**MNUM – PROJEKT, zadanie 2.15**

**Wojciech Rokicki gr. AR2**

**Zadanie 1**  
Obliczanie wartości własnych macierzy nieosobliwych metodą rozkładu QR w wersji z przesunięciami oraz bez przesunięć

Idea pojedynczego kroku metody bez przesunięć (przekształcenie do ):

Ponieważ jest ortogonalna, więc

Macierz jest macierzą przekształconą przez podobieństwo, więc ma te same wartości własne. Można pokazać, że dla macierzy symetrycznej , macierz będzie zbiegać do macierzy diagonalnej .

Algorytm metody z przesunięciami

Za przyjmuje się bliższą wartość własną podmacierzy z prawego dolnego rogu macierzy . Po wyzerowaniu wszystkich elementów ostatniego wiersza poza ostatnim elementem (diagonalnym) postępujemy analogicznie z macierzą zmniejszoną do wymiarowości .

Algorytm metody QR bez przesunięć (EigvalQRNoShift.m):

function eigenvalues = EigvalQRNoShift(D,tol, imax)

%tol - tolerancja (górna granica wartosci) elementów zerowanych

%imax - maksymalna liczba iteracji

n=size(D, 1);

i=1;

while i <= imax && max(max(D-diag(diag(D)))) > tol

[Q1, R1]=qr(D);

D=R1\*Q1; %macierz przekszta?cona

i=i+1;

end

if i > imax

error('imax exceeded program terminated');

end

eigenvalues=diag(D);

disp(i);

end

Algorytm metody QR z przesunięciami (EigvalQRShifts.m):

function eigenvalues = EigvalQRShifts(A, tol, imax)

%tol - tolerancja (gorna granica wartosci) elementow zerowych

%imax - max liczba iteracji dla liczenia jednej wartosci wlasnej

n=size(A,1);

eigenvalues=diag(zeros(n));

INITIALsubmatrix=A; %macierz poczatkowa (oryginalna)

for k=n:-1:2

DK=INITIALsubmatrix; %macierz startowa dla jednej wartosci wlasnej

i=0;

while i <= imax && max(abs(DK(k,1:k-1))) > tol

DD=DK(k-1:k,k-1:k); %2x2 podmacierz prawego dolnego rogu

[ev1,ev2]=quadpolynroots(1,-(DD(1,1)+DD(2,2)),DD(2,2)\*DD(1,1)-DD(2,1)\*DD(1,2));

if abs(ev1-DD(2,2)) < abs(ev2-DD(2,2))

shift=ev1; %najblizsza DK(k,k) wartosc wlasna podmacierzy DD

else

shift=ev2; %najblizsza DK(k,k) wartosc wlasna podmacierzy DD

end

DK=DK-eye(k)\*shift; %macierz przesunieta

[Q1,R1]=qr(DK); %faktoryzacja QR

DK=R1\*Q1+eye(k)\*shift; %macierz przeksztalcona

i=i+1;

end

if i > imax

error('imax exceeded program terminated');

end

eigenvalues(k)=DK(k,k);

if k >2

INITIALsubmatrix=DK(1:k-1,1:k-1); %deflacja macierzy

else

eigenvalues(1)=DK(1,1); %ostatnia wartosc wlasna

end

end

end

Algorytm rozkładu QR - zmodyfikowany algorytm Grama-Shmidta (QRDecomposition.m):

function [Q,R] = QRDecomposition(A)

%Rozklad QR (waski) zmodyfikowanym alg. Grama-Schmidta dla

%macierzy mxn (m>=n) o rzedzie n, rzeczywistej lub zespolonej

[m, n]=size(A);

Q=zeros(m,n); R=zeros(n,n); d=zeros(1,n);

%rozklad z kolumnami Q ortogonalnymi (nie ortonormalnymi):

for i =1:n

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

R(i,i)=1;

d(i)=Q(:,i)'\*Q(:,i);

for j=i+1:n

R(i,j)=(Q(:,i)'\*A(:,j))/d(i);

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

end

end

%normowanie rozkladu (kolumny Q ortonomralne):

for i=1:n

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

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

R(i,i:n)=R(i,i:n)\*dd;

end

end

Kod algorytmu oblicznia pierwiastków równania kwadratowego (plik: quadpolynroots.m) :

function [ x1, x2 ] = quadpolynroots( a,b,c )

%quadpolynroots funkcja zwracajaca pierwistki wielomianu stopnia 2

% a,b,c - wspolczynniki wielomianu

l1 = -b + sqrt(b\*b - 4\*a\*c);

l2 = -b - sqrt(b\*b - 4\*a\*c);

%licznik o wiekszym module

if abs(l1) > abs(l2)

licznik = l1;

else

licznik = l2;

end

x1 = licznik/(2\*a);

%drugi pierwiastek (wzory Viete'a)

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

end

**Badane będzie 30 losowych macierzy o rozmiarach 5x5, 10x10 i 20x20. Maksymalna liczba iteracji (imax) została ustawiona na 10 000, tolerancja 0.00001.**

Rozmiar 5x5

W tym przypadku wszystkie metody były zbieżne.

|  |  |  |
| --- | --- | --- |
| Macierz i metoda | Średnia liczba iteracji | Liczba nieudanych prób |
| Macierz symetryczna  Algorytm bez przesunięć | 33.4333 | 0 |
| Macierz symetryczna  Algorytm z przesunięciami | 7.7667 | 0 |
| Macierz niesymetryczna  Algorytm z przesunięciami | 9.5667 | 0 |

Rozmiar 10x10

W jednym przypadku metoda bez przesunięć okazała się rozbieżna. Algorytm z przesunięciami był zbieżny dla wszystkich macierzy

|  |  |  |
| --- | --- | --- |
| Macierz i metoda | Średnia liczba iteracji | Liczba nieudanych prób |
| Macierz symetryczna  Algorytm bez przesunięć | 226.2333 | 0/1 (przy imax = 1000) |
| Macierz symetryczna  Algorytm z przesunięciami | 14.0667 | 0 |
| Macierz niesymetryczna  Algorytm z przesunięciami | 20.6333 | 0 |

Rozmiar 20x20

W jednym przypadku metoda bez przesunięć okazała się rozbieżna. Algorytm z przesunięciami był zbieżny dla wszystkich macierzy

|  |  |  |
| --- | --- | --- |
| Macierz i metoda | Średnia liczba iteracji | Liczba nieudanych prób |
| Macierz symetryczna  Algorytm bez przesunięć | 578.4000 | 0/1 (przy imax = 1000) |
| Macierz symetryczna  Algorytm z przesunięciami | 28.6667 | 0 |
| Macierz niesymetryczna  Algorytm z przesunięciami | 45.1333 | 0 |

**Wnioski – porównanie metod:**  
 Rozkład QR w wersji bez przesunięć wymaga znacznie większej liczby iteracji, nie będzie działał także w przypadku macierzy niesymetrycznych. Algorytm w wersji z przesunięciami wymaga mniejszego nakładu obliczeniowego, mimo większej złożoności każdego kroku, gdyż jest szybciej zbieżny. Jest on ponadto bardziej uniwersalny ze względu na obsługę macierzy niesymetrycznych.

**Wyniki – porównanie z funkcją eig():**  
Przetestowane zostały pojedyncze przypadkowe macierze o danych rozmiarach i wyniki powyższych algorytmów zostały porównane z wynikami otrzymanymi za pomocą wbudowanej funkcji eig(). W tabeli przedstawiono średnią różnic wartości własnych otrzymanych tymi dwoma sposobami

|  |  |  |  |
| --- | --- | --- | --- |
|  | 5x5 | 10x10 | 20x20 |
| Macierz symetryczna  Algorytm bez przesunięć | 1,56104379422928e-11 | 9,60175688957676e-12 | 3,69694691526590e-12 |
| Macierz symetryczna  Algorytm z przesunięciami | 1,26287869051112e-15 | 1,18366427770411e-12 | 6,32764327046509e-11 |
| Macierz niesymetryczna  Algorytm z przesunięciami | 2,85144689765145e-08 | 8,60442913801266e-08 | 1,00270065059626e-07 |

**Wnioski:**  
Jak widać rezultaty otrzymane za pomocą powyższych algorytmów można uznać za dokładne w stosunku do wbudowanej funkcji eig(). Dla macierzy niesymetrycznych uzyskujemy najmniejszą dokładność ze względu na występowanie wśród wartości własnych liczb zespolonych.

**Zadanie 2**  
Wyznaczanie metodą najmniejszych kwadratów funkcji wielomianowej najlepiej aproksymującej dane.

Definiując macierz jako macierz , gdzie N – ilość próbek, n – stopień wielomianu, dla której

Rozwiązanie zadania najmniejszych kwadratów polega na znalezieniu wektora zawierającego współczynniki wielomianu. W tym przypadku zrobimy to na dwa sposoby:

Układ równań normalnych:

Układ równań wynikający z rozkładu :

Algorytm aproksymujący (Approximation.m):

function [ a, res ] = Approximation( x, y, n, meth )

% n - stopien wielomianu

% meth : 1 - uklad rownan normalnych; 2 - qr

N = size(x,1);

A = zeros(N,n);

%Wypelniamy macierz A odpowiednimi potêgami x

for i=1:N

for j = 1:n

A(i,j) = x(i,1)^(j-1);

end

end

%uklad rownan normalnych

if meth == 1

ata = A'\*A;

aty = A'\*y;

a = ata\aty;

res = norm(aty - ata\*a);

%uklad wynikajacy z rozkladu QR

elseif meth == 2

[Q,R] = QRDecomposition(A);

a =R\Q'\*y;

res = norm(R\*a - Q'\*y);

end

end

Algorytm obliczający wartości wielomianu ( pval.m):

function [ w ] = pval(a , x)

% pval oblicza wartosci wielomianu o wspolczynnikach a w   
% punktach x (a(1) odpowiada x^0)

% w - wartosci wielomianu w danych punktach x

ilprobek = size(x,1);

stwiel = size(a,1);

w = zeros(ilprobek,1);

for i = 1: ilprobek

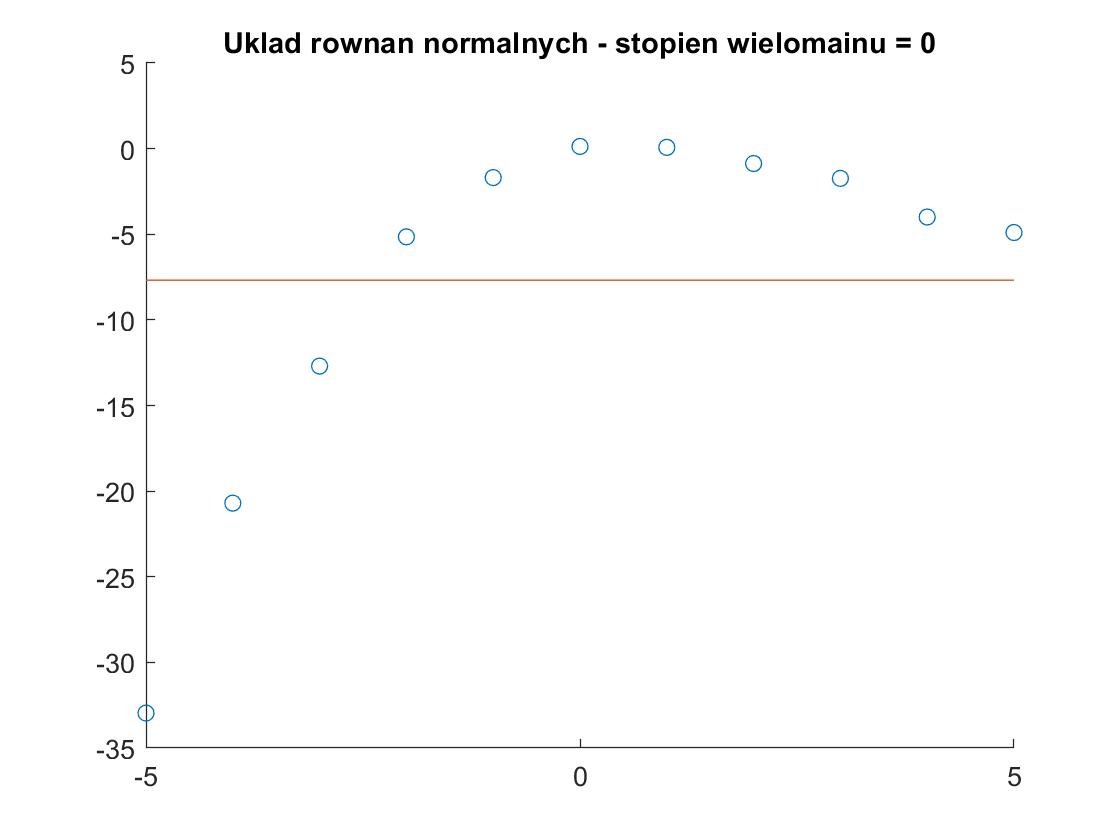
for j = 1:stwiel

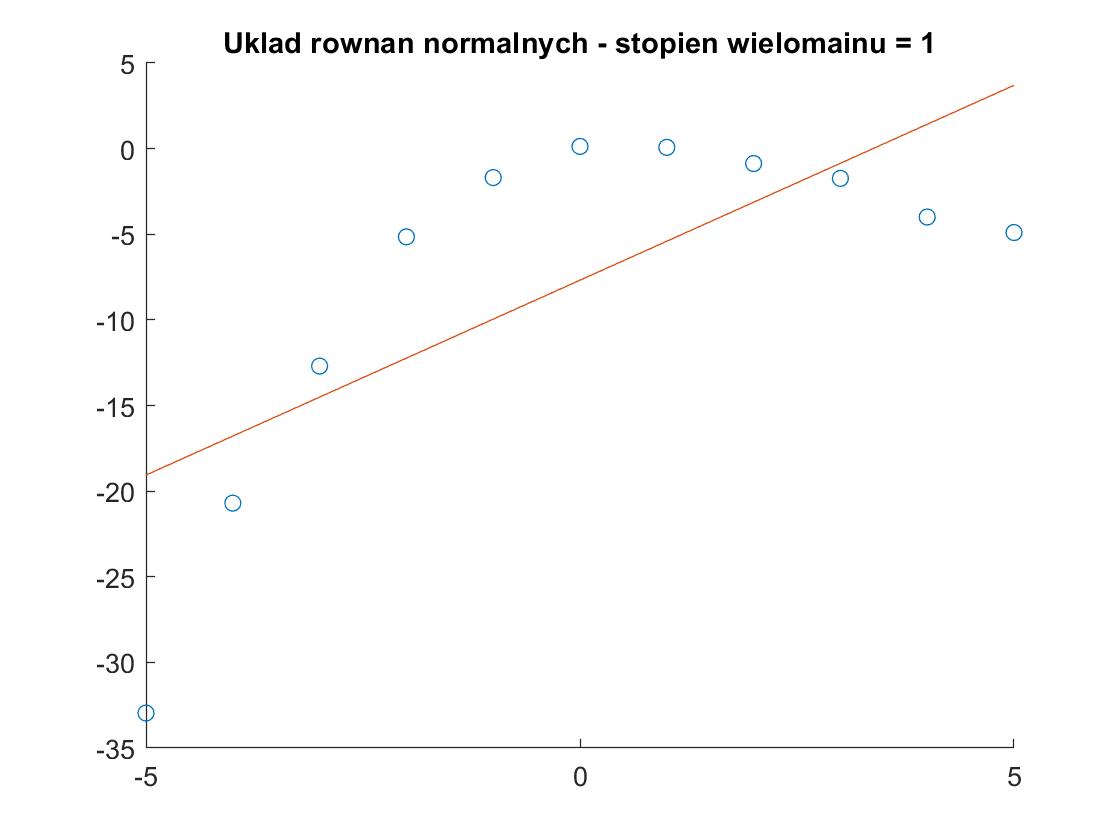
w(i) = w(i) + a(j) \* x(i)^(j-1);

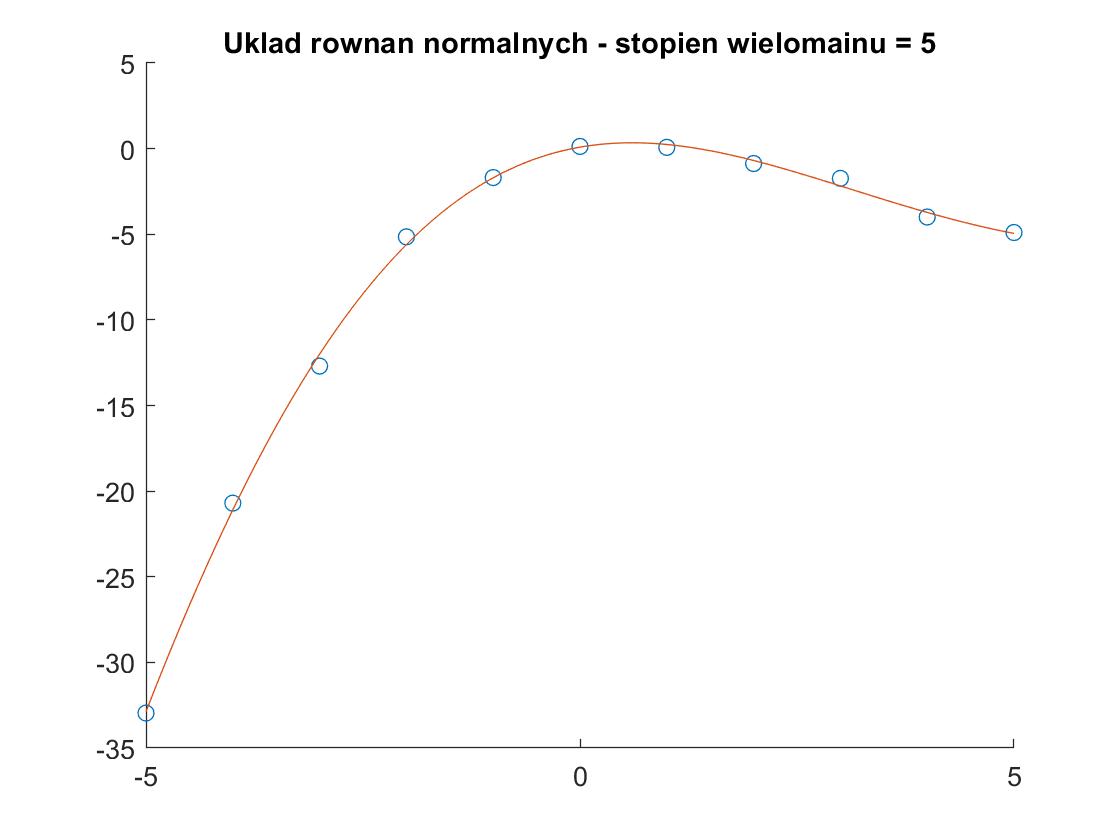
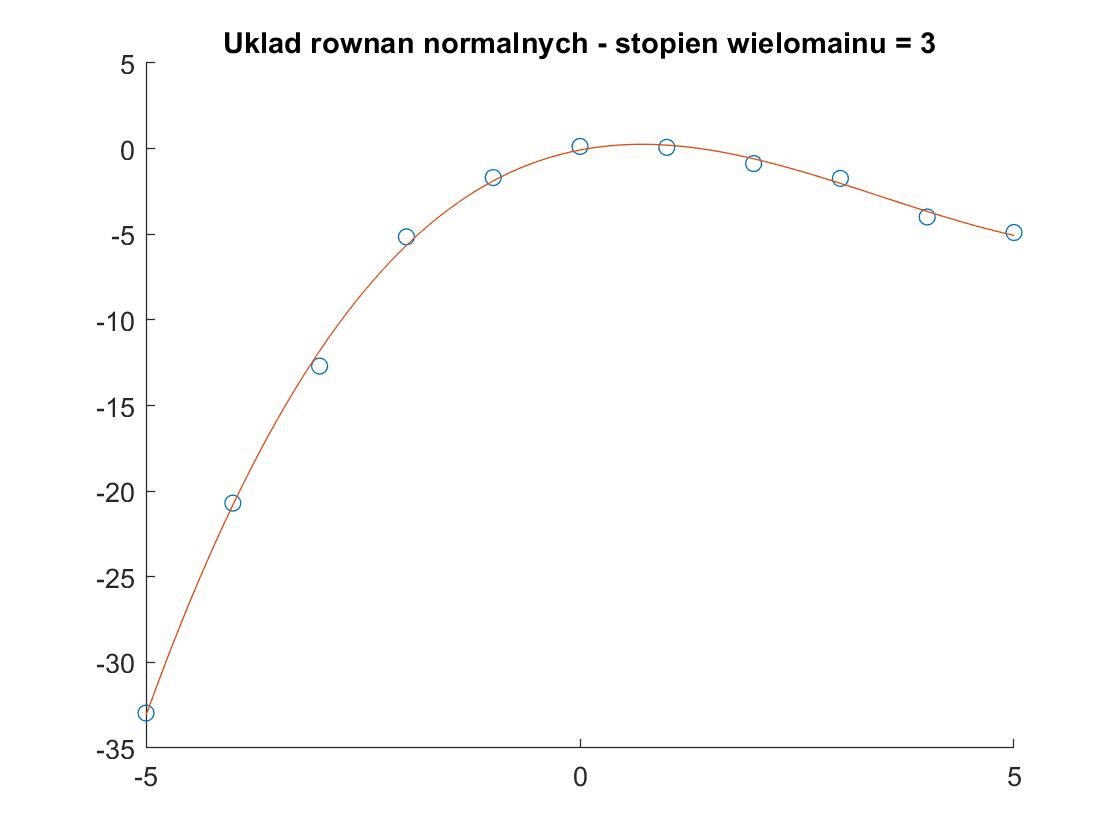
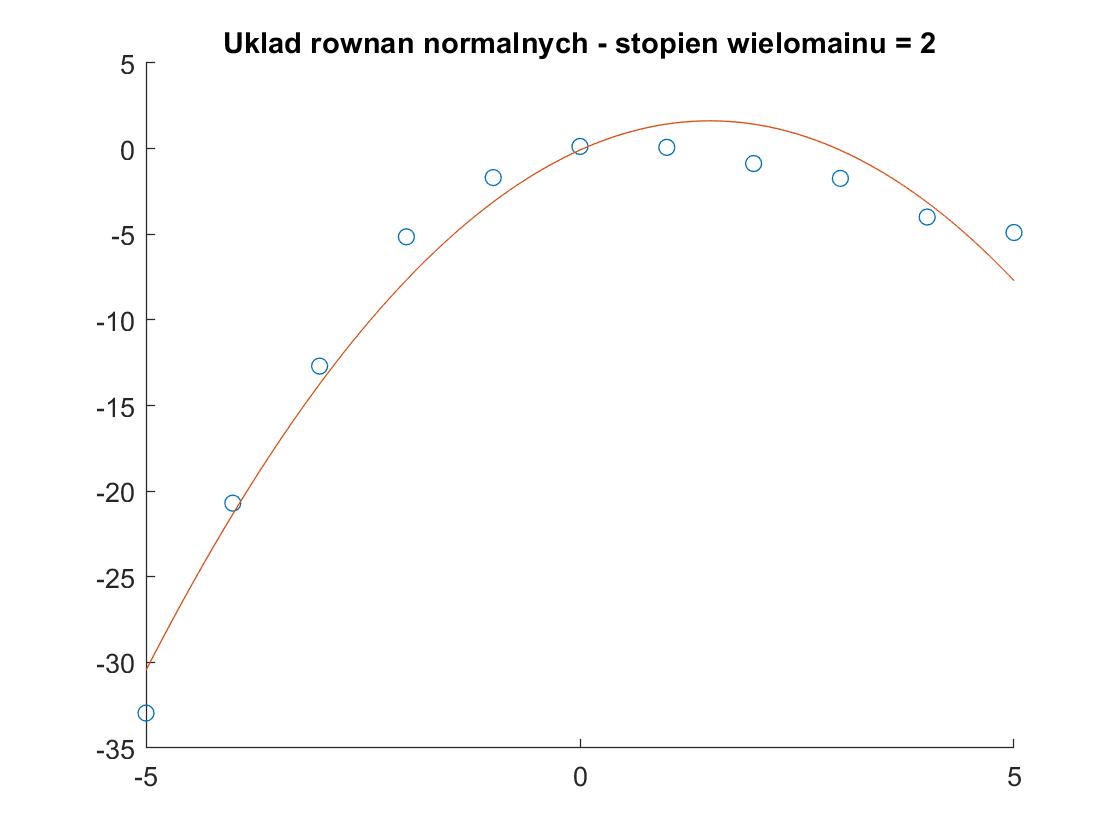
end

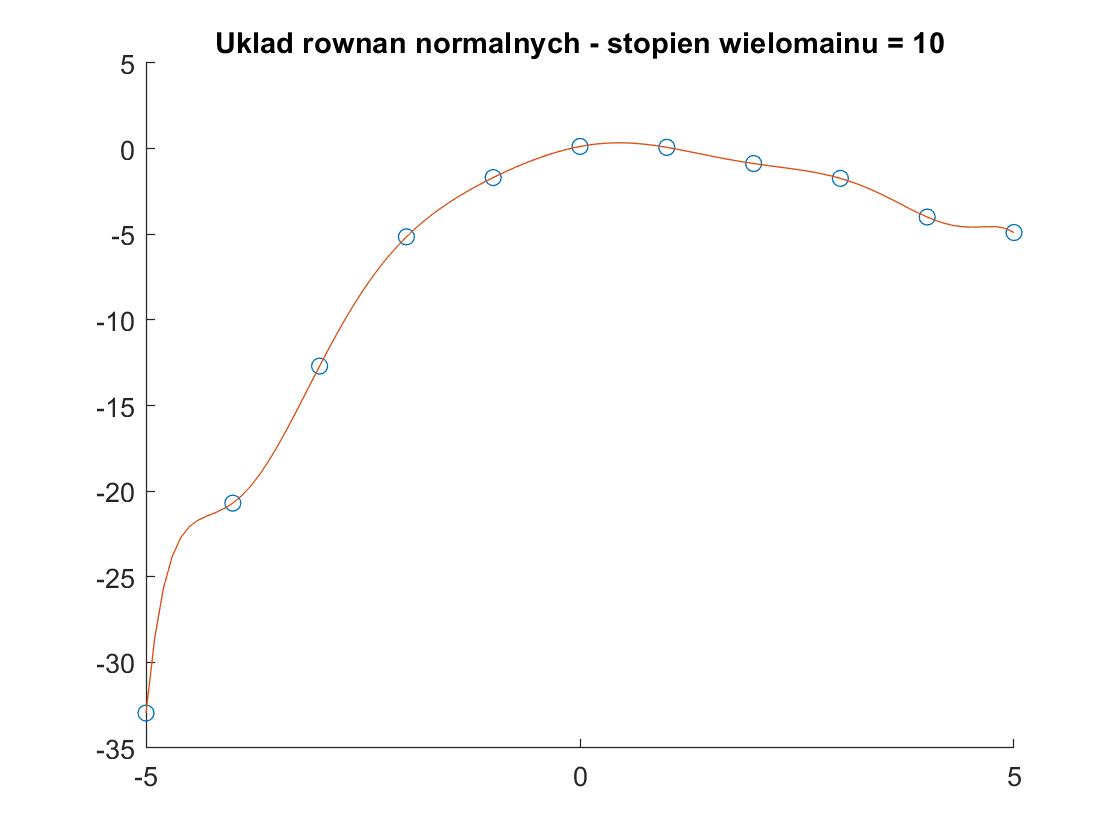
end

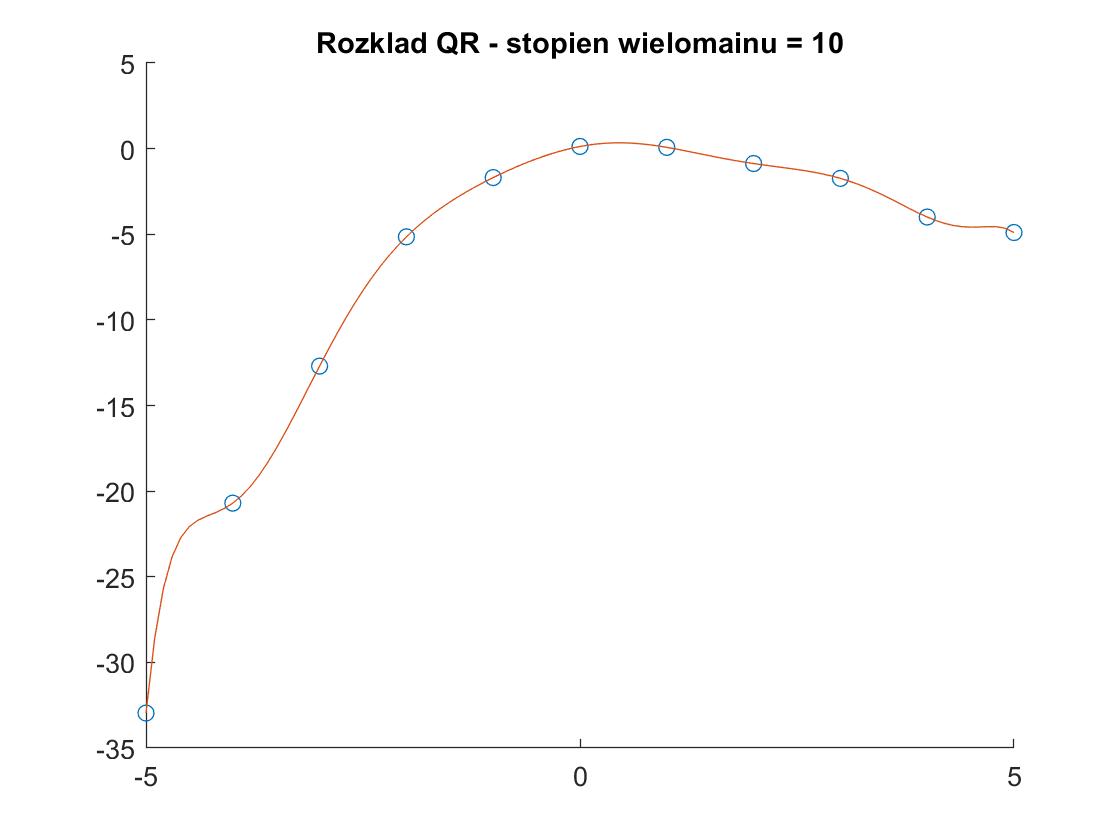
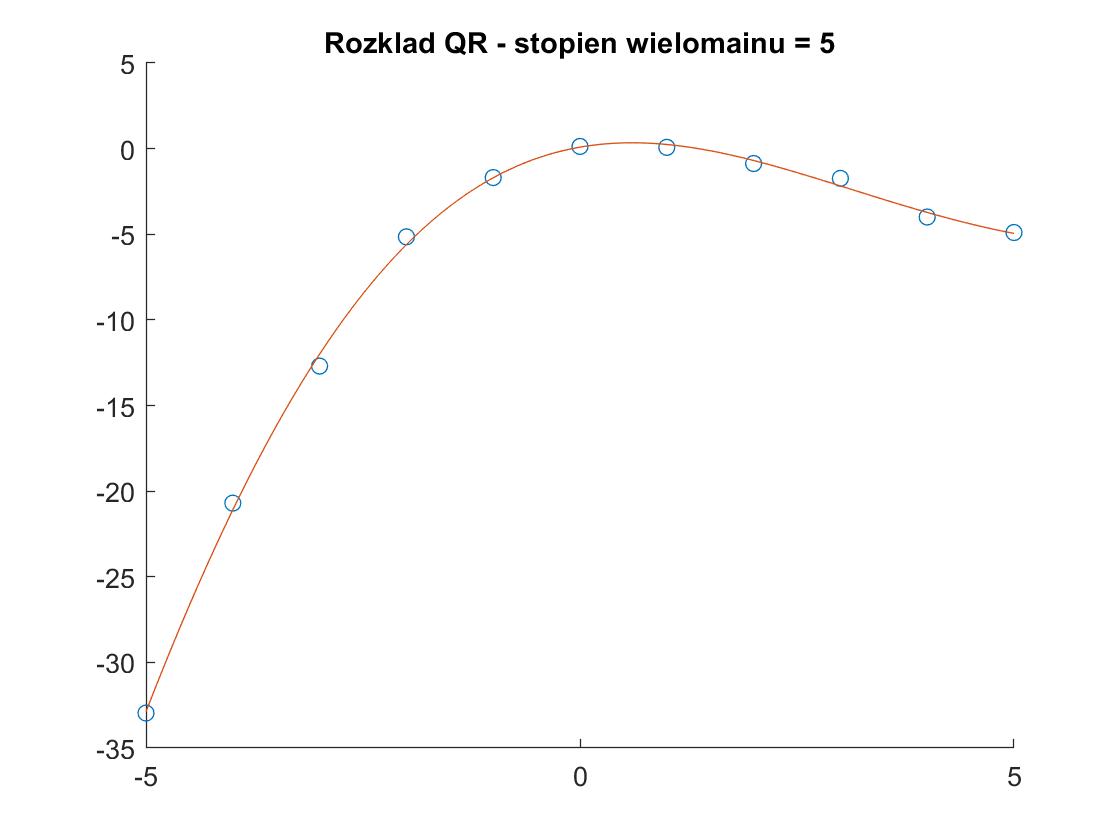
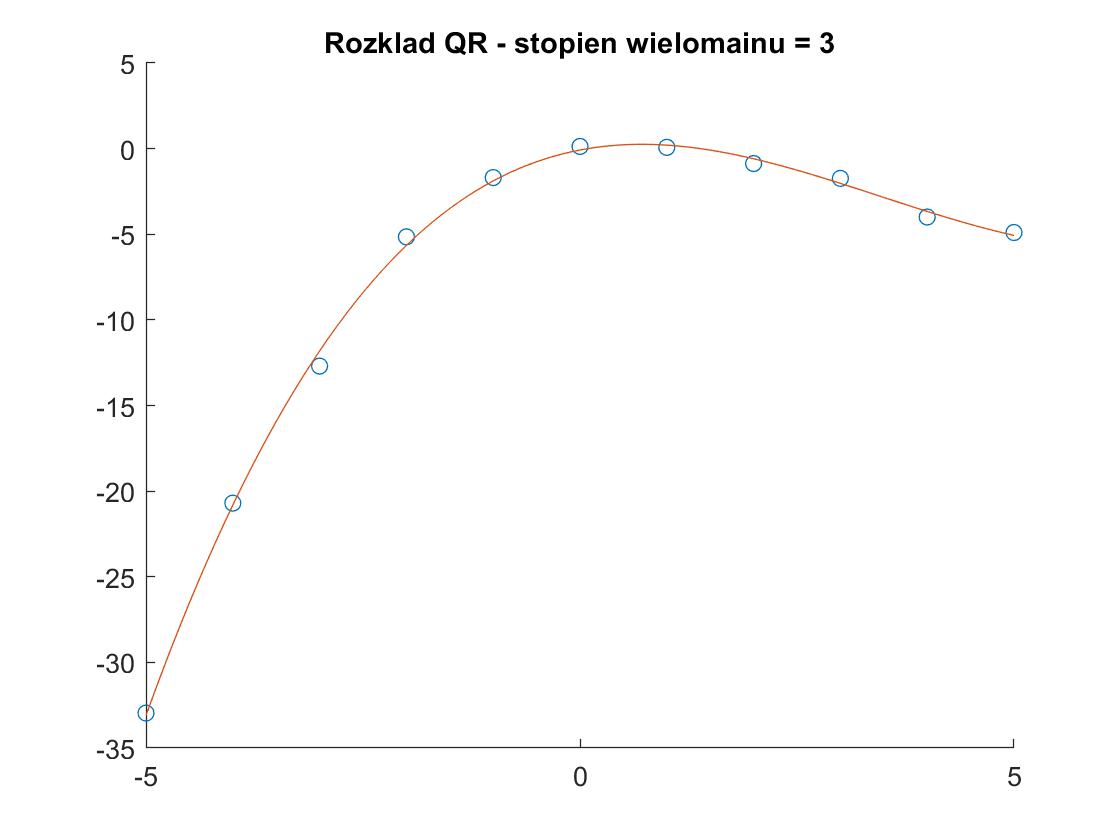
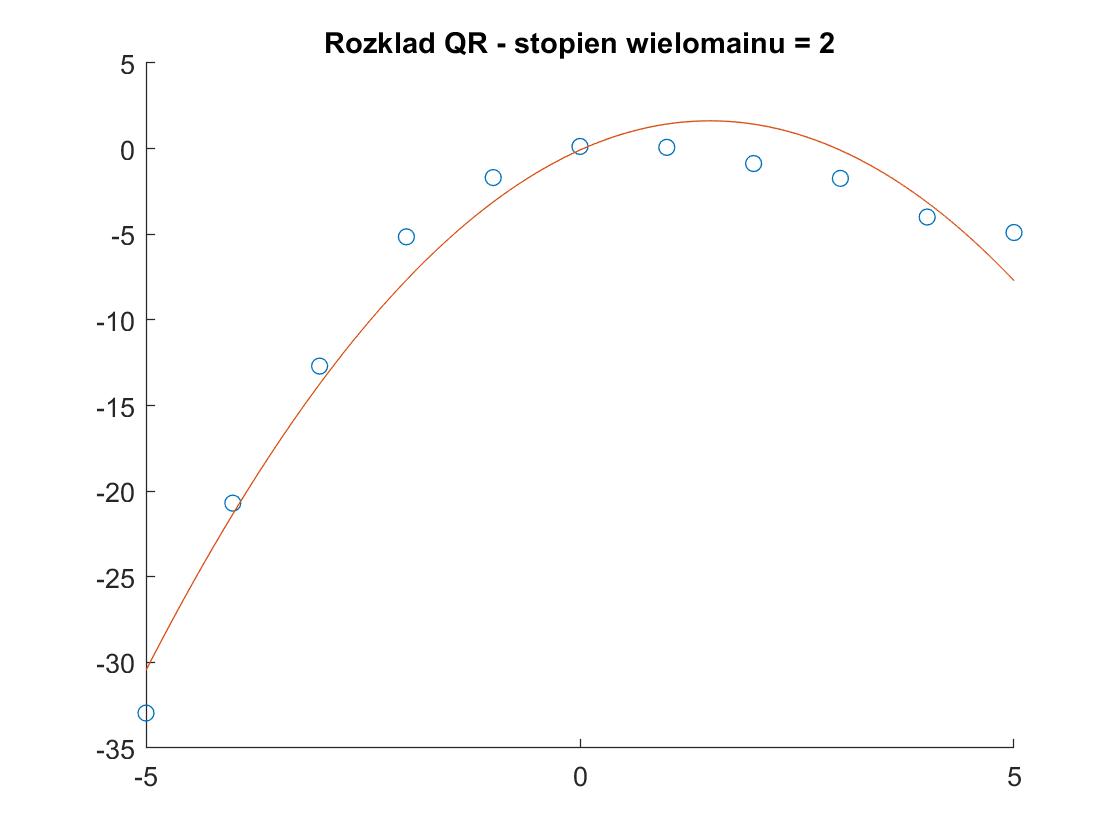
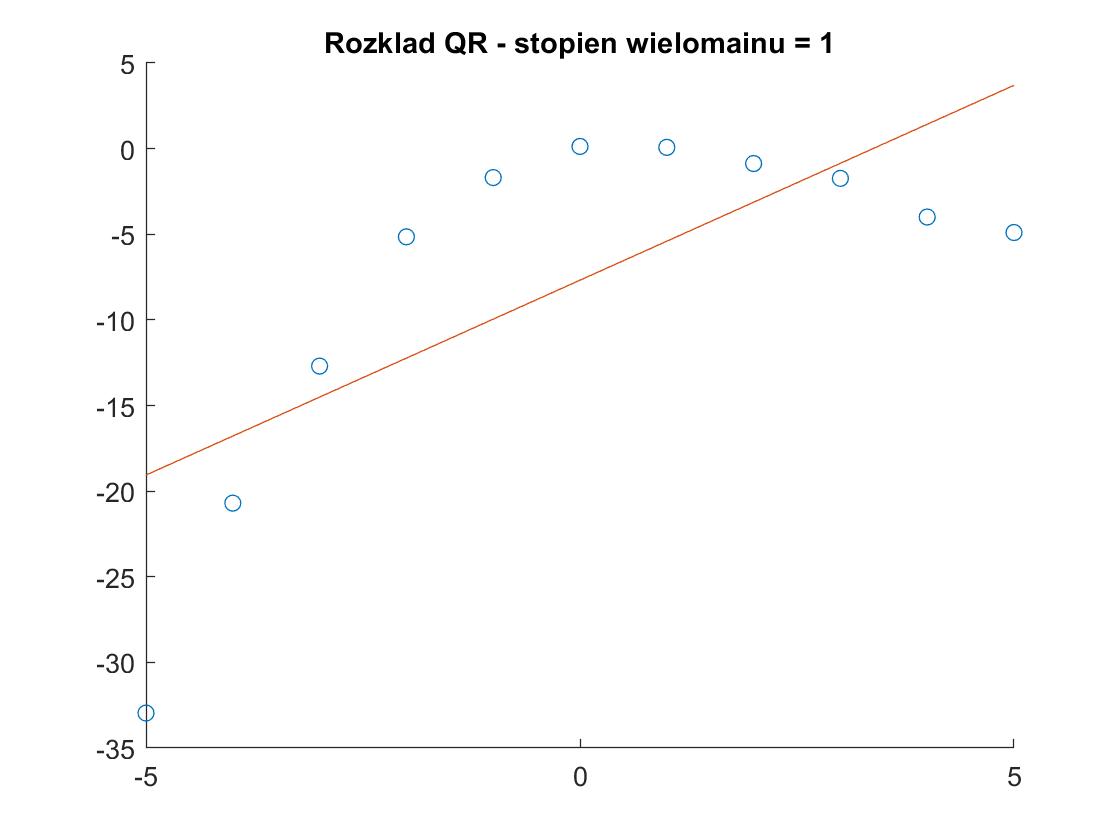
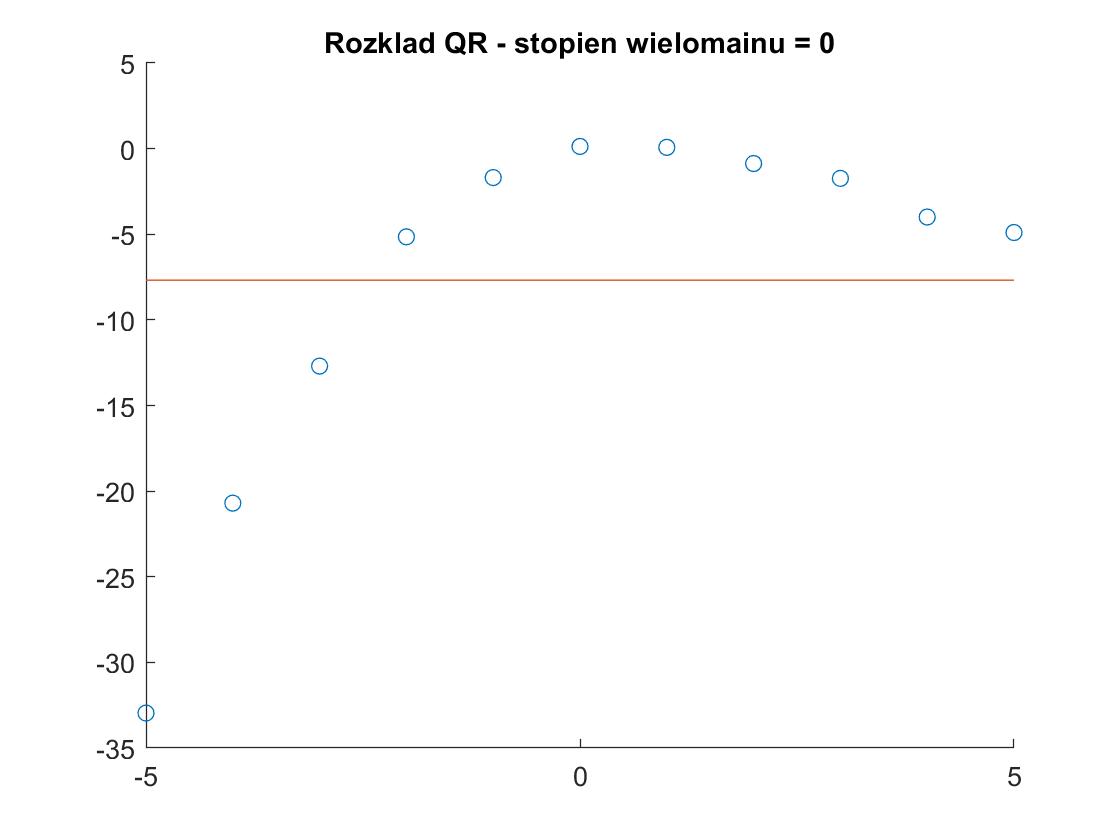
end

**Zadanie 2 – Wykresy (funkcja aproksymacji na tle danych)** 









Błędy rozwiązania obliczone jako norma residuum:

|  |  |  |
| --- | --- | --- |
| Stopień wielomianu | Błąd rozwiązania  Układ równań normalnych | Błąd rozwiązania  Rozkład QR |
| 0 | 0 | 0 |
| 1 | 2.8422e-14 | 1.0049e-14 |
| 2 | 2.8422e-14 | 8.7023e-15 |
| 3 | 9.0949e-13 | 6.2172e-15 |
| 5 | 1.4552e-11 | 6.2096e-15 |
| 10 | 4.1833e-07 | 2.5108e-13 |

Błędy aproksymacji:

|  |  |  |
| --- | --- | --- |
| Stopień wielomianu | Maksymalny błąd aproksymacji  Układ równań normalnych | Maksymalny błąd aproksymacji  Rozkład QR |
| 0 | 25.2773 | 25.2773 |
| 1 | 13.8985 | 13.8985 |
| 2 | 2.5216 | 2.5216 |
| 3 | 0.8845 | 0.8845 |
| 5 | 0.6992 | 0.6992 |
| 10 | 7.0487e-11 | 2.6128e-12 |

**Wnioski:**

Jak widać układ równań wynikający z rozkładu QR zachowuje dobre uwarunkowanie dłużej   
w przeciwieństwie do układu równań normalnych, który szybko traci dokładność. Mimo tego dla stopni wielomianów stopnia większego bądź równego 10 przebieg funkcji aproksymującej w obu przypadkach zaczyna odbiegać od poprzednich rezultatów. Wynika to z faktu, że w pewnym momencie przestajemy aproksymować funkcję a jedynie dane pomiarowe. Jak widać w obu przypadkach błędy aproksymacji maleją wraz z zwiększaniem stopnia wielomianu.