MNUM-PROJEKT, zadanie 2.43 Łukasz Świtaj, 283777

Spis treści

Zadanie 1. M*etoda QR obliczania wartości własnych* 2

Zadanie 2. *Aproksymacja funkcji*  3

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

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

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

Celem zadania jest:

1. napisanie programu obliczającego wartości własne macierzy metodą rozkładu QR:
   1. z przesunięciami
   2. 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.

Koncepcja rozwiązania

1. Do wykonania rozkładu QR macierzy wykorzystana została ortogonalizacja Grama-Schmidta. Algorytm samego rozkładu można opisać następująco:
2. Na podstawie macierzy wejściowej wyznacz rozmiar macierzy Q oraz R.
3. Dokonaj ortogonalizacji Grama-Schmidta(algorytm opisany w poprzednim projekcie). Q - macierz zortogonalizowana, R – macierz współczynników.
4. Znormalizuj otrzymane macierze Q oraz R.
   1. Algorytm wyznaczania wartości własnych bez przesunięć:
      1. Dopóki nie zostanie osiągnięta **tolerancja (0,00001) lub maksymalna liczba iteracji (200)** dopóty
      2. Dokonaj rozkładu macierzy wejściowej (A) na macierz *Q* i *R*.
   2. Algorytm wyznaczania wartości własnych z przesunięciami:
      1. wykonuj do momentu uzyskania macierzy 2x2
      2. wyznacz macierz dolnego prawego rogu (2x2) - *M*
      3. oblicz wartości własne macierzy *M*
      4. wybierz wartość własną bliższą elementowi prawego dolnego rogu (*przesunięcie*)
      5. Dokonaj rozkładu QR na macierzy *A – I\*przesunięcie*
      6. odtwórz macierz A jako *R\*Q + I\*przesunięcie*
      7. po osiągnięciu wymaganej dokładności zapisz otrzymaną wartość własną (dolny prawy róg)
      8. dokonaj deflacji macierzy
5. Została napisana funkcja tworząca macierze nie-/symetryczne o zadanym wymiarze i losowych wartościach. Dla ułatwienia zadania wartości losowane są liczbami całkowitymi z przedziału 0-100. Do sprawdzenia została użyta wbudowana matlabowa funkcja *qr()* wyznaczająca rozkład QR z dokładnością co do znaku oraz funkcja *eig()* wyznaczająca wektor wartości własnych.
6. Prócz wyznaczenia średniej liczby iteracji potrzebnej do uzyskania wyniku o zadanej dokładności sprawdzona została z ciekawości potencjalna korelacja ilości potrzebnych iteracji do uwarunkowania macierzy.

Sprawdzenie

\*\*

Komentarz

\*\*

Zadanie 2. **Aproksymacja funkcji**

Celem zadania jest napisanie programu, który będzie aproksymował wilomianową 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):

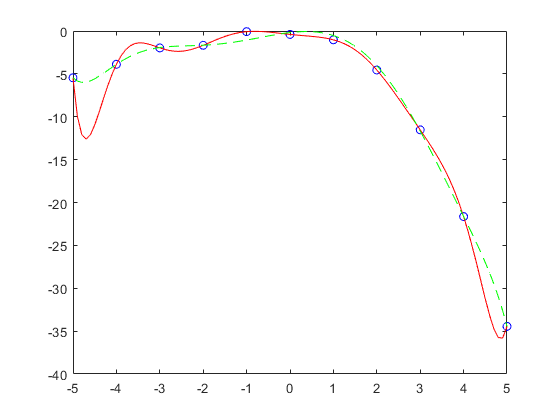
dane = [-5 -5.4606;-4 -3.8804;-3 -1.9699;-2 -1.6666;-1 -0.0764;0 -0.3971;1 -1.0303;2 -4.5483;3 -11.528;4 -21.6417;5 -34.4458];

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.

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;

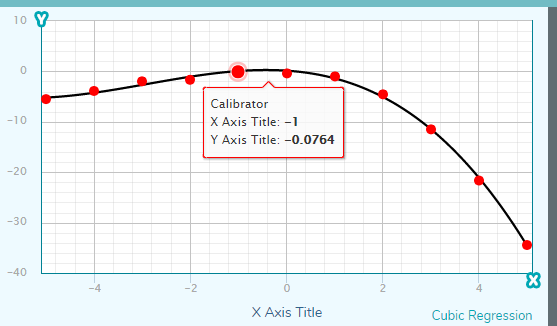
nr = norm(residuum);

end

do sprawdzenia poprawności:

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