

Spis treści

Zadanie 1. <i>Epsilon maszynowy</i>	2
Zadanie 2. <i>Eliminacja Gaussa z pełnym wyborem elementu podstawowego</i>	3
Zadanie 3. <i>Metoda Gaussa-Seidel'a</i>	8
Załącznik 1. <i>Kod źródłowy zadania 1.</i>	10
Załącznik 2. <i>Kod źródłowy zadania 2.</i>	11
Załącznik 3. <i>Kod źródłowy zadania 3.</i>	15

Zadanie 1. Epsilon maszynowy

Celem zadania jest wyznaczenie dokładności maszynowej komputera.

Teoria

Dokładność maszynowa nazywana również epsilon maszynowym (w skrócie *eps*) to najmniejsza liczba g ze zbioru liczb mogących być zapisanych w komputerze (za pomocą określonej precyzji) spełniająca nierówność:

$$1+g>1 \quad (1.01)$$

W liczbach rzeczywistych nierówność 1.01. nie ma rozwiązania, jednakże przez to, że komputer jest maszyną opartą na systemie binarnym dysponujemy tylko wycinkiem zbioru liczb rzeczywistych. Dzięki temu wyznaczenie liczby g jest możliwe.

Koncepcja rozwiązania

Notacją stosowaną w komputerach jest zapis liczby za pomocą mantysy oraz wyznacznika. Dlatego do wyznaczenia liczby *eps* sprowadza się do jednej pętli.

```
1. while (1 + 2^i > 1)
2.     i = i - 1;
3. end
4. i = i + 1;                                     (1.02)
```

W linii nr 1 sprawdzamy czy warunek na liczbę g został spełniony, jeśli nie - sprawdzamy dalej.

Gdy warunek zostanie spełniony osiągniemy liczbę 2^x , gdzie $1+2^x \leq 1$ oraz $1+2^{(x+1)} > 1$ dlatego w linii 4 musimy pamiętać o zwiększeniu wyznacznika o 1. Na skutek tego otrzymamy i będące wyznacznikiem liczby $g = 2^i$.

Sprawdzenie

Otrzymany wynik (Lenovo Thinkpad T420, Windows 10, 64 bit) to $2.2204e-16$.

W celu sprawdzenia wyniku można wywołać wbudowaną w MatLaba funkcję `eps`. W powyższym przypadku dała ten sam wynik.

Komentarz

W celu uniknięcia jednej iteracji z listingu 1.02 można przepisać podaną pętlę do formatu przedstawionego na listingu 1.03 i otrzymać ten sam efekt.

```
1. while (1 + 2^(i-1) > 1)
2.     i = i - 1;
3. end                                           (1.03)
```

Nie zostało to jednak zrobione w tym ćwiczeniu, aby dokładnie przedstawić rozwiązanie.

Zadanie 2. Eliminacja Gaussa z pełnym wyborem elementu podstawowego

Celem zadania jest:

1. Napisanie programu rozwiązującego równania liniowe postaci $Ax = b$ przy użyciu metody eliminacji Gaussa z pełnym wyborem elementu podstawowego.
2. Przetestowanie skuteczności działania dla 3 typy równań liniowych dla różnej ilości równań:

$$a. \quad a_{ij} = \begin{cases} 6 & \text{dla } i = j \\ 2 & \text{dla } i = j - 1 \text{ lub } i = j + 1, \\ 0 & \text{dla pozostałych} \end{cases} \quad b = 9 + 0,5 * i;$$

$$b. \quad a_{ij} = 5 * (i - j) + 1; \quad a_{ii} = 1/8; \quad b = -3 + 0,5 * i;$$

$$c. \quad a_{ij} = 4 / [5 * (i + j - 1)]; \quad b_{i=2n+1} = 1/2 * i; \quad b_{i=2n} = 0;$$

3. Wyznaczenie błędu rozwiązania (norma residuum).
4. Sporządzenie wykresu przedstawiającego zależność błędu od liczby równań.

Teoria

1. Do rozwiązywania równań została zaimplementowana klasyczna metoda eliminacji Gaussa z pełnym wyborem elementu podstawowego.

2. Przykładowe macierze równań typu:

$$a. \quad \begin{bmatrix} 6 & 2 & 0 & 0 \\ 2 & 6 & 2 & 0 \\ 0 & 2 & 6 & 2 \\ 0 & 0 & 2 & 6 \end{bmatrix} \begin{bmatrix} 9,5 \\ 10 \\ 10,5 \\ 11 \end{bmatrix}$$

$$b. \quad \begin{bmatrix} \frac{1}{8} & -4 & -9 & -14 \\ 6 & \frac{1}{8} & -4 & -9 \\ 11 & 6 & \frac{1}{8} & -4 \\ 16 & 11 & 6 & \frac{1}{8} \end{bmatrix} \begin{bmatrix} -2,5 \\ -2 \\ -1,5 \\ -1 \end{bmatrix}$$

$$c. \quad \begin{bmatrix} \frac{4}{5} & \frac{4}{4} & \frac{4}{4} & \frac{4}{4} \\ \frac{10}{4} & \frac{15}{4} & \frac{20}{4} & \frac{25}{4} \\ \frac{15}{20} & \frac{20}{25} & \frac{25}{30} & \frac{30}{35} \\ \frac{4}{20} & \frac{4}{25} & \frac{4}{30} & \frac{4}{35} \end{bmatrix} \begin{bmatrix} 0,5 \\ 0 \\ 1,5 \\ 0 \end{bmatrix}$$

3. Norma residuum została obliczona jako norma wektora $Ax - b$ gdzie:

- a. **A** to macierz wejściowa ze zmienioną kolejnością wierszy i kolumn (bez jakichkolwiek zmian wartości) tak jak było to realizowane w algorytmie Gaussa.
- b. **x** to wektor rozwiązań po zastosowaniu eliminacji Gaussa z pełnym wyborem elementu głównego.
- c. **b** to wektor wyrazów wolnych służący do sprawdzenia poprawności otrzymanych wyników (z przedstawionymi wierszami analogicznie jak A).

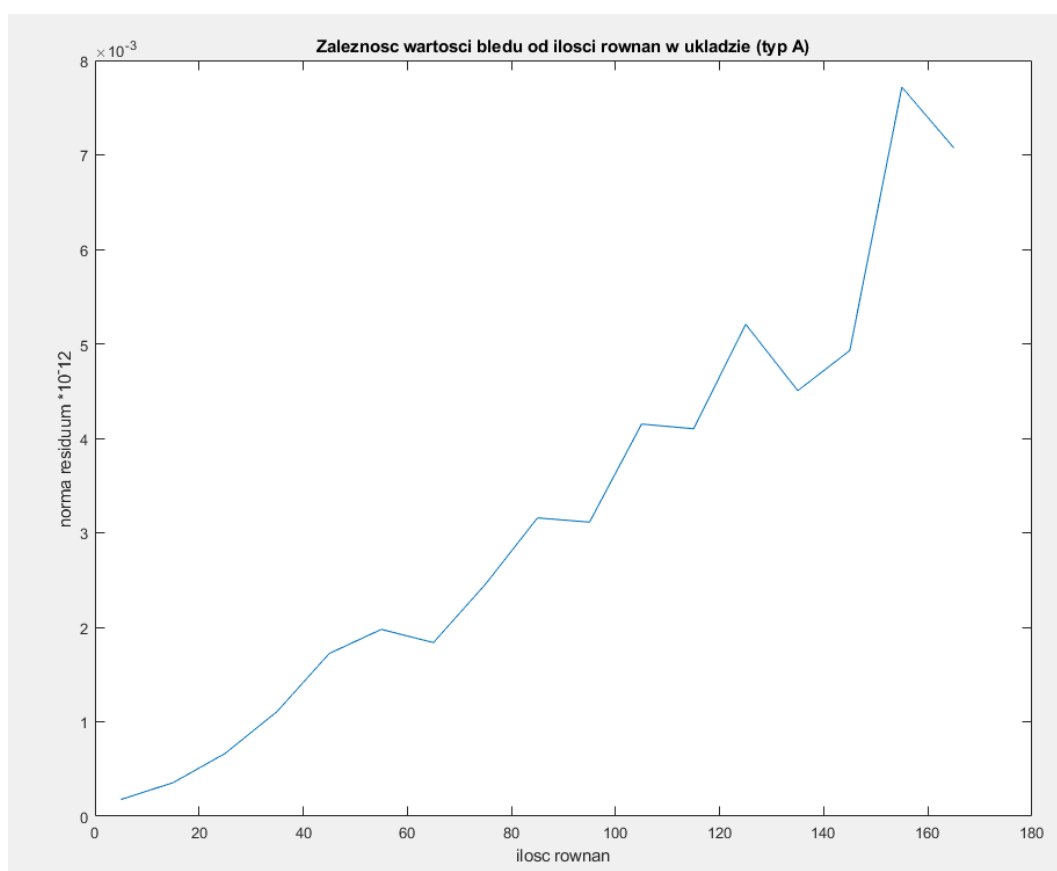
Koncepcja rozwiązania

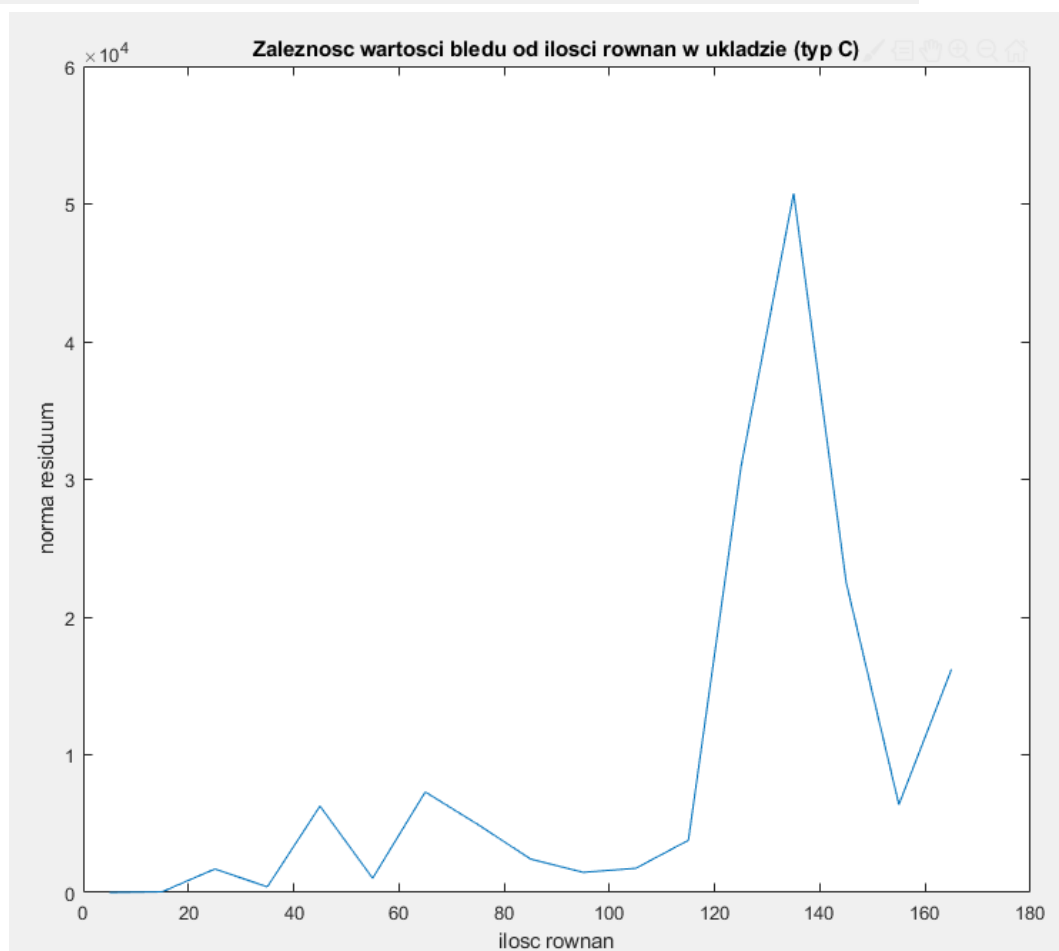
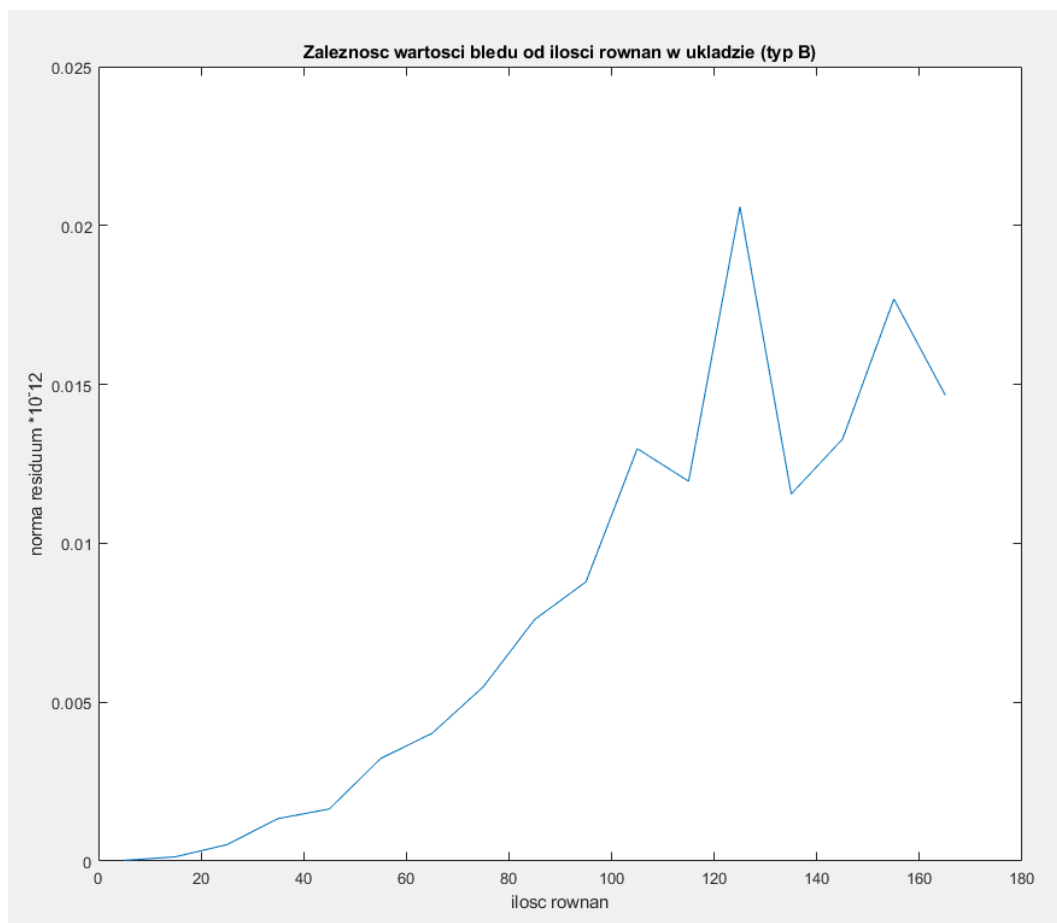
1. Dla wejścia w postaci macierzy współczynników *wspolczynniki* oraz wektora wyrazów wolnych *rozw* algorytm działa następująco:
 - a. Tworzenie zmiennej iteracyjnej *i* mówiącej o tym, w którym kroku algorytmu aktualnie jesteśmy oraz *rozmiar* mówiącej m.in. o wymiarach macierzy współczynników.
 - b. W każdym kolejnym kroku rozważana jest macierz kwadratowa będąca podmacierzą *wspolczynniki* składająca się z wierszy oraz kolumn od *i* do *rozmiar* (analogicznie dla wektora *rozw*).
 - c. Na wybranej podmacierzy dokonywany jest wybór elementu maksymalnego.
 - d. Zostaje dokonana zamiana kolumny oraz wiersza, w których jest element maksymalny z kolumną i wierszem o współrzędnych *i,i* – tak aby element maksymalny był w lewym górnym rogu wybranej podmacierzy.
 - e. Dla wierszy od *i+1* do *rozmiar* zostają wyznaczone współczynniki (*w*), dzięki którym możliwe będzie wyzerowanie elementów poniżej aktualnego elementu maksymalnego.
 - f. Wiersz główny podmacierzy zostaje przemnożony przez $(-1)^w$ (korespondujący współczynnik „zerujący”). Odejmowanie z wykorzystaniem współczynników ma również miejsce na kolumnie wyrazów wolnych.
 - g. Po wykonaniu kroków od b do f dla całej macierzy otrzymana zostaje macierz schodkowa, w której element o współrzędnych *rozmiar,rozmiar* jest znany. Wykorzystując tą wiedzę wyznaczony zostaje element *rozmiar-1,rozmiar-1* analogicznie eskalując w górę metodę aż do komórki *1,1*.
 - h. Dla zachowania przejrzystości rozwiązanie zostaje wyświetlone w wektorze *rozw*.
2. Podczas tworzenia macierzy współczynników oraz macierzy wyrazów wolnych została zwrócona uwaga na zminimalizowanie liczby obliczeń do minimum, a w zamian za to zwiększenie liczby przypisań. Przykładowo:
 - Typ a – kolumna wyrazów wolnych powstaje poprzez wpisanie jako pierwszy element 9,5. Każdy kolejny powstaje przez dodanie $r = 0.5$. Skutek – brak mnożenia.
 - Typ b – początkowo zostaje obliczona pierwsza i ostatnia kolumna (analogicznie do typu a). W kolejnym etapie wartości z odpowiednich komórek zostają przepisane zgodnie ze strzałkami zaznaczonymi w nagłówku *Teoria*.
 - Typ c – możliwe do zaimplementowania w podobny sposób elementy zostały napisane analogicznie. Również następuje „strzałkowe” przypisanie. Po

obliczeniu wartości dla pierwszej kolumny następuje przepisanie odpowiednich elementów zaś wartość na dole każdej kolejnej kolumny jest wyliczana.

Sprawdzenie

1. Sprawdzenie zgodności względem algorytmu Gaussa-Seidela.
2. Sprawdzenie algorytmu tworzenia macierzy pod względem poprawności implementacji oraz wizualne – brak newralgicznych miejsc, w których mógłby nie działać.
3. -
4. Wykresy:





Komentarz

1. Operacje mające na celu zamianę wierszy/kolumny macierzy można zastąpić mnożeniem przez zmodyfikowaną macierz jednostkową.
Przykładowo liniijkę:
`wspolczynniki(:, [1, index_max]) = wspolczynniki(:, [index_max, 1]);`
można zastąpić mnożeniem prawostronnym przez macierz jednostkową z zamienioną kolumną 1 z kolumną o indeksie `index_max`.
Mechanizm mnożenia przez zmodyfikowane macierze jednostkowe nie został zaimplementowany z dwóch względów:
 - a. Lepsza czytelność kodu algorytmu eliminacji Gaussa – a to on jest kluczowym elementem zadania.
 - b. Do utworzenia zmodyfikowanej macierzy jednostkowej można użyć pętli, co negatywnie wpłynęłoby na zrozumienie całości kodu dodając dodatkowy element konieczny do zrozumienia. Gdyby chcieć uprościć tworzenie macierzy zamiany wierszy/kolumn należałoby użyć liniijki:
`macierz_zmiany(:, [1, index_max]) =
macierz_jedn(:, [index_max, 1]);`
Która sama w sobie wykorzystuje wbudowany mechanizm zamiany wierszy/kolumn. Dlatego zostało uznane, że lepiej wykorzystywać ten mechanizm bezpośrednio na macierzy współczynników.
2. Algorytmy tworzące macierze typu A-C można wyeksportować do oddzielnego pliku jednakże chciałem ująć każde zadanie w osobnym, niezależnym pliku.
3. Wniosek: Łatwo zauważyć, że błąd jest tym większy im większa jest ilość równań w układzie (a) przy czym momentalnie występują piki (b), np. dla $x \approx 124$ dla typu A, dla $x \approx 127$ dla typu B oraz dla $x \approx 136$ dla typu C. Ciekawym jest fakt, że błędy między typem A i B w stosunku do C to ponad 16 rzędów wielkości (c).
 - a. Wielkość błędu rośnie wraz z ilością równań ponieważ taka jest specyfika metody Gaussa – wyznaczamy „dolną wartość”, która potem eskaluje dalej. Przez to właśnie jeśli wyznaczona zostanie wartość x_n z pewnym błędem kolejne wartości będą zwiększać błąd dotyczący pewnego x_i gdyż będzie on wyznaczany na podstawie już na wstępie błędnej wartości.
 - b. Piki na wykresach prawdopodobnie wynikają z niewymierności, które zostają w pewnym momencie wyolbrzymione, gdyż stają się podstawą do dalszych obliczeń tak jak napisane w punkcie a.
 - c. Jak łatwo zauważyć macierz A i B zawiera jedynie elementy całkowite lub wartości 2^{-3} . W opozycji do tego stoi macierz C, która już w trzecim wierszu pierwszej kolumny przyjmie niewymierną wartość równą $4/15$.

Zadanie 3. Metoda Gaussa-Seidel'a

Celem zadania jest:

1. Napisanie programu rozwiązującego równania liniowe postaci $Ax = b$ przy użyciu iteracyjnej metody - Gaussa-Seidel'a.
2. Zastosować zaimplementowany algorytm do:
 - a. Zadanego układu równań.
 - b. Układów równań z poprzedniego zadania.
3. Sprawdzić dokładność rozwiązania dla zadanego układu równań.

Teoria

1. Do rozwiązywania równań została zaimplementowana metoda Gaussa-Seidel'a z warunkiem stopu precyzji 0,1% oraz początkową wartością wszystkich współczynników na 0.
2. -
3. Błąd zostanie obliczony jako norma residuum analogicznie jak w *Zadaniu 2*.

Koncepcja rozwiązania

1. Metoda Gaussa-Seidel'a w tej implementacji:
 - a. Inicjuje wszystkie niewiadome zerami.
 - b. Rozbija macierz na sumę $L + D + U$.
 - c. Odwraca macierz D obliczając odwrotności wyrazów na przekątnej.
 - d. Oblicza iloczyny $D^{-1} * b$, $D^{-1} * L$, $D^{-1} * U$.
 - e. Zespala w jedną macierz $(-1) * (D^{-1} * L + D^{-1} * U)$.
 - f. Do czasu osiągnięcia wymaganej precyzji oblicza x^i iteracyjnie jako iloczyn składających się na niego wyrazów i ich dotychczasowych wartości.

Sprawdzenie

1. Algorytm będzie poprawny gdy jego wyniki będą zbliżone do wyników osiągniętych przez algorytm z zad. 2, a norma residuum będzie dostatecznie mała. Po sprawdzeniu na zadanych przykładach efekt ten został osiągnięty.
2. b. Zgodnie z oczekiwaniami algorytm Gaussa-Seidel'a jest zdecydowanie wolniejszy od Gaussa gdyż musi on osiągnąć warunek stopu i liczba kroków nie jest z góry znana. przy założonym rozmiarze macierzy 50 obliczenia wydawały się nie kończyć. Dlatego ostatecznie został przetestowany tylko dla pętli rozmiar=5:3:15.

Błąd dla macierzy A i C wynosi kolejno:

```
blad_res_a = 5x2
      0      0
0.0130  5.0000
0.0183  8.0000
0.0214 11.0000
0.0242 14.0000
```



```
blad_res_c = 5x2
0          0
1.9508     5.0000
3.9384     8.0000
7.0873    11.0000
9.3216    14.0000
```

Z kolei dla macierzy B nie można określić dokładności, gdyż osiągnął się wynik NaN co jest związane z tym, że macierz B nie spełnia warunków diagonalnej dominacji.

3. Dla zadanego układu równań norma residuum wynosi 0.0211.

Załącznik 1. Kod źródłowy zadania 1.

obliczenie epsa maszynowego

```
i = 0;
while (1 + 2^i > 1)
    i = i - 1;
end
i = i + 1;
fprintf('epsilon maszynowy to %e\n', 2^i);
fprintf('czyli 2^%d\n', i);
```

wywołanie funkcji matlaba obliczającej eps maszynowy

```
fprintf('epsilon maszynowy wyznaczony przez MatLaba to: %e\n', eps);
```

sprawdzenie

```
if (eps == 2^i)
    disp("eps wyznaczony poprawnie")
else
    disp("błąd wyznaczenia epsa")
end
```

Output

```
epsilon maszynowy to 2.220446e-16
czyli 2^-52
epsilon maszynowy wyznaczony przez MatLaba to: 2.220446e-16
eps wyznaczony poprawnie
```

Załącznik 2. Kod źródłowy zadania 2.

zaimportowanie macierzy

```
clear;
clc;
typ = 3; %a-1,b-2,c-3
blad_res_a = zeros(17,2);
blad_res_b = zeros(17,2);
blad_res_c = zeros(17,2);
for r = 5:10:165
    for typ = 1:3
        if(typ == 1)
            temp = macierzA(r);
        elseif(typ == 2)
            temp = macierzB(r);
        elseif(typ == 3)
            temp = macierzC(r);
        end
        A = temp(:, 1:end-1);
        b = temp(:,end);
        % wywołanie funkcji Gaussa oraz residuum
        temp = gauss(A, b);
        A = temp(:, 1:end-2);
        rozwiazanie = temp(:, end-1);
        b = temp(:, end);
        a = norma_residuum(A, rozwiazanie, b);

        if(typ == 1)
            blad_res_a((r+5)/10,:) = [a*1000000000000, r];
        elseif(typ == 2)
            blad_res_b((r+5)/10,:) = [a*1000000000000, r];
        elseif(typ == 3)
            blad_res_c((r+5)/10,:) = [a, r];
        end
    end
end
end
```

utworzenie wykresow

```
plot(blad_res_a(:,2),blad_res_a(:,1));
title('Zaleznosc wartosci bledu od ilosci rownan w układzie (typ A)')
xlabel('ilosc rownan')
ylabel('norma residuum *10^-12')
plot(blad_res_b(:,2),blad_res_b(:,1));
title('Zaleznosc wartosci bledu od ilosci rownan w układzie (typ B)')
xlabel('ilosc rownan')
ylabel('norma residuum *10^-12')
plot(blad_res_c(:,2),blad_res_c(:,1));
title('Zaleznosc wartosci bledu od ilosci rownan w układzie (typ C)')
xlabel('ilosc rownan')
ylabel('norma residuum')
```

algorytm Gaussa

```

function g = gauss(wspolczynniki, rozw)
    rozmiar = size(rozw);
    pierwotne_b = rozw;
    pierwotne_wspolczynniki = wspolczynniki;
    % A B C
    % D E F
    % G H I
    % a - element uznany za glowny w danej kolumnie,
    % w tym przypadku kolejno: A, E
    % wspolczynniki(1,i) - wspolczynnik "pod schodkiem",
    % czyli tutaj: D/A, G/A, H/E
    for i = 1:(rozmiar-1)
        % wybor elementu glownego
        podmacierz = wspolczynniki(i:rozmiar, i:rozmiar);
        [el_glowny, ind_glowny] = max(podmacierz(:));
        [ind_wiersz, ind_kolumna] = ind2sub(size(podmacierz), ind_glowny);
        % z powyzszej linii uzyskujemy indexy w podmacierzy
        % wiec musimy dodac 'i-1'
        wspolczynniki([i,ind_wiersz+i-1],:) = wspolczynniki([ind_wiersz+i-
1,i],:);
        pierwotne_wspolczynniki([i,ind_wiersz+i-1],:) =
pierwotne_wspolczynniki([i,ind_wiersz+i-1],:);
        rozw([i,ind_wiersz+i-1],:) = rozw([ind_wiersz+i-1,i],:);
        pierwotne_b([i,ind_kolumna+i-1], :) = pierwotne_b([i,ind_kolumna+i-1],
:);
        wspolczynniki(:, [i,ind_kolumna+i-1]) = wspolczynniki(:, [ind_kolumna+i-
1,i]);
        pierwotne_wspolczynniki(:, [i,ind_kolumna+i-1]) =
pierwotne_wspolczynniki(:, [ind_kolumna+i-1,i]);
        % wyznaczenie rozkladu LU
        a = wspolczynniki(i,i);
        for l = (i+1):rozmiar
            wspolczynniki(l,i) = wspolczynniki(l,i) / a;
            wspolczynniki(l, (i+1):end) = wspolczynniki(l, (i+1):end) -
wspolczynniki(l,i) * wspolczynniki(i, (i+1):end);
            rozw(l) = rozw(l) - wspolczynniki(l,i) * rozw(i);
        end
    end

    % U * x = rozw -> wyznaczenie x
    for i = (rozmiar:-1:1)
        % Ax=b; -> odejmowanie wartosci znanych ze strony rozwiazan
        for j = (i+1):(rozmiar)
            rozw(i) = rozw(i) - wspolczynniki((i),(j))*wspolczynniki((j),(j));
        end
        wspolczynniki(i,i) = rozw(i)/wspolczynniki(i,i);
    end

    %przepisanie rozwiazan do macierzy rozw
    for i = (1:rozmiar)
        rozw(i) = wspolczynniki(i,i);
    end
    g = [pierwotne_wspolczynniki, rozw, pierwotne_b];
end

```

norma residuum

```
function nr = norma_residuum(wspolczynniki, x, rozw)
    residuum = wspolczynniki*x - rozw;
    nr = norm(residuum);
end
```

tworzenie macierzy typu a o zadanym rozmiarze rozmiar_a

```
function mac_a = macierzA(rozmiar_a)
    macierz_a = zeros(rozmiar_a);

    for j = 1:rozmiar_a
        for i = 1:rozmiar_a
            if (i < j-1) || (i > j+1)
                macierz_a(i,j) = 0;
            elseif (i == j-1) || (i == j+1)
                macierz_a(i,j) = 2;
            else
                macierz_a(i,j) = 6;
            end
        end
    end

    wolne_a = zeros(rozmiar_a, 1);
    % deklaracja pierwszej warości, aby pozostale
    % mogly powstac przez dodawanie zamiast mnozenia
    wolne_a(1) = 9.5;

    for i = 2:rozmiar_a
        wolne_a(i) = wolne_a(i-1) + 0.5;
    end

    mac_a = [macierz_a, wolne_a];
end
```

tworzenie macierzy typu b o zadanym rozmiarze rozmiar_b

```
function mac_b = macierzB(rozmiar_b)
    macierz_b = zeros(rozmiar_b);
    % wygenerowanie pierwszej i ostatniej kolumny pozwoli w pozniejszych
    % etapach przekopiowywac wartosci zamiast obliczac je na nowo
    macierz_b(1,1) = 1/8;
    macierz_b(end,end) = 1/8;
    % pierwsza kolumna
    macierz_b(2, 1) = 6;
    for i = 3:rozmiar_b
        macierz_b(i, 1) = macierz_b(i-1, 1) + 5;
    end
    % ostatnia kolumna
    macierz_b(end - 1, end) = -4;
    for i = (rozmiar_b-2):-1:1
        macierz_b(i, end) = macierz_b(i + 1, end) - 5;
    end
end
```

```

% wykorzystanie wartosci ze skrajnych kolumn
for j = 2:(rozmiar_b-1)
    for i = 1:rozmiar_b
        if (i < j)
            macierz_b(i,j) = macierz_b(end-(j-i),end);
        else
            macierz_b(i,j) = macierz_b(i-1,j-1);
        end
    end
end

wolne_b = zeros(rozmiar_b, 1);
% deklaracja pierwszej wartosci, aby pozostale
% mogly powstac przez dodawanie zamiast mnozenia
wolne_b(1) = -2.5;

for i = 2:rozmiar_b
    wolne_b(i) = wolne_b(i-1) + 0.5;
end

mac_b = [macierz_b, wolne_b];
end

```

tworzenie macierzy typu c o zadany rozmiarze rozmiar_c

```

function mac_c = macierzC(rozmiar_c)
    macierz_c = zeros(rozmiar_c);

    % wygenerowanie pierwszej kolumny
    % kolumny od 2 do end będą przekopiowywac
    % (rozmiar_c-1) wartosci z poprzednich kolumn
    % i wyliczac jedna własna
    for i = 1:rozmiar_c
        macierz_c(i, 1) = 4/(5*i);
    end

    % wykorzystanie wartosci z pierwszej kolumny
    for j = 2:rozmiar_c
        for i = 1:(rozmiar_c-1)
            macierz_c(i,j) = macierz_c(i+1, j-1);
        end
        macierz_c(rozmiar_c, j) = 4/(5*(rozmiar_c+j-1));
    end
    wolne_c = zeros(rozmiar_c, 1);

    % deklaracja pierwszej wartosci, aby pozostale
    % mogly powstac przez dodawanie zamiast mnozenia
    wolne_c(1) = 0.5;

    for i = 3:2:rozmiar_c
        wolne_c(i) = wolne_c(i-2) + 1;
    end
    mac_c = [macierz_c, wolne_c];
end

```

Załącznik 3. Kod źródłowy zadania 3.

dane z zadania

```
A = [12 2 1 -6;  
      4 -15 2 -5;  
      2 -1 8 -2;  
      5 -2 1 -8];  
b = [6; 8; 20; 2];  
e = 0.001;
```

wywołanie funkcji

```
display(gauss_seidel(A, b, e));  
norma_residuum(A, gauss_seidel(A, b, e), b)
```

algorytm

```
function gs = gauss_seidel(wspolczynniki, rozw, err)  
    rozmiar = (size(wspolczynniki,1));  
    x = zeros(rozmiar, 1);  
    blad = zeros(rozmiar, 1);  
    % zeby warunek petli byl spelniony  
    blad(1) = 100;  
    % rozklad L + D + U  
    L = zeros(rozmiar);  
    U = zeros(rozmiar);  
    for i = 1:rozmiar  
        L(i, 1:i-1) = wspolczynniki(i, 1:i-1);  
        U(i, i+1:rozmiar) = wspolczynniki(i, i+1:rozmiar);  
    end  
    % wyznaczenie D^-1  
    D_odwr = zeros(rozmiar);  
    for i = 1:rozmiar  
        D_odwr(i,i) = 1/wspolczynniki(i,i);  
    end  
    % wyznaczenie D^-1 * b, D^-1 * L, D^-1 * U  
    rozw = D_odwr * rozw;  
    L = D_odwr * L;  
    U = D_odwr * U;  
    % polaczenie odwroconych macierzy L i U  
    % dla latwosci wykonywania petli  
    LU = (-1) * (L + U);  
  
    while(max(abs(blad)) > err)  
        for i = 1:rozmiar  
            poprzednia_wartosc = x(i);  
            x(i) = rozw(i);  
            for j = 1:(rozmiar)  
                x(i) = x(i) + LU(i,j) * x(j);  
            end  
            blad(i) = (x(i)-poprzednia_wartosc)/x(i);  
        end  
    end  
    gs = x;
```

end

Output

```
0.5609
-0.2105
2.4484
0.4592
norma residuum = 0.0211
```

(dla tych samych danych output algorytmu z zad. 2, wnioski w sekcji „Komentarz” zadania 3):

```
0.5605
2.4483
-0.2104
0.4590
```