Tomasz Bocheński

Metody numeryczne projekt 1 – dokumentacja

Zadanie 1

Treść:

Proszę napisać program wyznaczający dokładność maszynową komputera i wyznaczyć ją na swoim komputerze.

Krótki opis zastosowanych algorytmów:

Opis algorytmu zastosowanego w tym zadaniu wynika bezpośrednio z definicji dokładności maszynowej. Dokładność maszynowa jest to najmniejsza liczba większa od zera, która dodana do jedności daje wynik większy od jedności. Zatem algorytm zaimplementowany w tym zadaniu musi działać w pętli. Niech ‘x’ początkowo wynosi 1. Najpierw należy sprawdzić, czy suma jedności i liczby ‘x’ jest większa od jedności. Jeśli tak jest, to należy zapamiętać aktualne ‘x’ jako dokładność maszynową (‘myEps’) i zmniejszyć ‘x’ dwukrotnie. W tym momencie należy wrócić do początku, czyli znowu sprawdzić, czy suma ‘x’ i jedności jest większa od jedności. W przypadku gdy tak jest, należy wykonać analogiczne operacje co wcześniej. Wyjście z pętli następuje, gdy suma ‘x’ i jedności nie jest większa od jedności. Oznacza to, że otrzymaliśmy liczbę, która dodana do jedności nie zmienia wyniku, zatem jest mniejsza od dokładności maszynowej komputera. Wynika z tego, ze wartość dokładności maszynowej przechowywana jest w zmiennej ‘myEps’.

Wydruk dobrze skomentowanych programów z implementacją użytych algorytmów:

getTask1Solution – służy do obliczania dokładności maszynowej, funkcja główna zadania 1:

```
function [ ] = getTask1Solution( )
    % ZADANIE 1
    % algorytm oparty jest bezposrednio na definicji dokladnosci maszynowej:
    % dokladnosc maszynowa jest najmniejsza liczba wieksza od 0 ktora
    % dodana do jednosci daje wynik wiekszy od jednosci
    % otwieram plik do ktorego zapisze wynik
    [id, kom] = fopen('wynikZad1.txt', 'wt');
    if id < 0
        disp(kom);
    end
    x = 1.0;
    % petla dzialajaca dopoki wynik jest wiekszy od jednosci
    while 1.0 + x > 1.0
        myEps = x;
        x = x/2;
    end
    % w tym momencie myEps przechowuje najmniejsza liczbe ktora dodana do
    % jednosci daje wynik wiekszy od jednosci
    fprintf('Dokladnosc maszynowa komputera wynosi: %g\n', myEps);
    if id > 0
        fprintf(id, 'Dokladnosc maszynowa komputera wynosi: %g\n', myEps);
    end
    fclose(id);
    % wynik myEps obliczony w tym programi zgadza sie z wartoscia eps ktora
    % otrzymamy po wpisaniu do matlaba 'eps' lub 'eps(1)'
end
```

Prezentacja otrzymanych wyników:

Dokladnosc maszynowa komputera wynosi: 2.22045e-16

Komentarz do otrzymanych wyników oraz wnioski z eksperymentów:

Otrzymany wynik (2.22045e-16) jest prawidłowy. Wynika to z faktu, że po wpisaniu do Matlab'a komendy eps lub eps(1) otrzymujemy dokładnie taki sam wynik: 2.2204e-16. Zmieniając format wyświetlanych liczb na 'long' można zauważyć zgodność wszystkich kolejnych cyfr. Algorytm zastosowany w tym zadaniu jest prosty i krótki, zatem nie istnieje wiele miejsc w których można by próbować go optymalizować. Chociaż w językach programowania takich jak C operacje przesunięcia bitowego zajmują znacznie mniej czasu (takt procesora), to w Matlab'ie trwają one dłużej i efektywniejsze jest dzielenie wartości 'x' przez dwa, niż wykonywanie operacji przesunięcia bitowego (wynika to z moich obserwacji). Dlatego uważam, że algorytm który napisałem jest efektywny i przemyślany. Pisząc ten program zauważyłem ponadto, że mnożąc eps przez liczbę większą od 0.5 i mniejszą od 1 (czyli w efekcie zmniejszając eps), a następnie dodając wynik do jedności, otrzymamy liczbę większą od jedności. Nie znaczy to jednak, że znaleźliśmy w ten sposób nowe eps. Wynik sumowania pomnożonego eps i jedności jest przybliżany do wyniku sumy eps i jedności. Można się o tym przekonać przyrównując do siebie oba te wyniki w Matlab'ie.

Zadanie 2

Treść:

Proszę napisać program rozwiązujący układ n równań liniowych **Ax=b** wykorzystując podaną metodę. Proszę zastosować program do rozwiązania podanych niżej układów równań dla rosnącej liczby równań n = 10,20,40,80,160,…. Liczbę tych równań proszę zwiększać aż do momentu, gdy czas potrzebny na rozwiązanie układu staje się zbyt duży (lub metoda zawodzi).

Metoda: eliminacji Gaussa z częściowym wyborem elementu podstawowego

Dane:

1) a_{ij} = 7 dla i = j; 1 dla i=j-1 lub i = j+1; 0 dla pozostałych; $b_i = 1.4 + 0.6*i$;

2) $a_{ij} = 2*(i-j)+1$; $a_{ii} = 0.2$; $b_i = 1 + 0.4 * i$;

3) $a_{ij} = 7/[9*(i+j+1)]$; $b_i = 7/(5*i)$ gdy i parzyste; $b_i = 0$ gdy i nieparzyste;

Krótki opis zastosowanych algorytmów:

W zadaniu tym używany jest algorytm implementujący metodę eliminacji Gaussa z częściowym wyborem elementu podstawowego. Jest to ulepszenie algorytmu eliminacji Gaussa. Podczas wykonywania kolejnych kroków w metodzie eliminacji Gaussa możemy trafić na blokującą obliczenia sytuacje, gdy jeden z elementów na diagonalnej wynosi 0. Metoda eliminacji Gaussa z częściowym wyborem elementu podstawowego pozwala uniknąć takich sytuacji. W każdym k-tym kroku wybierany jest wiersz z elementem o największej co do modułu wartości spośród elementów w k-tej kolumnie, poczynając od elementu w wierszu k. Następnie wiersz ten zamieniany jest z k-tym wierszem kolejnością. W ten sposób na diagonalnej w kolumnie k znajduje się element, którego moduł jest większy od modułu każdego elementu znajdującego się pod nim. Jeśli wynosi on 0 oznacza to, że nie da się jednoznacznie określić wszystkich rozwiązań równania. Jednak w układach równań w których istnieje jedno rozwiązanie (jeden wektor ‘x’), sytuacja taka nie wystąpi (a właśnie takie układy równań rozwiązuje się tym algorytmem). Zatem nigdy element na diagonalnej nie będzie wynosił 0 i będzie można bezpiecznie przez niego dzielić. Ponadto metoda ta jest lepsza od zwykłego algorytmu eliminacji Gaussa, ponieważ prowadzi do mniejszych błędów numerycznych.

Napisana przeze mnie do tego celu funkcja pobiera dwa argumenty. Pierwszym jest macierz ‘A’ reprezentującą współczynniki przy odpowiednich wartościach ‘x’, a drugim jest macierz wyników ‘b’. Na samym początku odczytywany jest jeden z wymiarów macierzy kwadratowej ‘A’, a wynik umieszczany w zmiennej ‘n’. Tworzona jest macierz ‘C’ poprzez dodanie na końcu macierzy ‘A’ dodatkowej kolumny z wartościami macierzy ‘b’. Następnie następuje przejście do głównej pętli funkcji, która odpowiedzialna jest za przekształcenie macierzy ‘C’. W każdym kolejnym k-tym kroku znajdujemy numer wiersza z największą co do modułu wartością wśród wartości leżących w k-tej kolumnie poczynając od wiersza o numerze k. Numer ten zapamiętujemy w zmiennej ‘maxLine’. Następnie zamieniane są miejscami całe wiersze o numerach ‘k’ oraz ‘maxLine’. Dalsze obliczenia przebiegają zgodnie z metodą eliminacji Gaussa. Obliczane są kolejne współczynniki $l_{ik} = a_{ik}/a_{kk}$, oraz macierz ‘C’ jest modyfikowana. Dla każdego wiersza o numerze i, gdzie i należy od k+1 do n, wykonywane są działania $w_i = w_i - l_{ik}*w_k$. Po wyjściu z pętli otrzymujemy macierz ‘C’ przekształconą do postaci macierzy trójkątnej górnej. Alokowana jest pamięć na wektor wyników ‘x’. Kolejne

wartości ‘x’ obliczane są bezpośrednio ze wzorów $x_n = b_n/a_{nn}$, $x_k = (b_k - \sum_{j=k+1}^{n} a_{kj} * x_j)/a_{kk}$, dla k = n-1, n-2, n-3, …, 1. Otrzymany w ten sposób wektor rozwiązań ‘x’ zwracany jest przez funkcje wraz z czasem wykonania.

Wydruk dobrze skomentowanych programów z implementacją użytych algorytmów:

CEG – służy do rozwiązywania układów równań metodą eliminacji Gaussa z częściowym wyborem elementu podstawowego:

```
% funkcja do rozwiazywania ukladu n rownan liniowych metoda:
% eliminacja Gaussa z czesciowym wyborem elementu podstawowego

% zalozenie: przekazywane macierze A i b maja odpowiednie wymiary
% pozwalajace wyznaczyc rozwiazania ukladu rownan, przy czym:
% A - macierz nieosobliwa n x n wspolczynnikow
% b - macierz n x 1 wynikow
function [ x, time ] = CEG(A, b)
    % ustawiam pomiar czasu wykonywania funkcji
    tic;
    % wczytuje rozmiar macierzy kwadratowej A
    [n,~] = size(A);
    % scalam macierze wspolczynnikow A i wynikowa B
    C = [A,b];
    % inicjuje glowna petle funkcji
    for k = 1: n
        % wybieram element o najwiekszym module w okreslonej kolumnie
        % zapisuje numer wiersza w ktorym sie znajduje w maxLine
        [~,maxLine] = max(abs(C(k:n,k)));
        % korygujw wartosc maxLine aby odnosila sie ona do macierzy C
        maxLine = maxLine + k -1;
        % podmieniam wiersze k-ty z numerem wiersza zapisanym w maxLine
        C([k,maxLine],:) = C([maxLine,k],:);
        % inicjuje petle ktora utworzy macierz trojkatna
        for t = k + 1: n
            C(t,k:n+1) = C(t,k:n+1) - C(t,k)/C(k,k) * C(k,k:n+1);
        end
    end
    % tworze pusta kolumne wynikowa x
    x = zeros(n, 1);
    % implementuje wzory:
    % xn = bn/ann
    % xk = (bk-suma:od j=k+1 do n(akj*xj))/akk, dla k = n-1, n-2,...,1
    x(n) = C(n,n+1)/C(n,n);
    for i = n-1:-1:1
        tmpSum = 0;
        for j= i+1:n
            tmpSum = tmpSum + C(i,j) * x(j);
        end
        x(i) = (C(i,n+1) - tmpSum)/C(i,i);
    end
    % koncze pomiar czasu
    time = toc;
% koncze wywolanie funkcji
end
```

getMatrixes – służy do generowania wybranego typu macierzy:

```
% funkcja zwracajaca macierze wspolczynnikow (A) i macierz wynikowa (b)
% utworzone wedlug jednego z 3 algorytmow podanych w tresci zadania
% laboratoryjnego
function [ A, b ] = getMatrixes( n, dataNumber )
    % uzupelniam obie macierze samymi zerami
    A = zeros(n, n);
    b = zeros(n, 1);
    switch dataNumber
        case 1
            % uzupelniam wektor A
            for i = 1: n
                A(i,i) = 7;
                if i < n
                    A(i,i+1) = 1;
                end
                if i > 1
                    A(i,i-1) = 1;
                end
            end
            % uzupelniam wektor b
            for i = 1: n
                b(i) = 1.4 + 0.6*i;
            end
        case 2
            % uzupelniam macierz A
            % najpierw uzupelniam wszystkie elementy macierzy w wierszu
            % wedlug wzoru aij = 2*(i-j)+1, nastpenie element aii
            % zastepuje 0.2 ilosc operacji wykonywanych w takim algorytmie
            % jest mniejsza niz gdyby za kazdym razem porownywac
            % indeksy i oraz j
            for i = 1: n
                for j = 1: n
                    A(i,j) = 2*(i-j)+1;
                end
                A(i,i) = 0.2;
            end
            % uzupelniam macierz b
            for i = 1: n
                b(i) = 1 + 0.4*i;
            end
        case 3
            % uzupelniam wektor A
            for i = 1: n
                for j = 1: n
                    A(i,j) = 7/(9*(i+j+1));
                end
            end
            % uzupelniam wektor b
            for i = 2: 2: n
                b(i) = 7/(5*i);
            end
    end
end
```

getTestResultsCEG – pomocnicza funkcja testująca:

```
% funkcja wywolujaca program dla okreslonego w dataNumber zestawu danych
% dla macierzy o wymiarach n1=10, n2=20 ... nx = 10* 2^(x-1), gdzie x
% okreslone jest przez parametr numberOfAttempts
function [ N, Err,TimeM ] = getTestResultsCEG(numberOfAttempts,dataNumber)
    % tworze trzy puste macierze zawierajace odpowiednio wymiary kolejnych
    % macierzy, bledy rozwiazania oraz czasy rozwiazania
    N = zeros( numberOfAttempts, 1);
    Err = zeros( numberOfAttempts, 1);
    TimeM = zeros( numberOfAttempts, 1);
    % glowna petla programu
    for i = 1: numberOfAttempts
        % obliczam wymiar aktualnej macierzy
        tempN = 10*2^(i-1);
        % uzupelniam macierz wymiarow macierzy
        N(i) = tempN;
        % generuje odpowiednia macierz
        [A, b] = getMatrixes( tempN, dataNumber);
        % wywoluje funkcje wyznaczajaca rozwiazanie ukladu rownan
        [x, time ] = CEG(A,b);
        % uzupelniam macierz czasow wykonania
        TimeM(i) = time;
        % wyznaczam residuum
        residuum = A*x - b;
        % wyznaczam norme residuum
        residuumNorm = getNorm2(residuum);
        % uzupelniam macierz bledow rozwiazania
        Err(i) = residuumNorm;
    end
end
```

getNorm2 – pomocnicza funkcja do wyznaczania drugiej normy macierzy:

```
% norma druga macierzy A jest to pierwiastek z najwiekszego modulu wartosci
% wlasnej macierzy powstalej z pomnozenia transponowanej macierzy
% A przez macierz A
function [ norm2 ] = getNorm2( A )
    % pomocniczo wyznaczam macierz T dla ktorej bede liczyl wartosci
    % wlasne
    T = A' * A;
    % w zmiennej maxEig przechowuje najwieksza wartosc wlasna macierzy T
    maxEig = max(abs(eig(T)));
    % obliczam norme druga jako pierwiastek z najwiekszej wartosci wlasnej
    norm2 = sqrt(maxEig);
end
```

getTask2Solution – funkcja główna programu:

```
function [ ] = getTask2Solution( )
    % ZADANIE 2
    % otwieram plik do ktorego zapisze wyniki
    [id, kom] = fopen('wynikiTestowZad2.txt', 'wt');
    if id < 0
        disp(kom);
    end
    % inicjuje zmienna okreslajaca ilosc prob
    numberOfAttempts = 7;
    % wywoluje testy dla wszystkich danych
    [n1, e1, t1] = testZad2(numberOfAttempts, 1);
    [n2, e2, t2] = testZad2(numberOfAttempts, 2);
    [n3, e3, t3] = testZad2(numberOfAttempts, 3);
    % wyniki testow dla danych 1
    fprintf('\tDANE 1:\n');
    if id > 0
        fprintf(id, '\tDANE 1:\n');
    end
    for i = 1: numberOfAttempts
        fprintf('-------------------------\n');
        fprintf('Wymiar macierzy: %d\n', n1(i));
        fprintf('Blad rozwiazania: %e\n', e1(i));
        fprintf('Czas rozwiazania: %f\n', t1(i));
        if id > 0
            fprintf(id, '-------------------------\n');
            fprintf(id, 'Wymiar macierzy: %d\n', n1(i));
            fprintf(id, 'Blad rozwiazania: %e\n', e1(i));
            fprintf(id, 'Czas rozwiazania: %f\n', t1(i));
        end
    end
    % wykres bledu od n dla danych 1
    subplot(3,1,1);
    plot(n1,e1,'r.');
    title('Dane 1');
    xlabel('Wymiar macierzy');
    ylabel('Blad');
    % wyniki testow dla danych 2
    fprintf('\tDANE 2:\n');
    if id > 0
        fprintf(id, '\tDANE 2:\n');
    end
    for i = 1: numberOfAttempts
        fprintf('-------------------------\n');
        fprintf('Wymiar macierzy: %d\n', n2(i));
        fprintf('Blad rozwiazania: %e\n', e2(i));
        fprintf('Czas rozwiazania: %f\n', t2(i));
        if id > 0
            fprintf(id, '-------------------------\n');
            fprintf(id, 'Wymiar macierzy: %d\n', n2(i));
            fprintf(id, 'Blad rozwiazania: %e\n', e2(i));
            fprintf(id, 'Czas rozwiazania: %f\n', t2(i));
        end
    end
    % wykres bledu od n dla danych 2
    subplot(3,1,2);
    plot(n2,e2,'m.');
    title('Dane 2');
    xlabel('Wymiar macierzy');
    ylabel('Blad');
```

```
    % wyniki testow dla danych 3
    fprintf('\tDANE 3:\n');
    if id > 0
        fprintf(id, '\tDANE 3:\n');
    end
    for i = 1: numberOfAttempts
        fprintf('--------------------------\n');
        fprintf('Wymiar macierzy: %d\n', n3(i));
        fprintf('Blad rozwiazania: %e\n', e3(i));
        fprintf('Czas rozwiazania: %f\n', t3(i));
        if id > 0
            fprintf(id, '--------------------------\n');
            fprintf(id, 'Wymiar macierzy: %d\n', n3(i));
            fprintf(id, 'Blad rozwiazania: %e\n', e3(i));
            fprintf(id, 'Czas rozwiazania: %f\n', t3(i));
        end
    end
    % wykres bledu od n dla danych 3
    subplot(3,1,3);
    plot(n3,e3,'b.');
    title('Dane 3');
    xlabel('Wymiar macierzy');
    ylabel('Blad');
    fclose(id);
end
```

Prezentacja otrzymanych wyników:

DANE 1:

Wymiar macierzy: 10
Blad rozwiazania: 4.440892e-16
Czas rozwiazania: 0.000491

Wymiar macierzy: 20
Blad rozwiazania: 2.589463e-15
Czas rozwiazania: 0.001610

Wymiar macierzy: 40
Blad rozwiazania: 9.101121e-15
Czas rozwiazania: 0.005828

Wymiar macierzy: 80
Blad rozwiazania: 2.724552e-14
Czas rozwiazania: 0.024276

Wymiar macierzy: 160
Blad rozwiazania: 7.094872e-14
Czas rozwiazania: 0.111438

Wymiar macierzy: 320
Blad rozwiazania: 1.734224e-13
Czas rozwiazania: 0.604741

Wymiar macierzy: 640
Blad rozwiazania: 4.576085e-13
Czas rozwiazania: 4.755779

Wymiar macierzy: 1280
Blad rozwiazania: 1.544016e-12
Czas rozwiazania: 40.980438

DANE 2:

Wymiar macierzy: 10
Blad rozwiazania: 1.857758e-15
Czas rozwiazania: 0.000473

Wymiar macierzy: 20
Blad rozwiazania: 7.246279e-15
Czas rozwiazania: 0.001574

Wymiar macierzy: 40
Blad rozwiazania: 2.740160e-14
Czas rozwiazania: 0.005819

Wymiar macierzy: 80
Blad rozwiazania: 7.216723e-14
Czas rozwiazania: 0.024403

Wymiar macierzy: 160
Blad rozwiazania: 2.805844e-13
Czas rozwiazania: 0.107879

Wymiar macierzy: 320
Blad rozwiazania: 2.890394e-12
Czas rozwiazania: 0.567217

Wymiar macierzy: 640
Blad rozwiazania: 3.571625e-12
Czas rozwiazania: 4.491675

Wymiar macierzy: 1280
Blad rozwiazania: 1.370155e-11
Czas rozwiazania: 38.289805

DANE 3:

Wymiar macierzy: 10
Blad rozwiazania: 8.332611e-04
Czas rozwiazania: 0.000464

Wymiar macierzy: 20
Blad rozwiazania: 8.853763e-01
Czas rozwiazania: 0.001697

Wymiar macierzy: 40
Blad rozwiazania: 8.871557e-01
Czas rozwiazania: 0.005852

Wymiar macierzy: 80
Blad rozwiazania: 2.378220e+00
Czas rozwiazania: 0.024613

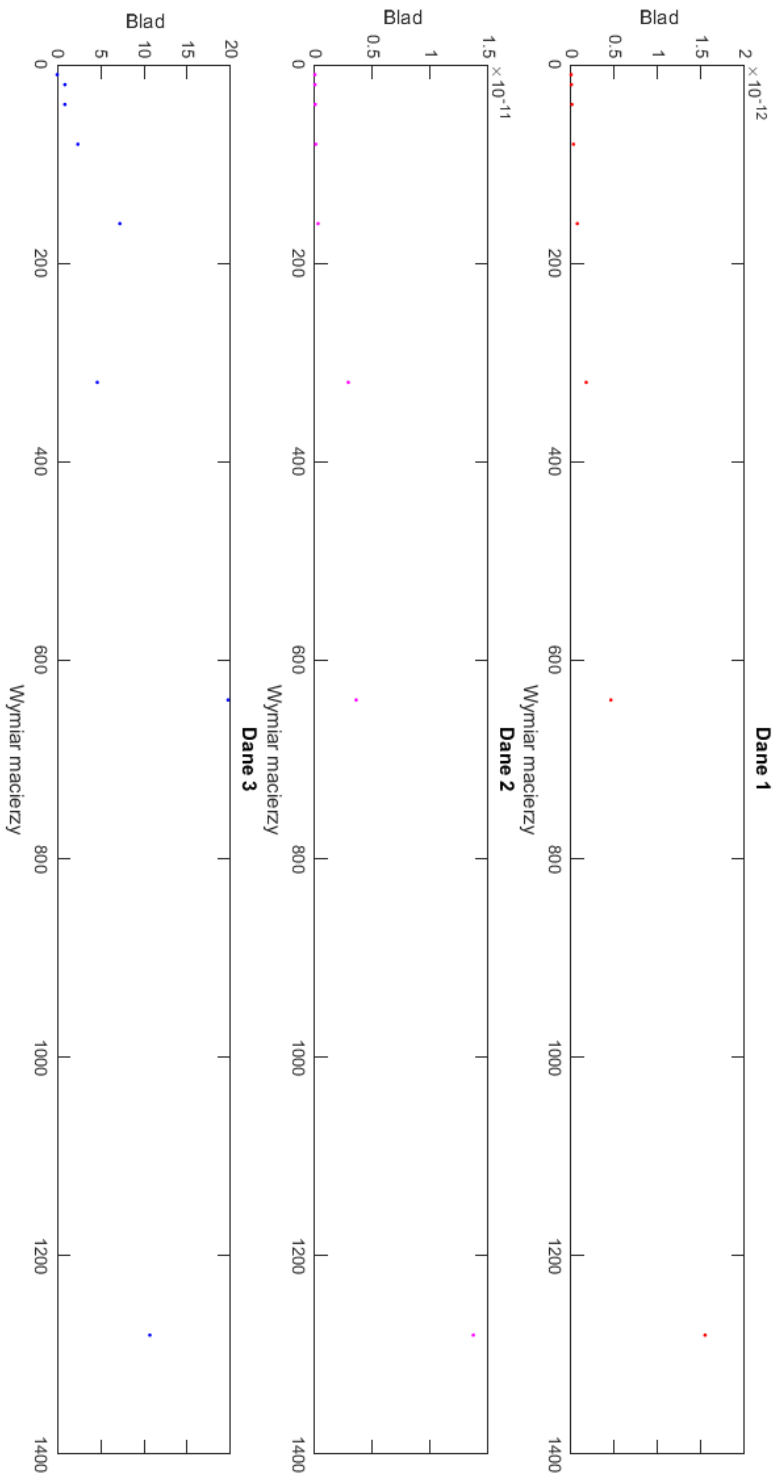
Wymiar macierzy: 160
Blad rozwiazania: 7.222390e+00
Czas rozwiazania: 0.107494

Wymiar macierzy: 320
Blad rozwiazania: 4.617822e+00
Czas rozwiazania: 0.556668

Wymiar macierzy: 640
Blad rozwiazania: 1.970035e+01
Czas rozwiazania: 4.475377

Wymiar macierzy: 1280
Blad rozwiazania: 1.067740e+01
Czas rozwiazania: 38.050090

Rysunek zależności błędu rozwiązania od liczby równań n:



Komentarz do otrzymanych wyników oraz wnioski z eksperymentów:

Najbardziej dokładne wyniki otrzymywane są dla danych nr 1. Macierz ‘A’ powstała z tych danych składa się z liczb całkowitych, natomiast macierz ‘b’ z liczb rzeczywistych z jedną cyfrą po przecinku. Jeśli chodzi o dokładność wyników dla danych nr 2, to również jest ona wystarczająca by stwierdzić, że układy równań są rozwiązane prawidłowo. Macierz ‘A’ powstała z tych danych składa się z liczb całkowitych i liczb rzeczywistych z jedną cyfrą po przecinku, natomiast macierz ‘b’ składa się z liczb rzeczywistych z jedną cyfrą po przecinku. Dla danych nr 3 błąd rozwiązania jest duży. Zarówno macierz ‘A’ jak i macierz ‘b’ powstałe z tych danych składają się z liczb rzeczywistych z dużą liczbą cyfr po przecinku. Większość z tych liczb nie da się zapisać w postaci dziesiętnej bez utraty dokładności. Duża liczba cyfr po przecinku oraz brak dokładnej reprezentacji większości liczb w systemie dziesiętnym odróżnia macierze powstałe z danych nr 3 od macierzy powstałych z danych nr 2 i 1. Jest to powodem powstawania tak dużych błędów podczas rozwiązywania układów równań równoważnych tym macierzom. Jeśli chodzi o czas rozwiązywania układów równań, to jest on podobny dla każdych danych. Bez względu na wartości macierzy ‘C’, liczba operacji jest z góry ustalona i zależy ona od liczby równań. Czasy rozwiązywania układów równań potrafią się od siebie różnić, a zależy to od komputera na którym uruchamiany jest program. Jednak bez względu na urządzenie można wyznaczyć granicę, przy której następuje gwałtowny skok ilości czasu potrzebnego na rozwiązanie układu. Tą granicą jest liczba równań równa 1280. Zaimplementowany tu algorytm jest algorytmem efektywnym, brak w nim zbędnych operacji. Działania wykonywane są na macierzach zamiast na pojedynczych wartościach.

Zadanie 3

Treść:

Proszę napisać program rozwiązujący układ n równań liniowych **Ax=b** wykorzystując metodę Jacobiego i użyć go do rozwiązania poniższego układu równań liniowych:

$15*x_1 + 3*x_2 – 2*x_3 – 8*x_4 = 5$

$3*x_1 – 12*x_2 – x_3 + 9*x_4 = -2$

$7*x_1 + 3*x_2 + 35*x_3 + 18*x_4 = 29$

$x_1 + x_2 + x_3 + 5*x_4 = 10$

Proszę sprawdzić dokładność rozwiązania oraz spróbować zastosować zaprogramowaną metodę do rozwiązania układów równań z zadania 2.

Krótki opis zastosowanych algorytmów:

W zadaniu tym używany jest algorytm implementujący metodę Jacobiego. Jest to jedna z iteracyjnych metod rozwiązywania układów równań liniowych. Metody takie polegają na powtarzaniu danych operacji określoną liczbę razy lub aż do spełnienia określonego warunku.

Napisana przeze mnie do tego celu funkcja pobiera trzy argumenty. Pierwszym jest macierz ‘A’ reprezentującą współczynniki przy odpowiednich wartościach ‘x’, drugim jest macierz wyników ‘b’ a ostatnim liczba iteracji jaka ma zostać wykonana. Na samym początku odczytywany jest jeden z wymiarów macierzy kwadratowej ‘A’, a wynik umieszczany w zmiennej ‘n’. Ponadto alokowana jest pamięć na macierz ‘x’ zawierającą wartości rozwiązań układu równań liniowych. Macierz ‘x’ ma wymiary n x 2. W pierwszej kolumnie umieszczane są najbardziej aktualne wyniki rozwiązania (pochodzące z aktualnej iteracji), natomiast w drugiej kolumnie znajdują się wartości ‘x’ z iteracji poprzedniej względem aktualnej, które potrzebne są do obliczenia wartości najnowszych. Po tych zabiegach następuje przejście do głównej pętli programu powtarzanej określoną przez trzeci argument funkcji liczbę razy. W pętli tej obliczane są najnowsze wartości macierzy ‘x’ (czyli jego pierwsza kolumna) ze wzoru $x_{j1} = -1/d_{jj}*(\sum_{k=1}^{n} (l_{jk} + u_{jk})*x_{k2} – b_j)$, gdzie ‘D’ – macierz diagonalna powstała z macierzy ‘A’; ‘L’ – macierz poddiagonalna powstała z macierzy ‘A’; ‘U’ – macierz naddiagonalna powstała z macierzy ‘A’. Ze względu na oszczędność pamięci nie są tworzone trzy nowe macierzy, jak to jest w teoretycznym modelu tej metody, tylko wykonywane są odpowiednie operacje na macierzy ‘A’. Suma ze wzoru liczona jest dla danego ‘j’ poprzez zsumowanie wszystkich iloczynów kolejnych elementów j-tego wiersza z odpowiadającymi im wartościami ‘x’ z drugiej kolumny. Następnie od tej sumy odejmowany jest iloczyn wartości leżącej na przekątnej z odpowiadającym jej ‘x’. W ten sposób otrzymuje mniejszą liczbę operacji, niż gdyby za każdym razem sprawdzać, czy dany element leży na diagonalnej macierzy czy nie. Jest to jeden z elementów optymalizacji. Kolejne obliczenia wykonywane są zgodnie ze wzorem opisanym wcześniej. Przed przejściem do kolejnej iteracji wartości pierwszej kolumny kopiowane są do kolumny drugiej. Gdy oczekiwana liczba iteracji zostanie zrealizowana, druga kolumna macierzy ‘x’ zostaje usunięta i powstały wektor zwracany jako wynik, razem z czasem wykonania.

Wydruk dobrze skomentowanych programów z implementacją użytych algorytmów:

JCB – służy do rozwiązywania układów równań iteracyjną metodą Jacobiego:

```
% funkcja do wyznaczania rozwiazania rownan liniowych metoda Jacobiego
% zalozenie: przekazywane macierze A i b maja odpowiednie wymiary
% pozwalajace wyznaczyc rozwiazania ukladu rownan, przy czym:
% A - macierz nieosobliwa n x n wspolczynnikow
% b - macierz n x 1 wynikow
% ponadto zakladam ze dla kazdego 'i' A(i,i) ~= 0
function [ x, time ] = JCB( A, b, maxIterations )
    % nie tworze oddzielnie macierzy L (poddiagonalna), D(diagonalna) i U
    % (naddiagonalna) aby niepotrzebnie nie marnowac pamieci, wszystkie
    % obliczenia beda wykonywane za pomoca macierzy A

    % ustawiam pomiar czasu wykonywania funkcji
    tic;
    % pobieram wymiar macierzy kwadratowej A
    [n,~] = size(A);
    % tworze macierz wynikowa uzupelniona samymi zerami: pierwsza kolumna
    % przechowuje najnowszy wektor x, druga kolumna przechowuje starszy
    % wektor x
    x = zeros(n, 2);
    % ustawiam poczatkowa wartosc parametru currentIter
    currentIter = 1;
    % dopoki liczba iteracji jest niewieksza od parametru maxIterations
    % w petli liczymy kolejne przyblizenia
    while currentIter <= maxIterations
        for i = 1: n
            sum = 0;
            % pierwszy etap obliczania sum
            for k = 1: n
                sum = sum + A(i,k) * x(k,2);
            end
            % drugi etap obliczania sum
            sum = sum - A(i,i) * x(i,2);
            sum = sum - b(i);
            x(i,1) = -1/A(i,i) * sum;
        end
        % po wyjsciu z petli for mam obliczone nowe wartosci macierzy x
        % aktualizuje currentIter
        currentIter = currentIter + 1;
        % uaktualniam druga kolumne wektora x aby gotowy byl do dalszej
        % iteracji
        x(:,2) = x(:,1);
    end
    % koryguje wektor x, usuwam niepotrzebna juz druga kolumne
    x(:,2) = [];
    % koncze pomiar czasu
    time = toc;
end
```

getMatrixes – służy do generowania wybranego typu macierzy:

```
% funkcja zwracajaca macierze wspolczynnikow (A) i macierz wynikowa (b)
% utworzone wedlug jednego z 3 algorytmow podanych w tresci zadania
% laboratoryjnego
function [ A, b ] = getMatrixes( n, dataNumber )
    % uzupelniam obie macierze samymi zerami
    A = zeros(n, n);
    b = zeros(n, 1);
    switch dataNumber
        case 1
            % uzupelniam wektor A
            for i = 1: n
                A(i,i) = 7;
                if i < n
                    A(i,i+1) = 1;
                end
                if i > 1
                    A(i,i-1) = 1;
                end
            end
            % uzupelniam wektor b
            for i = 1: n
                b(i) = 1.4 + 0.6*i;
            end
        case 2
            % uzupelniam macierz A
            % najpierw uzupelniam wszystkie elementy macierzy w wierszu
            % wedlug wzoru aij = 2*(i-j)+1, nastpenie element aii
            % zastepuje 0.2 ilosc operacji wykonywanych w takim algorytmie
            % jest mniejsza niz gdyby za kazdym razem porownywac
            % indeksy i oraz j
            for i = 1: n
                for j = 1: n
                    A(i,j) = 2*(i-j)+1;
                end
                A(i,i) = 0.2;
            end
            % uzupelniam macierz b
            for i = 1: n
                b(i) = 1 + 0.4*i;
            end
        case 3
            % uzupelniam wektor A
            for i = 1: n
                for j = 1: n
                    A(i,j) = 7/(9*(i+j+1));
                end
            end
            % uzupelniam wektor b
            for i = 2: 2: n
                b(i) = 7/(5*i);
            end
    end
end
```

getTestResultsJBC – pomocnicza funkcja testująca:

```
% funkcja wywolujaca program dla okreslonego w dataNumber zestawu danych
% dla macierzy o wymiarach n1=10, n2=20 ... nx = 10* 2^(x-1), gdzie x
% okreslone jest przez parametr numberOfAttempts
function [ N, Err, TimeM] = getTestResultsJCB( numOfA, dataNr, maxIter)
    % tworze trzy puste macierze zawierajace odpowiednio wymiary
    % kolejnych macierzy, bledy rozwiazania oraz czasy rozwiazania
    N = zeros( numOfA, 1);
    Err = zeros( numOfA, 1);
    TimeM = zeros( numOfA, 1);
    % glowna petla programu
    for i = 1: numOfA
        % obliczam wymiar aktualnej macierzy
        tempN = 10*2^(i-1);
        % uzupelniam macierz wymiarow macierzy
        N(i) = tempN;
        % generuje odpowiednia macierz
        [A, b] = getMatrixes( tempN, dataNr);
        % wywoluje funkcje wyznaczajaca rozwiazanie ukladu rownan
        [x, time ] = JCB(A, b, maxIter);
        % uzupelniam macierz czasow wykonania
        TimeM(i) = time;
        % wyznaczam residuum
        residuum = A*x - b;
        % wyznaczam norme residuum
        residuumNorm = getNorm2(residuum);
        % uzupelniam macierz bledow rozwiazania
        Err(i) = residuumNorm;
    end
end
```

getNorm2 – pomocnicza funkcja do wyznaczania drugiej normy macierzy:

```
% norma druga macierzy A jest to pierwiastek z najwiekszego modulu wartosci
% wlasnej macierzy powstalej z pomnozenia transponowanej macierzy
% A przez macierz A
function [ norm2 ] = getNorm2( A )
    % pomocniczo wyznaczam macierz T dla ktorej bede liczyl wartosci
    % wlasne
    T = A' * A;
    % w zmiennej maxEig przechowuje najwieksza wartosc wlasna macierzy T
    maxEig = max(abs(eig(T)));
    % obliczam norme druga jako pierwiastek z najwiekszej wartosci wlasnej
    norm2 = sqrt(maxEig);
end
```

getDataForGraphJCB – funkcja zwracająca informacje potrzebne do wykresu:

```
% funkcja zwracajaca dwie macierze: Iter przechowuje kolejne liczby
% iteracji, Errors przechowuje kolejne bledy rozwiazania. Na podstawie tych
% informacji utworzony zostanie wykres bledu rozwiazania od liczby iteracji
% funkcja pobiera trzy argumenty: macierz wspolczynnikow A, macierz
% wynikowa b oraz liczbe iteracji dla ktorych nalezy obliczyc blad
function [Iter, Errors] = getDataForGraphJCB( A, b, numbOfIter )
    % alokuje pamiec na macierz Errors
    Errors = zeros(numbOfIter,1);
    % poniewaz od razu wiem jak bedzie wygladala macierz Iter uzupelniam ja
    Iter = 1 : numbOfIter;
    % glowna petla okreslajaca liczbe iteracji
    for i = 1: numbOfIter
        % wyznaczam rozwiazanie ukladu dla liczby iteracji i
        [x,~] = JCB(A, b, i);
        % wyznaczam residuum
        residuum = A*x - b;
        % wyznaczam norme residuum
        residuumNorm = getNorm2(residuum);
        % uzupelniam macierz Errors
        Errors(i) = residuumNorm;
    end
end
```

getTask3Solution – funkcja główna programu:

```
function [ ] = getTask3Solution( )
    % ZADANIE 3
    % otwieram plik do ktorego zapisze wyniki
    [id, kom] = fopen('wynikiTestowZad3.txt', 'wt');
    if id < 0
        disp(kom);
    end
    % inicjuje zmienna okreslajaca maksymalna ilosc iteracji
    maxIter = 100;
    fprintf('\tILOSC ITERACJI: %d\n', maxIter);
    if id > 0
        fprintf(id, '\tILOSC ITERACJI: %d\n', maxIter);
    end
    % uklad rownan podany w otrzymanym zadaniu
    A = [ 15 3 -2 -8; 3 -12 -1 9; 7 3 35 18; 1 1 1 5];
    b = [ 5; -2; 29; 10];
    [x,~] = JCB(A, b, maxIter );
    % wyznaczam residuum
    residuum = A*x - b;
    % wyznaczam norme residuum
    residuumNorm = norm(residuum);
    fprintf('--------------------------\n');
    fprintf('Rozwiazanie ukladu rownan podanego w zadaniu:\n');
    fprintf('x1 = %f, x2 = %f, x3 = %f, x4 = %f\n', x(1),x(2),x(3),x(4));
    fprintf('Blad rozwiazania ukladu rownan podanego w zadaniu:\n');
    fprintf('blad = %g\n\n', residuumNorm);
    fprintf('--------------------------\n');
    if id > 0
        fprintf(id, '--------------------------\n');
        fprintf(id, 'Rozwiazanie ukladu rownan podanego w zadaniu:\n');
        fprintf(id, 'x1 = %f, x2 = %f, x3 = %f, x4 = %f\n', x(1),x(2),x(3),x(4));
        fprintf(id, 'Blad rozwiazania ukladu rownan podanego w zadaniu:\n');
        fprintf(id, 'blad = %g\n\n', residuumNorm);
        fprintf(id, '--------------------------\n');
    end
    % rysuje wykres zaleznosci bledu rozwiazania od ilosci iteracji dla
    % ukladu rownan podanego w otrzymanym zadaniu
    [I,E] = getDataForGraphJCB(A,b,15);
    plot(I,E,'r.');
    title('Wykres zaleznosci bledu od liczby iteracji');
    xlabel('Liczba iteracji');
    ylabel('Blad');
    % inicjuje zmienna okreslajaca ilosc prob
    numberOfAttempts = 10;
    % wywoluje testy dla wszystkich danych
    [n1, e1, t1] = testZad3(numberOfAttempts, 1, maxIter);
    [n2, e2, t2] = testZad3(numberOfAttempts, 2, maxIter);
    [n3, e3, t3] = testZad3(numberOfAttempts, 3, maxIter);
    % wyniki testow dla danych 1
    fprintf('\tDANE 1:\n');
    if id > 0
        fprintf(id, '\tDANE 1:\n');
    end
    for i = 1: numberOfAttempts
        fprintf('-------------------------\n');
        fprintf('Wymiar macierzy: %d\n', n1(i));
        fprintf('Blad rozwiazania: %e\n', e1(i));
        fprintf('Czas rozwiazania: %f\n', t1(i));
        if id > 0
```

```
            fprintf(id, '--------------------------\n');
            fprintf(id, 'Wymiar macierzy: %d\n', n1(i));
            fprintf(id, 'Blad rozwiazania: %e\n', e1(i));
            fprintf(id, 'Czas rozwiazania: %f\n', t1(i));
        end
    end
    % wyniki testow dla danych 2
    fprintf('\tDANE 2:\n');
    if id > 0
        fprintf(id, '\tDANE 2:\n');
    end
    for i = 1: numberOfAttempts
        fprintf('--------------------------\n');
        fprintf('Wymiar macierzy: %d\n', n2(i));
        fprintf('Blad rozwiazania: %e\n', e2(i));
        fprintf('Czas rozwiazania: %f\n', t2(i));
        if id > 0
            fprintf(id, '--------------------------\n');
            fprintf(id, 'Wymiar macierzy: %d\n', n2(i));
            fprintf(id, 'Blad rozwiazania: %e\n', e2(i));
            fprintf(id, 'Czas rozwiazania: %f\n', t2(i));
        end
    end
    % wyniki testow dla danych 3
    fprintf('\tDANE 3:\n');
    if id > 0
        fprintf(id, '\tDANE 3:\n');
    end
    for i = 1: numberOfAttempts
        fprintf('--------------------------\n');
        fprintf('Wymiar macierzy: %d\n', n3(i));
        fprintf('Blad rozwiazania: %e\n', e3(i));
        fprintf('Czas rozwiazania: %f\n', t3(i));
        if id > 0
            fprintf(id, '--------------------------\n');
            fprintf(id, 'Wymiar macierzy: %d\n', n3(i));
            fprintf(id, 'Blad rozwiazania: %e\n', e3(i));
            fprintf(id, 'Czas rozwiazania: %f\n', t3(i));
        end
    end
    fclose(id);
end
```

Prezentacja otrzymanych wyników:

ILOSC ITERACJI: 100

Rozwiazanie ukladu rownan podanego w zadaniu:
x1 = 0.821044, x2 = 1.577824, x3 = -0.281683, x4 = 1.576563
Blad rozwiazania ukladu rownan podanego w zadaniu:
blad = 2.51215e-15

DANE 1:

Wymiar macierzy: 10
Blad rozwiazania: 1.616509e-15
Czas rozwiazania: 0.000495

Wymiar macierzy: 20
Blad rozwiazania: 3.411114e-15
Czas rozwiazania: 0.000943

Wymiar macierzy: 40
Blad rozwiazania: 8.668276e-15
Czas rozwiazania: 0.002454

Wymiar macierzy: 80
Blad rozwiazania: 2.997849e-14
Czas rozwiazania: 0.008522

Wymiar macierzy: 160
Blad rozwiazania: 6.062590e-14
Czas rozwiazania: 0.035216

Wymiar macierzy: 320
Blad rozwiazania: 1.902346e-13
Czas rozwiazania: 0.135435

Wymiar macierzy: 640
Blad rozwiazania: 5.200474e-13
Czas rozwiazania: 0.683518

Wymiar macierzy: 1280
Blad rozwiazania: 1.298347e-12
Czas rozwiazania: 4.265375

Wymiar macierzy: 2560
Blad rozwiazania: 3.577841e-12
Czas rozwiazania: 22.801696

Wymiar macierzy: 5120
Blad rozwiazania: 1.215037e-11
Czas rozwiazania: 93.111219

DANE 2:

Wymiar macierzy: 10
Blad rozwiazania: 5.896373e+246
Czas rozwiazania: 0.000522

Wymiar macierzy: 20
Blad rozwiazania: 3.778943e+307
Czas rozwiazania: 0.000922

Wymiar macierzy: 40
Blad rozwiazania: NaN
Czas rozwiazania: 0.002458

Wymiar macierzy: 80
Blad rozwiazania: NaN
Czas rozwiazania: 0.008448

Wymiar macierzy: 160
Blad rozwiazania: NaN
Czas rozwiazania: 0.031248

Wymiar macierzy: 320
Blad rozwiazania: NaN
Czas rozwiazania: 0.126067

Wymiar macierzy: 640
Blad rozwiazania: NaN
Czas rozwiazania: 0.667694

Wymiar macierzy: 1280
Blad rozwiazania: NaN
Czas rozwiazania: 4.155492

Wymiar macierzy: 2560
Blad rozwiazania: NaN
Czas rozwiazania: 22.757346

Wymiar macierzy: 5120
Blad rozwiazania: NaN
Czas rozwiazania: 93.249703

DANE 3:

Wymiar macierzy: 10
Blad rozwiazania: 3.049410e+91
Czas rozwiazania: 0.000530

Wymiar macierzy: 20
Blad rozwiazania: 9.972097e+122
Czas rozwiazania: 0.000962

Wymiar macierzy: 40
Blad rozwiazania: 3.323045e+153
Czas rozwiazania: 0.002557

Wymiar macierzy: 80
Blad rozwiazania: 4.811256e+183
Czas rozwiazania: 0.008661

Wymiar macierzy: 160
Blad rozwiazania: 5.169718e+213
Czas rozwiazania: 0.031455

Wymiar macierzy: 320
Blad rozwiazania: 4.982524e+243
Czas rozwiazania: 0.123103

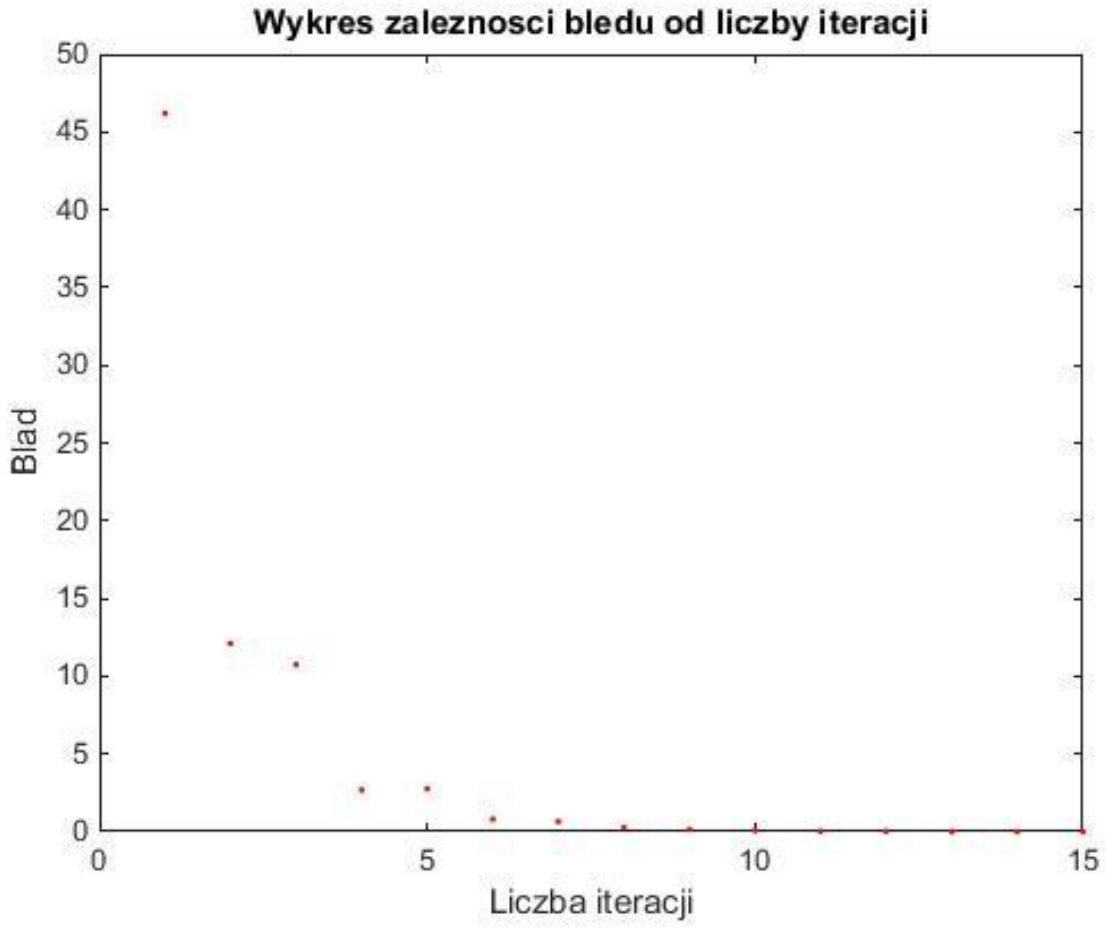
Wymiar macierzy: 640
Blad rozwiazania: 4.605547e+273
Czas rozwiazania: 0.669343

Wymiar macierzy: 1280
Blad rozwiazania: 4.184830e+303
Czas rozwiazania: 4.165612

Wymiar macierzy: 2560
Blad rozwiazania: NaN
Czas rozwiazania: 22.770975

Wymiar macierzy: 5120
Blad rozwiazania: NaN
Czas rozwiazania: 93.391954

Rysunek zależności błędu rozwiązania od ilości iteracji dla układu równań podanego w zadaniu:



Komentarz do otrzymanych wyników oraz wnioski z eksperymentów:

Układ równań podany w tym zadaniu jest za pomocą tej metody rozwiązywany prawidłowo. Błąd rozwiązania jest bardzo mały. Promień spektralny macierzy $M = -D^{-1}(L+U)$ jest mniejszy od jedności, zatem warunek zbieżności ciągu kolejnych rozwiązań jest spełniony. Jeśli chodzi o układy równań z zadania 2, to tylko pierwszy z nich da się rozwiązać za pomocą tej metody. Błędy rozwiązań są względnie małe, biorąc pod uwagę rozmiary macierzy (ilość równań). W tym przypadku warunek konieczny zbieżności kolejnych rozwiązań jest również spełniony. Spełniony jest tu też warunek dostateczny zbieżności metody Jacobiego, który zakłada silną diagonalną dominację macierzy. W przypadku pozostałych układów równań z zadania 2, metoda ta nie sprawdza się. Otrzymujemy bardzo duże błędy (w przypadku danych nr 3) i 'NaN' (w przypadku danych nr 2). W żadnym z tych przypadków warunek konieczny zbieżności nie jest spełniony, czyli promień spektralny macierzy $M = -D^{-1}(L+U)$ jest większy od jedności. Błędy rozwiązań dążą do nieskończoności. Gdyby liczba iteracji została zwiększona, w obu tych przypadkach błąd rozwiązania wynosiłby 'NaN'. Wynika to z faktu, iż kolejne rozwiązania układu są co do modułu coraz większe, aż w końcu osiągają 'inf' lub '–inf'. Ponieważ w macierzy 'A' generowanej przez dane nr 2 są elementy ujemne i dodatnie, to po wielu iteracjach rozwiązania osiągają wartości '–inf' lub 'inf'. Wynikiem działania 'inf' – 'inf' jest 'NaN', a działanie takie może się zdarzyć w dwóch miejscach w algorytmie. Jeśli chodzi o macierz z danymi nr 3, to są tam tylko liczby dodatnie. Zatem po wielu iteracjach w wektorze wyników 'x' będą znajdować się dodatnie nieskończoności ('inf'). Gdyby implementować wzór zgodny z teoretycznymi założeniami, to wynik ten nie zmienił by się w dalszych iteracjach (błąd pozostałby równy 'inf'). Jednak wprowadzona przeze mnie optymalizacja opisana we wcześniejszym z punktów sprawia, że jeden ze składników sumy (każdy z tych składników jest równy nieskończoność) należy od niej odjąć. W ten sposób otrzymujemy działanie 'inf' – 'inf', co prowadzi do powstania 'NaN'. Ponieważ jednak algorytm ten przeznaczony jest do rozwiązywania tylko niektórych układów równań oraz biorąc pod uwagę fakt, ze wynik 'inf' również nie jest prawidłowym wynikiem, to uważam że wprowadzona przeze mnie optymalizacja jest dobrym rozwiązaniem. Jeśli chodzi o dokładność wyników, to dokładność jest tym większa, im większa jest liczba iteracji. Trudno określić optymalną liczbę iteracji, ponieważ zależy to od rozpatrywanego przypadku. Im promień spektralny mniejszy, tym szybkość zbieżności, czyli szybkość malenia błędu, większa, a co za tym idzie, liczba iteracji potrzebnych do otrzymania dobrego wyniku mniejsza.