MNUM-PROJEKT, zadanie 2.43 Łukasz Świtaj, 283777

Spis treści

Zadanie 1. **Metoda QR obliczania wartości własnych**

Celem zadania jest:

1. napisanie programu obliczającego wartości własne macierzy metodą rozkładu QR:
    a. z przesunięciami
    b. bez przesunięć
2. przetestowanie napisanych funkcji na 30 macierzach kwadratowych o rozmiarze 5, 10 i 20
3. wyznaczenie średniej liczby iteracji potrzebnej do uzyskania wyniku o zadanej dokładności.

Teoria

**

Koncepcja rozwiązania

**

Sprawdzenie

**

Komentarz

**

Zadanie 2. **Aproksymacja funkcji**

Celem zadania jest napisanie programu, który będzie aproksymował 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.

Teoria

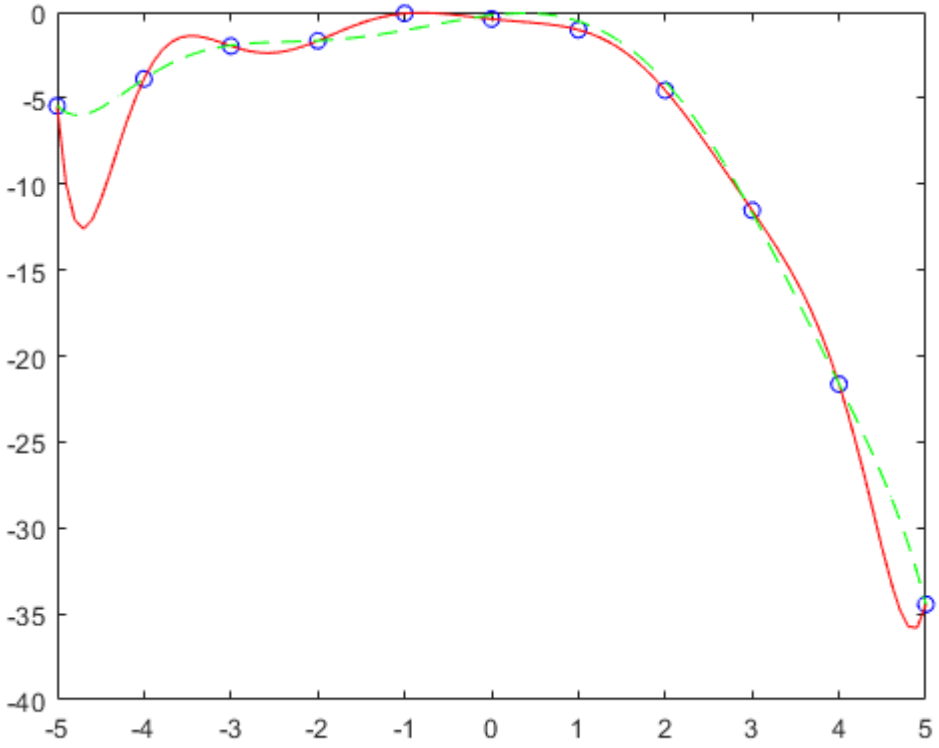
**

Koncepcja rozwiązania

**

Sprawdzenie

Przybliżenie funkcji generowane dla 10-stopnia – czerwona to układ rownan normalnych, zielona to QR.



Komentarz

**

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

program

```
clc;
clear;
% for rozmiar = [5 10 20]
%     for i = 1:30
%         macierz_symetryczna(rozmiar);
%         macierz_niesymetryczna(rozmiar);
%     end
% end
A = [1 1;2 -1; -2 4];
%A = [1 1 2; -1 -2 4];
A = macierz_symetryczna(5);
A
% [q r] = qr_rozklad(A)
% [Q R] = qr(A)
eig(A)
[B i] = qr_bezprzesuniec(A)
[B i] = qr_przesuniecia(A)
```

wyswietlenie bledow

```
% blad_res_a
```

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 < 1000+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 < 1000+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];
funkcja = uklad_rownan_normalnych(dane, 10)
x = linspace(-5,5,100);
y = fun(funkcja, x);
funkcja2 = uklad_qr(dane, 10)
y2 = fun(funkcja2, x);
plot(dane(:,1),dane(:,2),'bo', x, y, 'r', x, y2, 'g--')
%A
```

wyswietlenie bledow

```
% blad_res_a
```

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);
    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
```

sprawdzenie tolerancji

```
function tol = tolerancja(A)
    rozmiar = size(A);
    A = abs(A);
    tol = 0;
    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
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
```

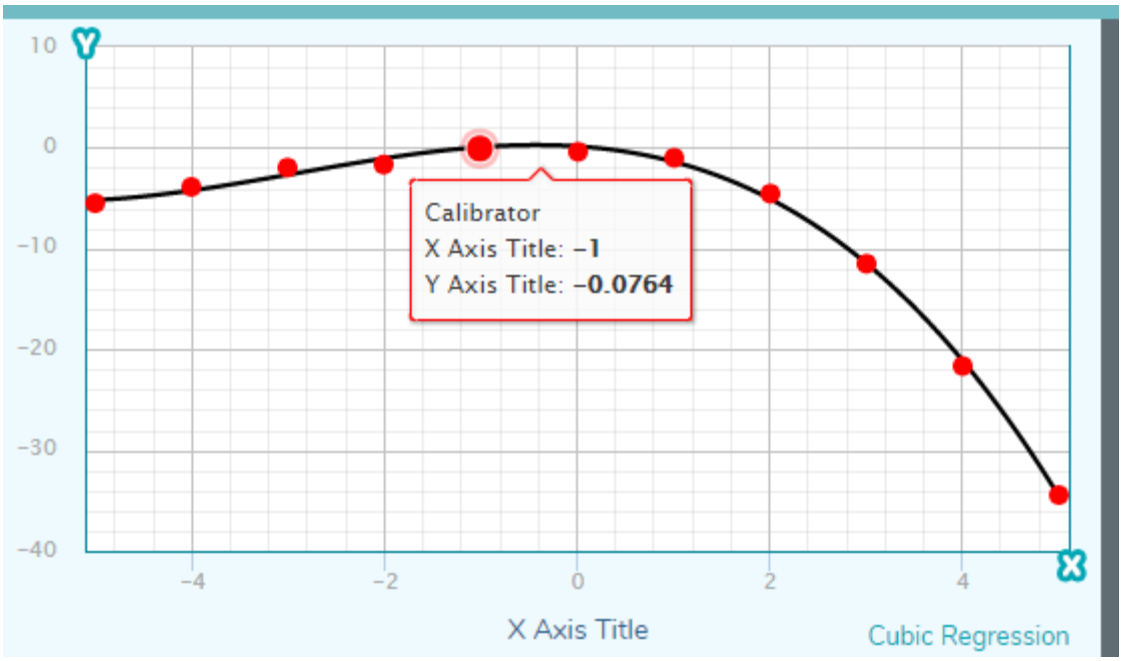
norma residuum

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

do sprawdzenia poprawności:

https://www.wolframalpha.com/input/?i=-0.09-0.8*x-0.65*x%5E2%2B0.13*x%5E3

oraz https://mycurvefit.com/



sprawdzana dokladnosc przy trzecim stopniu