1. **Za pomocą metody iteracyjnej Gaussa – Seidela rozwiąż następujące równania ( sprawdzać za każdym razem czy metoda jest zbieżna). Zbadać zależność szybkości zbieżności i poprawności rozwiązania od przybliżenia początkowego ( przyjąść dokładności rozwiązania e = 10-2 , 10 -4, 10-10 ). Sporządzić wykres relacji między błędem a liczbą iteracji oraz dokonać porównania skuteczności wykorzystywanych metod.**
2. -2x + y + 5z = 15

4x – 8y + z = -21

4x – y + +z = 7

1. 4x + 3y + 2z = 5

3x + 4y + 3z = 8

2 x + 3y + 4z = 10

1. **Opis metody**

Metoda Gaussa – Seidela jest metodą iteracyjną. Tego typu metody polegają na obraniu dowolnego początkowego wektora rozwiązań ( w naszym przypadku orano wektor zerowy) i następnie w kolejnych krokach wartość ta jest ulepszana i coraz bardziej zbliżana do rzeczywistej wartości rozwiązania. Aby istniało rozwiązanie algorytm ten musi być zbieżny. Dla metody Gaussa-Seidla algorytm jest zbieżny jeśli macierz A jest silnie diagonalnie dominująca lub silnie diagonalnie dominująca kolumnowo . Problem są także zera występujące na diagonali. Należy odpowiednio zamienić wiersze w celu pozbycia się ich z diagonali . Jeśli nie da się tego zrobić to algorytm na pewno nie będzie zbieżny.

Ilość iteracji niezbędną do otrzymania rządanej dokładności nie jest łatwo przewidzieć. Nie zakłada się zwykle konkretnej liczby iteracji lecz dobrze jest wbudować zabezpieczenie aby uniknąć ewentualnych zapętleń programu lub braku zbieżności. Stosuje się następujące warunki przerwania działania pętli Są to warunki konieczne i wystarczające. Jako warunki stopu tych algorytmów przyjmuje się:

tempo zbieżności:

Wielkość residuum:

W tej metodzie macierz po sprawdzeniu zbieżności i ewentualnej jej modyfikacji należy rozłożyć na macierz diagonalną, naddiagonalną i poddiagonalną. Wzór na kolejne przybliżenie w iteracji wyraża się wzorem:

xk+1 = - ( D + L)-1  Uxk + (D +L) -1b

1. **Opis programu**

W naszym programie główny skrypt nosi nazwe ,, MojGaussSeil” i przy jego pomocy uruchamia się progam wprwadzając wcześniej macierz A, wektor b oraz żądaną dokładność e.

Na początku jest wywołany skrypt ,, sprawdz\_poprawnosc”. Sprawdza on czy macierz A i wektor mają odpowiednie rozmiary a także czy macierz A jest nieosobliwa. Jeśli warunki są spełnione można przejść do dalszej części programu. Skrypt:

W macierzy A na przekątnej nie mogą wsytępować zera. Wywołany jest skrypt „ znajdz\_zera\_i\_sume\_w\_kolumnie ”. Szuka on zer na przekątnej macierzy A oraz zapisuje indeksy w których ewentualnie występuje taka sytuacja oraz liczbę zer w tej kolumnie. Trzeba tak zmieniać wiersze aby nie było zer na przekątnych zaczynając od modyfikacji tych punktów dla których ilość zer jest największa w kolumnie. Wiersze które uległy zamianie są potem zablokowane do zmiany. Wiersze zamienia się tak aby liczba o możliwie największym module znalazła się na przekątnej ( przydatne do warunków zbieżności ). Operacje te zostały dokonane za pomocą skryptu ,, zamien\_wiersze”.

Następnym krokiem jest sprawdzenie zbieżności przy użyciu skryptu „ sprawdz\_zbieznosc”. Jeśli jest stwierdzona zbieżność to można przejść dalej. W przeciwnym razie jest podejmowana próba przestawienia wierszy i kolumn tak aby został spełniony jeden z warunków zbieżności.

Gdy wiadomo , że warunek zbieżności jest spełniony to wyliczane są macierze diagonalna, naddiagonalna i poddiagonalna oraz początkowy wektor jest ustalany na wartość zero. Następują kolejne iteracje i po spełnieniu wystarczających warunków otrzymujemy końcowy wynik.

1. Rozwiązanie
2. Pierwszy przypadek był zbieżny.

Dla dokładnoście = 10-2 proces zakończył się po 4 iteracjach i wyniki wyniosły:

x = 1.9931

y = 3.9736

z = 3.0012

Dla dokładnoście = 10-4 proces zakończył się po 7 iteracjach i wyniki wyniosły:

x =2.0000

y = 3.9999

z = 3.0001

Dla dokładnoście = 10-10  zakończył się po 14 iteracjach dla wszystkich żądanych dokładności. Wyniki rozwiązania:

x = 2

y = 4

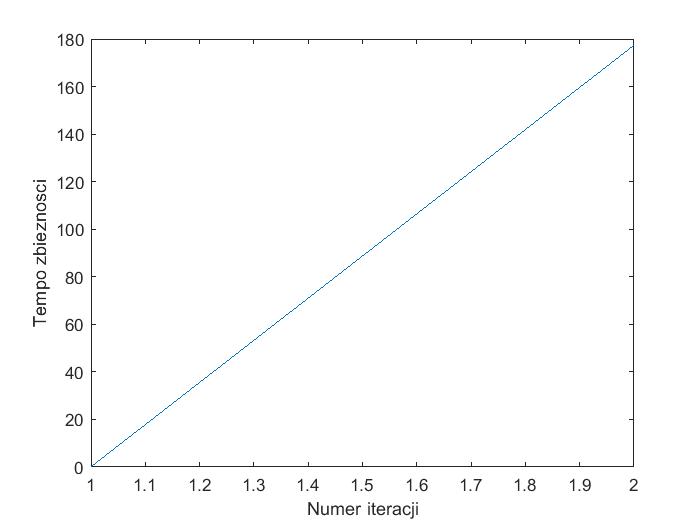
z = 3

Widać, że wyniki są bardzo dokładne już po kilku iteracjach.

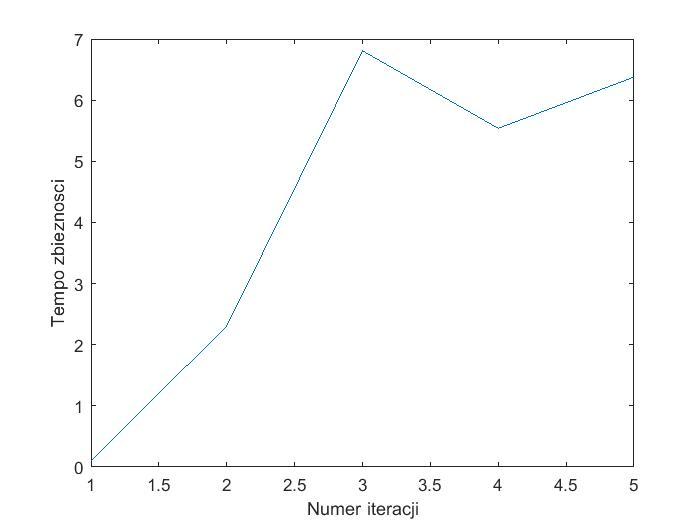
Następnie sprawodzo jak punkt początkowy wpływa na tempo zbieżności i poprawność rozwiązania.

1. Dokładność e = 10-2

Dla zerowego punktu początkowego:



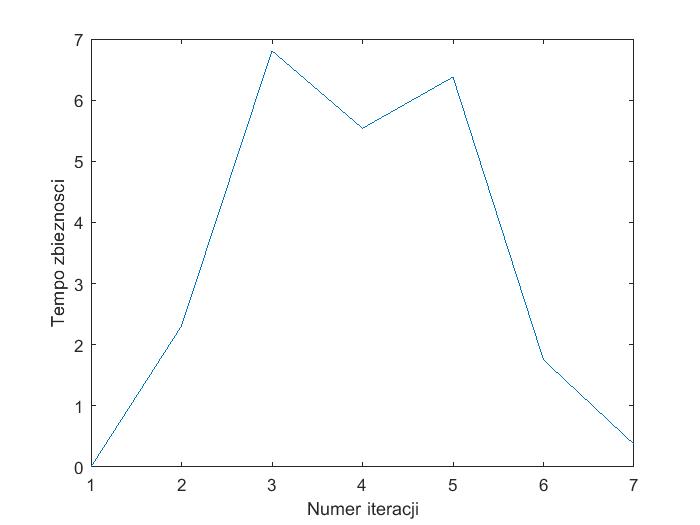
Dla punktu początkowego równego 10000:



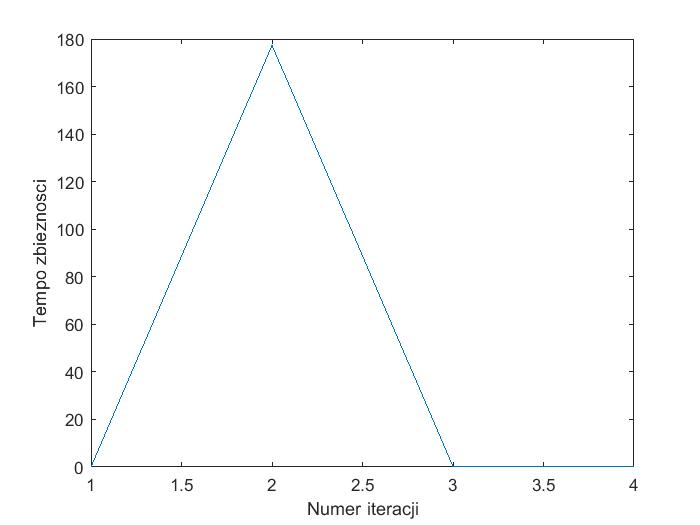
Z powodu małej dokładności dokładności otrzymano niewielką liczbę literacji. Dla zerowego punktu tempo rośnie lecz są tylo 2 iteracje więc ciężko wyciągnąć jakies, dla większych wartości punktu na początku rośnie gwałtownie a potem zaczyna spadać i znowu wzrastać.

1. Dokładność e = 10-4

Dla punktu początkowego równego 10000:

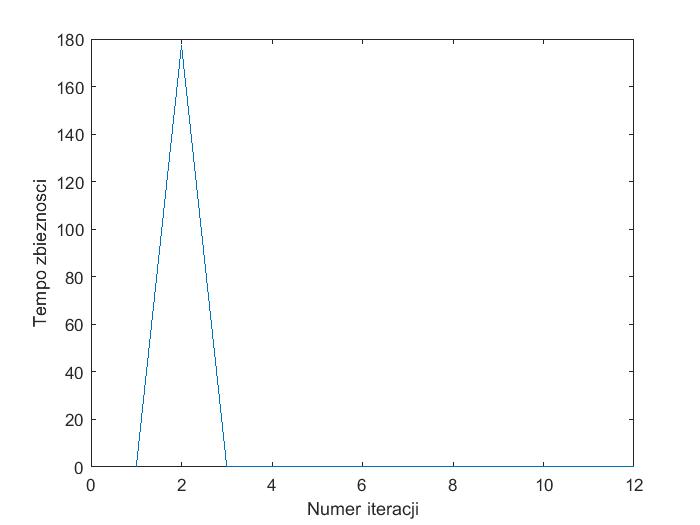


Dla zerowego punktu początkowego:

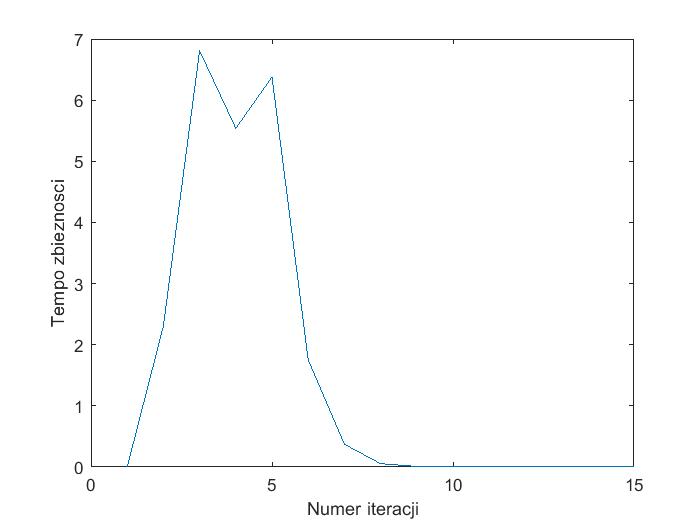


Zwiększyła się liczba iteracji dzięki czemu poznaliśmy dalsze przebiegi. Dla obu przypadków tempo gwałtownie spadło w dalszych częściach wykresów.

Dla zera:



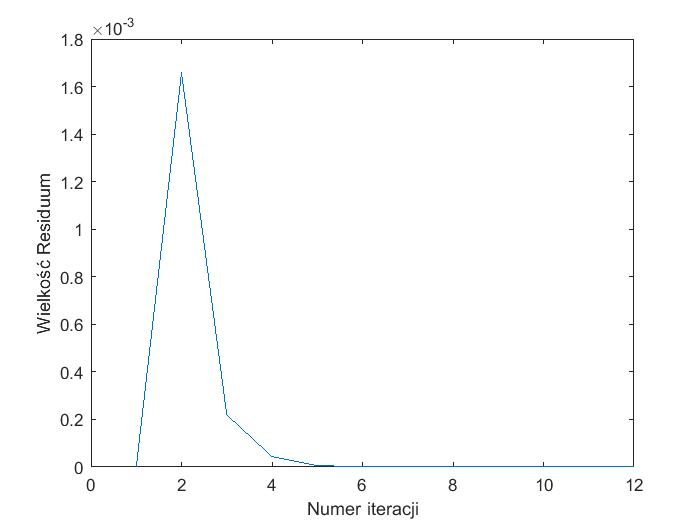
Dla 10000:



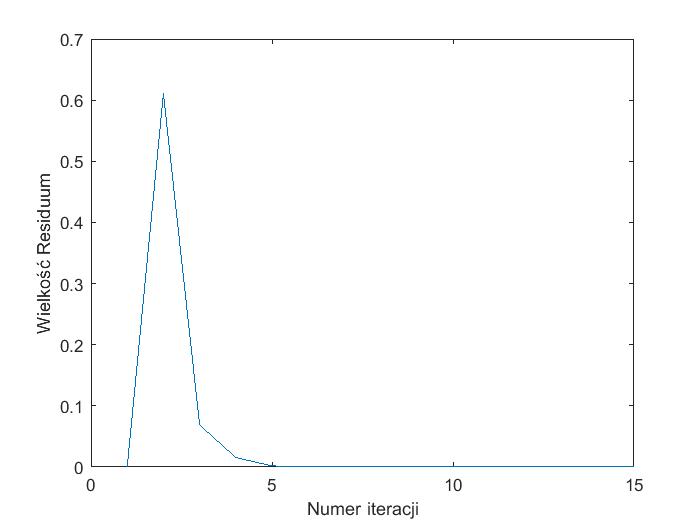
W kolejnych iteracjach tempo spada w okolice zera.

Następnie sprawdzono jak zmieniały się wartości Residuum w kolejnych iteracjach.

Dla 0:

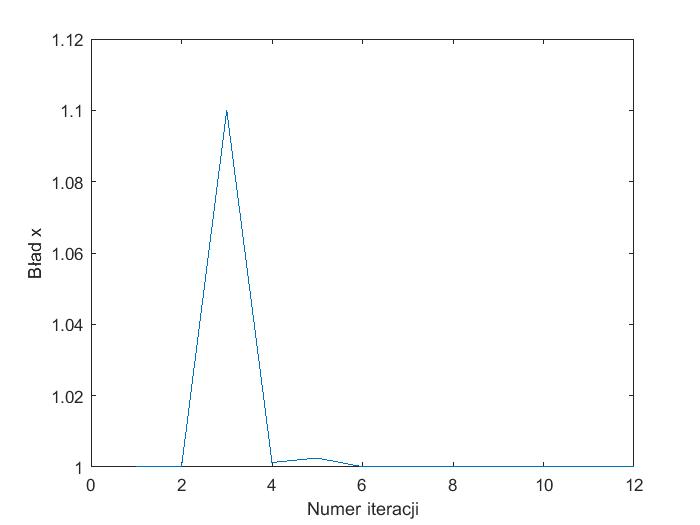
Dla 0:

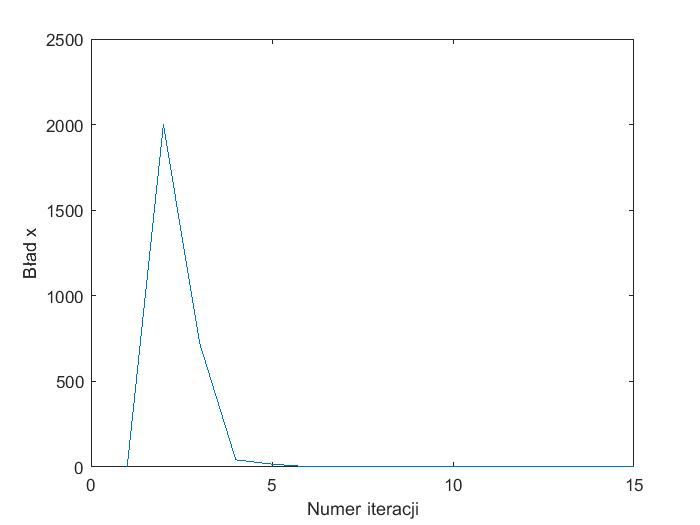
Dla 10000:



Widać, że wartość znacznie bardziej oddalona od rozwiązania równania przyjmuje znacznie większe wartości residuum chociaż kształt tych funkcji jest identyczny i są one tylko przeskalowane – mają te same zachowania lecz inne wartości.

Błedy dla x od iteracji:





Na początku błędy są bardzo duże ale są bardzo szybko redukowane i schodzą do bardzo małej wartośćci. Oczywiście dla punktu znacznie bardziej oddalonego od rozwiązanie wartości tych błędów są znacznie większe. Wykresy dla wartości y i z są podobne.

1. Dla tego przykładu metoda nie jest zbieżna. Nie można więc policzyć rozwiązań przy użyciu tej metody tego równania.

Jeżeli macierz A jest zbieżna, funkcja ta pozwala dość dokładnie wyznaczyć wartości rozwiązań. Sprawdza się dobrze przy rozwiązywaniu dużych układów równań. Jest bardziej skuteczna przy wielkich układach niż metody dokładne. Już przy niewielu iteracjach błąd jest minimalny.

IV Skrypty

MojGaussSeidla

%% Skrypt rozwiązujący układ równań metodą iteracyjną Gaussa-Seidela

%% o postaci Ax=b, należy podać macierz A , wektor b oraz żadaną dokładność e

%A = [ -2 1 5 ; 4 -8 1 ; 4 -1 1];

%b = [15; -21 ; 7];

%e = 10^(-10);

sprawdz\_poprawnosc;

if poprawnosc == 1

znajdz\_zera\_i\_sume\_w\_kolumnie;

if ile\_zer\_A > 0 % Jeśli znaleziono zera na przekątnej to trzeba zamienic te wiersze

zamien\_wiersze;

end

sprawdz\_zbieznosc;

if zbieznosc == 1

D = zeros(rozmiar\_A ,rozmiar\_A); % macierz diagonalna

for i = 1 : 1: rozmiar\_A

D( i , i) = A ( i , i);

end

L = zeros(rozmiar\_A ,rozmiar\_A); % macierz poddiagonalna

for i = 2 : 1: rozmiar\_A

for j = 1 : 1 : i - 1

L( i , j) = A ( i , j);

end

end

U = zeros(rozmiar\_A ,rozmiar\_A); % macierz naddiagonalna

for i = 1 : 1: rozmiar\_A - 1

for j = i + 1 : 1 : rozmiar\_A

U( i , j) = A ( i , j);

end

end

x\_wczesniejsze = zeros(rozmiar\_A ,1); % Przyjmujemy narazie, że rozwiązanie to wektor zerowy

x\_kolejne = zeros(rozmiar\_A ,1);

x\_poczatkowe = zeros(rozmiar\_A ,1);

numer\_iteracji = 1;

maksymalna\_ilosc\_iteracji = 100;

e1\_dop = e;

e2\_dop = e;

e1 = 10 \* e ;

e2 = 10 \* e;

while (numer\_iteracji <= maksymalna\_ilosc\_iteracji) && (e1(numer\_iteracji) > e1\_dop) && (e2(numer\_iteracji) > e2\_dop)

numer\_iteracji = numer\_iteracji + 1;

x\_wczesniejsze = x\_kolejne ;

x\_kolejne = - (D+L) ^(-1) \* U \* x\_wczesniejsze + ( D + L ) ^ (-1) \* b;

e1( numer\_iteracji) = ( norm (x\_kolejne - x\_wczesniejsze ) )/( norm (x\_kolejne) );

e2 (numer\_iteracji) = ( norm (A \* x\_kolejne - b) ) / ( norm ( A \* x\_poczatkowe - b ) );

end

disp('Rozwiązanie zakończono po');

numer\_iteracji

disp(' Wynik: ')

if zmiana\_kolejnosci\_zmiennych == 1

for i = 1 : 1 : rozmiar\_A

if i ~= kolejnosc\_zmiennych(i) % jest zamieniona kolejnosc na tej pozycji

x\_porzadek(i) = x\_kolejne( kolejnosc\_zmiennych(i) ) ;

else % na swojej pozycji

x\_porzadek(i) = x\_kolejne(i);

end

end

x\_porzadek

else % nie była zmieniana kolejność kolumn

x\_kolejne

end

disp(' e1 = ')

disp(e1 ( numer\_iteracji ))

disp(' e2 = ')

disp(e2 ( numer\_iteracji ))

else

disp('Warunki zbieżnośći nie zostały spełnione');

end

end

,,Aktualizuj tablice”

% Skrypt zmniejsza rozmiar tablicy o 1 i porządluje ja

if numer\_wiersza\_najwiekszego == ile\_zer\_A % jeśli ostani element w tablicy to zmiejszamy tylko wielko tablicy o 1

ile\_zer\_A = ile\_zer\_A - 1 ;

end

else

for k = najwiecej\_zer : 1 : ile\_zer\_A - 1

zera (1 , k ) = zera (1 , k + 1) ;

zera (2 , k ) = zera (2 , k + 1) ;

end

ile\_zer\_A = ile\_zer\_A -1 % Zmniejszamy rozmiar tablic o 1

,,gdzie\_nie\_zbiezne\_kolumny”

% Tworzy tablice z suma kolumn i tab z informacja gdzie potrzebna zamiana i

% ilosc do złych kolumn

sumy\_w\_kolumnach = zeros(1, rozmiar\_A);

ile\_zlych = 0;

for i = 1 : 1 : rozmiar\_A

for j = 1 : 1 : rozmiar\_A

sumy\_w\_kolumnach(i) = sumy\_w\_kolumnach(i) + abs(A ( j , i) );

end

if abs( A(i , i) ) <= sumy\_w\_kolumnach(i) - abs( A( i , i ) )

ile\_zlych = ile\_zlych + 1;

indeksy\_zlych( ile\_zlych ) = i ;

end

end

,,gdzie\_nie\_zbiezne\_wiersze”

% Tworzy tablice z suma kolumn i tab z informacja gdzie potrzebna zamiana i

% ilosc do złych wierszy

sumy\_w\_wierszach = zeros( rozmiar\_A, 1); % tablica z sumami w wierszach

ile\_zlych = 0; % ile znaleziono złych wierszy

for j = 1 : 1 : rozmiar\_A

for i = 1 : 1 : rozmiar\_A

sumy\_w\_wierszach(i) = sumy\_w\_wierszach(i) + abs(A ( i , j) );

end

if abs( A(j, j) ) <= sumy\_w\_wierszach(i) - abs( A( j , j ) )

ile\_zlych = ile\_zlych + 1;

indeksy\_zlych( ile\_zlych ) = j ;

end

end

,,Nowa\_suma\_kolumn”

% Liczy sumy w kolumnach dla nowego rozwiązania B

sumy\_w\_kolumnach = zeros(1, rozmiar\_A);

for i = 1 : 1 : rozmiar\_A

for j = 1 : 1 : rozmiar\_A

sumy\_w\_kolumnach(i) = sumy\_w\_kolumnach(i) + abs(B ( j , i) );

end

end

,,Nowa\_suma\_wiersza”

% Liczy sumy w wierszach dla nowego rozwiązania B

sumy\_w\_wierszach = zeros( rozmiar\_A, 1);

for i = 1 : 1 : rozmiar\_A

for j = 1 : 1 : rozmiar\_A

sumy\_w\_wierszach(i) = sumy\_w\_wierszach(i) + abs(B ( i , j) );

end

end

,,Sprawdz poprawność”

% Skrypt sprawdza poprawnosc wprowadzonej macierzy A oraz wektora b

rozmiar\_b = size(b); % rozmiar wektora b

rozmiar\_A = size(A); % rozmair macierzy A

poprawnosc = 1 ;

if rozmiar\_A(1,1) ~= rozmiar\_b(1,1) || rozmiar\_b(1,2)>1 || rozmiar\_A(1 ,1) ~= rozmiar\_A(1,2) % sprawdzenie poprawności A i b

disp('Błednie podane macierze')

poprawnosc = 0;

return

end

if det(A) == 0 % Jeśli wyznacznik A = 0 to nie można rozwiącać ukłądu równań

disp('Macierz A jest osobliwa. Nie można rozwiązać układu równań');

return;

end

rozmiar\_A(:, 2) = []; % macierz jest kwadratowa więc dla ułatwienia

,,Sprawdz\_zbieznosc”

% Srawdzenia warunku zbieżności, jeśli jest zbiezny to zbieznosc = 1

zbieznosc = 1 ;

% Sprawdzamy czy silnie diagonalnie dominujaca

for i = 1 : 1 : rozmiar\_A

suma\_wiersza = 0;

for j = 1 : 1 : rozmiar\_A

suma\_wiersza = suma\_wiersza + abs( A ( i , j) );

end

if abs( A(i,i) ) <= suma\_wiersza - abs( A(i,i) )

zbieznosc = 0;

break

end

end

% Jeśli nie to sprawdzamy czy silnie diagonalnie dominujaca kolumnowo

if zbieznosc == 0

zbieznosc = 1;

for i = 1 : 1 : rozmiar\_A

suma\_kolumny = 0;

for j = 1 : 1 : rozmiar\_A

suma\_kolumny = suma\_kolumny + abs( A ( j , i) );

end

if abs( A(i,i) ) <= suma\_kolumny - abs( A(i,i) )

zbieznosc =0;

break

end

end

end

if zbieznosc == 0 % Jeśli żaden z warunków zbieżności jest niespełniony to zamieniamy wiersze

gdzie\_nie\_zbiezne\_kolumny; % Najpierw próbujemy tak aby była silnie diagonalnie dominujaca kolumnowo

Zamien\_wiersze\_zbieznosc\_kolumny;

if zbieznosc == 0 % Jeśli przestawienie wierszy nie pomogło to próbujemy przestawić kolumny

zmiana\_kolejnosci\_zmiennych = 0; % czy kolejnosc zmiennych została zmieniona

Zamien\_kolumny\_zbieznosc\_kolumny;

end

end

if zbieznosc == 0 % Jeśli operacja ta nie pomogła to zmieniamy tak aby była silnie diagonalnie dominujaca

gdzie\_nie\_zbiezne\_wiersze;

Zamien\_wiersze\_zbieznosc\_wiersze;

if zbieznosc == 0 % Jeśli zamiana wierszy nie pomogła to zamieniamy kolumny

zmiana\_kolejnosci\_zmiennych = 0; % czy kolejnosc zmiennych została zmieniona

Zamien\_kolumny\_zbieznosc\_wiersze;

end

end

,,Zamien\_kolumny\_zbieznosc\_kolumny”

gdzie\_nie\_zbiezne\_kolumny;

kolejnosc\_zmiennych = 1: rozmiar\_A ;

for i = 1 : 1 : ile\_zlych % wszystkie złe kolumny

for j = 1 : 1 : rozmiar\_A % szukamy po wierszu elementu nadającego się

if 2 \* abs( B( indeksy\_zlych(i ) , j ) ) > sumy\_w\_kolumnach(indeksy\_zlych(i ) ) % Jeśli to rozwiązuje problem kolumny

% Macierz testowa i rozpatrujemy rozwiazanie ze zmiana

% kolumn

Testowa\_macierz = B;

Testowa\_macierz(:, [ indeksy\_zlych(i ) j] ) = Testowa\_macierz(:,[ j indeksy\_zlych(i )] ); % Zamienamy kolumne diagonalną ze znaleziona kolumna

% Sprawdzamy ile w tym rozwiazaniu mamy zlych kolumn

nowa\_ilosc\_zlych = 0;

for k = 1 : 1 : rozmiar\_A

if abs( Testowa\_macierz(k,k) ) <= sumy\_w\_kolumnach(k) - abs( Testowa\_macierz(k,k) )

nowa\_ilosc\_zlych = nowa\_ilosc\_zlych + 1;

end

end

if nowa\_ilosc\_zlych < ilosc\_zlych\_najlepszego % Jeśli to rozwiazanie jest lepsze

najlepsze\_rozwiazanie = Testowa\_macierz; % To zapisujemy to rozwiazanie

ilosc\_zlych\_najlepszego = nowa\_ilosc\_zlych;

zmiana\_kolejnosci\_zmiennych = 1; % zmieniono kolejność zmiennych

kolejnosc\_zmiennych(: ,[ indeksy\_zlych(i ) j ] ) = kolejnosc\_zmiennych(: , [ j indeksy\_zlych(i )] ); % zmiana kolejnosc zmiennych

if ilosc\_zlych\_najlepszego == 0 % Jeśli mamy kompletne rozwiązanie to wychodzimy

break;

end

end

end

end

B = najlepsze\_rozwiazanie; % przeszliśmy cała kolumne i zapisujemy najlepsze rozwiazanie w niej

Nowa\_suma\_kolumn;

if ilosc\_zlych\_najlepszego == 0 % Jeśli już jest ok to wychodzimy

A = B ; % Mamy dobre rozwiązanie więc aktualizujemy macierz A do tej postaci

zbieznosc = 1;

break;

end

zbieznosc = 0;

end

,,Zamien\_kolumny\_zbieznosc\_kolumny”

gdzie\_nie\_zbiezne\_wiersze;

kolejnosc\_zmiennych = 1: rozmiar\_A ;

for j = 1 : 1 : ile\_zlych % wszystkie złe wiersze

for i = 1 : 1 : rozmiar\_A % po wierszu szukamy elementu nadającego sie

if (2 \*abs( B( indeksy\_zlych(j) , i ) )) > sumy\_w\_wierszach(indeksy\_zlych(j)) % zamiana tego elementu rozwiązuje problem w tym wierszu

% Tworzymy testowa macierz i rozpatrujemy rozwiazanie ze zmiana

% kolumn

Testowa\_macierz = B;

Testowa\_macierz(:, [indeksy\_zlych(j ) i ] ) = Testowa\_macierz(:,[ i indeksy\_zlych(j ) ]); % Zamienamy kolumne diagonalną ze znaleziona kolumna

% Sprawdzamy ile w tym rozwiazaniu mamy zlych wierszy

nowa\_ilosc\_zlych = 0;

for k = 1 : 1 : rozmiar\_A

if abs( Testowa\_macierz(k,k) ) <= sumy\_w\_wierszach(k) - abs( Testowa\_macierz(k,k) )

nowa\_ilosc\_zlych = nowa\_ilosc\_zlych + 1;

end

end

if nowa\_ilosc\_zlych < ilosc\_zlych\_najlepszego % Jeśli to rozwiazanie jest lepsze

najlepsze\_rozwiazanie = Testowa\_macierz; % To zapisujemy to rozwiazanie

ilosc\_zlych\_najlepszego = nowa\_ilosc\_zlych;

zmiana\_kolejnosci\_zmiennych = 1; % zmieniono kolejność zmiennych

kolejnosc\_zmiennych(: ,[ indeksy\_zlych(j ) i ] ) = kolejnosc\_zmiennych(: , [ i indeksy\_zlych(j )] ); % zmiana kolejnosc zmiennych

if ilosc\_zlych\_najlepszego == 0 % Jeśli mamy kompletne rozwiązanie to wychodzimy

break;

end

end

end

end

B = najlepsze\_rozwiazanie; % przeszliśmy cała kolumne i zapisujemy najlepsze rozwiazanie w niej

Nowa\_suma\_wiersza;

if ilosc\_zlych\_najlepszego == 0 % Jeśli już jest ok to wychodzimy

A = B ; % MAmy dobre rozwiązanie więc aktualizujemy macierz A do tej postaci

zbieznosc = 1;

break;

end

zbieznosc = 0;

end

,,Zamien\_wiersze”

% Skrypt na podstawie tablicy z zapisanymi indeksami położenia zer oraz ich

% ilości w danych kolumnach znajduje kolumne z największą ilością zer i

% zamienia wiersze tak aby komórka z największym modułem znalazła się na

% przekątnej. Powtarza takie działanie dopókią są jeszcze zera na

% przekątnej

ile\_powtorzen = ile\_zer\_A ;

for int i = 1 : 1 : ile\_powtorzen % trzeba zaminiec tyle wierszy ile znaleziono zer

% zaczynamy od znlezenia z wybranych kolumn tej gdzie jest najwiecej zer

najwiecej\_zer = 1; % Narazie przyjmujemy, że w ideksie zapisanym w 1 komórce jest najwiecej 0

zera\_kolumna = zera(2, 1); % ile jest zer w kolumnie w której jest jej nawięcej

if ile\_zer\_A > 1 % Jeśli mamy jedno zero to jest ono pod 1 indeksem, jeśli nie to szukamy

znajdz\_kolumne;

end

numer\_kolumny = zera( 1, najwiecej\_zer) ; % nr rozpatrywanej kolumny

wartosc\_najwiekszego = 0;

numer\_wiersza\_najwiekszego = 0;

znajdz\_najwiekszy\_elemnt;

zabronione ( numer\_wiersza\_najwiekszego ) = 1; % dodajemy ten wiersz do zabronionych

A( [ numer\_wiersza\_najwiekszego numer\_kolumny ], : ) = A( [ numer\_kolumny numer\_wiersza\_najwiekszego] , : ); % zamiana wierszy

aktualizuj\_tablice;

b([ numer\_wiersza\_najwiekszego numer\_kolumny ], : ) = b ( [ numer\_kolumny numer\_wiersza\_najwiekszego] , : ); % zamiana wektora b

end

,,Zamien\_wiersze\_zbieznosc\_kolumny”

% Próbuje zamienic wiersze tak by A była silnie diagonalna dominująca

najlepsza\_ilosc\_zlych\_kolumna = ile\_zlych; % Będzie przechowywać ilość złych kolumn macierzy testowej

najlepsze\_rozwiazanie = A; % Przechowuje najlepsze znalezione rozwiązanie

ilosc\_zlych\_najlepszego = ile\_zlych; % Będzie przechowywać ilość złych kolumn najlepszej znalezionej macierzy

B = A; % kopia do operacji na niej

b\_najlepsze = b; % zamiana wierszy zmieni nam też wektor b

for i = 1 : 1 : ile\_zlych % wszystkie złe kolumny

for j = 1 : 1 : rozmiar\_A % po kolumnie szukamy elementu nadającego sie

if abs( B( j , indeksy\_zlych(i ) ) ) > sumy\_w\_kolumnach(indeksy\_zlych(i ) ) % zamiana tego elementu rozwiązuje problem w tej kolumnie

% Tworzymy testowa macierz i rozpatrujemy rozwiazanie ze zmiana

% wierszy

Testowa\_macierz = B;

Testowa\_macierz( [ indeksy\_zlych(i ) j ] , :) = Testowa\_macierz([ j indeksy\_zlych(i )], :); % Zamienamy wiersz diagonalny ze znalezionym

% Sprawdzamy ile w tym rozwiazaniu mamy zlych kolumn

nowa\_ilosc\_zlych = 0;

for k = 1 : 1 : rozmiar\_A

if abs( Testowa\_macierz(k,k) ) <= sumy\_w\_kolumnach(k) - abs( Testowa\_macierz(k,k) )

nowa\_ilosc\_zlych = nowa\_ilosc\_zlych + 1;

end

end

if nowa\_ilosc\_zlych < ilosc\_zlych\_najlepszego % Jeśli to rozwiazanie jest lepsze

najlepsze\_rozwiazanie = Testowa\_macierz; % To zapisujemy to rozwiazanie

b\_schowek = b\_najlepsze( indeksy\_zlych(i ) ) ; % Trzeba zaktualizować wektor b

b\_najlepsze( indeksy\_zlych(i ) ) = b\_najlepsze(j);

b\_najlepsze(j) = b\_schowek;

ilosc\_zlych\_najlepszego = nowa\_ilosc\_zlych;

if ilosc\_zlych\_najlepszego == 0 % Jeśli mamy kompletne rozwiązanie to wychodzimy

break;

end

end

end

end

B = najlepsze\_rozwiazanie; % przeszliśmy cała kolumne i zapisujemy najlepsze rozwiazanie w niej

Nowa\_suma\_kolumn;

if ilosc\_zlych\_najlepszego == 0 % Jeśli już jest ok to wychodzimy

A = B ; % MAmy dobre rozwiązanie więc aktualizujemy macierz A do tej postaci

b = b\_najlepsze;

zbieznosc = 1;

break;

end

zbieznosc = 0;

end

,,Zamien\_wiersze\_zbieznosc\_wiersza”

% Próbuje zamienic wiersze tak by A była silnie diagonalna ( suma wierszy)

najlepsza\_ilosc\_zlych\_wiersz = ile\_zlych; % Będzie przechowywać ilość złych wierszy macierzy testowej

najlepsze\_rozwiazanie = A; % Przechowuje najlepsze znalezione rozwiązanie

ilosc\_zlych\_najlepszego = ile\_zlych; % Będzie przechowywać ilość złych wierszy najlepszej znalezionej macierzy

B = A; % kopia do operacji na niej

for j = 1 : 1 : ile\_zlych % wszystkie złe wiersze

for i = 1 : 1 : rozmiar\_A % po wierszu szukamy elementu nadającego sie

if abs( B( indeksy\_zlych(j) , i ) ) > sumy\_w\_wierszach(indeksy\_zlych(j)) % zamiana tego elementu rozwiązuje problem w tym wierszu

% Tworzymy testowa macierz i rozpatrujemy rozwiazanie ze zmiana

% wierszy

Testowa\_macierz = B;

Testowa\_macierz([ i indeksy\_zlych(j)], :) = Testowa\_macierz([ indeksy\_zlych(j) i ], :)

% Sprawdzamy ile w tym rozwiazaniu mamy zlych wierszy

nowa\_ilosc\_zlych = 0;

for k = 1 : 1 : rozmiar\_A

if abs( Testowa\_macierz(k,k) ) <= sumy\_w\_wierszach(k) - abs( Testowa\_macierz(k,k) )

nowa\_ilosc\_zlych = nowa\_ilosc\_zlych + 1;

end

end

if nowa\_ilosc\_zlych < ilosc\_zlych\_najlepszego % Jeśli to rozwiazanie jest lepsze

najlepsze\_rozwiazanie = Testowa\_macierz; % To zapisujemy to rozwiazanie

b\_schowek = b\_najlepsze( indeksy\_zlych(i ) ) ; % Trzeba zaktualizować wektor b

b\_najlepsze( indeksy\_zlych(i ) ) = b\_najlepsze(j);

b\_najlepsze(j) = b\_schowek;

ilosc\_zlych\_najlepszego = nowa\_ilosc\_zlych;

if ilosc\_zlych\_najlepszego == 0 % Jeśli mamy kompletne rozwiązanie to wychodzimy

break;

end

end

end

end

B = najlepsze\_rozwiazanie; % przeszliśmy cała kolumne i zapisujemy najlepsze rozwiazanie w niej

Nowa\_suma\_wiersza;

if ilosc\_zlych\_najlepszego == 0 % Jeśli już jest ok to wychodzimy

A = B ; % MAmy dobre rozwiązanie więc aktualizujemy macierz A do tej postaci

b = b\_najlepsze;

zbieznosc = 1;

break;

end

zbieznosc = 0;

end

,,Znajdz\_kolumne”

% Skrypt szuka w której kolumnie z wybranych jest najwięcej zer

for i = 2 : 1 : ile\_zer\_A +1

if zera (2 , i ) > zera\_kolumna

najwiecej\_zer = i ; % nowy indeks gdzie jest najwiecej zer

zera\_kolumna = zera(2, i); % nowa ilosc największej ilości zer

end

end

,,znajdz\_najwiekszy\_elemnt”

% Skrypt szuka najwiekszego elemntu wybranej kolumny który można zamienić

for i = 1 : 1 : rozmiar\_A

if abs( A( i , numer\_kolumny) ) > wartosc\_najwiekszego && zabronione( i ) = 0 % Jak znajdziemy wiekszy w kolumnie i dozwolony

numer\_wiersza\_najwiekszego = i;

wartosc\_najwiekszego = abs( A( i , numer\_kolumny) );

end

end

,, znajdz\_zera\_i\_sume\_w\_kolumnie”

% Skrypt szuka zer na przekatnej, zapisuje ich indeksy oraz liczbe zer w

% tych kolumnach

% zera = zeros( 2, rozmiar\_A(1,1) ); % Przechowuje indeksy w 1 wierszu tego gdzie sa zera a 2 wierszu ile ich jest

ile\_zer\_A = 0;

for i =1 : 1 : rozmiar\_A % wektor do pamiętania którego wiersza nie można ruszyć, 0 - niezabronione, 1 - zabronione

zabronione(i) = 0;

end

for i = 1 : 1 : rozmiar\_A % Sprawdzamy czy na przekątnej macierzy A występuje wartość 0

if A(i , i) == 0 % Jeśli znaleziono zero to zapisujemy gdzie jest i ile jest w ich w kolumnie

zera(1 , ile\_zer + 1 ) = i;

suma = 0;

for j = 1 : 1 : rozmiar\_A

if A( j , i) == 0

suma = suma + 1;

end

end

zera(2 , ile\_zer + 1 ) = suma;

ile\_zer\_A = ile\_zer\_A + 1; % zwiekszamy ilość przechowywanych danych o 1

end

end