

[MNUM] Metody Numeryczne
Semestr 2021L
Projekt 1
Sprawozdanie

Zadanie 1.

Treść:

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

Rozwiązanie:

Dokładność maszynowa (inaczej precyzja maszynowa) to maksymalny błąd względny reprezentacji zmiennoprzecinkowej. Oznaczamy ją przez ϵ .

Można wykazać, że dokładność maszynowa zależy od liczby bitów mantysy t : $\epsilon = 2^{-t}$ [1].

Inna definicja dokładności maszynowej:

Dokładność maszynowa ϵ to najmniejsza dodatnia liczba maszynowa g taka, że zachodzi relacja $fl(1 + g) > 1$, tzn. $\epsilon \stackrel{\text{def}}{=} \min\{g \in M : fl(1 + g) > 1, g > 0\}$ [1].

Algorytm w postaci listy kroków:

- 1) Przyjmij początkowo $\epsilon := 1$ oraz $\exp := 0$
- 2) Dopóki $1 + \epsilon > 1$, wykonuj:
 - 2a) $\epsilon := \epsilon / 2$
 - 2b) $\exp := \exp - 1$
- 3) $\epsilon := \epsilon * 2$
- 4) $\exp := \exp + 1$

Komentarz do kroków 3) oraz 4) algorytmu:

Po wyjściu z pętli, pamiętamy największą liczbę dodatnią eps , która nie spełnia warunku $fl(1 + eps) > 1$.

Aby uzyskać poszukiwaną dokładność maszynową, należy przemnożyć wartość eps przez 2, natomiast wartość t zwiększyć o 1.

Listing (Zad1.m):

```
% eps - poszukiwana dokładność maszynowa
% t - liczba całkowita, dla której 2^t = eps

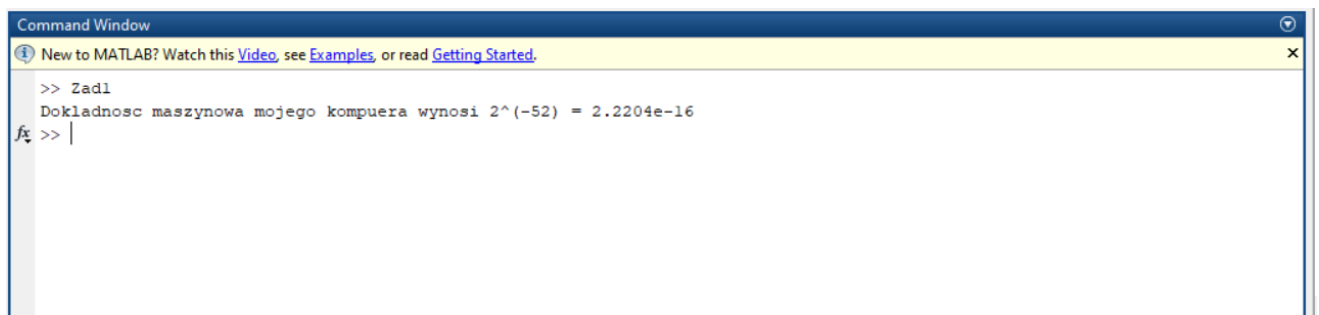
format long;

eps = 1;
t = 0;
while 1 + eps > 1
    eps = eps/2;
    t = t - 1;
end

eps = eps*2;
t = t + 1;

fprintf( 'Dokładność maszynowa mojego komputera wynosi 2^(%s) = %s\n',
num2str(t), num2str(eps) );
```

Wywołanie:



```
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.
>> Zad1
Dokładność maszynowa mojego komputera wynosi 2^(-52) = 2.2204e-16
fx >> |
```

Dokładność maszynowa mojego komputera wynosi $2^{-52} \approx 2.2204 \cdot 10^{-16}$.

Odpowiada to dokładności maszynowej dla liczb zmiennoprzecinkowych podwójnej precyzji wg standardu IEEE 754 (mantysa ma wtedy długość 52 bitów).

Zadanie 2.

Treść:

Napisać uniwersalną procedurę (solwer) rozwiązującą układ n równań liniowych $Ax = b$, wykorzystując podaną metodę (parametry wejściowe: A, b, n ; parametr wyjściowy: x).

Proszę najpierw sprawdzić działanie swojego solwera dla zadania wymiaru $n = 5$ z gęstą macierzą $A = GG^T$, gdzie macierz G została wygenerowana poleceniem $10 \cdot \text{rand}(5)$ i wektorem b wygenerowanym poleceniem $30 \cdot \text{rand}(5,1)$.

Proszę zastosować go następnie w programie do rozwiązania podanych niżej układów równań dla ich rosnącej liczby n : 5, 10, 50, 100, 200.

Metoda: eliminacji Gaussa-Jordana

Dane:

(a)

$$a_{ij} = \begin{cases} 12 & \text{dla } i = j \\ 3.8 & \text{dla } i = j - 1 \text{ lub } i = j + 1 \\ 0 & \text{dla pozostałych} \end{cases}$$
$$b_i = 4.5 - 0.5i$$

(b) $a_{ij} = 3(i - j) + 3$; $a_{ii} = \frac{1}{3}$; $b_i = 0.5 - 0.6i$

Dla każdego rozwiązania proszę obliczyć błąd rozwiązania $\varepsilon_1 = \|Ax - b\|$ i dla każdego układu równań proszę wykonać rysunek zależności tego błędu od liczby równań n .

Rozwiązanie:

Metoda Gaussa-Jordana jest jedną z metod skończonych rozwiązywania układów równań liniowych. Składa się ona z n kroków. W i -tym kroku ($i = 1, 2, \dots, n$) algorytmu dzielimy i -ty wiersz macierzy A oraz i -ty element wektora b przez element macierzowy a_{ii} .

Ponadto, w i -tym kroku algorytmu, dla każdego $j \neq i$ ($j = 1, 2, \dots, n$), wykonujemy, co następuje: j -ty element wektora b zmniejszamy o wartość $a_{ji} \cdot b_i$, natomiast od j -tego wiersza macierzy A odejmujemy, przemnożony przez element macierzowy a_{ji} , i -ty wiersz macierzy A . Po wykonaniu powyższych kroków algorytmu, rozwiązaniem x jest wektor b .

W opisie metody Gaussa-Jordana w postaci listy kroków przyjmujemy, że A_i – i -ty wiersz macierzy A .

Algorytm eliminacji Gaussa-Jordana w postaci listy kroków:

- 1) Dla $i := 1$ aż do n wykonuj:
 - 1.1) $b_i := b_i \div a_{ii}$
 - 1.2) $A_i := A_i \div a_{ii}$
 - 1.3) Dla $j := 1$ aż do n wykonuj:
 - 1.3.1) jeśli $j \neq i$, to:
 - 1.3.1.1) $b_j := b_j - a_{ji} \cdot b_i$
 - 1.3.1.2) $A_j := A_j - a_{ji} \cdot A_i$
- 2) $x := b$

Listing solwera (GJ.m):

```
function x = GJ(A, b, n)
% Metoda eliminacji zupełnej Gaussa-Jordana
% Założenie: A(i,i) ~= 0 dla i=1,2,...,n

for i=1:n

    b(i) = b(i) / A(i,i);
    A(i,:) = A(i,:) / A(i,i);

    for j=1:n
        if j ~= i
            b(j) = b(j) - A(j,i) * b(i);
            A(j,:) = A(j,:) - A(j,i) * A(i,:);
        end
    end

end

x = b;

end
```

Sprawdzenie działania solwera

Listing (Zad2test.m):

```
% n = 5
% A = G * G^T, gdzie G = 10*rand(5)
% b = 30 * rand(5,1)

format long;

n = 5;
G = 10 * rand(5);
A = G * G.'; % A = G * G^T
b = 30 * rand(5,1);

tic;
x = GJ(A,b,n);
toc;

eps1 = norm( A*x - b );
fprintf('Bład obliczonego rozwiązania wynosi eps1 = %s\n', num2str(eps1));
```

Wywołanie:

The screenshot shows the MATLAB Command Window interface. At the top, there is a yellow banner with a question mark icon and the text "New to MATLAB? Watch this Video, see Examples, or read Getting Started." Below the banner, the command window displays the output of running the script "Zad2test". The output consists of three lines: ">> Zad2test", "Elapsed time is 0.000082 seconds.", and "Bład obliczonego rozwiązania wynosi eps1 = 3.8514e-13". The prompt "fx >>|" is visible at the bottom left of the window.

```
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.
>> Zad2test
Elapsed time is 0.000082 seconds.
Bład obliczonego rozwiązania wynosi eps1 = 3.8514e-13
fx >>|
```

Solwer znalazł rozwiązanie w czasie $0,000082 \text{ s} = 82 \mu\text{s}$.

Ponadto, norma błędu rozwiązania wyniosła $\varepsilon_1 = 3,8514 \cdot 10^{-13}$.

Można zatem przyjąć, że algorytm działa poprawnie.

Układ równań typu (a)

Listing skryptu wywołującego (Zad2a.m):

```
n = [5 10 50 100 200]';
eps1 = zeros( length(n), 1 );
T = zeros( length(n), 1 );

for k=1:length(n)

    % Generowanie macierzy A
    % A(i,j) = 12 dla i=j
    % A(i,j) = 3.8 dla i=j-1 lub i=j+1
    % A(i,j) = 0 dla pozostałych
    A = zeros( n(k) );

    for i=1:n(k)
        A(i,i) = 12;
    end

    for j=2:n(k)
        A(j-1,j) = 3.8;
    end

    for j=1:n(k)-1
        A(j+1,j) = 3.8;
    end

    % Generowanie wektora b
    % b(i) = 4.5 - 0.5*i
    b = zeros( n(k), 1 );

    for i=1:n(k)
        b(i) = 4.5 - 0.5*i;
    end

    tic;
    x = GJ( A, b, n(k) );
    T(k) = toc;

    eps1(k) = norm( A*x - b );
end

T = T .* 1000; % aby wyniki były w ms
```

```

% Wykres bledu rozwiazania od liczby rownan
figure(1);
hold on;

plot(n,eps1,'r.', 'MarkerSize', 25);
grid on;

title('Zaleznosc bledu rozwiazania od liczby rownan (czesc a)');
xlabel('Liczba rownan n');
ylabel('Blad rozwiazania eps1');

hold off;

% Wykres czasu dzialania algorytmu od liczby rownan
figure(2);
hold on;

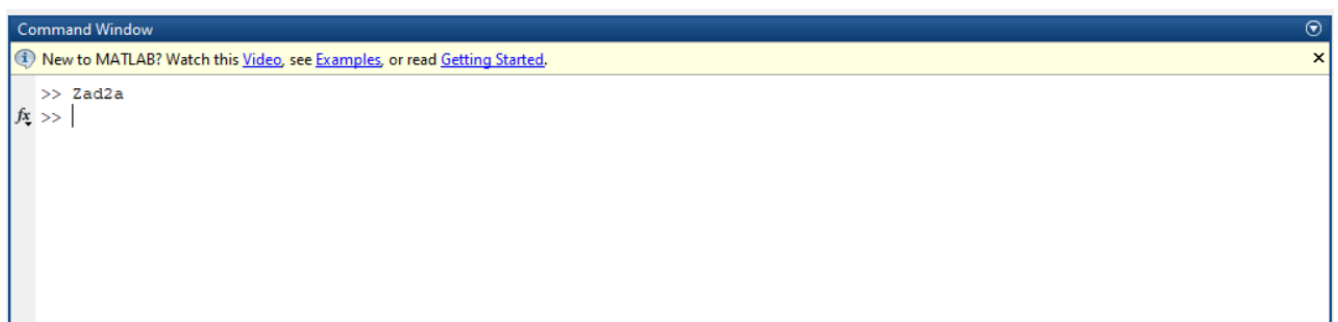
plot(n,T,'r.', 'MarkerSize', 25);
grid on;

title('Zaleznosc czasu dzialania algorytmu od liczby rownan (czesc a)');
xlabel('Liczba rownan n');
ylabel('Czas dzialania algorytmu [ms]');

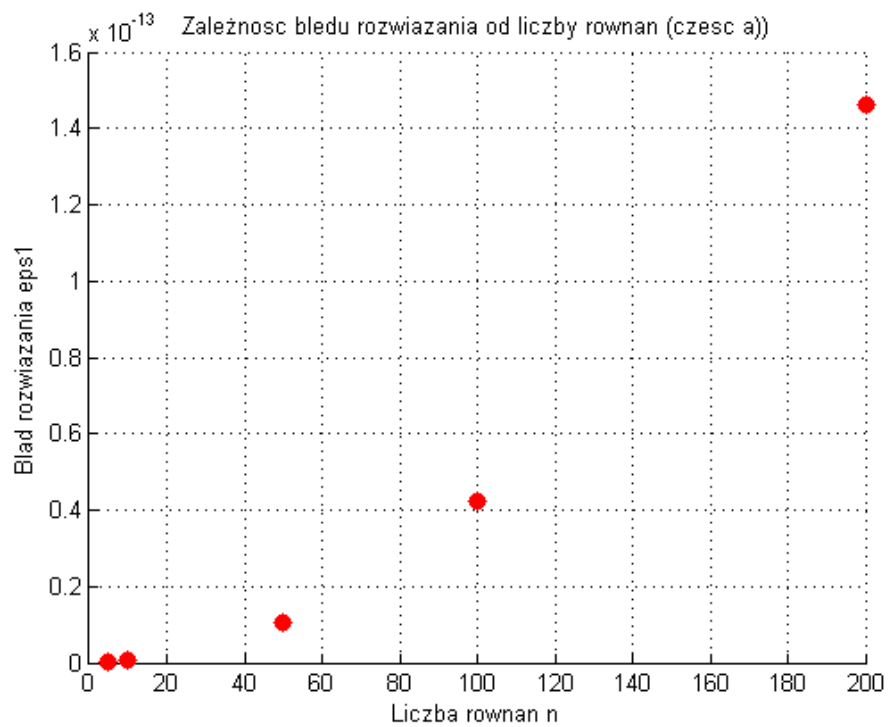
hold off;

```

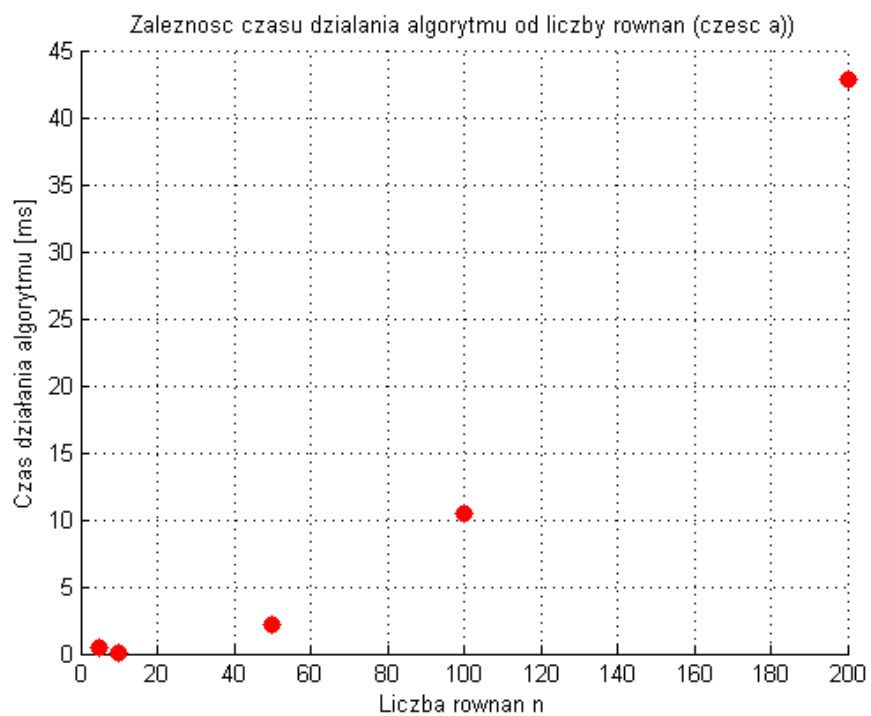
Wywołanie:



Wykresy:



Wykres 1. Zależność błędu rozwiązania ϵ_1 od liczby równań n



Wykres 2. Zależność czasu działania algorytmu od liczby równań n

Komentarz:

Wraz ze wzrostem parametru n , wzrasta (nieliniowo) wartość błędu ε_1 ,
jak również wzrasta (nieliniowo) czas działania solwera.

Niemniej, maksymalny błąd jest rzędu 10^{-13} , natomiast czas działania solwera
nie przekracza kilkudziesięciu ms.

Możemy więc powiedzieć, że solwer daje zadowalające wyniki
w relatywnie krótkim czasie.

Układ równań typu (b)

Listing skryptu wywołującego (Zad2b.m):

```
n = [5 10 50 100 200]';  
eps1 = zeros( length(n), 1 );  
T = zeros( length(n), 1 );  
  
for k=1:length(n)  
  
    % Wygenerowanie macierzy A  
    % A(i,j) = 1/3 dla i=j  
    % A(i,j) = 3*(i-j)+3 dla pozostałych  
    A = zeros( n(k) );  
  
    for i=1:n(k)  
        for j=1:n(k)  
            if i==j  
                A(i,j) = 1/3;  
            else  
                A(i,j) = 3*(i-j)+3;  
            end  
        end  
    end  
end
```

```

% Wygenerowanie wektora b
%  $b(i) = 0.5 - 0.6 \cdot i$ 
b = zeros( n(k), 1 );

for i=1:n(k)
    b(i) = 0.5 - 0.6*i;
end

tic;
x = GJ( A, b, n(k) );
T(k) = toc;

eps1(k) = norm( A*x - b );
end

T = T .* 1000; % aby wyniki byly w ms

% Wykres bledu rownian od liczby rownan
figure(1);
hold on;

plot(n,eps1,'r.', 'MarkerSize', 25);
grid on;

title('Zaleznosc bledu rownian od liczby rownan (czesc b)');
xlabel('Liczba rownan n');
ylabel('Bled rownian eps1');

hold off;

% Wykres czasu dzialania algorytmu od liczby rownan
figure(2);
hold on;

plot(n,T,'r.', 'MarkerSize', 25);
grid on;

title('Zaleznosc czasu dzialania algorytmu od liczby rownan (czesc b)');
xlabel('Liczba rownan n');
ylabel('Czas dzialania algorytmu [ms]');

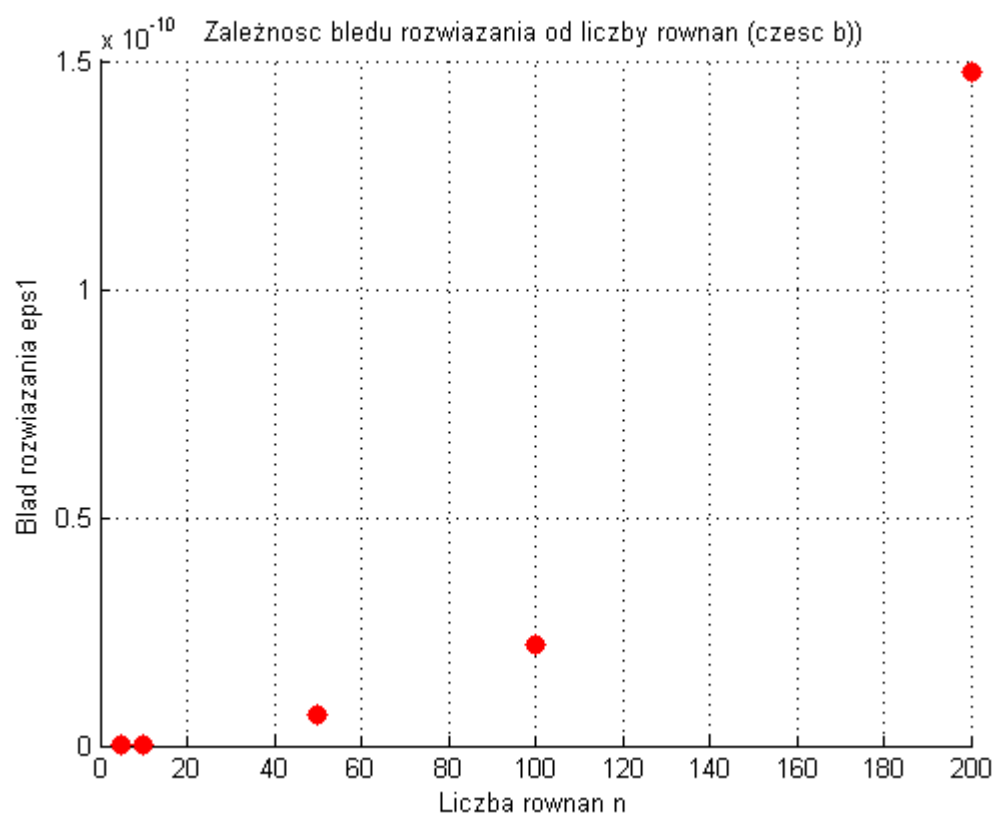
hold off;

```

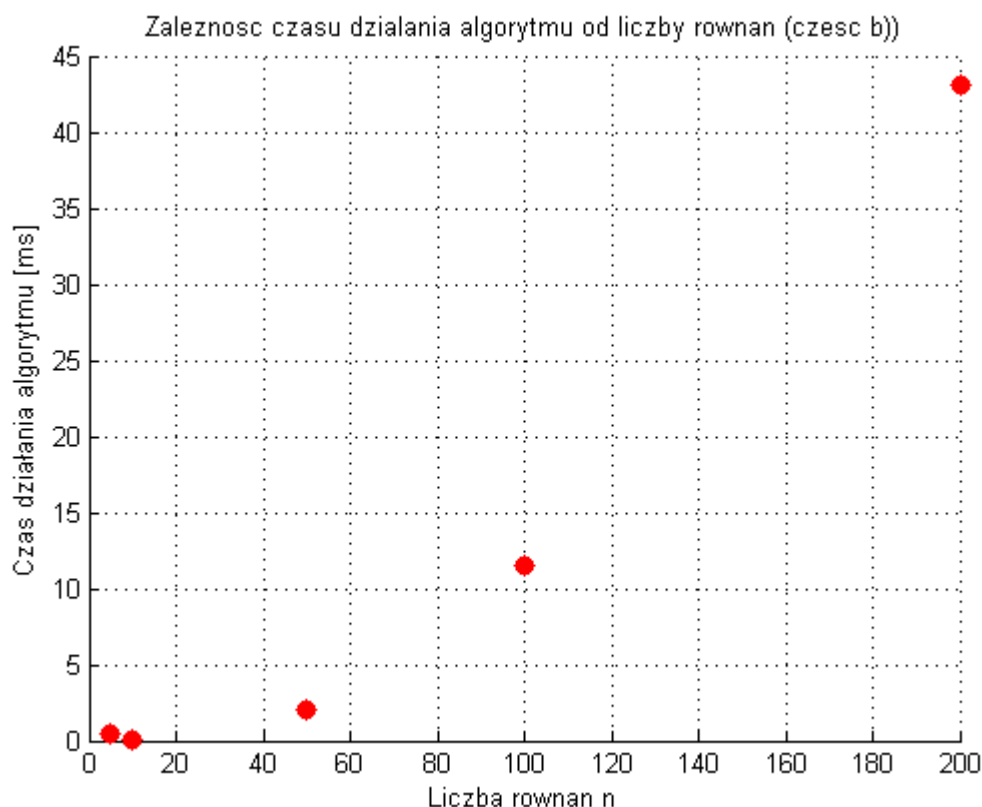
Wywołanie:

```
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.
>> Zad2b
fx >>
```

Wykresy:



Wykres 3. Zależność błędu rozwiązania ε_1 od liczby równań n



Wykres 4. Zależność czasu działania algorytmu od liczby równań n

Komentarz:

Wraz ze wzrostem parametru n , wzrasta (nieliniowo) wartość błędu ε_1 , jak również wzrasta (nieliniowo) czas działania solwera.

Niemniej, maksymalny błąd jest rzędu 10^{-10} , natomiast czas działania solwera nie przekracza kilkudziesięciu ms.

Możemy więc powiedzieć, że solwer daje dość zadowalające wyniki w relatywnie krótkim czasie.

W porównaniu do punktu (a), maksymalny błąd jest o 3 rzędy wielkości większy, natomiast czas działania jest zbliżony. Większe błędy mogą wynikać z gorszego uwarunkowania macierzy w punkcie (b).

Zadanie 3.

Treść: Napisać solwer rozwiązujący układ n równań liniowych $Ax = b$, wykorzystując metodę iteracyjną Gaussa-Seidela. Jego parametry wejściowe powinny zawierać błąd graniczny ε_2 liczony jako norma euklidesowa różnicy między kolejnymi przybliżeniami rozwiązania. Proszę spróbować zastosować swój solwer do rozwiązywania układów równań z p. 2a, 2b.

Rozwiązanie:

Metoda Gaussa-Seidela jest metodą iteracyjną służącą do rozwiązywania układów równań [1].

Iterację $(k + 1)$ -szą rozwiązania opisuje zależność:

$$Dx^{(k+1)} = -Lx^{(k+1)} - Ux^{(k)} + b, \quad k = 0, 1, 2, \dots [1].$$

Oczywiście, $x^{(0)}$ to przybliżenie początkowe rozwiązania.

Warto zaznaczyć, że macierze L, D, U to (odpowiednio) macierze poddiagonalna, diagonalna oraz naddiagonalna, powstałe z dekompozycji macierzy układu równań A .

Jeśli zdefiniujemy $w^{(k)} := Ux^{(k)} - b$,

to i -ty ($i = 1, 2, \dots, n$) element wektora $x^{(k+1)}$ ma wzór

$$x_i^{(k+1)} = \frac{1}{d_{ii}} \cdot \left(-w_i^{(k)} - \sum_{j=1}^{i-1} l_{ij} \cdot x_j^{(k+1)} \right).$$

W ramach testu stopu, możemy sprawdzać, czy norma euklidesowa różnicy kolejnych iteracji rozwiązania nie przekracza pewnej ustalonej wartości ε_2 lub nie osiągnęliśmy ustalonej liczby iteracji max_iter .

Algorytm iteracyjny Gaussa-Seidela w postaci listy kroków:

- 1) Zdekomponuj macierz A do macierzy L, D, U
- 2) Przyjmij przybliżenie początkowe rozwiązania $x := x_0$
- 3) Ustaw licznik iteracji na 1 ($iter := 1$)
- 4) Wykonuj w pętli nieskończonej:
 - 4a) $w := U \cdot x - b$
 - 4b) $xNext_i := \frac{1}{d_{ii}} \cdot \left(-w_i - \sum_{j=1}^{i-1} l_{ij} \cdot xNext_j \right)$
 - 4c) Jeśli $\|xNext - x\| < \varepsilon_2$ lub $iter > max_iter$, to przyjmij rozwiązanie $x := xNext$ i zakończ działanie funkcji
 - 4d) Jeśli nie zachodzi warunek z punktu 4c), to:
 - 4d.1) $x := xNext$
 - 4d.2) $iter := iter + 1$

Listing solwera (GS.m):

```
function x = GS(A, b, n, eps2, max_iter)
% Metoda iteracyjna Gaussa-Seidela
% Zalozenie: A(i,i)~=0 dla i=1,2,...,n

% Dekompozycja macierzy A
% L - macierz poddiagonalna
% D - macierz diagonalna
% U - macierz naddiagonalna
L = zeros(n);
D = zeros(n);
U = zeros(n);

for i=1:n
    for j=1:n
        if i==j
            D(i,i) = A(i,i);
        elseif i < j
            U(i,j) = A(i,j);
        else
            L(i,j) = A(i,j);
        end
    end
end

x = zeros(n,1); % przyblizenie startowe
xNext = zeros(n,1); % zmienna pomocnicza: kolejne przyblizenie rozwiazania

iter = 1;

while 1
    w = U*x - b;

    for i=1:n

        sum = 0;

        for j=1:i-1
            sum = sum + L(i,j) * xNext(j);
        end

        xNext(i) = (-w(i) - sum) / D(i,i);

    end

    if norm( xNext - x ) < eps2 || iter > max_iter
        x = xNext;
        break;
    else
        x = xNext;
        iter = iter + 1;
    end
end

end
```

Układ równań typu (a)

Listing skryptu wywołującego (Zad3a.m):

```
n = [5 10 50 100 200]';
eps1 = zeros( length(n), 1 );
eps2 = 1e-6;
max_iter = 100;

T = zeros( length(n), 1 );

for k=1:length(n)

    % Generowanie macierzy A
    % A(i,j) = 12 dla i=j
    % A(i,j) = 3.8 dla i=j-1 lub i=j+1
    % A(i,j) = 0 dla pozostałych
    A = zeros( n(k) );

    for i=1:n(k)
        A(i,i) = 12;
    end

    for j=2:n(k)
        A(j-1,j) = 3.8;
    end

    for j=1:n(k)-1
        A(j+1,j) = 3.8;
    end

    % Generowanie wektora b
    % b(i) = 4.5 - 0.5*i
    b = zeros( n(k), 1 );

    for i=1:n(k)
        b(i) = 4.5 - 0.5*i;
    end

    tic;
    x = GS( A, b, n(k), eps2, max_iter );
    T(k) = toc;

    eps1(k) = norm( A*x - b );
end
```

```

T = T .* 1000; % aby czasy byly w ms

% Wykres bledu rozwiazania od liczby rownan
figure(1);
hold on;

plot(n,eps1,'r.', 'MarkerSize', 25);
grid on;

title('Zaleznosc bledu rozwiazania od liczby rownan (czesc a)');
xlabel('Liczba rownan n');
ylabel('Blad rozwiazania eps1');

hold off;

% Wykres czasu dzialania algorytmu od liczby rownan
figure(2);
hold on;

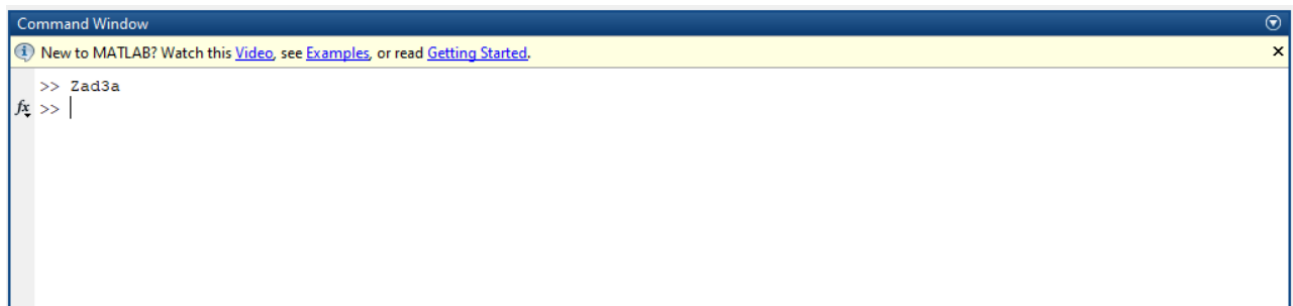
plot(n,T,'r.', 'MarkerSize', 25);
grid on;

title('Zaleznosc czasu dzialania algorytmu od liczby rownan (czesc a)');
xlabel('Liczba rownan n');
ylabel('Czas dzialania algorytmu [ms]');

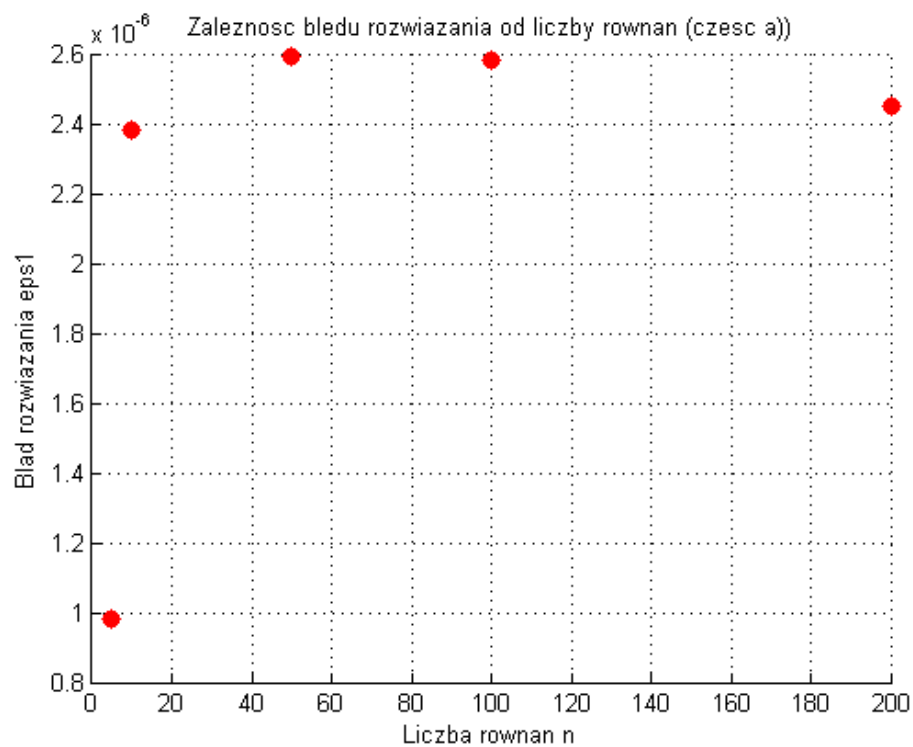
hold off;

```

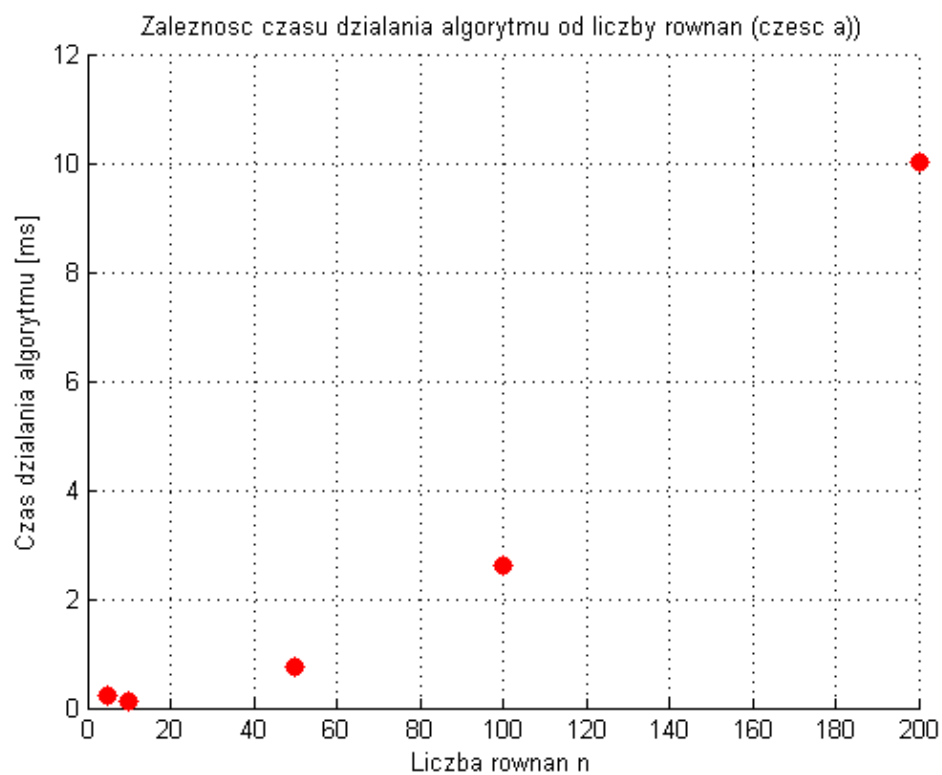
Wywołanie:



Wykresy:



Wykres 5. Zależność błędu rozwiązania ε_1 od liczby równań n



Wykres 6. Zależność czasu działania algorytmu od liczby równań n

Komentarz:

Analizując wykres 5., możemy stwierdzić, że dla małych liczb równań, norma błędu znalezionej rozwiązania wzrasta. Z kolei idąc w kierunku większych liczb równań, obserwujemy stabilizację normy błędu rozwiązania, a nawet niewielki spadek. Analizując wykres 6., możemy stwierdzić, że czas działania algorytmu wzrasta (nieliniowo) wraz ze wzrostem liczby równań. Ponadto, maksymalny czas działania nie przekracza 10 ms, co jest wynikiem lepszym od maksymalnego czasu działania metody Gaussa-Jordana (między 40 a 45 ms). Widzimy więc, że stosując metodę iteracyjną, istnieje możliwość znalezienia rozwiązania szybciej niż w przypadku użycia metody dokładnej.

Układ równań typu (b)

Listing skryptu wywołującego (Zad3b.m):

```
n = [5 10 50 100 200]';
eps1 = zeros( length(n), 1 );
eps2 = 1e-6;
max_iter = 100;

T = zeros( length(n), 1 );

for k=1:length(n)

    % Wygenerowanie macierzy A
    % A(i,j) = 1/3 dla i=j
    % A(i,j) = 3*(i-j)+3 dla pozostałych
    A = zeros( n(k) );

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

```

% Wygenerowanie wektora b
%  $b(i) = 0.5 - 0.6 \cdot i$ 
b = zeros( n(k), 1 );

for i=1:n(k)
    b(i) = 0.5 - 0.6*i;
end

tic;
x = GS( A, b, n(k), eps2, max_iter );
T(k) = toc;

eps1(k) = norm( A*x - b );
end

T = T .* 1000; % aby czasy byly w ms

% Wykres bledu rozwiazania od liczby rownan
figure(1);
hold on;

plot(n,eps1,'r.', 'MarkerSize', 25);
grid on;

title('Zaleznosc bledu rozwiazania od liczby rownan (czesc b)');
xlabel('Liczba rownan n');
ylabel('Blad rozwiazania eps1');

hold off;

% Wykres czasu dzialania algorytmu od liczby rownan
figure(2);
hold on;

plot(n,T,'r.', 'MarkerSize', 25);
grid on;

title('Zaleznosc czasu dzialania algorytmu od liczby rownan (czesc b)');
xlabel('Liczba rownan n');
ylabel('Czas dzialania algorytmu [ms]');

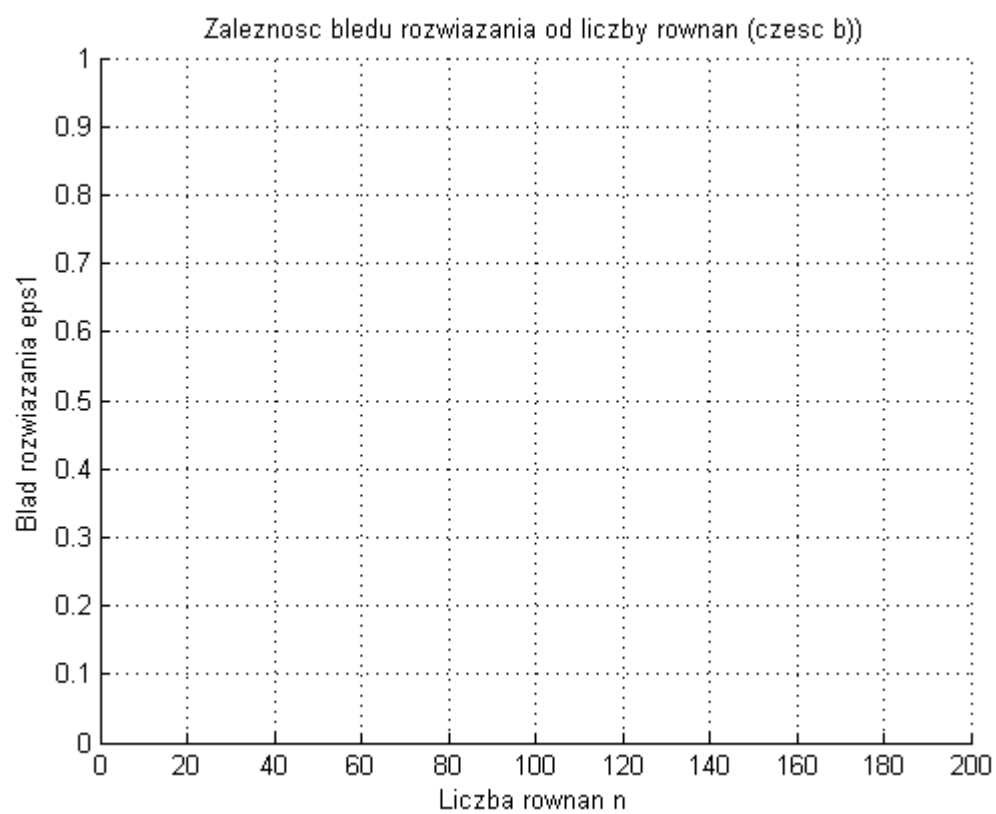
hold off;

```

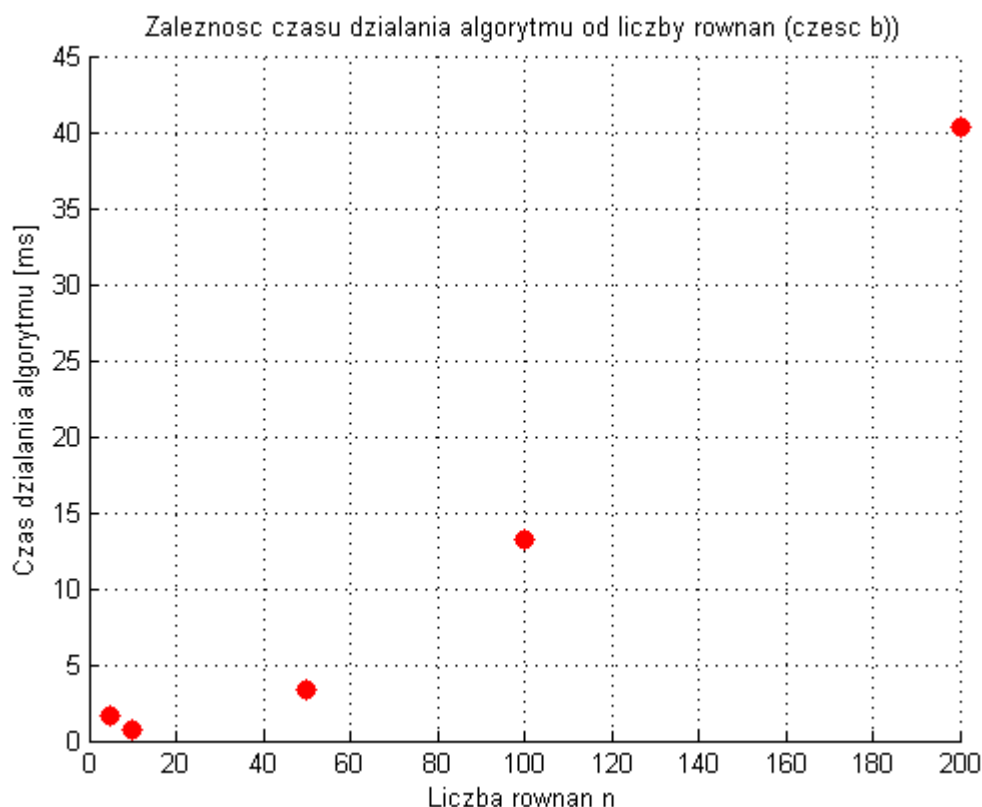
Wywołanie:

```
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.
>> Zad3b
fx >> |
```

Wykresy:



Wykres 7. Zależność błędu rozwiązania ε_1 od liczby równań n



Wykres 8. Zależność czasu działania algorytmu od liczby równań n

Komentarz:

Przykład ten dowodzi, że metoda Gaussa-Seidela nie zawsze jest zbieżna (zbieżność osiągamy, gdy macierz A jest silnie diagonalnie dominująca lub jest symetryczna i dodatnio określona [1]). Świadczy o tym brak punktów na wykresie 7. Skoro nie została osiągnięta zbieżność, to dla każdej liczby równań n wykonana została maksymalna liczba iteracji max_iter ; analizując wykres 8., widzimy, że dla rosnącej liczby równań, rośnie (nieliniowo) czas działania algorytmu.

Zaprezentuję krótkie uzasadnienie, że macierz A nie spełnia warunków zbieżności dla metody Gaussa-Seidela (dla liczby równań $n \geq 2$).

Dla przypomnienia, $a_{ij} = 3(i - j) + 3$; $a_{ii} = \frac{1}{3}$ ($i, j = 1, 2, \dots, n$; $i \neq j$).

Zauważmy, że macierz A nie jest symetryczna

(bo np. $a_{12} = 3 \cdot (1 - 2) + 3 = 0 \neq a_{21} = 3 \cdot (2 - 1) + 3 = 6$).

Dalej, macierz A nie jest silnie diagonalnie dominująca wierszowo

(bo np. $|a_{22}| = \left|\frac{1}{3}\right| = \frac{1}{3} < |a_{21}| = |3 \cdot (2 - 1) + 3| = 6$).

Zauważmy również, że macierz A nie jest silnie diagonalnie dominująca kolumnowo

(bo np. $|a_{11}| = \left|\frac{1}{3}\right| = \frac{1}{3} < |a_{21}| = |3 \cdot (2 - 1) + 3| = 6$).

Bibliografia:

[1] Piotr Tatjewski „Metody Numeryczne”, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2013