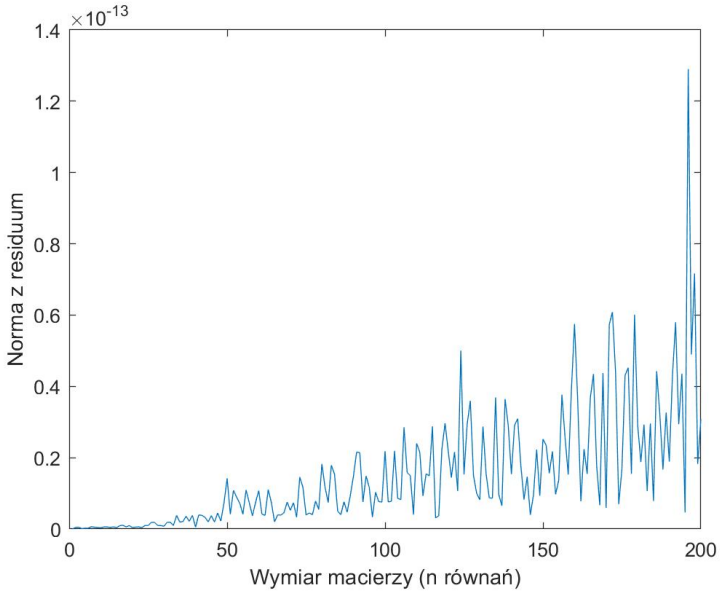
Następnie obliczam normę z $r^{(1)}$

```
%Norma z residuum
r=U*x-b;
disp(r);
norma=norm(r);
```

Zad 2a

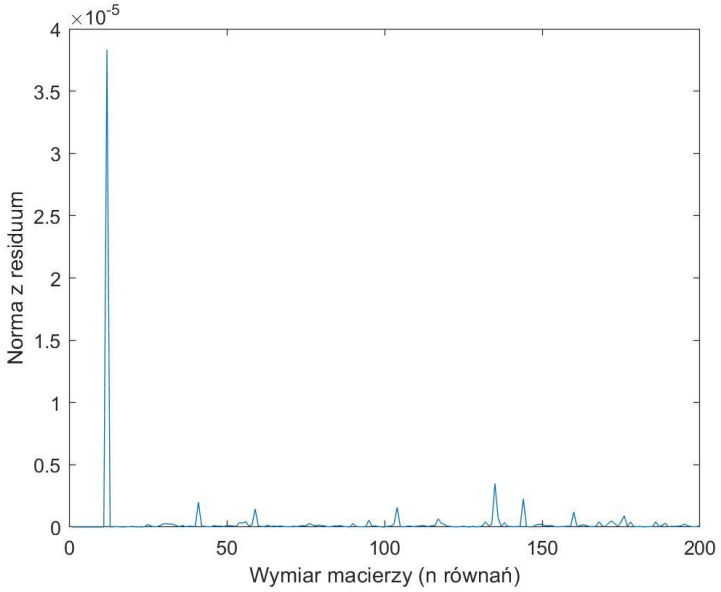


Zad 2b



Wniosek dla 2a i 2b: Im wiekszy nakład obliczeń (wiekszy rozmiar macierzy) tym wieksza norma z residuum. Zapewne wynika to z niedokładności reprezentacji danych (czasami dane układają się tak że program wylicza dokładniej). W 2a jest regularniejszy przyrost niż w 2b.

Zad 2c



Wniosek: Dla macierzy o małych rozmiarach norma z residuum jest nieregularna, przy dużych wymiarach macierzy jest mała.

3. **Metoda Jacobiego** została zrealizowana za pomocą następującego algorytmu:
-Stosujemy rozkład macierzy na macierze: L (poddiagonalna), D (diagonalna), U(naddiagonalna) t.że. $A = L + D + U$

Przykład:

A =

-7	1	1	1
1	11	-5	2
2	3	16	-3
1	2	-6	-15

L =

0	0	0	0
1	0	0	0
2	3	0	0
1	2	-6	0

D =

-7	0	0	0
0	11	0	0
0	0	16	0
0	0	0	-15

U =

0	1	1	1
0	0	-5	2
0	0	0	-3
0	0	0	0

-korzystamy ze wzoru:

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

lub skalarnie

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

```
A=[-7, 1, 1, 1; 1, 11, -5, 2; 2, 3, 16, -3; 1, 2, -6, -15]
b=[3; 17; 9; -26]
x=[1;1;1;1];
xi=ones(4,1);
e=0.001;    %zadany blad przyblizenia
L=zeros(4);
D=zeros(4);
U=zeros(4);
for i=1:4   %rozklad na L D U (A=L+D+U)
    for j=1:4
        if(i==j)
            D(i,j)=A(i,j);
        end
        if(i>j)
           L(i,j)=A(i,j);
        end
        if(i<j)
            U(i,j)=A(i,j);
        end
    end
end
i=1;
while(1)
xi=-inv(D)*(L+U)*x+inv(D)*b;
if norm(xi-x)<e     %Jeśli norma zeuklidesowa roznicy miedzy kolejnymi przyblizeniami rozwiazania
    break           %jest mniejsza od zadanego bledu przyblizenia to przerwij petle
end
i=i+1;      %zliczanie iteracji potrzebnych do zrealizowania przyblizenia
x=xi;
end

disp(xi);
disp(i);
```

Przykład (z błędem przybliżenia, liczonym jako norma euklidesowa różnicy między kolejnymi przybliżeniami rozwiązania) dla e=0.001:

A =

-7	1	1	1
1	11	-5	2
2	3	16	-3
1	2	-6	-15

b =

3
17
9
-26

x =

0.1122
1.4930
0.5881
1.7044

i = 9

Wnioski: Dla mniejszego błędu przyblizenia jest więcej iteracji, metoda przybliża bardzo skutecznie. Dla macierzy danej w zad 2a metoda sprawdziła się (16 iteracji), natomiast przy ułamkach dla macierzy 2b program długo pracował, a uruchomienie nie powiodło się. (testy byly prowadzone na macierzach o wymiarach 5x5).
Metoda Jacobiego nie jest skuteczna dla liczb ułamkowych, za to metoda eliminacji Gaussa jest uniwersalna i radzi sobie z małymi ułamkami.