Tomasz Bocheński

MNUM Projekt 2 – DOKUMENTACJA cz. 2

Wydruk skomentowanych programów:

getEigenvalNoShiftQR – funkcja obliczająca wartości własne macierzy metodą QR bez przesunięć:

```
% funkcja wyznaczajaca wartosci wlasne macierzy symetrycznej metoda QR bez
% przesuniec
% funkcja pobiera trzy argumenty: pierwszy argument (A) jest to macierz
% ktorej wartosci wlasne sa poszukiwane, drugi argument (precision) okresla
% precyzje z jaka wartosci wlasne maja zostac wyznaczone (czyli najwieksza
% dopuszczalna wartosc dla liczb lezacych pod i nad diagonala), trzeci
% argument (maxIter) okresla maksymalna liczbe iteracji, ktora moze zostac
% przeprowadzona w celu znalezienia wartosci wlasnych
% funkcja zwraca 4 argumenty: pierwszy (eval) to wartosci wlasne macierzy,
% drugi (iter) okresla wykonana liczbe iteracji, trzeci (prAch) okresla czy
% podana precyzja zostala osiagnieta, czwarty (time) to czas wykonania
% programu
function [eval,iter,prAch,time]=getEigenvalNoShiftQR(A,precision,maxIter)
    % zaczynam mierzyc czas wykonania
    tic;
    % inicjalizuje zmienna prAch na 1; wartosc 1 oznacza ze podana
    % zostala osiagnieta, wartosc 0 ze nie zostala osiagnieta; dokonuje
    % tutaj zalozenia ze podana precyzja zostanie osiagnieta, ewentualna
    % korekta zostanie wykonana na koncu programu (mniejsza liczba skokow)
    prAch = 1;
    % inicjalizuje zmienna currIter
    currIter = 0;
    % petla dzialajaca do momentu wykonania maksymalnej mozliwej liczby
    % iteracji lub do momentu otrzymania satysfakcjonujacej nas dokladnosci
    while currIter < maxIter && max(max(abs(A - diag(diag(A)))))>precision
        % wyznaczam rozklad QR macierzy A (faktoryzacja)
        [Q,R] = getQRmodGramSchmidt(A);
        % wyznaczam macierz przeksztalcona
        A = R * Q;
        % zwiekszam licznik iteracji
        currIter = currIter + 1;
    end
    % zapamietuje w zmiennej eval wyznaczone wartosci wlasne
    eval = diag(A);
    % ustawiam wartosc iter
    iter = currIter;
    % sprawdzam czy po wyjsciu z petli osiagnelismy podana precyzje,
    % koryguje zmienna precisionAchieved
    if max(max(abs(A- diag(diag(A))))) > precision
        prAch = 0;
    end
    % odczytuje czas wykonania
    time = toc;
end
```

getEigenvalShiftQR – funkcja obliczająca wartości własne macierzy metodą QR z przesunięciami:

```
% funkcja wyznaczajaca wartosci wlasne macierzy symetrycznej i macierzy
% niesymetrycznej metoda QR z przecunieciami
```

```
% funkcja pobiera trzy argumenty: pierwszy argument (A) jest to macierz
% ktorej wartosci wlasne sa poszukiwane, drugi argument (precision) okresla
% precyzje z jaka wartosci wlasne maja zostac wyznaczone (czyli najwieksza
% dopuszczalna wartosc dla liczb lezacych pod i nad diagonala), trzeci
% argument (maxIterPer) okresla maksymalna liczbe iteracji, ktora moze
zostac
% przeprowadzona w celu znalezienia pojedynczej wartosci wlasnej
% funkcja zwraca 4 argumenty: pierwszy (eval) to wartosci wlasne macierzy,
% drugi (iter) okresla wykonana liczbe iteracji, trzeci (prAch) okresla czy
% podana precyzja zostala osiagnieta, czwarty (time) to czas wykonania
% programu
function [eval,iter,prAch,time] =
getEigenvalShiftQR(A,precision,maxIterPer)
    % zaczynam mierzyc czas wykonania
    tic;
    % inicjalizuje zmienna prAch na 1; wartosc 1 oznacza ze podana
    % zostala osiagnieta, wartosc 0 ze nie zostala osiagnieta; dokonuje
    % tutaj zalozenia ze podana precyzja zostanie osiagnieta, ewentualna
    % korekta zostanie wykonana na koncu programu (mniejsza liczba skokow)
    prAch = 1;
    % inicjalizuje zmienna iter, okresla ona calkowita liczbe iteracji
    iter = 0;
    % odczytuje stopien macierzy kwadratowej A i zapisuje go w zmiennej n
    [n,~] = size(A);
    % alokuje pamiec na macierz wynikowa eval
    eval = zeros(n,1);
    % inicjalizuje macierz submatrixA, ktora przechowuje aktualna dla
    % kazdego kroku k podmacierz
    submatrixA = A;
    % petla zewnetrzna, w kazdym jej przebiegu wyznaczana jest kolejna
    % wartosc wlasna macierzy
    for k = n : -1 : 2
        % inicjalizuje zmienna AK na wartosc poczatkowa submatrixA
        AK = submatrixA;
        % inicjalizuje currIter
        currIter = 0;
        % petla wewnetrzna, dzialajaca do momentu wykonania maksymalnej
        % mozliwej liczby iteracji lub do momentu otrzymania
        % satysfakcjonujacej nas dokladnosci
        while currIter < maxIterPer && max(abs(AK(k,1:k-1))) > precision
            % wyznaczam wartosc przesuniecia jako wartosc wlasna
            % podmacierzy 2x2 z prawego dolnego rogu macierzy AK blizsza
            % wartosci o wspolrzednych (2,2) tej podmacierzy
            M2x2 = AK(k-1:k,k-1:k);
            EigenvalM = getEigenvalOf2x2Matrix(M2x2);
            % inicjalizuje zmienna shift pierwsza wartoscia wlasna, w razie
            % potrzeby zostanie ona skorygowana
            shift = EigenvalM(2);
            % koryguje wartosc shift gdy zachodzi taka koniecznosc
            if abs(EigenvalM(1) - M2x2(2,2))<abs(EigenvalM(2) - M2x2(2,2))
                shift = EigenvalM(1);
            end
            % wyznaczam macierz przesunieta
            AK = AK - eye(k)*shift;
            % wyznaczam rozklad QR macierzy AK (faktoryzacja)
            [Q,R] = getQRmodGramSchmidt(AK);
            % wyznaczam macierz przeksztalcona
            AK = R * Q + eye(k)*shift;
            % zwiekszam licznik iteracji
            currIter = currIter + 1;
            % zwiekszam ilosc wszystkich iteracji
```

```
            iter = iter + 1;
        end
        % koryguje zmienna prAch
        if prAch == 1 && max(abs(AK(k,1:k-1))) > precision
            prAch = 0;
        end
        % uzupelniam macierz eval o obliczona w tym kroku wartosc wlasna
        eval(k) = AK(k,k);
        % jesli stopien macierzy kwadratowej jest wiekszy od 2 to
        % opuszczamy ostatni wiersz i ostatnia kolumne (deflacja)
        if k > 2
            submatrixA = AK(1:k-1,1:k-1);
        % w przeciwnym wypadku byl to ostatni przebieg petli zewnetrznej,
        % a ostatnia poszukiwana wartosc wlasna znajduje sie w AK(1,1)
        else
            eval(1) = AK(1,1);
        end
    end
    % odczytuje czas wykonania
    time = toc;
end
```

getQRmodGramSchmidt – funkcja wyznaczająca rozkład wąski QR za pomocą zmodyfikowanego algorytmu Gram'a Schmidt'a:

```
% funkcja do wyznaczania rozkladu QR waskiego zmodyfikowanym algorytmem
% Gram'a Schmidt'a; standardowy algorytm Gram'a Schmidt'a ma niekorzystne
% wlasnosci numeryczne, w zmodyfikowanym algorytmie proces ortogonalizacji
% przeprowadzany jest w innej kolejnosci; standardowy algorytm
% ortogonalizuje kolumny macierzy po kolei, w algorytmie zmodyfikowanym po
% wyznaczeniu kolejnej kolumny ortogonalnej od razu ortogonalizuje sie
% wobec niej wszystkie nastepne kolumny
% algorytm dziala dla macierzy o pelnym rzedzie i wyznacza rozklad QR waski
function [ Q, R ] = getQRmodGramSchmidt( A )
    % odczytuje wymiary macierzy
    [m,n] = size(A);
    % alokuje pamiec na macierz Q
    Q = zeros(m,n);
    % alokuje pamiec na macierz R
    R = zeros(n,n);
    % alokuje pamiec na pomocnicza macierz AUX
    AUX = zeros(1,n);
    % petla zewnetrzna programu
    for i = 1 : n
        Q(:,i) = A(:,i);
        R(i,i) = 1;
        AUX(i) = Q(:,i)' * Q(:,i);
        % petla wewnetrzna programu
        for j = i + 1 : n
            R(i,j) = (Q(:,i)' * A(:,j))/AUX(i);
            A(:,j) = A(:,j) - R(i,j) * Q(:,i);
        end
    end
    % normowanie rozkladu
    for i = 1: n
        norm2 = norm(Q(:,i));
        Q(:,i) = Q(:,i)/norm2;
        R(i,i:n) = R(i,i:n) * norm2;
    end
```

```
end
```

getEigenvalOf2x2Matrix – funkcja wyznaczająca wartości własne macierzy 2x2:

```
% funkcja pomocnicza do wyznaczania wartosci wlasnych macierzy kwadratowej
% stopnia 2
% wartosci wlasne sa wylicznane bezposrednio z definicji, czyli szukam
% takich x, ktore spelniaja rownania det(A-xI) = 0
% niech A = [A(1,1), A(1,2); A(2,1), A(2,2)]
% wtedy rownanie charakterystyczne jest postaci:
% (A(1,1)-x) * (A(2,2)-x) - A(1,2) * A(2,1) = 0
% postac po przeksztalceniach:
% x^2 + x[-A(1,1)-A(2,2)] + [A(1,1)*A(2,2)-A(1,2)*A(2,1)]
% delta po przeksztalceniach:
% delta = A^2(1,1)+A^2(2,2)-2*A(1,1)*A(2,2)+4*A(1,2)*A(2,1)
% wartosci x1 i x2 po przeksztalceniach:
% x1 = [A(1,1)+A(2,2)-sqrt(delta)]/2
% x2 = [A(1,1)+A(2,2)+sqrt(delta)]/2
% funkcja pobiera jeden argument: maicerz kwadratowa stopnia 2 ktorej
% wartosci wlasne nalezy wyznaczyc
% funkcja zwraca macierz wyznaczonych wartosci wlasnych
function [ eval ] = getEigenvalOf2x2Matrix( A )
    % alokuje pamiec na macierz wynikowa eval
    eval = zeros(2,1);
    % obliczam delte rownania charakterystycznego
    delta = A(1,1)*A(1,1)+A(2,2)*A(2,2)-2*A(1,1)*A(2,2)+4*A(1,2)*A(2,1);
    % obliczam wartosci wlasne macierzy
    eval(1) = (A(1,1)+A(2,2)-sqrt(delta))/2;
    eval(2) = (A(1,1)+A(2,2)+sqrt(delta))/2;
end
```

getRandomMatrix – funkcja zwracająca losową macierz:

```
% funkcja generujaca macierz o losowych wartosciach
% funkcja przyjmuje dwa argumenty: pierwszy (n) okresla stopien macierzy
% kwadratowej ktora ma zostac wygenerowana, drugi (flag) okresla czy
% macierz ma byc symetryczna czy niesymetryczna; gdy flag wynosi 0
% generowana jest macierz niesymetryczna, w przeciwnym przypadku zostaje
% stworzona macierz symetryczna
function [ M ] = getRandomMatrix( n, flag )
    % generuje losowa macierz M
    M = randi([-2,1],n,n) + rand(n,n);
    % w razie potrzeby przeksztalcam ja do macierzy symetrycznej
    if flag ~= 0
        M = M + M';
    end
end
```

getAverageIterationTime – pomocnicza funkcja do testowania:

```
% funkcja zwracajaca srednia liczbe iteracji i sredni czas obliczen
% funkcja pobiera trzy argumenty: pierwszy (numbOfTests) okresla ilosc
% testow ktore nalezy wykonac, drugi (sym) okresla czy testowane macierze
% maja byc symetryczne czy nie ( sym == 0: macierz jest niesymetryczna),
% trzeci (shift) okresla czy obliczenia wykonywane sa za pomoca metody z
% przesunieciami czy bez przesuniec (shift == 0: bez przesuniec)
% funkcja zwraca trzy macierze: pierwsza okresla srednia liczbe iteracji i
% sredni czas dla macierzy 5x5, druga dla macierzy 10x10 i trzecia dla
% macierzy 20x20
```

```
function [M5,M10,M20] = getAverageIterationsTime(nmbOfTests,sym,shift,prec)
    % alokuje pamiec na macierze M5, M10, M20
    M5 = zeros(2,1);
    M10 = zeros(2,1);
    M20 = zeros(2,1);
    % petla odliczajaca liczbe testow
    for i = 1 : nmbOfTests
        A5 = getRandomMatrix(5,sym);
        A10 = getRandomMatrix(10,sym);
        A20 = getRandomMatrix(20,sym);
        if shift == 0
            [~,iter5,~,time5] = getEigenvalNoShiftQR(A5,prec,intmax);
            [~,iter10,~,time10] = getEigenvalNoShiftQR(A10,prec,intmax);
            [~,iter20,~,time20] = getEigenvalNoShiftQR(A20,prec,intmax);
        else
            [~,iter5,~,time5] = getEigenvalShiftQR(A5,prec,intmax);
            [~,iter10,~,time10] = getEigenvalShiftQR(A10,prec,intmax);
            [~,iter20,~,time20] = getEigenvalShiftQR(A20,prec,intmax);
        end
        % sumuje ogolna liczbe iteracji i ogolny czas
        M5(1) = M5(1) + iter5;
        M5(2) = M5(2) + time5;
        M10(1) = M10(1) + iter10;
        M10(2) = M10(2) + time10;
        M20(1) = M20(1) + iter20;
        M20(2) = M20(2) + time20;
    end
    % wyznaczam srednia dzielac przez liczbe testow
    M5 = M5 / nmbOfTests;
    M10 = M10 / nmbOfTests;
    M20 = M20 / nmbOfTests;
end
```

getTask1Solution – główna funkcja testująca:

```
% zadanie 1: glowna funkcja, zajmuje sie obliczaniem wynikow, ich
% prezentacja oraz zapisywaniem ich do pliku
function [ ] = getTask1Solution( )
    % otwieram plik do ktorego zapisze wyniki
    [id, kom] = fopen('wynikiTestowZad1P2.txt', 'wt');
    if id < 0
        disp(kom);
    end
    % wyznaczam srednie ilosci iteracji

    % SYMETRYCZNA, BEZ PRZESUNIEC
    % precyzja: 0.1
    [M5_1, M10_1, M20_1] = getAverageIterationsTime(30,1,0,0.1);
    % precyzja: 0.01
    [M5_2, M10_2, M20_2] = getAverageIterationsTime(30,1,0,0.01);

    % SYMETRYCZNA, Z PRZESUNIECIAMI
    % precyzja: 0.00001
    [M5_3, M10_3, M20_3] = getAverageIterationsTime(30,1,1,0.00001);
    % precyzja: 0.0000001
    [M5_4, M10_4, M20_4] = getAverageIterationsTime(30,1,1,0.0000001);

    % NIESYMETRYCZNA, Z PRZESUNIECIAMI
    % precyzja: 0.00001
    [M5_5, M10_5, M20_5] = getAverageIterationsTime(30,0,1,0.00001);
```

```
% precyzja: 0.0000001
[M5_6, M10_6, M20_6] = getAverageIterationsTime(30,0,1,0.0000001);


% drukowanie wynikow na ekran (i do pliku)
fprintf('---------------------------------------\n');
fprintf('---------------------------------------\n');
fprintf('MACIERZ SYMETRYCZNA, BEZ PRZESUNIEC\n');
fprintf('---------------------------------------\n');
fprintf('PRECYZJA: 0.1\n');
fprintf('Macierze 5x5\n');
fprintf('Srednia ilosc iteracji: %d\n', M5_1(1));
fprintf('Sredni czas wykonania: %g\n', M5_1(2));
fprintf('Macierze 10x10\n');
fprintf('Srednia ilosc iteracji: %d\n', M10_1(1));
fprintf('Sredni czas wykonania: %g\n', M10_1(2));
fprintf('Macierze 20x20\n');
fprintf('Srednia ilosc iteracji: %d\n', M20_1(1));
fprintf('Sredni czas wykonania: %g\n', M20_1(2));
fprintf('---------------------------------------\n');
fprintf('PRECYZJA: 0.01\n');
fprintf('Macierze 5x5\n');
fprintf('Srednia ilosc iteracji: %d\n', M5_2(1));
fprintf('Sredni czas wykonania: %g\n', M5_2(2));
fprintf('Macierze 10x10\n');
fprintf('Srednia ilosc iteracji: %d\n', M10_2(1));
fprintf('Sredni czas wykonania: %g\n', M10_2(2));
fprintf('Macierze 20x20\n');
fprintf('Srednia ilosc iteracji: %d\n', M20_2(1));
fprintf('Sredni czas wykonania: %g\n', M20_2(2));
fprintf('---------------------------------------\n');
fprintf('---------------------------------------\n');
fprintf('MACIERZ SYMETRYCZNA, Z PRZESUNIECIEM\n');
fprintf('---------------------------------------\n');
fprintf('PRECYZJA: 0.00001\n');
fprintf('Macierze 5x5\n');
fprintf('Srednia ilosc iteracji: %d\n', M5_3(1));
fprintf('Sredni czas wykonania: %g\n', M5_3(2));
fprintf('Macierze 10x10\n');
fprintf('Srednia ilosc iteracji: %d\n', M10_3(1));
fprintf('Sredni czas wykonania: %g\n', M10_3(2));
fprintf('Macierze 20x20\n');
fprintf('Srednia ilosc iteracji: %d\n', M20_3(1));
fprintf('Sredni czas wykonania: %g\n', M20_3(2));
fprintf('---------------------------------------\n');
fprintf('PRECYZJA: 0.0000001\n');
fprintf('Macierze 5x5\n');
fprintf('Srednia ilosc iteracji: %d\n', M5_4(1));
fprintf('Sredni czas wykonania: %g\n', M5_4(2));
fprintf('Macierze 10x10\n');
fprintf('Srednia ilosc iteracji: %d\n', M10_4(1));
fprintf('Sredni czas wykonania: %g\n', M10_4(2));
fprintf('Macierze 20x20\n');
fprintf('Srednia ilosc iteracji: %d\n', M20_4(1));
fprintf('Sredni czas wykonania: %g\n', M20_4(2));
fprintf('---------------------------------------\n');
fprintf('---------------------------------------\n');
fprintf('MACIERZ NIESYMETRYCZNA, Z PRZESUNIECIEM\n');
fprintf('---------------------------------------\n');
fprintf('PRECYZJA: 0.00001\n');
fprintf('Macierze 5x5\n');
```

```
fprintf('Srednia ilosc iteracji: %d\n', M5_5(1));
fprintf('Sredni czas wykonania: %g\n', M5_5(2));
fprintf('Macierze 10x10\n');
fprintf('Srednia ilosc iteracji: %d\n', M10_5(1));
fprintf('Sredni czas wykonania: %g\n', M10_5(2));
fprintf('Macierze 20x20\n');
fprintf('Srednia ilosc iteracji: %d\n', M20_5(1));
fprintf('Sredni czas wykonania: %g\n', M20_5(2));
fprintf('-------------------------------------\n');
fprintf('PRECYZJA: 0.0000001\n');
fprintf('Macierze 5x5\n');
fprintf('Srednia ilosc iteracji: %d\n', M5_6(1));
fprintf('Sredni czas wykonania: %g\n', M5_6(2));
fprintf('Macierze 10x10\n');
fprintf('Srednia ilosc iteracji: %d\n', M10_6(1));
fprintf('Sredni czas wykonania: %g\n', M10_6(2));
fprintf('Macierze 20x20\n');
fprintf('Srednia ilosc iteracji: %d\n', M20_6(1));
fprintf('Sredni czas wykonania: %g\n', M20_6(2));
fprintf('-------------------------------------\n');

if id > 0
    fprintf(id, '-------------------------------------\n');
    fprintf(id, '-------------------------------------\n');
    fprintf(id, 'MACIERZ SYMETRYCZNA, BEZ PRZESUNIEC\n');
    fprintf(id, '-------------------------------------\n');
    fprintf(id, 'PRECYZJA: 0.1\n');
    fprintf(id, 'Macierze 5x5\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M5_1(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M5_1(2));
    fprintf(id, 'Macierze 10x10\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M10_1(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M10_1(2));
    fprintf(id, 'Macierze 20x20\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M20_1(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M20_1(2));
    fprintf(id, '-------------------------------------\n');
    fprintf(id, 'PRECYZJA: 0.01\n');
    fprintf(id, 'Macierze 5x5\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M5_2(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M5_2(2));
    fprintf(id, 'Macierze 10x10\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M10_2(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M10_2(2));
    fprintf(id, 'Macierze 20x20\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M20_2(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M20_2(2));
    fprintf(id, '-------------------------------------\n');
    fprintf(id, '-------------------------------------\n');
    fprintf(id, 'MACIERZ SYMETRYCZNA, Z PRZESUNIECIEM\n');
    fprintf(id, '-------------------------------------\n');
    fprintf(id, 'PRECYZJA: 0.00001\n');
    fprintf(id, 'Macierze 5x5\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M5_3(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M5_3(2));
    fprintf(id, 'Macierze 10x10\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M10_3(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M10_3(2));
    fprintf(id, 'Macierze 20x20\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M20_3(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M20_3(2));
```

```
    fprintf(id, '-------------------------------------\n');
    fprintf(id, 'PRECYZJA: 0.0000001\n');
    fprintf(id, 'Macierze 5x5\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M5_4(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M5_4(2));
    fprintf(id, 'Macierze 10x10\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M10_4(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M10_4(2));
    fprintf(id, 'Macierze 20x20\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M20_4(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M20_4(2));
    fprintf(id, '-------------------------------------\n');
    fprintf(id, '-------------------------------------\n');
    fprintf(id, 'MACIERZ NIESYMETRYCZNA, Z PRZESUNIECIEM\n');
    fprintf(id, '-------------------------------------\n');
    fprintf(id, 'PRECYZJA: 0.00001\n');
    fprintf(id, 'Macierze 5x5\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M5_5(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M5_5(2));
    fprintf(id, 'Macierze 10x10\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M10_5(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M10_5(2));
    fprintf(id, 'Macierze 20x20\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M20_5(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M20_5(2));
    fprintf(id, '-------------------------------------\n');
    fprintf(id, 'PRECYZJA: 0.0000001\n');
    fprintf(id, 'Macierze 5x5\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M5_6(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M5_6(2));
    fprintf(id, 'Macierze 10x10\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M10_6(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M10_6(2));
    fprintf(id, 'Macierze 20x20\n');
    fprintf(id, 'Srednia ilosc iteracji: %d\n', M20_6(1));
    fprintf(id, 'Sredni czas wykonania: %g\n', M20_6(2));
    fprintf(id, '-------------------------------------\n');
end

% porownanie przykladowych wynikow
A = getRandomMatrix(7,1);
WA1 = getEigenvalNoShiftQR(A,0.00001,intmax);
WA2 = eig(A);

B = getRandomMatrix(7,1);
WB1 = getEigenvalShiftQR(B,0.00001,intmax);
WB2 = eig(B);

C = getRandomMatrix(7,0);
WC1 = getEigenvalShiftQR(C,0.00001,intmax);
WC2 = eig(C);

fprintf('-------------------------------------\n');
fprintf('-------------------------------------\n');
fprintf('POROWNANIE WYNIKOW LOSOWYCH MACIERZY:\n');
fprintf('-------------------------------------\n');
fprintf('Macierz symetryczna, bez przesuniec\n');
fprintf('Wynik zaimplementowanej funkcji:\n');
s = size(WA1,1);
for i = 1 : s
    fprintf('%f\t',WA1(i));
```

```
end
fprintf('\nWynik wywolania eig:\n');
s = size(WA2,1);
for i = 1 : s
    fprintf('%f\t',WA2(i));
end
fprintf('\n---------------------------------------\n');
fprintf('Macierz symetryczna, z przesunieciami\n');
fprintf('Wynik zaimplementowanej funkcji:\n');
s = size(WB1,1);
for i = 1 : s
    fprintf('%f\t',WB1(i));
end
fprintf('\nWynik wywolania eig:\n');
s = size(WB2,1);
for i = 1 : s
    fprintf('%f\t',WB2(i));
end
fprintf('\n---------------------------------------\n');
fprintf('Macierz niesymetryczna, z przesunieciami\n');
fprintf('Wynik zaimplementowanej funkcji:\n');
s = size(WC1,1);
for i = 1 : s
    fprintf('%f\t',WC1(i));
end
fprintf('\nWynik wywolania eig:\n');
s = size(WC2,1);
for i = 1 : s
    fprintf('%f\t',WC2(i));
end
fprintf('\n---------------------------------------\n');

if id > 0
    fprintf(id, '---------------------------------------\n');
    fprintf(id, '---------------------------------------\n');
    fprintf(id, 'POROWNANIE WYNIKOW LOSOWYCH MACIERZY:\n');
    fprintf(id, '---------------------------------------\n');
    fprintf(id, 'Macierz symetryczna, bez przesuniec\n');
    fprintf(id, 'Wynik zaimplementowanej funkcji:\n');
    s = size(WA1,1);
    for i = 1 : s
        fprintf(id, '%f\t',WA1(i));
    end
    fprintf(id, '\nWynik wywolania eig:\n');
    s = size(WA2,1);
    for i = 1 : s
        fprintf(id, '%f\t',WA2(i));
    end
    fprintf(id, '\n---------------------------------------\n');
    fprintf(id, 'Macierz symetryczna, z przesunieciami\n');
    fprintf(id, 'Wynik zaimplementowanej funkcji:\n');
    s = size(WB1,1);
    for i = 1 : s
        fprintf(id, '%f\t',WB1(i));
    end
    fprintf(id, '\nWynik wywolania eig:\n');
    s = size(WB2,1);
    for i = 1 : s
        fprintf(id, '%f\t',WB2(i));
    end
    fprintf(id, '\n---------------------------------------\n');
```

```
        fprintf(id, 'Macierz niesymetryczna, z przesunieciami\n');
        fprintf(id, 'Wynik zaimplementowanej funkcji:\n');
        s = size(WC1,1);
        for i = 1 : s
            fprintf(id, '%f\t',WC1(i));
        end
        fprintf(id, '\nWynik wywolania eig:\n');
        s = size(WC2,1);
        for i = 1 : s
            fprintf(id, '%f\t',WC2(i));
        end
        fprintf(id, '\n---------------------------------------\n');
    end
    fclose(id);
end
```