MNUM-PROJEKT, zadanie 1.2
Raport

Autor: Tomasz Jakubczyk

1. Dokładność maszynowa
   Epsilon maszynowy – Istnieją dwie lekko różniące się definicje: największa nieujemna liczba w zmiennoprzecinkowej reprezentacji maszynowej której dodanie do 1 daje 1, lub najmniejsza liczba nieujemna w reprezentacji maszynowej, która dodana do 1 daje wynik różny od 1. Mimo, że te definicja wydają się być matematycznie równoważne, dają one nieco różniące się wyniki. Wykorzystam definicję drugą ze względu na jej popularność w środowiskach programistycznych.
   Dokładność maszynowa zależy od precyzji reprezentacji, architektury maszyny oraz implementacji.
   Wykorzystywany komputer jest w architekturze ***x86_64*** i posługuje się liczbami zmiennoprzecinkowymi w standardzie ***IEE754***. Środowisko ***Matlab*** udostępnia dwa typy zmiennoprzecinkowe: ***double*** i ***single***.
   Matlab posiada funkcję ***eps*** zwracającą epsilon maszynowy i posłuży za porównanie.
   Epsilon maszynowy (aproksymację) znajduję iteracyjnie zmniejszając zmienną o połowę poczynając od jedynki.

```
%Aproksymacja epsilonu maszynowego dla podwójnej precyzji
%normalnie domyœlnym typem jest double
dx=1;
while dx+1.0>1.0
    dx=dx*0.5;
end
MyEpsDouble=dx*2.0;
disp(sprintf('EpsylonDouble = %1.20e (2^%d)\n',MyEpsDouble,log2(MyEpsDouble)));
```

   Druga metoda uzyskiwania epsilona maszynowego polega na wykorzystaniu faktu, że w standardzie IEE754 kolejne wartości zmiennoprzecinkowe różnią się binarnie o 1. Algorytm ten wykorzystuje rzutowanie typów i przesunięcie o 1 w typie całkowitoloczbowym.

```
%Wyliczenie epsilona maszynowego przez ró¿nicê kolejnych wartoœci
%reprezentacji zmiennoprzecinkowej w standardzie IEE754
%Epsilon dla double
a=1;
b=typecast(a,'int64');
b=b+int64(1);
c=typecast(b,'double');
MyEps2=c-a;
```

```
disp(sprintf('EpsylonSingle = %1.20e (2^%d)\n',MyEps2,log2(MyEps2)));
```

Związek epsilona maszynowego z liczbą bitów wykorzystanych do reprezentacji liczby zmiennoprzecinkowej: epsilon odpowiada liczbie pozycji w mantysie. Dwie kolejne liczby mogą się różnić tylko 2^-t. (Jest to wyprowadzone np. w podręczniku w rozdziale 1, zatem nie będę tego wklejał.)

Uzyskane wyniki:

```
>> GenericEpsylon
EpsylonSingle = 1.19209289550781250000e-07 (2^-23)

EpsylonDouble = 2.22044604925031310000e-16 (2^-52)

>> MyEpsylon
Aproksymacja Epsilonu Maszynowego:

EpsylonSingle = 1.19209289550781250000e-07 (2^-23)

EpsylonDouble = 2.22044604925031310000e-16 (2^-52)

Uzyskanie Epsilona Maszynowego przez rzutowanie typów:

EpsylonDouble = 1.19209289550781250000e-07 (2^-23)

EpsylonSingle = 2.22044604925031310000e-16 (2^-52)
```

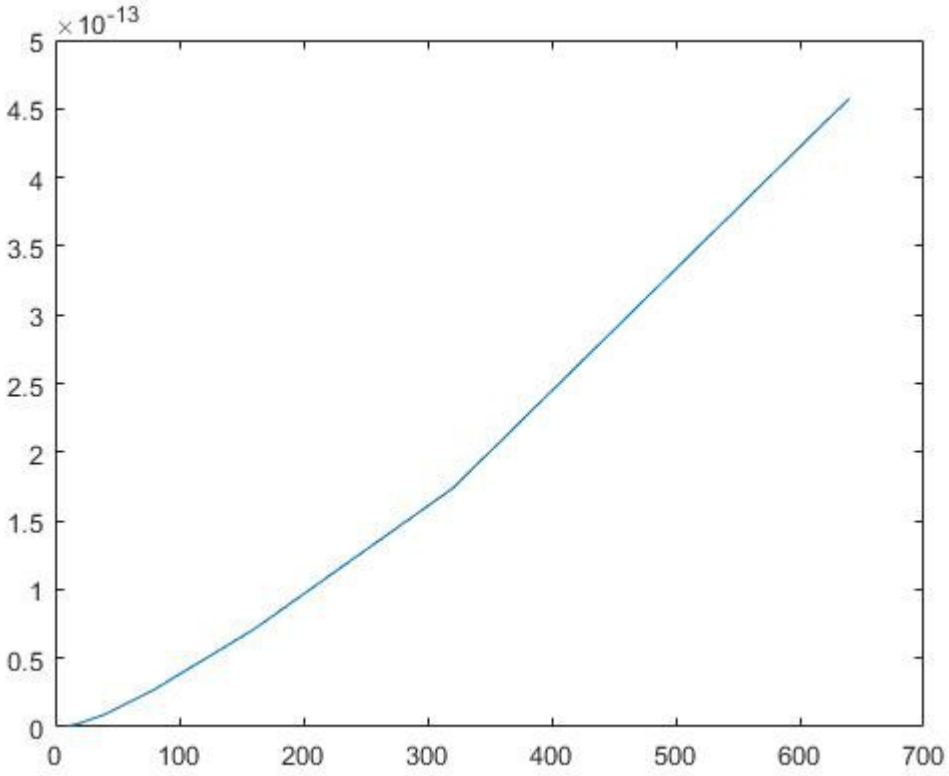
Epsilon maszynowy stanowi górne ograniczenie dla błędu względnego w reprezentacji liczby. abs((x-y)/x)<=eps gdzie y jest maszynową reprezentacją x. Epsilon można wykorzystać, żeby wyznaczyć maksymalny błąd jakiejś operacji dodając go do argumentów, a potem zobaczyć jak się w tej operacji propagował. Uzyskane wyniki są ze sobą zgodne. Metoda z rzutowaniem typów jest wydajniejsza i moim zdaniem elegantsza. Zwykle powinno się wykorzystać metodę wbudowaną w język lub środowisko programistyczne.

2. Rozwiązywanie układu równań metodą eliminacji Gaussa
   Metoda eliminacji Gaussa z częściowym wyborem elementu podstawowego jest metodą skończoną. Jej złożoność obliczeniowa to O(n^3). Wybór elementu skończonego polega na wybieraniu wiersza w którym pierwszy element jest największy. Pozwala to ograniczyć błędy numeryczne.
   Macierz sprowadza się do postaci trójkątnej w taki sposób, że najpierw wybiera się wiersz którego pierwszy element jest największy, przestawia się go do góry, a następnie dodaje się ten wiersz odpowiednio pomnożony do pozostałych wierszy tak, że ich pierwsze elementy stają się zerami. Raz wybrany wiersz już zostaję, a następne wiersze są tak redukowane, że mają na początku coraz więcej zer, ąz powstanie macierz trójkątna.

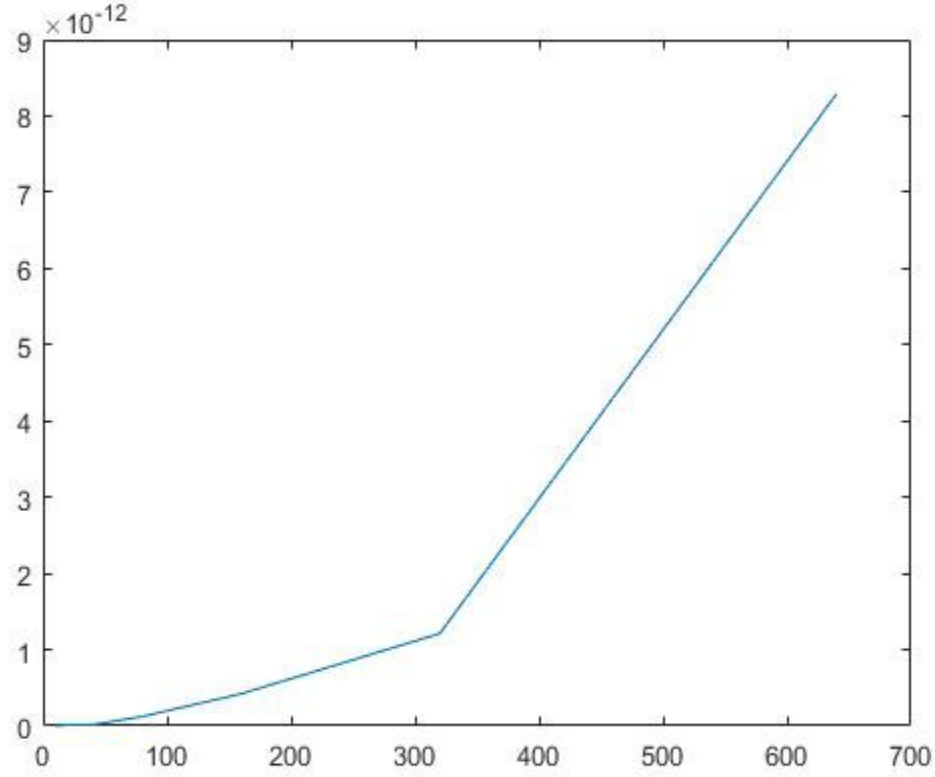
```
function [X, ex] = GE(A,b)
%funkca rozwi¹zuj¹ca zadany uk³ad równañ metod¹ Gaussa-Crouta
```

```
%X - rozwi¹zanie
%ex - b³¹d rozwi¹zania
AB=[A,-b];
n=size(A,1);
W=1:n;%numery wierszy
for i=1:n-1
    k=i;
    %wyszukujemy element o najwiêkszym module
    for j=i+1:n
        if abs(AB(W(k),i))<abs(AB(W(j),i))%to chyba da siê zrobiæ w jednej operacji
max
            k=j;
        end
    end
    %w wektorze zamieniamy numery kolumn wg znalezionego elementu
    tmp=W(k);
    W(k)=W(i);
    W(i)=tmp;
    if abs(AB(W(i),i))<eps(1e-6)%jeœli znaleziony element jest zerem, koñczymy
        %error('wybrany element jest zerem');
        ex=Inf;
        %return;
        break;
    end
    for j=i+1:n
        m=-AB(W(j),i)/AB(W(i),i);%wyznaczamy mno¿nik
        for kk=i:n+1 %sumujemy wiersz j-ty z wierszem i-tym przemno¿onym przez m
            AB(W(j),kk)=AB(W(j),kk)+m*AB(W(i),kk);%to chyba da siê zrobiæ w jednej
operacji
        end
    end
    %AB(W(:),:)
end
% wyliczamy kolejne niewiadome
for i=n:-1:1
    if abs(AB(W(i),i))<eps(1e-6)
        %error('dzielnik zero\n')
        ex=Inf;
        %return;
        break;
    end
    s=AB(W(i),n+1);
    for j=n:-1:i+1
        s=s+AB(W(i),j)*X(j);
    end
    X(i)=-s/AB(W(i),i);
end
%ex=n^3*2^(n-1)*eps;
r=A*X'-b;
ex=norm(r);
end
```

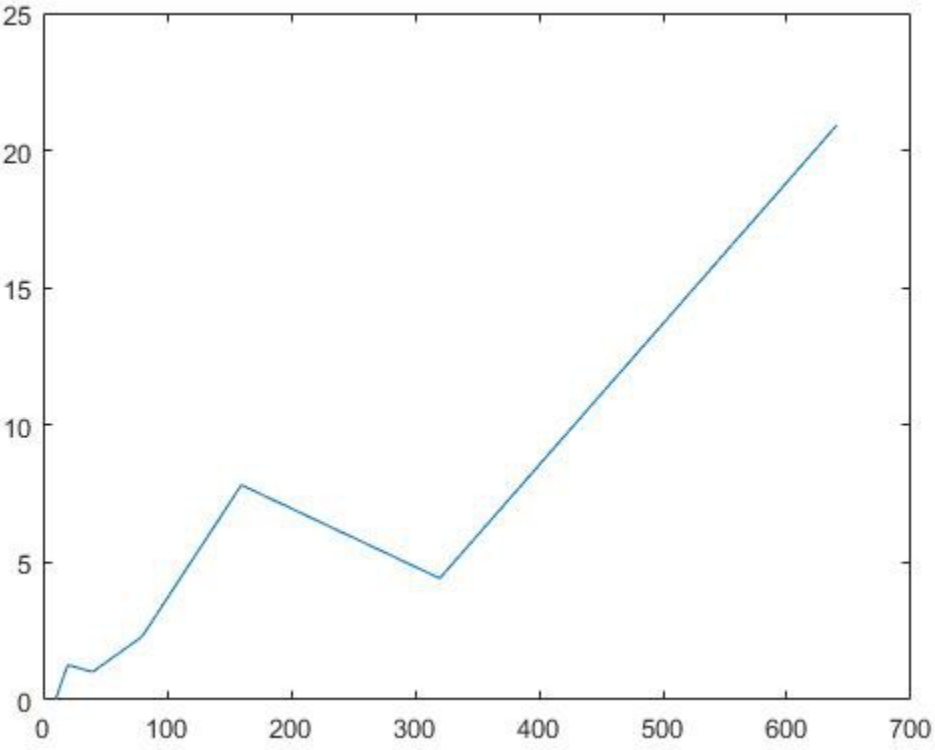
Wykresy zależności błędów rozwiązania od rozmiarów układów równań:



Błędy dla układów 1



Błędy dla układów 2

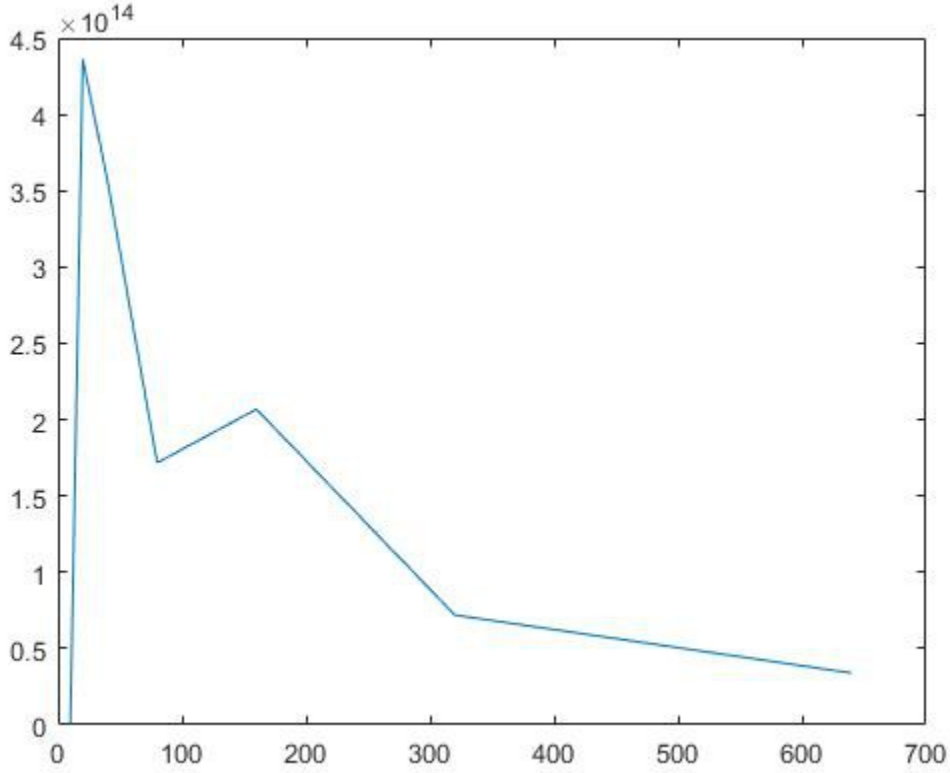


Błędy dla układów 3

Dla układów 3 ze wzrostem liczby układów błędy wzrastają najbardziej. Wydaje mi się że jest to spowodowane wzorem $a_{ij}=7/[9(i+j+1)]$ dającym zróżnicowane liczby dodatnie mniejsze od jedynki, co może pogarszać błędy. Błędy te są zbliżone do pesymistycznego oszacowania z góry dla metody eliminacji Gaussa z częściowym wyborem elementu głównego. Algorytm dokonuje dzieleń przez bardzo małe liczby.

Zadanie może być źle uwarunkowane. Sprawdzam to:

```
[ A3, b3 ] = EquationSystem3( n );
[X3,eX3(i)]=GE(A3,b3);
DA3=A3+eps;
Db3=b3+eps;
DAB3=[DA3,-Db3];
AB3=[A3,-b3];
[DX3,DeX3]=GE(DA3,Db3);
cond(i)=(norm(DX3-X3,Inf)/norm(X3,Inf))/(norm(DAB3-AB3,Inf)/norm(AB3,Inf));
```



Dla układów dla n>10 wskaźnik uwarunkowania jest raczej duży. Z takimi wskaźnikami uwarunkowania można się spodziewać dokładności od 2 miejsc dokładności 1-2 dla niższych wskaźników, do 1-2 rzędów wielkości dokładności rozwiązania. Zadanie jest źle uwarunkowane, bo niewielkie zaburzenia danych powodują duże zmiany rozwiązania.

3. Rozwiązywanie układu równań metodą Jacobiego
   Metoda Jacobiego jest metodą iteracyjną i polega na zamienianiu rozwiązań na

coraz dokładniejsze. Metoda ta może być niezbieżna i nie dać dobrego rozwiązania. Metoda może też czasem zbiegać mimo nie spełniaenia warunków zbieżności.

Wzór metody Jacobiego z podręcznika:

$$\mathbf{D}\mathbf{x}^{(i+1)} = -(\mathbf{L}+\mathbf{U})\,\mathbf{x}^{(i)} + \mathbf{b}, \qquad i = 0, 1, 2, \ldots$$

D – macierz diagonalna, w kodzie odpowiada jej M

L,U – dolne i górne macierze trójkątne, w kodzie (L+U)x wyliczane jest jako delta

```
function [ xk, ex ] = JacobiMethod( A, b )
%JacobiMethod implementacja metody Jacobiego
%   Rozwi¹zywanie ukadw równañ liniowych
n=size(A,1);
xk=zeros(n,1);
k=0;
stop=false;
t0=tic;
M=diag(diag(A));
Z=tril(A,-1)+triu(A,1);
P=M^-1*Z;
p=norm(P,Inf)%promieñ zbie¿noœci
while ~stop
    xkm1=xk;
    for i=1:n
        delta=0;
        for j=1:n
            if j~=i
                delta=delta+A(i,j)*xk(j);
            end
        end
        xk(i)=(b(i)-delta)/A(i,i);
        if isnan(xk(i)) || isinf(xk(i))%w tym przypadku nie ma co dalej liczyæ
            stop=true;
            xk=xkm1;
            break;
        end
    end
    k=k+1;
    if norm(xk-xkm1)<=1e-12%bardzo ma³a poprawa
        r=A*xk-b;
        if norm(r)<=1e-12%wynik wystarczaj¹co dok³adny
            stop=true;
        end
    end
    if toc(t0)>5
        stop=true
    end
end
r=A*xk-b;
ex=norm(r);
end
```

Rozwiązanie podanego układu:

X = 0.8210 1.5778 -0.2817 1.5766

Błąd tego rozwiązania to
ex = 9.0351e-13
Promień zbieżności:
p = 1.0833

Rozwiązując tą metodą układy równań z punktu 2 widać, że metoda dobrze działa dla układu 1 i 3, a dla układu 2 nie. Metoda Jacobiego dla układu 2 jest niezbieżna, a warunek wystarczający nie jest spełniony. Układy 2 mają promienie zbieżności znacznie większe od 1, to znaczy, że metoda iteracyjna może nie znaleźć rozwiązania, i nie znajduje. Dla układów 3 promień zbieżności jest trochę większy od 1, a metoda znajduje średnio dokładne rozwiązania wychodząc z czasowym warunkiem stopu. Wszystkie układy 1 mają promienie zbieżności mniejsze od 1 i znajdują rozwiązania z zadaną dokładnością.

Wyniki dla n = 10
układ1:
promień zbieżności
p = 0.2857
błąd rozwiązania
ex1 = 1.7886e-13
układ 2:
promień zbieżności
p = 495
błąd rozwiązania
ex2 = 2.7539e+301
układ 3
promień zbieżności
p = 12.1351
wyjście z powodu przekroczenia czasu
błąd rozwiązania
ex3 = 0.3588