MNUM-PROJEKT, zadanie 3.43 Łukasz Świtaj, 283777

Spis treści

Zadanie 1. Metody rozwiązywania równań nieliniowych z jedną niewiadomą 2

1. *metoda bisekcji* 4
2. *metoda siecznych* 6
3. *metoda Newtona (stycznych)*
4. porównanie

Zadanie 2. *Metoda Mullera MM1*  8

Załącznik 1. *Kod źródłowy zadania 1.* 15

Załącznik 2. *Kod źródłowy zadania 2.* 20

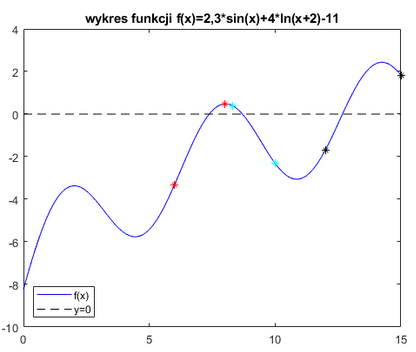
Zadanie 1. **Metody rozwiązywania równań nieliniowych z jedną niewiadomą**

Celem zadania jest napisanie programu obliczającego wszystkie zera funkcji  
*f(x) = 2.3\*sin(x)+4\*ln(x+2)–11* w przedziale [2, 12].

1. Wybór przedziałów startowych oraz ograniczeń.
2. Implementacja metod (oraz stworzenie warunków, w których minimum jedna z nich zawodzi):
   1. bisekcji
   2. siecznych
   3. stycznych (Newtona)
3. Porównanie zaimplementowanych metod.

Koncepcja rozwiązania

1. Początkowo napisana została metoda wspomagająca rysowanie wykresu funkcji (rysunek 1).



rysunek 1

Przedziały zaś zostały dobrane wg. dwóch kryteriów:

1. maksymalna wielkość
2. możliwość poprawnego wykonania każdej metody dla zadanej funkcji w zadanym przedziale.

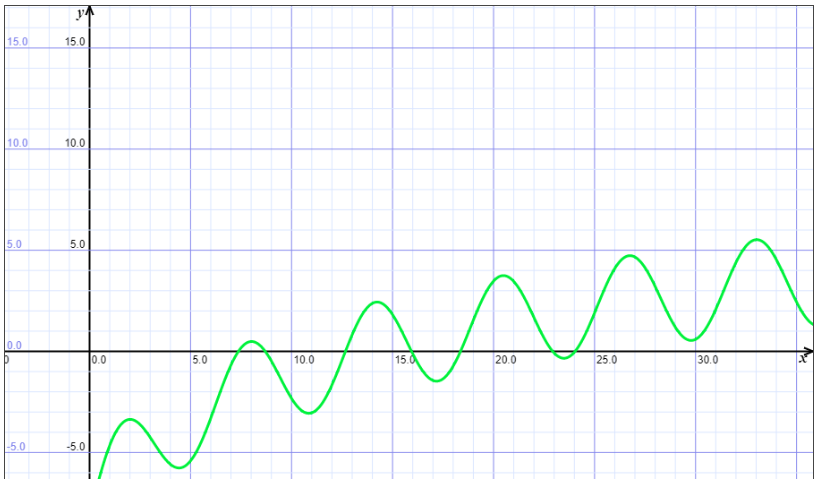
Ostateczne przedziały startowe zostały wybrane *metodą inżynierskiej intuicji* oraz *prób i błędów* jak przedstawiono na rysunku 1 - [6;8], [8.3;10], [12; 15].

Ograniczenia:

* **Każda z metod** została ograniczona parametrem globalnym **dokladnosc\_zer** określającym maksymalny moduł liczby mogącej być uznawanej w przybliżeniu za równą 0.
* **Metoda bisekcji oraz siecznych** zostały również ograniczone parametrem globalnym **wielkosc\_przedzialu**, aby zapobiegać przypadkom, w którym dla funkcji o małym nachyleniu pierwiastki zostaną wyznaczone niedokładnie.
* **Metoda stycznych (Newtona)** została ograniczona ze względu na maksymalną liczbę iteracji - **ilosc\_iteracji**.
* **Metoda stycznych (Newtona)** zostaje przerwana gdy któryś z kolejno wyznaczonych punktów wychodzi poza początkowy przedział – funkcja **nowy\_przedzial\_sieczny** chroni przed pochodną o zbyt małym nachyleniu zgłaszając błąd w niedozwolonym przypadku.

Sprawdzenie

1. Do sprawdzenia poprawności wyznaczonego wykresu użyty został generator wykresów ze strony:  
   <matemaks.pl/program-do-rysowania-wykresow-funkcji.html>



rysunek 2

Wygenerowany został rysunek 2 pokrywający się z rysunkiem 1.

Komentarz

1. W celu zautomatyzowania wyboru przedziałów startowych dla metody bisekcji oraz siecznych można napisać funkcję **wybierz\_przedziały** działającą wg. listy kroków:
   1. Podziel badany przedział na 100 części – małych przedziałów. Przedziały te powinny nachodzić na siebie z dokładnością do epsilona, tak aby nie było przypadku, w którym jeden kończy się na miejscu zerowym, a drugi na nim zaczyna (przypadek nieobsłużony przez metody).
   2. Dla każdego małego przedziału przeprowadź test **sprawdzenie\_przedziału** sprawdzający warunek f(x1)\*f(x2)<0.
   3. W przypadku spełnienia warunku dopisz go do wektora przedziałów, w których znajduje się pierwiastek funkcji.
   4. Na podstawie utworzonego wektora oblicz miejsca zerowe.

W projekcie metoda ta nie została zaimplementowana ze względu na wymóg doboru szerokich przedziałów startowych.

Zadanie 1a. **Metoda bisekcji**

Koncepcja rozwiązania

Zaimplementowana została klasyczna metoda bisekcji:

1. Z przedziału [x1,x2] wyznacz punkt c = (x1+x2)/2.
2. Jeśli c jest miejscem zerowym oraz przedział jest odpowiednio mały zwróć c.
3. Jeśli nie za pomocą metody sprawdź\_przedział wybierz [x1,c] lub [c,x2], w którym znajduje się miejsce zerowe.
4. Powróć do kroku i.

Sprawdzenie

Dla parametrów dokladnosc\_zer=0.001 oraz wielkosc\_przedzialu=0.1.

metoda bisekcji przedzial nr 1 ([6;8])

x=7.000000 y=-0.700033

x=7.500000 y=0.162567

x=7.250000 y=-0.208420

x=7.375000 y=-0.006645

x=7.437500 y=0.082156

x=7.406250 y=0.038790

x=7.390625 y=0.016329

x=7.382813 y=0.004906

x=7.378906 y=-0.000853

metoda bisekcji przedzial nr 2 ([8.3;10])

x=9.150000 y=-0.730176

x=8.725000 y=-0.028380

x=8.512500 y=0.229330

x=8.618750 y=0.110034

x=8.671875 y=0.043095

x=8.698438 y=0.007909

x=8.711719 y=-0.010100

x=8.705078 y=-0.001061

x=8.701758 y=0.003432

x=8.703418 y=0.001187

x=8.704248 y=0.000064

metoda bisekcji przedzial nr 3 (12;15)

x=13.500000 y=1.812064

x=12.750000 y=0.184950

x=12.375000 y=-0.775508

x=12.562500 y=-0.295103

x=12.656250 y=-0.054088

x=12.703125 y=0.065796

x=12.679688 y=0.005930

x=12.667969 y=-0.024062

x=12.673828 y=-0.009061

x=12.676758 y=-0.001564

x=12.678223 y=0.002183

x=12.677490 y=0.000310

Komentarz

1. Metoda bisekcji działa zgodnie z oczekiwaniami – w każdym kroku widać, że o połowę zmniejsza zadany w danej iteracji przedział. Co ciekawe jest to metoda, która w każdym kolejnym kroku nie zawsze zmniejsza wartość y na co przykładem jest przedział nr 1.

Warunki, w których metoda nie działa

1. Metoda bisekcji jest najprostszą metodą, która może najłatwiej zawieźć przez błąd programisty – zły dobór przedziału startowego (niespełniający warunku f(x1)\*f(x2)<0. Gdy uruchomiłem program właśnie dla takiego przedziału zapętlił on po ok. 20 iteracjach jeden wynik.

Przypuszczałem, że metoda bisekcji może zawieźć gdy przedziałem startowym będzie [x1 pierwiastek\_funkcji] zaś dokładność\_zer będzie mała. Uruchomiłem więc następującą funkcję:

x\_bisekcja = bisekcja([0 7.3794839], 0.000001, wielkosc\_przedzialu)

output

ilosc iteracji 24 x\_zero 7.379483e+00f, wartosc -3.645547e-07f

Jak widać metoda bisekcji nawet w takim przypadku szybko zbiega do poprawnego wyniku.

Zadanie 1b. **Metoda siecznych**

Koncepcja rozwiązania

Zaimplementowana została metoda siecznych w postaci:

1. Na podstawie przedziału [x1,x2] wyznacz funkcję liniową zawierającą te punkty.
2. Wyznacz miejsce zerowe tej funkcji korzystając z wzoru:

c = (x2\*y1-x1\*y2)/(y1-y2);

1. Jeśli c jest miejscem zerowym oraz przedział jest odpowiednio mały zwróć c.
2. Jeśli nie za pomocą metody sprawdź\_przedział wybierz [x1,c] lub [c,x2], w którym znajduje się miejsce zerowe.
3. Powróć do kroku i.

Sprawdzenie

Dla parametrów dokladnosc\_zer=0.001 oraz wielkosc\_przedzialu=0.1.

metoda bisekcji przedzial nr 1 ([6;8])

x=7.000000 y=-0.700033

x=7.500000 y=0.162567

x=7.250000 y=-0.208420

x=7.375000 y=-0.006645

x=7.437500 y=0.082156

x=7.406250 y=0.038790

x=7.390625 y=0.016329

x=7.382813 y=0.004906

x=7.378906 y=-0.000853

metoda bisekcji przedzial nr 2 ([8.3;10])

x=9.150000 y=-0.730176

x=8.725000 y=-0.028380

x=8.512500 y=0.229330

x=8.618750 y=0.110034

x=8.671875 y=0.043095

x=8.698438 y=0.007909

x=8.711719 y=-0.010100

x=8.705078 y=-0.001061

x=8.701758 y=0.003432

x=8.703418 y=0.001187

x=8.704248 y=0.000064

metoda bisekcji przedzial nr 3 (12;15)

x=13.500000 y=1.812064

x=12.750000 y=0.184950

x=12.375000 y=-0.775508

x=12.562500 y=-0.295103

x=12.656250 y=-0.054088

x=12.703125 y=0.065796

x=12.679688 y=0.005930

x=12.667969 y=-0.024062

x=12.673828 y=-0.009061

x=12.676758 y=-0.001564

x=12.678223 y=0.002183

x=12.677490 y=0.000310

Komentarz

1. Metoda bisekcji działa zgodnie z oczekiwaniami – w każdym kroku widać, że o połowę zmniejsza zadany w danej iteracji przedział. Co ciekawe jest to metoda, która w każdym kolejnym kroku nie zawsze zmniejsza wartość y na co przykładem jest przedział nr 1.

Warunki, w których metoda nie działa

1. Metoda bisekcji jest najprostszą metodą, która może najłatwiej zawieźć przez błąd programisty – zły dobór przedziału startowego (niespełniający warunku f(x1)\*f(x2)<0. Gdy uruchomiłem program właśnie dla takiego przedziału zapętlił on po ok. 20 iteracjach jeden wynik.

Przypuszczałem, że metoda bisekcji może zawieźć gdy przedziałem startowym będzie [x1 pierwiastek\_funkcji] zaś dokładność\_zer będzie mała. Uruchomiłem więc następującą funkcję:

x\_bisekcja = bisekcja([0 7.3794839], 0.000001, wielkosc\_przedzialu)

output

ilosc iteracji 24 x\_zero 7.379483e+00f, wartosc -3.645547e-07f

Jak widać metoda bisekcji nawet w takim przypadku szybko zbiega do poprawnego wyniku.

Zadanie 2. **Aproksymacja funkcji**

Celem zadania jest napisanie programu, który będzie aproksymował wielomianową funkcję na podstawie zadanych punktów dwiema metodami:

1. układu równań normalnych
2. układu równań liniowych wynikającego z rozkładu QR

Ponadto dla każdego układu należy obliczyć błąd rozwiązania jako normę residuum.

Koncepcja rozwiązania

Dla zestawu (x,y):

dane = [-5 -5.4606;-4 -3.8804;-3 -1.9699;-2 -1.6666;-1 -0.0764;0 -0.3971;1 -1.0303;2 -4.5483;3 -11.528;4 -21.6417;5 -34.4458];

wyznaczone zostały aproksymacje na podstawie obu metod. Wykorzystane do tego zostały funkcje, które za parametr przyjmują wektor x, y oraz stopień wielomianu, który ma służyć jako przybliżenie funkcji tworzącej dane.

1. Algorytm układu równań normalnych:
   1. wyznaczenie macierzy (*G*)Grama jako iloczynu przekształceń – w tym przypadku sumowanie potęg x
   2. wyznaczenie macierzy prawej strony (*P*) jako *przekształcenie\*korespondująca wartość wyjściowa (y)*
   3. obliczenie równania *GX = P.*komentarz: pozwoliłem sobie użyć wbudowanej w Matlaba funkcji \, gdyż przy poprzednim zadaniu samodzielnie pisałem funkcję obliczającą równania tego typu.
   4. potraktuj wyjście jako zbiór współczynników kolejnych potęg x.
2. Algorytm oparty na rozkładzie QR różni się jedynie tym, że w kroku 3 nie zostaje wykonane obliczenie równania *GX = P* tylko *Rx=QTP.*

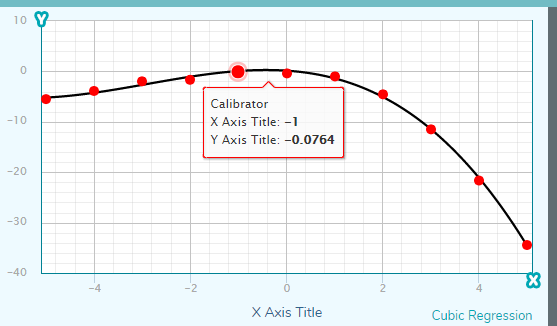
Komentarz: do dokonania rozkładu QR została wykorzystana napisana przeze mnie metoda rozkładu QR z poprzedniego zadania.

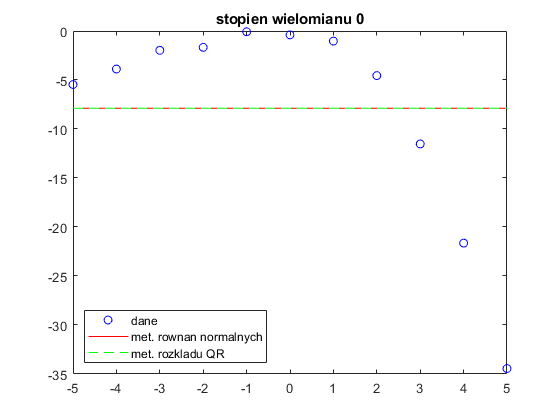
Sprawdzenie

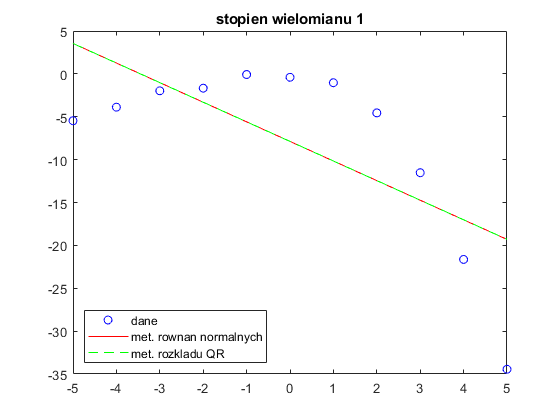
W celu sprawdzenia poprawności wykreowanych rozwiązań skorzystałem z:

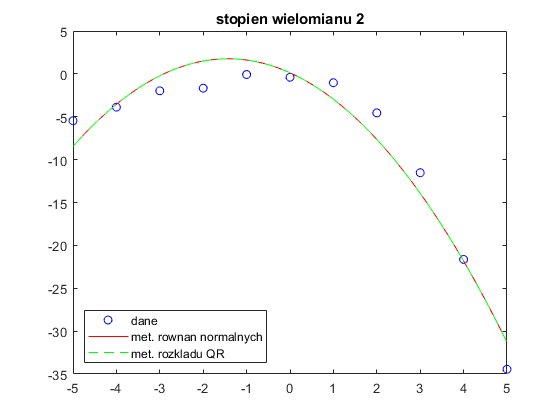
- efekt finalny: obliczania normy residuum

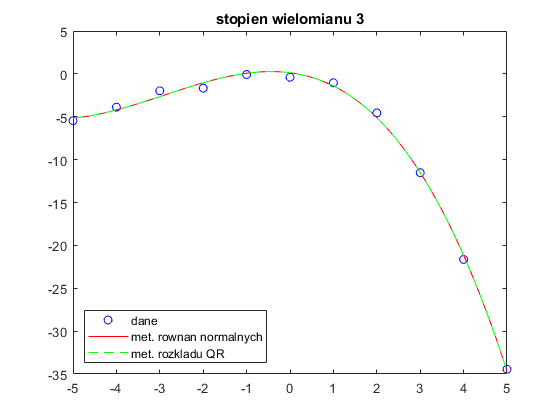
- poprawności rysowanych wykresów: strona, która na podstawie wyjścia generowanego przez mój program tworzyła wykresy, np. [https://www.wolframalpha.com/input/?i=-0.09-0.8\*x-0.65\*x%5E2%2B0.13\*x%5E3](https://www.wolframalpha.com/input/?i=-0.09-0.8*x-0.65*x%5E2%2B0.13*x%5E3)

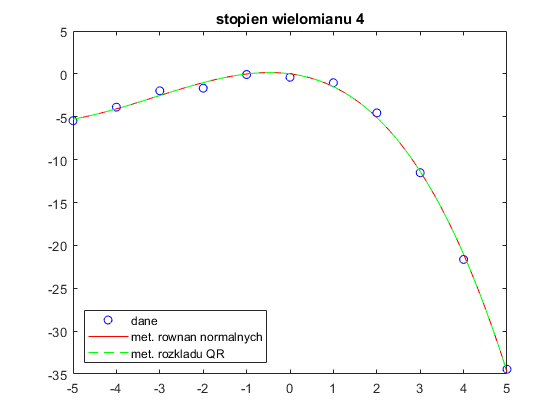
- dokładność przybliżenia: na podstawie danych weryfikowałem czy wykres jest dość dokładny jak na zadane wejście za pomocą <https://mycurvefit.com/> (przykład generowany przez stronę dla 3 stopnia)

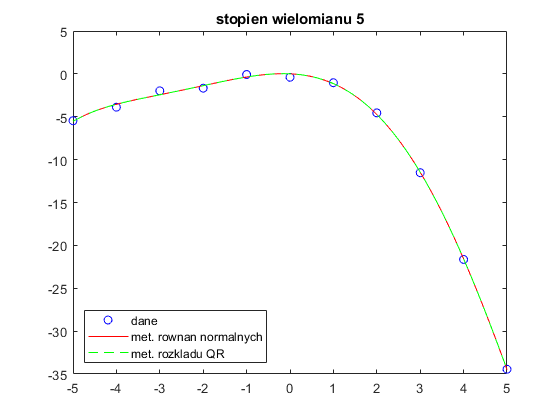


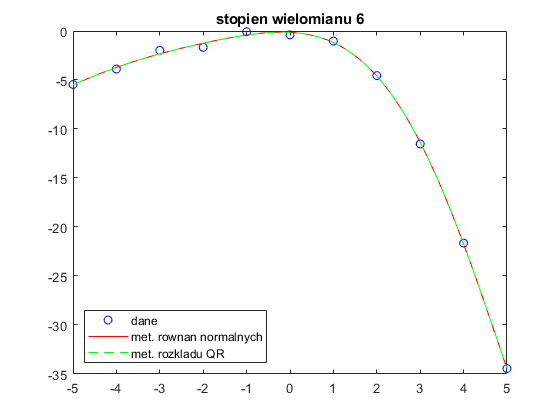


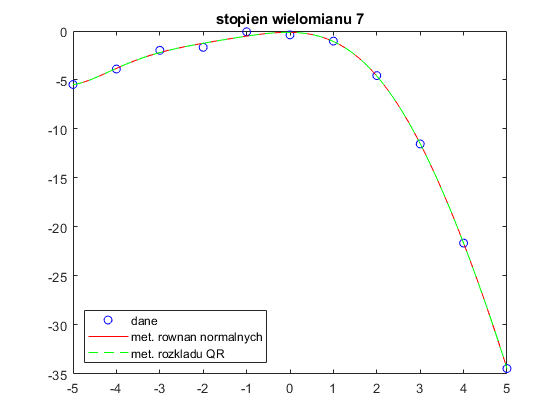


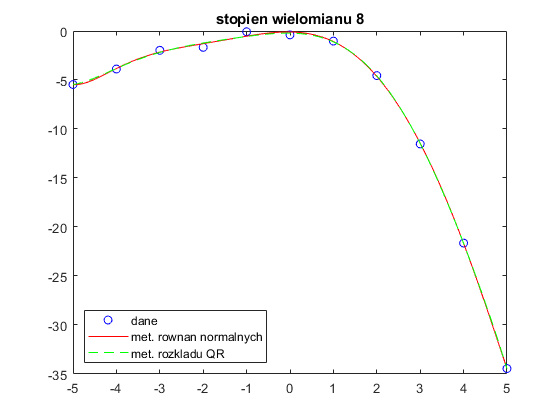












norma\_res = 9×2

**met. rown norm met QR**

34.3326 34.3326 – stopień 0

24.5832 24.5832 – stopień 1

7.3647 7.3647 – stopień 2

1.4390 1.4390 – stopień 3

1.3958 1.3958 – stopień 4

0.8501 0.8501 – stopień 5

0.7595 0.7595 – stopień 6

0.7069 0.7069 – stopień 7

0.6997 0.7181 – stopień 8

Komentarz

Obie metody dobrze dokonują aproksymacji dla zadanego zestawu danych już przy 3 stopniu wielomianu. Do 7. stopnia wielomianu różnice pomiędzy obiema metodami są wręcz niezauważalne – zarówno wykresy jak i norma residuum są takie same. Przy większym stopniu metoda QR traci nieznacznie na rzecz metody równań normalnych. Wynikać to może ze stosowania dodatkowego rozkładu, który przy większych macierzach nieco traci na dokładności – może to dziać się np. na etapie ortogonalizacji Grama-Schmidta gdyż występuje coraz większe pole do popełnienia błędów np. w elemencie sumy.

Załącznik 1. **Kod źródłowy zadania 1.**

program

clc;

clear;

iteracje\_sym\_bezprzes = zeros(30);

iteracje\_sym\_przes = zeros(30);

cond\_sym = zeros(30);

cond\_niesym = zeros(30);

iteracje\_niesym\_przes = zeros(30);

for rozmiar = [5 10 20]

for i = 1:30

A = macierz\_symetryczna(rozmiar);

[B iteracje\_sym\_bezprzes(i,rozmiar)] = qr\_bezprzesuniec(A);

[B iteracje\_sym\_przes( i,rozmiar)] = qr\_przesuniecia(A);

cond\_sym(i,rozmiar) = cond(A);

%eig(A);

A = macierz\_niesymetryczna(rozmiar);

[B iteracje\_niesym\_przes(i,rozmiar)] = qr\_przesuniecia(A);

cond\_niesym(i,rozmiar) = cond(A);

end

end

statystyki i czyszczenie danych

for rozmiar = [5 10 20]

rozmiar

sr\_sym\_bezprzes = iteracje\_sym\_bezprzes(:,rozmiar)

sr\_sym\_przes = iteracje\_sym\_przes(:,rozmiar)

sr\_niesym\_przes = iteracje\_niesym\_przes(:,rozmiar)

for i = 1:3

[wart indx] = max(cond\_sym(:,rozmiar));

iteracje\_sym\_bezprzes(indx,rozmiar) = 0;

iteracje\_sym\_przes(indx,rozmiar) = 0;

cond\_sym(indx,rozmiar) = 0;

[wart indx] = max(cond\_niesym(:,rozmiar));

iteracje\_niesym\_przes(indx,rozmiar) = 0;

cond\_niesym(indx,rozmiar) = 0;

end

end

for rozmiar = [5 10 20]

rozmiar

sr\_sym\_bezprzes = mean(iteracje\_sym\_bezprzes(:,rozmiar))

sr\_sym\_przes = mean(iteracje\_sym\_przes(:,rozmiar))

sr\_niesym\_przes = mean(iteracje\_niesym\_przes(:,rozmiar))

end

wykresy

for rozmiar = [5 10 20]

figure

plot(cond\_sym(:,rozmiar),iteracje\_sym\_bezprzes(:,rozmiar),'bo', cond\_sym(:,rozmiar), iteracje\_sym\_przes(:,rozmiar), 'r\*');

title(['Macierz symetryczna o rozmiarze ' num2str(rozmiar)])

xlabel('wskaznik uwarunkowania')

ylabel('ilosc potrzebnych iteracji')

legend({'bez przesuniec','z przesunieciami'},'Location','northeast');

figure

plot(cond\_niesym(:,rozmiar),iteracje\_niesym\_przes(:,rozmiar),'go');

title(['Macierz niesymetryczna o rozmiarze ' num2str(rozmiar)])

xlabel('wskaznik uwarunkowania')

ylabel('ilosc potrzebnych iteracji')

legend({'z przesunieciami'},'Location','northeast');

end

**funkcje pomocnicze**

rozklad QR

function [Q R] = qr\_rozklad(A)

[r\_wiersze r\_kolumny] = size(A);

Q = zeros(r\_wiersze);

if r\_wiersze > r\_kolumny

R = eye(r\_wiersze);

Q = eye(r\_wiersze);

else

R = eye(r\_kolumny);

Q = eye(r\_wiersze);

end

%Gram-Schmidt

for i = 1:r\_kolumny

Q(:,i) = A(:,i);

for j = 1:(i-1)

R(j,i) = mydot(Q(:,j),A(:,i))/mydot(Q(:,j),Q(:,j));

Q(:,i) = Q(:,i) - R(j,i)\*Q(:,j);

end

end

Q = Q(1:r\_wiersze,1:r\_kolumny);

%normalizacja

N = zeros(r\_wiersze);

for i = 1:r\_kolumny

N(i,i) = norm(Q(:,i));

Q(:,i) = Q(:,i)/N(i,i);

end

R = N\*R;

if r\_wiersze > r\_kolumny

R = R(1:r\_kolumny,1:r\_kolumny);

else

R = R(1:r\_wiersze,1:r\_wiersze);

end

end

algorytm obliczania wartosci wlasnych metoda QR bez przesuniec

function [wart\_wlasne i] = qr\_bezprzesuniec(A)

i = 0;

while tolerancja(A) > 0.00001 & i < 200+1

[Q R] = qr\_rozklad(A);

A = R \* Q;

i = i+1;

end

wart\_wlasne = wektor(A);

end

algorytm obliczania wartosci wlasnych metoda QR z przesunieciami

function [wart\_wlasne i] = qr\_przesuniecia(A)

rozmiar = size(A,1);

i = 0;

wart\_wlasne = zeros(rozmiar);

wart\_wlasne = wart\_wlasne(:,1);

for j = rozmiar:-1:2

while max(abs(A(j,1:j-1))) > 0.00001 & i < 200+1

mala\_macierz = A(j-1:j,j-1:j); % macierz 2x2,

[x1 x2] = pierw\_f\_kwadratowej(mala\_macierz);

przesuniecie = blizsza\_liczba(mala\_macierz(2,2), x1, x2); % z ktorej wyznaczana jest najlepsza wart. wlasna

A = A - eye(j)\*przesuniecie;

[Q R] = qr\_rozklad(A);

A = R \* Q + eye(j)\*przesuniecie;

i = i+1;

end

wart\_wlasne(j) = A(j,j);

if j > 2

A = A(1:j-1,1:j-1); %deflacja

else

wart\_wlasne(1) = A(1,1);

end

end

end

wyznaczanie pierw f. kwadratowej

function [x1 x2] = pierw\_f\_kwadratowej(mala\_macierz)

a = 1;

b = -(mala\_macierz(1,1)+mala\_macierz(2,2));

c = (mala\_macierz(1,1)\*mala\_macierz(2,2))-(mala\_macierz(2,1)\*mala\_macierz(1,2));

x1 = (-b + sqrt(b\*b - 4\*a\*c))/(2\*a);

x2 = (-b - sqrt(b\*b - 4\*a\*c))/(2\*a);

if abs(x2) > abs(x1)

x1 = x2;

end

%drugi pierwiastek ze wzorów Viete'a

x2 = ((-b)/a) - x1;

end

wybor pierwiastka blizszego d(n,n)

function x = blizsza\_liczba(wlasciwa, x1, x2)

if abs(wlasciwa-x1) < abs(wlasciwa-x2)

x = x1;

else

x = x2;

end

end

autorska implementacja matlabowej funkcji dot()

function md = mydot(A,B)

rozmiar = size(A);

md = 0;

for i = 1:rozmiar

md = md + A(i)\*B(i);

end

end

funkcja wektoryzujaca macierz diagonalna

function w = wektor(A)

rozmiar = size(A);

for i = 1:rozmiar

w(i,1) = A(i,i);

end

end

sprawdzenie tolerancji

function tol = tolerancja(A)

rozmiar = size(A);

A = abs(A);

tol = 0;

if rozmiar > 2

for i = 1:rozmiar

if max(A(i,i+1:end)) > tol

tol = max(A(i,i+1:end));

end

if max(A(i,1:i-1)) > tol

tol = max(A(i,1:i-1));

end

end

else

tol = 0;

end

end

tworzenie macierzy symetrycznej o zadanym rozmiarze

function mac\_sym = macierz\_symetryczna(rozmiar)

mac\_sym = randi([0 50],rozmiar,rozmiar);

mac\_sym = mac\_sym + mac\_sym';

end

tworzenie macierzy niesymetrycznej o zadanym rozmiarze

function mac\_nsym = macierz\_niesymetryczna(rozmiar)

mac\_nsym = randi([0 100],rozmiar,rozmiar);

end

norma residuum

function nr = norma\_residuum(wspolczynniki, x, rozw)

residuum = wspolczynniki\*x - rozw;

nr = norm(residuum);

end

Załącznik 2. **Kod źródłowy zadania 2.**

program

clc;

clear;

dane = [-5 -5.4606;-4 -3.8804;-3 -1.9699;-2 -1.6666;-1 -0.0764;0 -0.3971;1 -1.0303;2 -4.5483;3 -11.528;4 -21.6417;5 -34.4458];

x = linspace(-5,5,100);

max\_stopien = 8;

norma\_res = zeros(max\_stopien);

norma\_res = norma\_res(:,1:2);

for stopien = 0:max\_stopien

figure

funkcja = uklad\_rownan\_normalnych(dane, stopien);

y = fun(funkcja, x);

funkcja2 = uklad\_qr(dane, stopien);

y2 = fun(funkcja2, x);

plot(dane(:,1),dane(:,2),'bo', x, y, 'r', x, y2, 'g--');

legend({'dane','met. rownan normalnych','met. rozkladu QR'},'Location','southwest');

title(['stopien wielomianu ' num2str(stopien)]);

norma\_res(stopien+1,1) = norm(dane(:,2) - fun(funkcja, dane(:,1)));

norma\_res(stopien+1,2) = norm(dane(:,2) - fun(funkcja2, dane(:,1)));

end

norma\_res

funkcje pomocnicze

uklad rownan normalnych

function wspolczynniki = uklad\_rownan\_normalnych(dane, st\_wielomianu)

% wyznaczanie macierzy Grama - <przeksztalcenie\_i,przeksztalcenie\_j>

[r\_wiersze, r\_kolumny] = size(dane);

st\_wielomianu = st\_wielomianu + 1;

macierz\_Grama = wyzn\_macierz\_Grama(dane, st\_wielomianu);

% wektor prawej strony

prawa\_strona = zeros(st\_wielomianu);

prawa\_strona = prawa\_strona(:,1);

for i = 1:st\_wielomianu

for k = 1:r\_wiersze

prawa\_strona(i) = prawa\_strona(i) + (dane(k,1))^(i-1)\*dane(k,2);

end

end

wspolczynniki = macierz\_Grama\prawa\_strona;

end

uklad wynikajacy z rozkladu QR

function wspolczynniki = uklad\_qr(dane, st\_wielomianu)

% wyznaczanie macierzy Grama - <przeksztalcenie\_i,przeksztalcenie\_j>

[r\_wiersze, r\_kolumny] = size(dane);

st\_wielomianu = st\_wielomianu + 1;

macierz\_Grama = wyzn\_macierz\_Grama(dane, st\_wielomianu);

% wektor prawej strony

prawa\_strona = zeros(st\_wielomianu);

prawa\_strona = prawa\_strona(:,1);

for i = 1:st\_wielomianu

for k = 1:r\_wiersze

prawa\_strona(i) = prawa\_strona(i) + (dane(k,1))^(i-1)\*dane(k,2);

end

end

[Q R] = qr\_rozklad(macierz\_Grama);

wspolczynniki = R\Q'\*prawa\_strona;

end

wyznaczenie macierzy Grama

function macierz\_Grama = wyzn\_macierz\_Grama(dane, st\_wielomianu)

% wyznaczanie macierzy Grama - <przeksztalcenie\_i,przeksztalcenie\_j>

macierz\_Grama = zeros(st\_wielomianu);

[r\_wiersze, r\_kolumny] = size(dane);

for i = 1:st\_wielomianu

for j = 1:st\_wielomianu

for k = 1:r\_wiersze

macierz\_Grama(i,j) = macierz\_Grama(i,j) + (dane(k,1))^(i+j-2);

end

end

end

end

wyznaczenie wyjsc dla podanych x-ow i zadanej funkcji

function y = fun(funkcja, x)

[temp rozmiar\_x] = size(x);

if rozmiar\_x == 1

x = x';

rozmiar\_x = temp;

end

st\_wielomianu = size(funkcja);

y = zeros(rozmiar\_x);

y = y(:,1);

for i = 1:rozmiar\_x

for j = 1:st\_wielomianu

y(i,1) = y(i,1) + funkcja(j,1)\*(x(1,i))^(j-1);

end

end

end

rozklad QR

function [Q R] = qr\_rozklad(A)

[r\_wiersze r\_kolumny] = size(A);

Q = zeros(r\_wiersze);

if r\_wiersze > r\_kolumny

R = eye(r\_wiersze);

Q = eye(r\_wiersze);

else

R = eye(r\_kolumny);

Q = eye(r\_wiersze);

end

%Gram-Schmidt

for i = 1:r\_kolumny

Q(:,i) = A(:,i);

for j = 1:(i-1)

R(j,i) = mydot(Q(:,j),A(:,i))/mydot(Q(:,j),Q(:,j));

Q(:,i) = Q(:,i) - R(j,i)\*Q(:,j);

end

end

Q = Q(1:r\_wiersze,1:r\_kolumny);

%normalizacja

N = zeros(r\_wiersze);

for i = 1:r\_kolumny

N(i,i) = norm(Q(:,i));

Q(:,i) = Q(:,i)/N(i,i);

end

R = N\*R;

if r\_wiersze > r\_kolumny

R = R(1:r\_kolumny,1:r\_kolumny);

else

R = R(1:r\_wiersze,1:r\_wiersze);

end

end

autorska implementacja matlabowej funkcji dot()

function md = mydot(A,B)

rozmiar = size(A);

md = 0;

for i = 1:rozmiar

md = md + A(i)\*B(i);

end

end

funkcja wektoryzujaca macierz diagonalna

function w = wektor(A)

rozmiar = size(A);

for i = 1:rozmiar

w(i,1) = A(i,i);

end

end