**MNUM – Projekt 2.15**

**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 .

Kod algorytmu metody QR bez przesunięć (plik: eigval.m):

function [ w, iteracje, success ] = eigval( A ,tol, imax )

%eigval Oblicznie wartosci wlasnych metoda rozkladu QR bez przesuniec

% tol - tolerancja

% imax - maksymalna liczba iteracji

% w - wektor wartości własnych

% iteracje - liczba iteracji potrzebnych do znalezienia wartości własnych

% success - czy udało się uzyskać wynik zanim liczba iteracji

% przekroczyła imax

success = 1;

z = zeros(size(A,1));

for i= 1:imax

b = A - diag(diag(A));

if (b == z) | max(max(abs(b))) < tol

break;

end

[q, r] = qrmgs(A);

A = r \* q; %macierz przekształcona

end

if i == imax

success = 0;

disp('Uwaga: osiągnięto imax');

end

w =diag(A);

iteracje = i;

end

Kod algorytmu metody QR z przesunięciami (plik: eigvalS.m):

function [ eigenvalues, iteracje,success] = eigvalS( A, tol, imax )

%eigvalS Oblicznie wartosci wlasnych metoda rozkladu QR z przesunieciami

% tol - tolerancja

% imax - maksymalna liczba iteracji

success = 1;

n=size(A,1);

eigenvalues = diag(zeros(n));

INITIALsubmatrix = A; %macierz początkowa (oryginalna)

iteracje = 0;

for k = n:-1:2

DK = INITIALsubmatrix; %macierz startowa dla jednej wart. własnej

i = 0;

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

DD = DK(k-1:k,k-1:k); %macierz 2x2 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));

%najbliższa DK(k,k) wartość własna podmacierzy DD

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

shift = ev1;

else

shift = ev2;

end

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

[Q1,R1] = qrmgs(DK); %faktoryzajca QR

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

i = i+1;

iteracje = iteracje + 1;

end

if i >imax

success = 0;

disp('imax reached')

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 wartość własna

end

end

end

Kod algorytmu rozkładu QR - zmodyfikowany algorytm Grama-Shmidta (plik: qrmgs.m):

function [ Q, R ] = qrmgs( A )

%qrmgs Rozkład QR zmodyfikowanym algorytmem Grama-Schmidta

[m, n] = size(A);

Q = zeros(m,n);

R = zeros(n,n);

d = zeros(1,n);

%Rozkład QR

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 układu

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);

%wybieramy licznik o wiekszym module

if abs(l1) > abs(l2)

licznik = l1;

else

licznik = l2;

end

x1 = licznik/(2\*a);

%drugi pierwiastek obliczmy ze wzorów Viete'a

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

end

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

Rozmiar 5x5

W tym przypadku wszystkie metody były zbieżne.

|  |  |  |
| --- | --- | --- |
| Macierz i metoda | Średnia liczba iteracji | Liczba nieudanych prób |
| Macierz symetryczna  Algorytm bez przesunięć | 413.8000 | 0 |
| Macierz symetryczna  Algorytm z przesunięciami | 7.3667 | 0 |
| Macierz niesymetryczna  Algorytm z przesunięciami | 9.4000 | 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 | 808.3667 | 1 |
| Macierz symetryczna  Algorytm z przesunięciami | 14.3333 | 0 |
| Macierz niesymetryczna  Algorytm z przesunięciami | 20.5667 | 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 | 1.9205e+03 | 1 |
| Macierz symetryczna  Algorytm z przesunięciami | 27.6333 | 0 |
| Macierz niesymetryczna  Algorytm z przesunięciami | 43.9667 | 0 |

**Wnioski:**  
 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:**  
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 :

Kod algorytmu aproksymującego (plik: aproksymacja.m):

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

%aproksymacja Summary of this function goes here

% n - stopien wielomianu

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

N = size(x,1);

A = zeros(N,n);

%Wypelniamy macierz A odpowiednimi potęgami 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] = qrmgs(A);

a =R\Q'\*y;

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

end

end

Kod algorytmu obliczającego wartości wielomianu (plik: 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

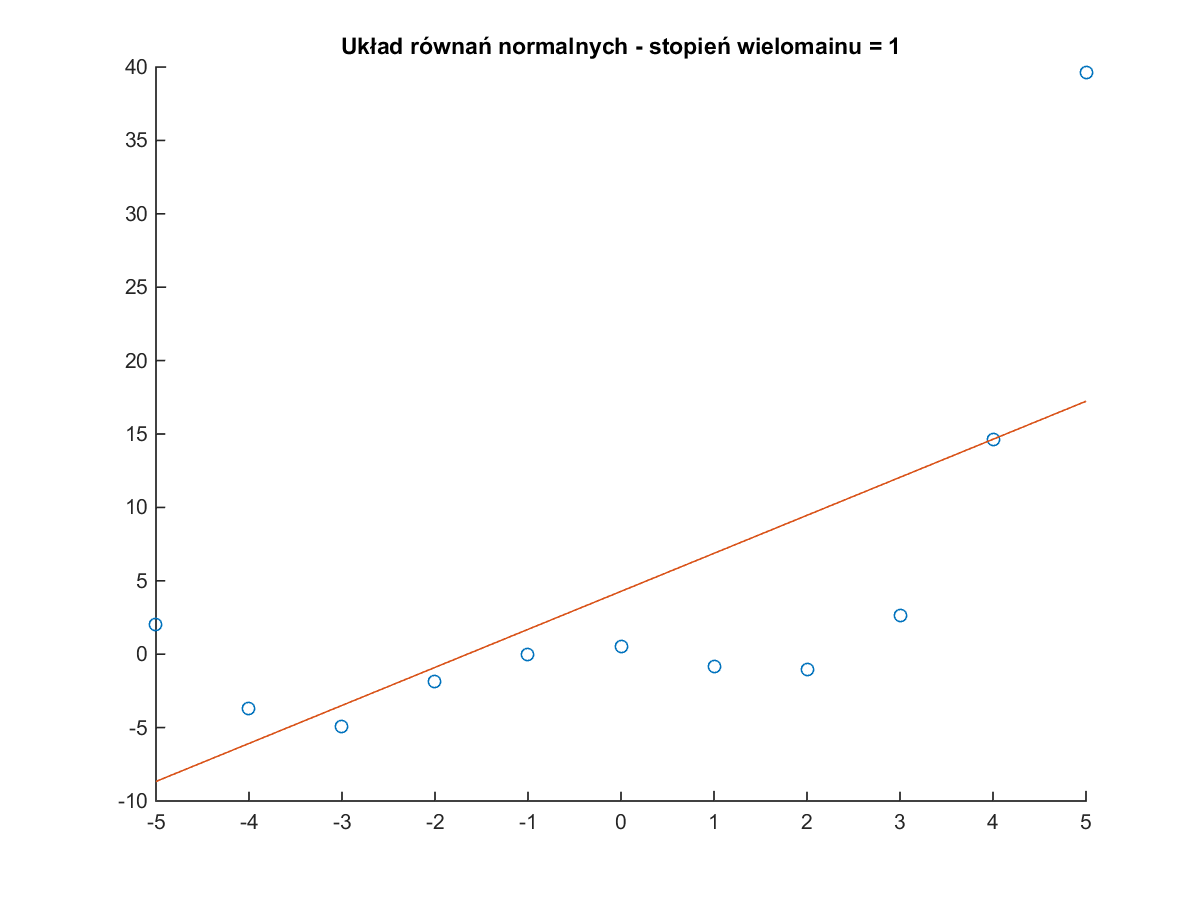
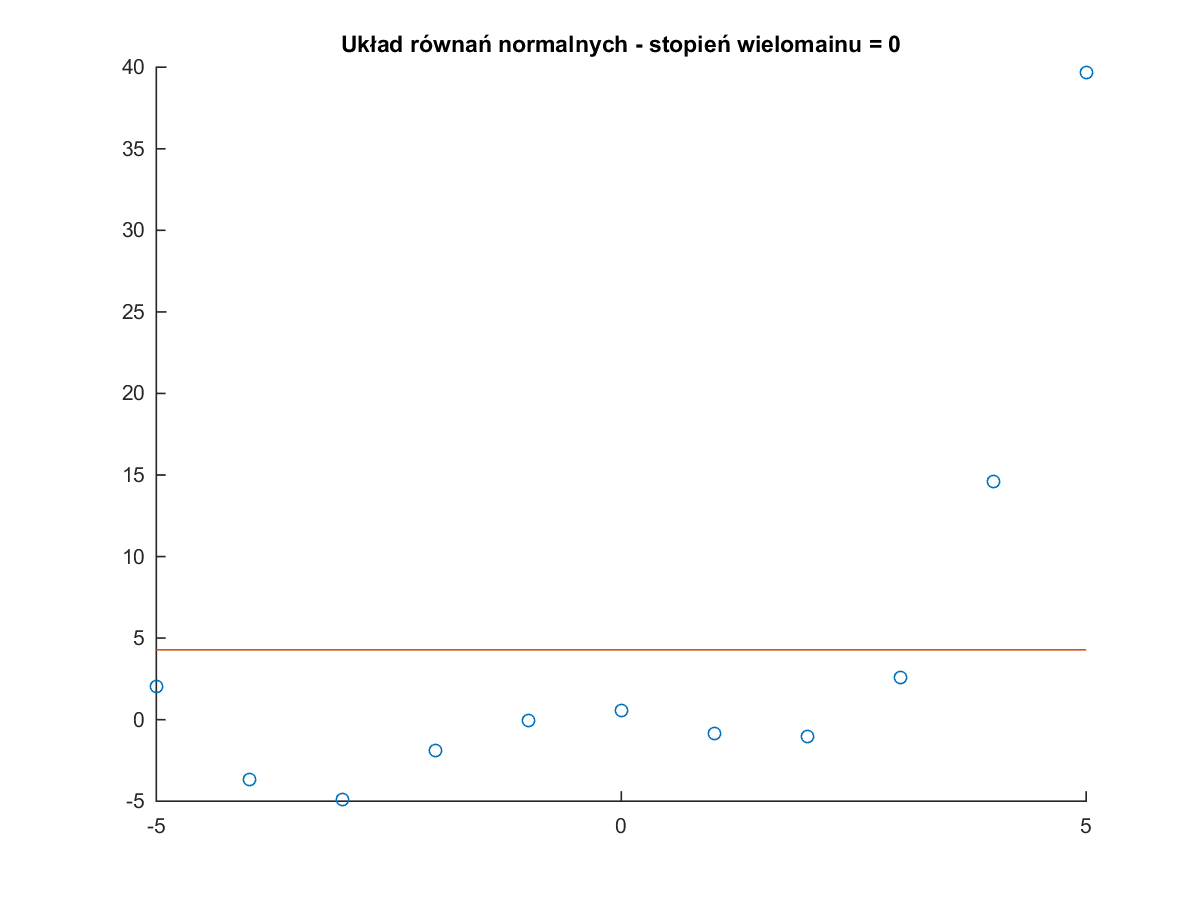
w(i) = w(i) + a(j) \* x(i)^(j-1);

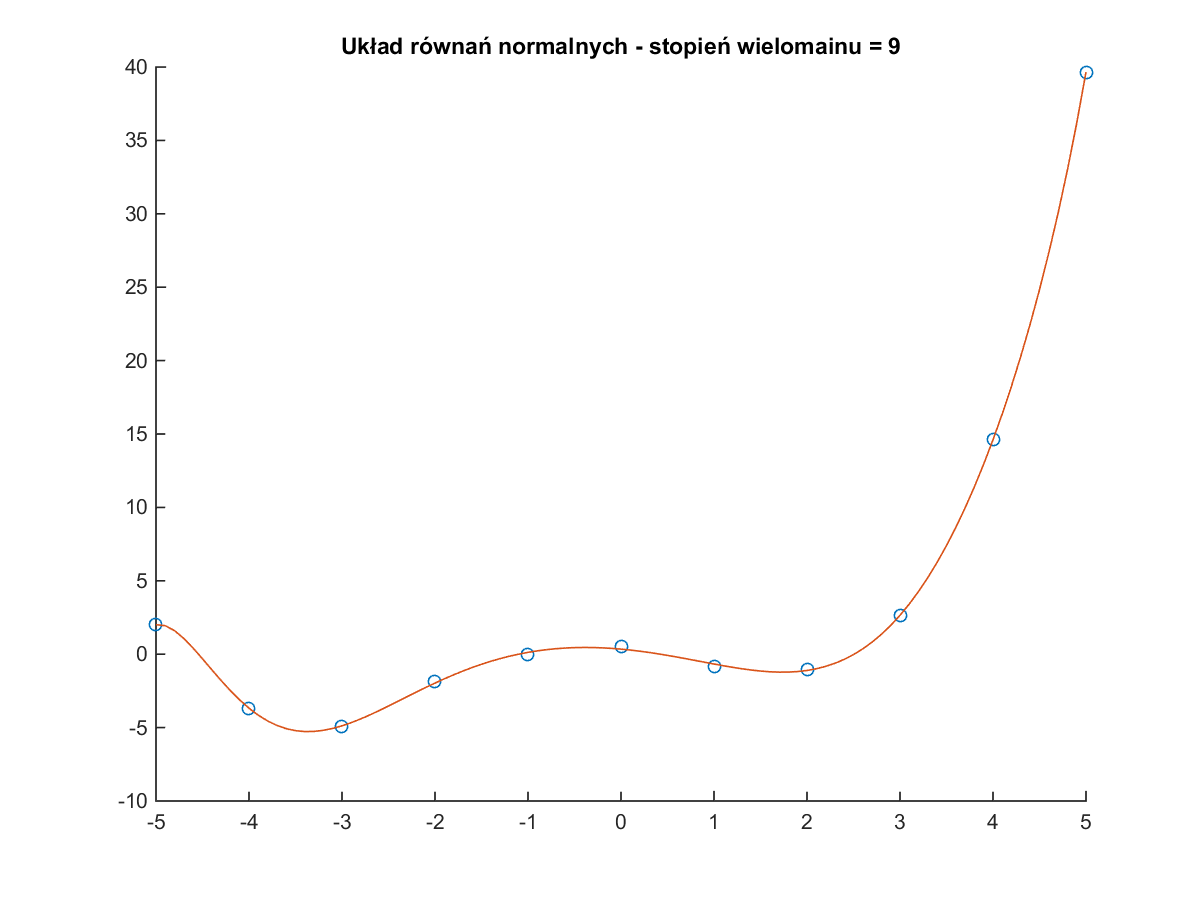
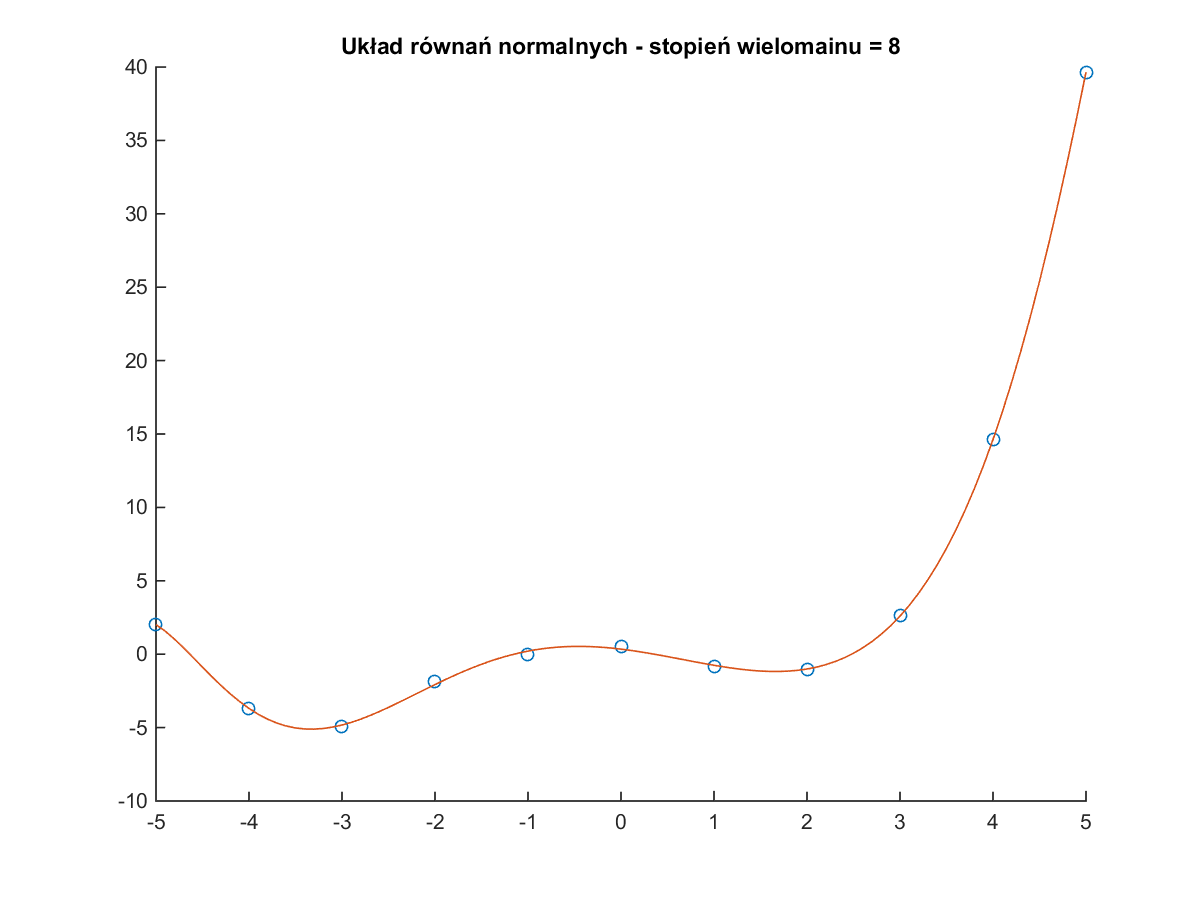
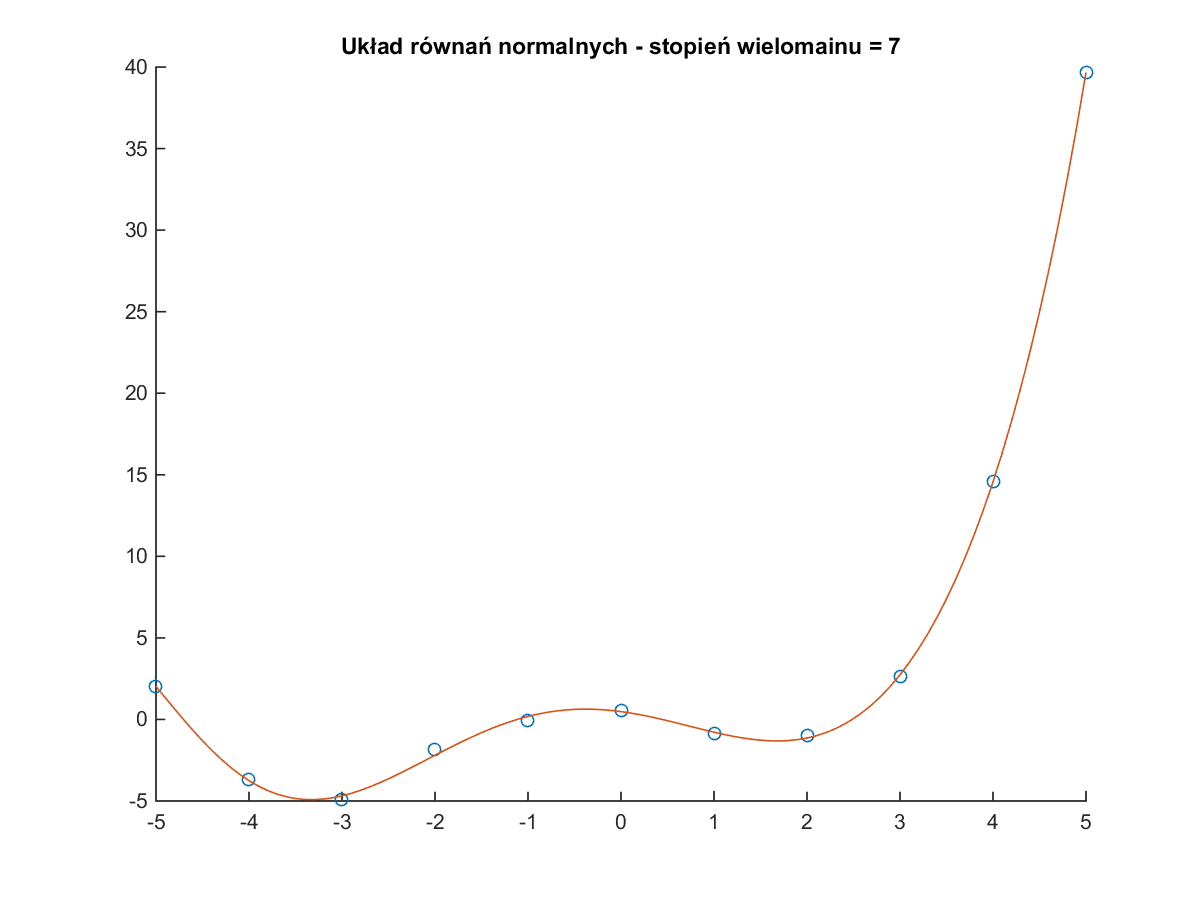
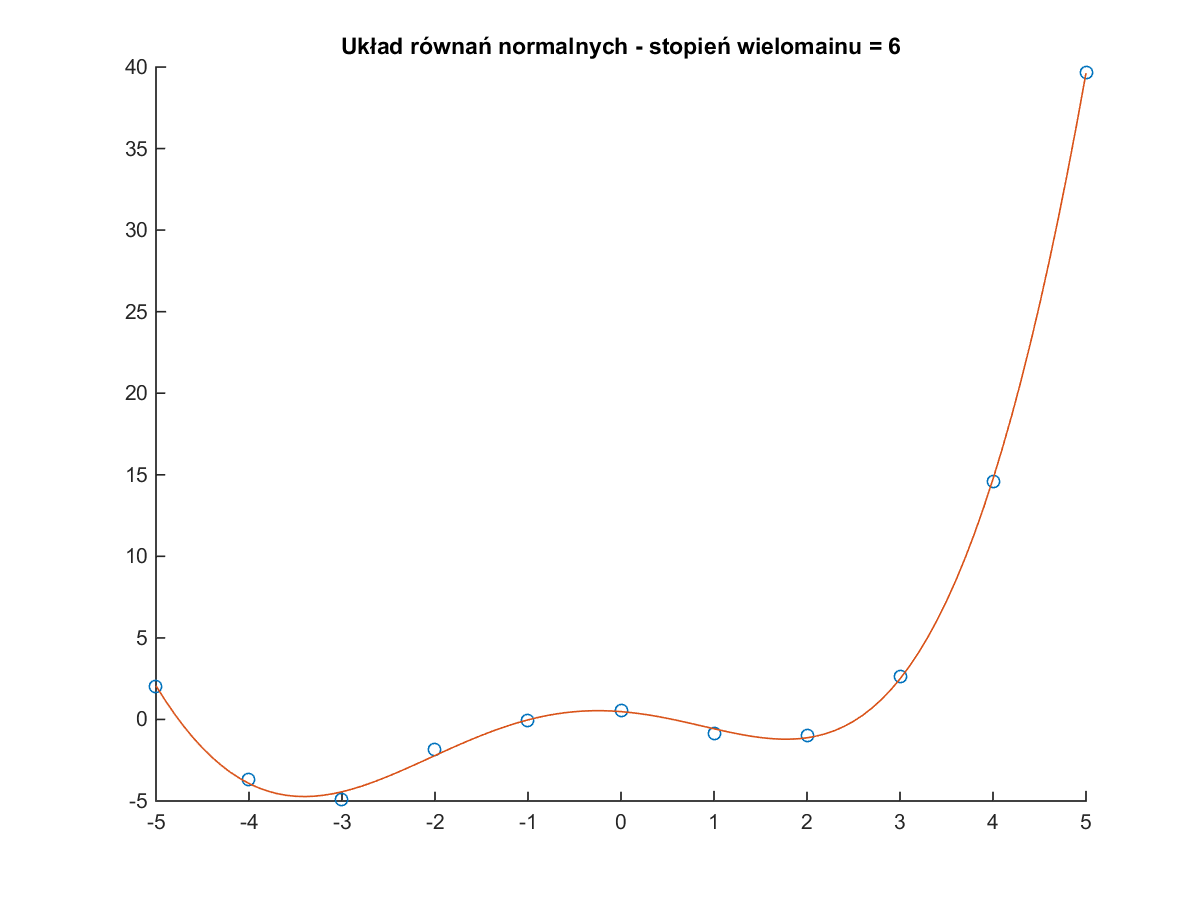
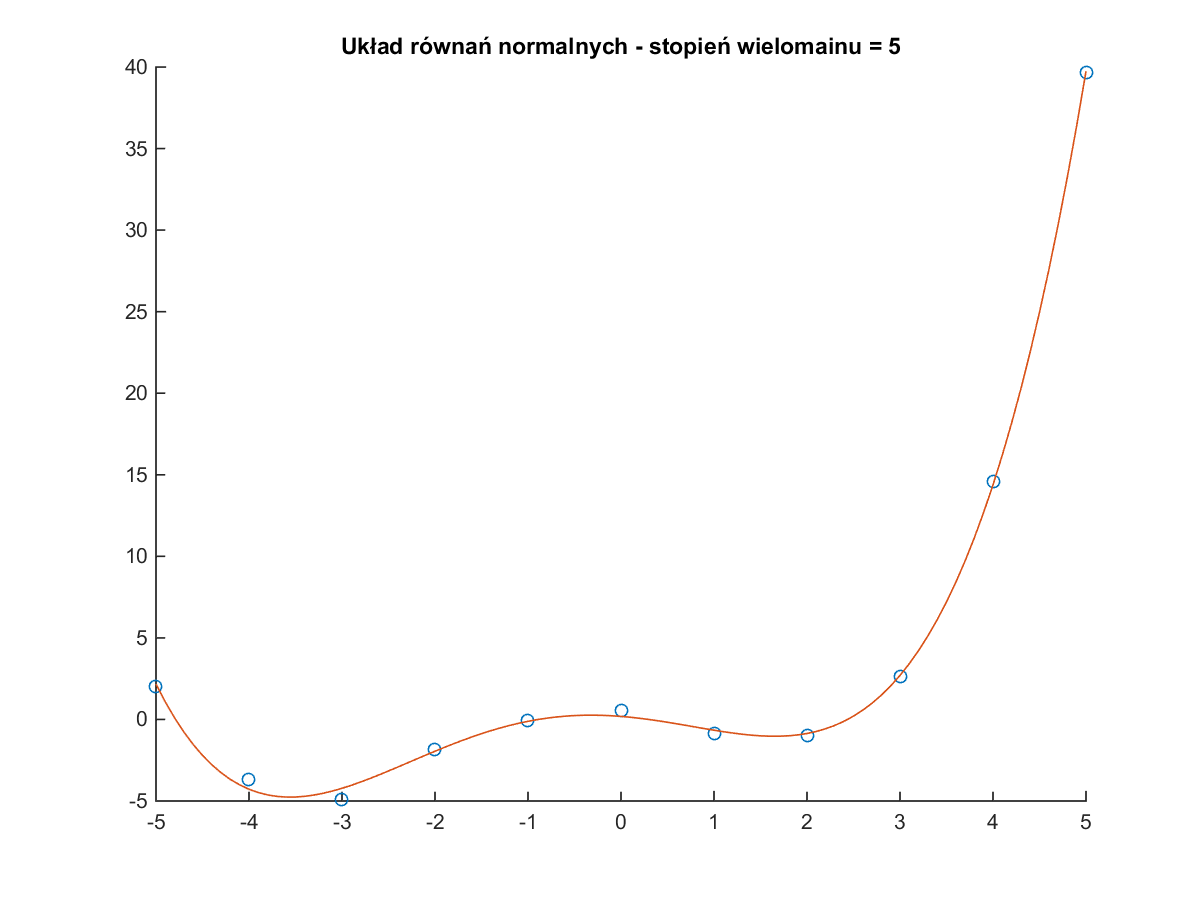
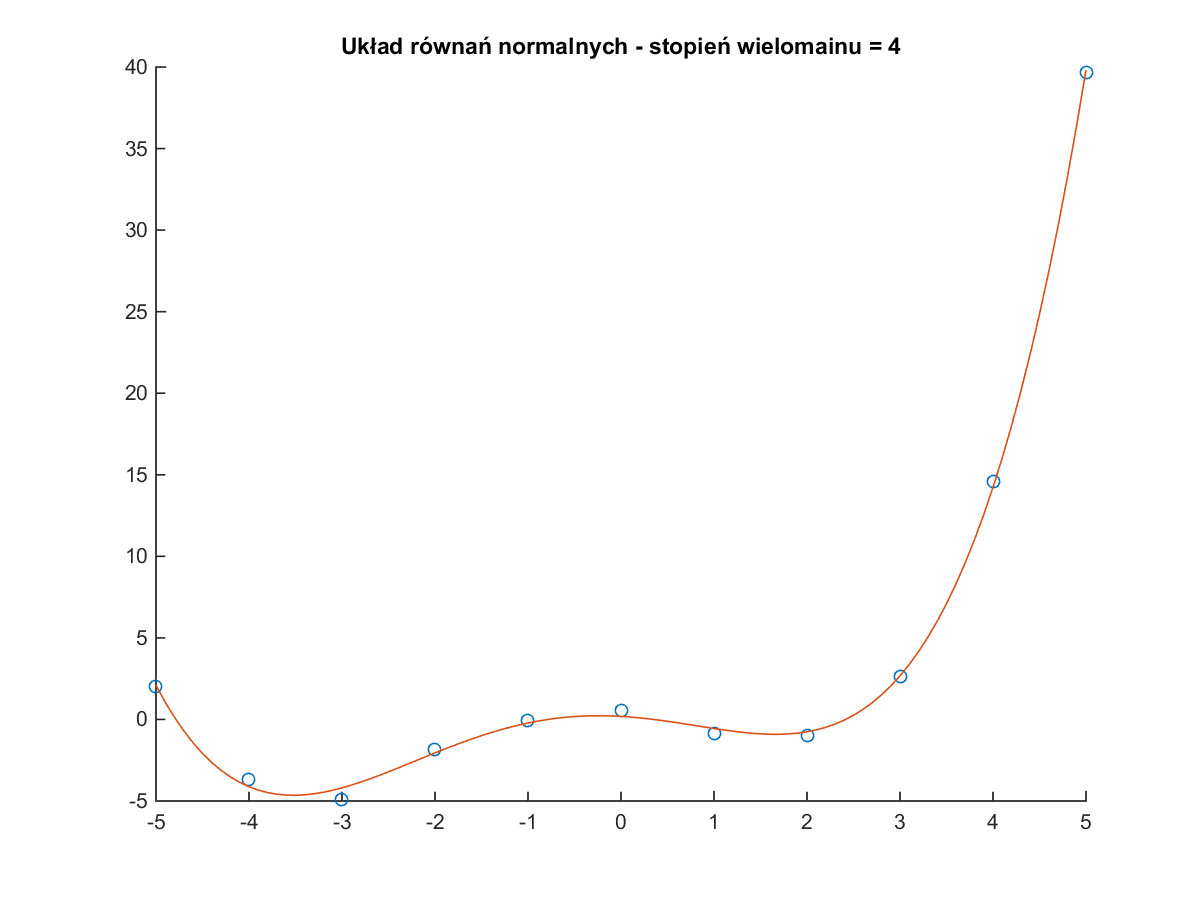
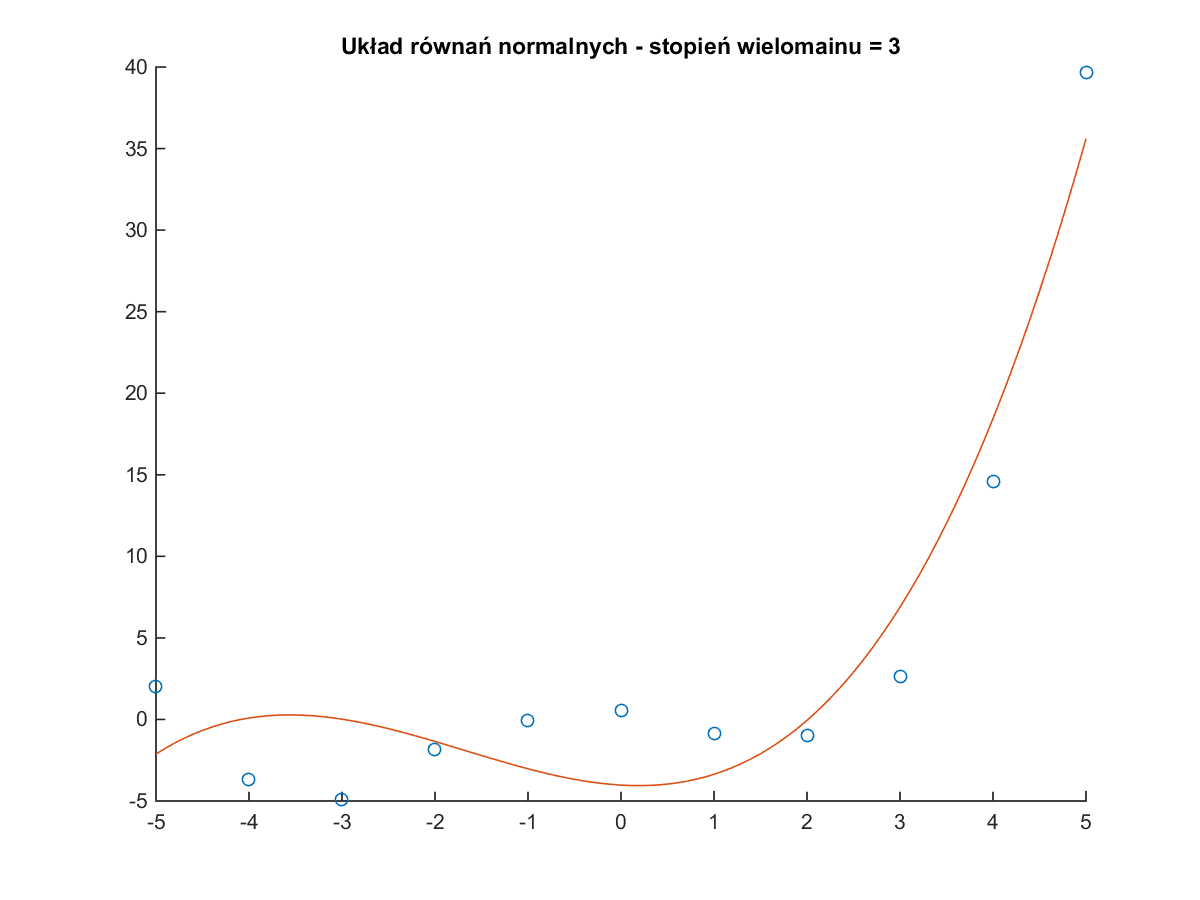
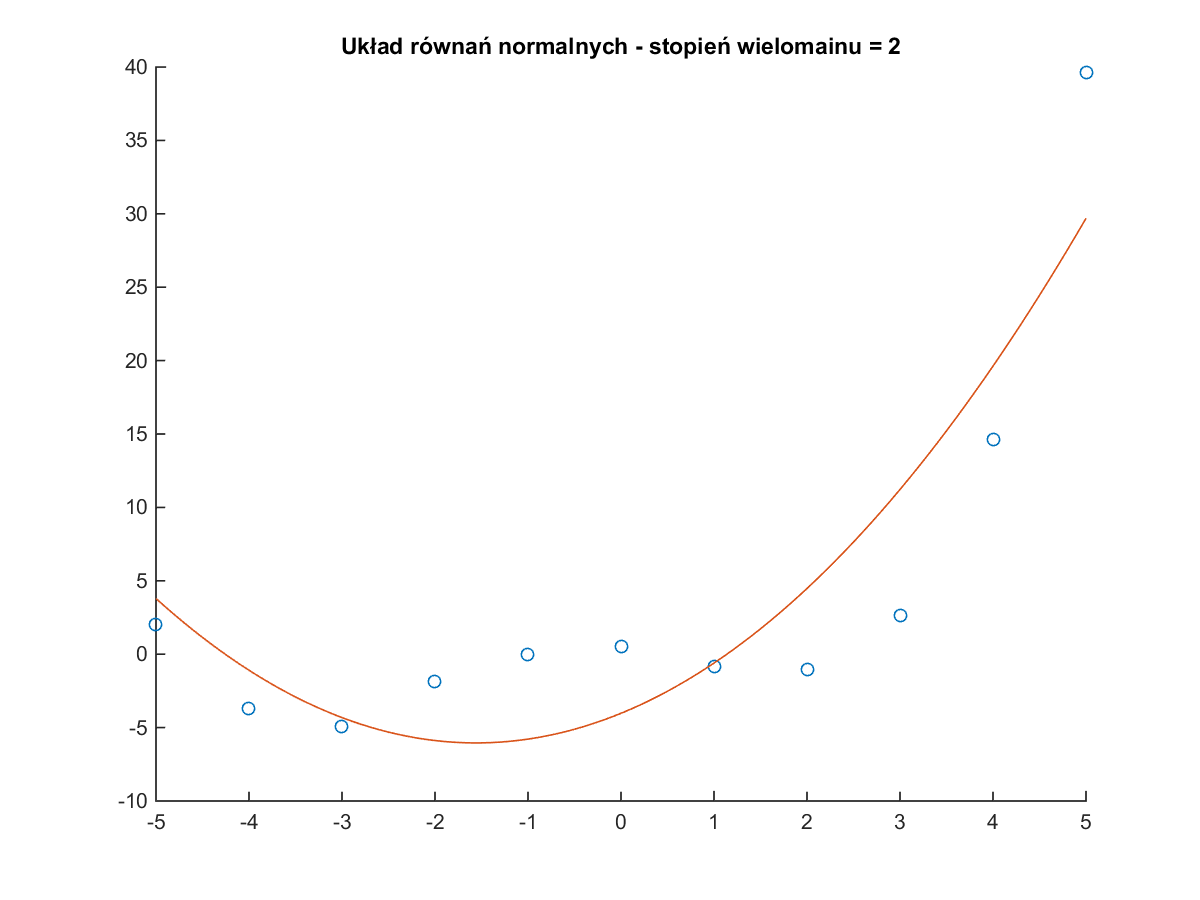
end

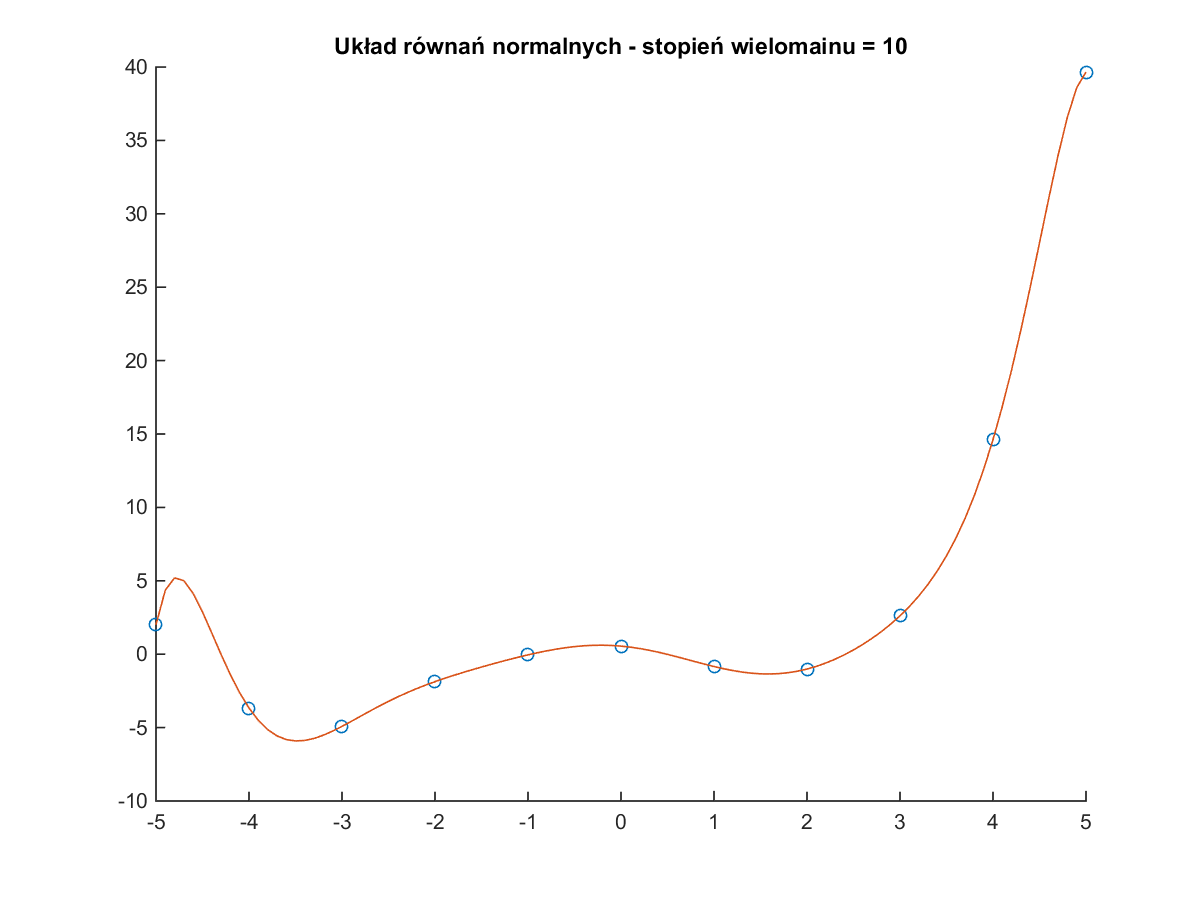
end

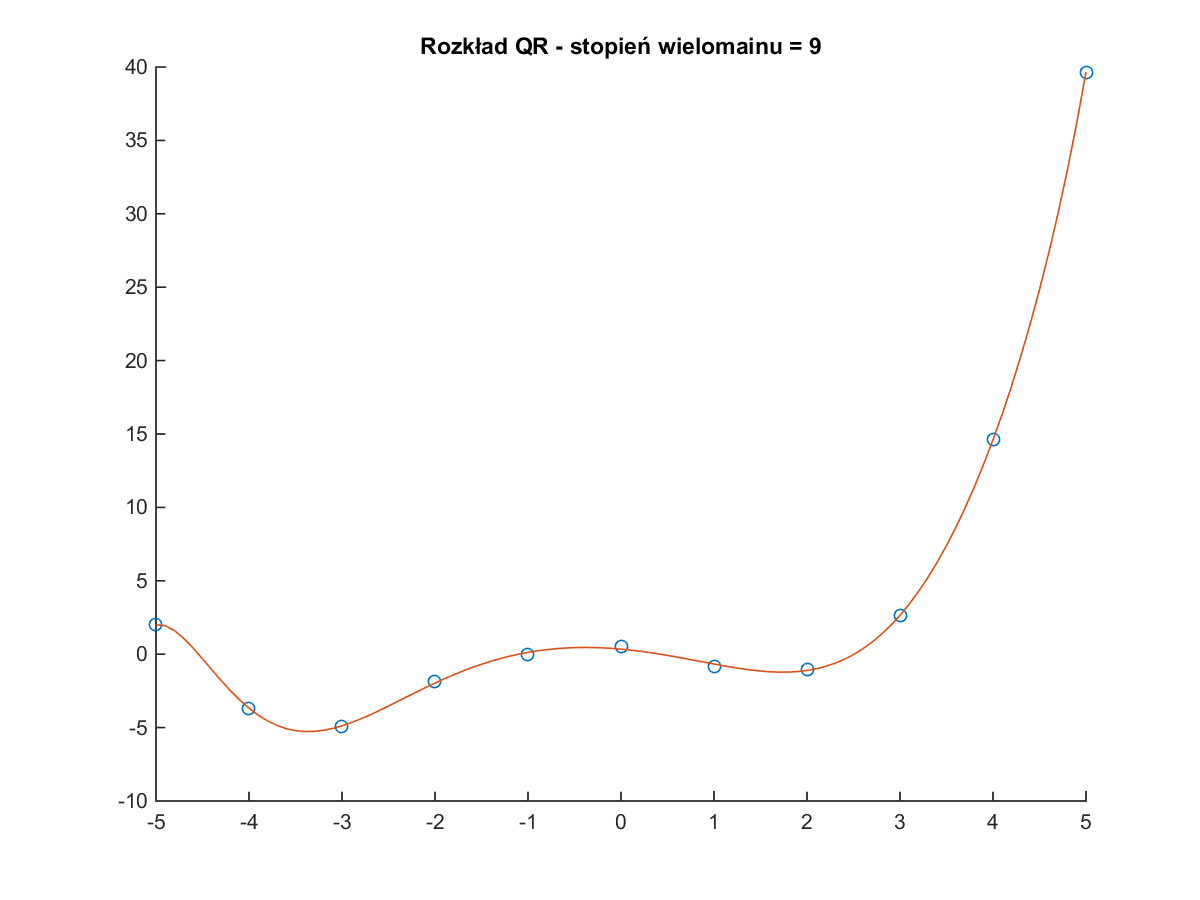
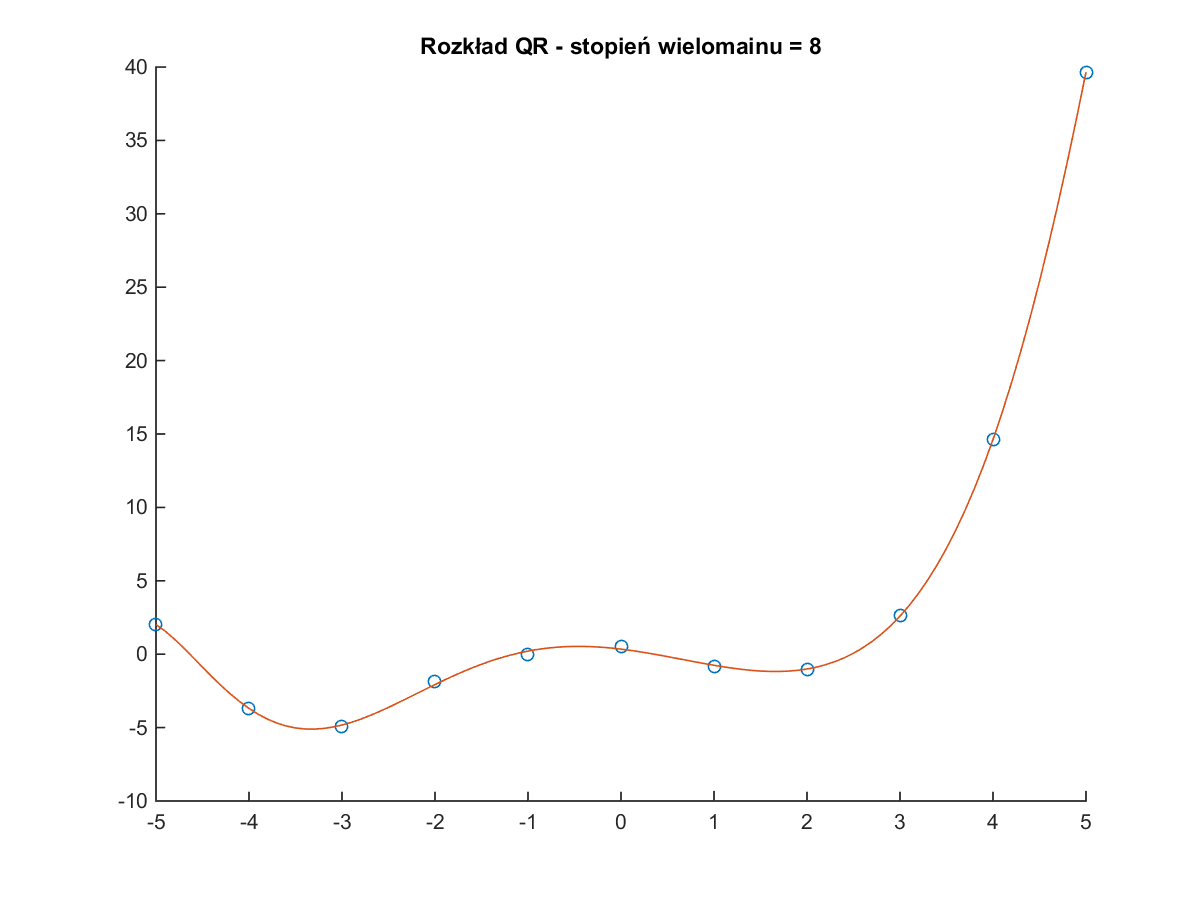
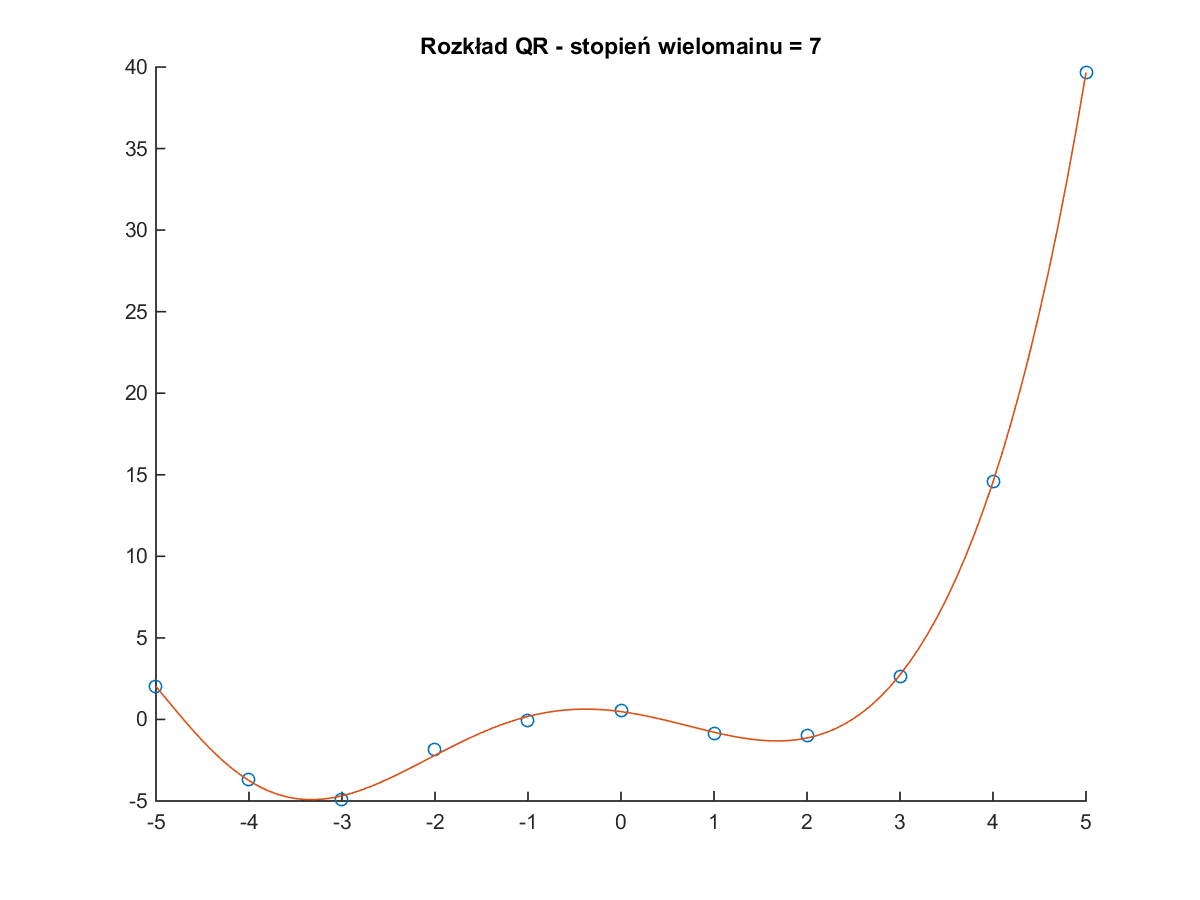
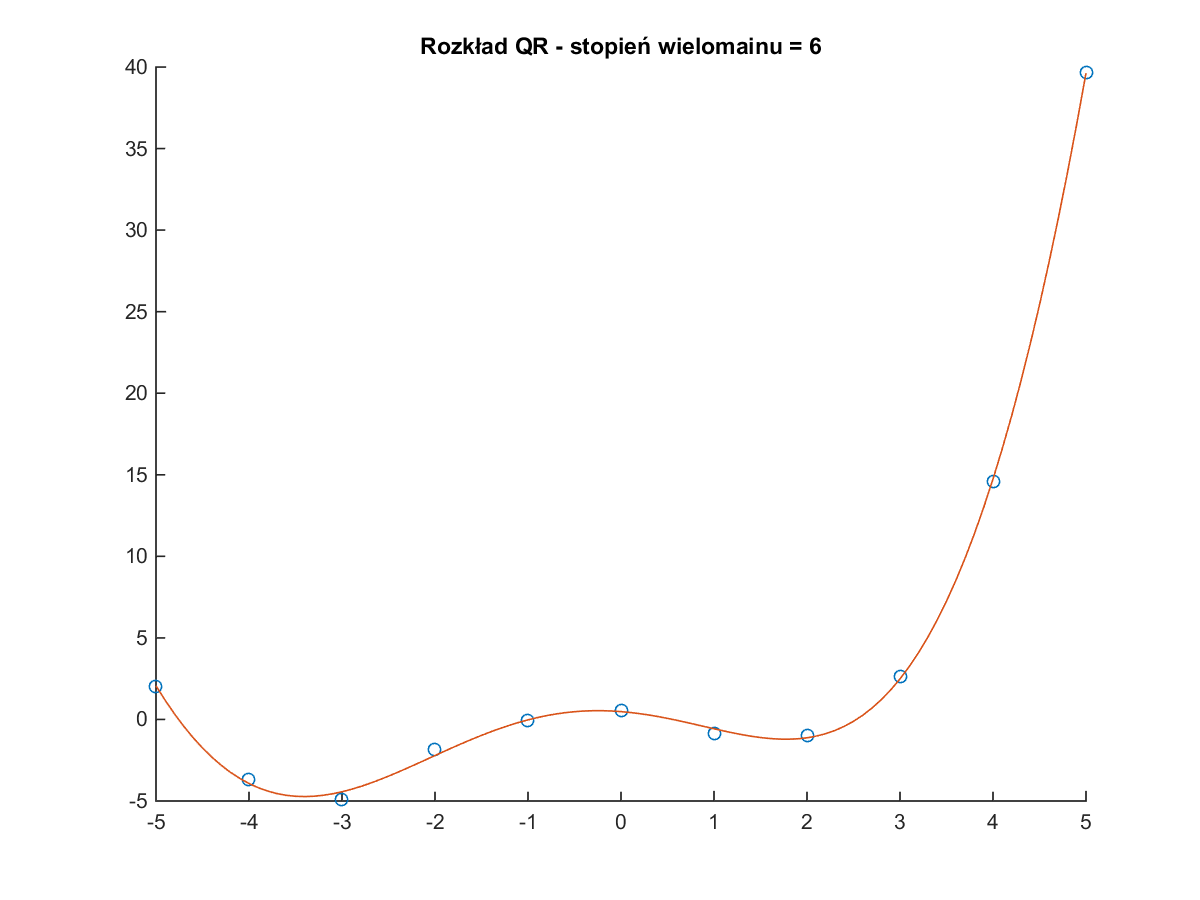
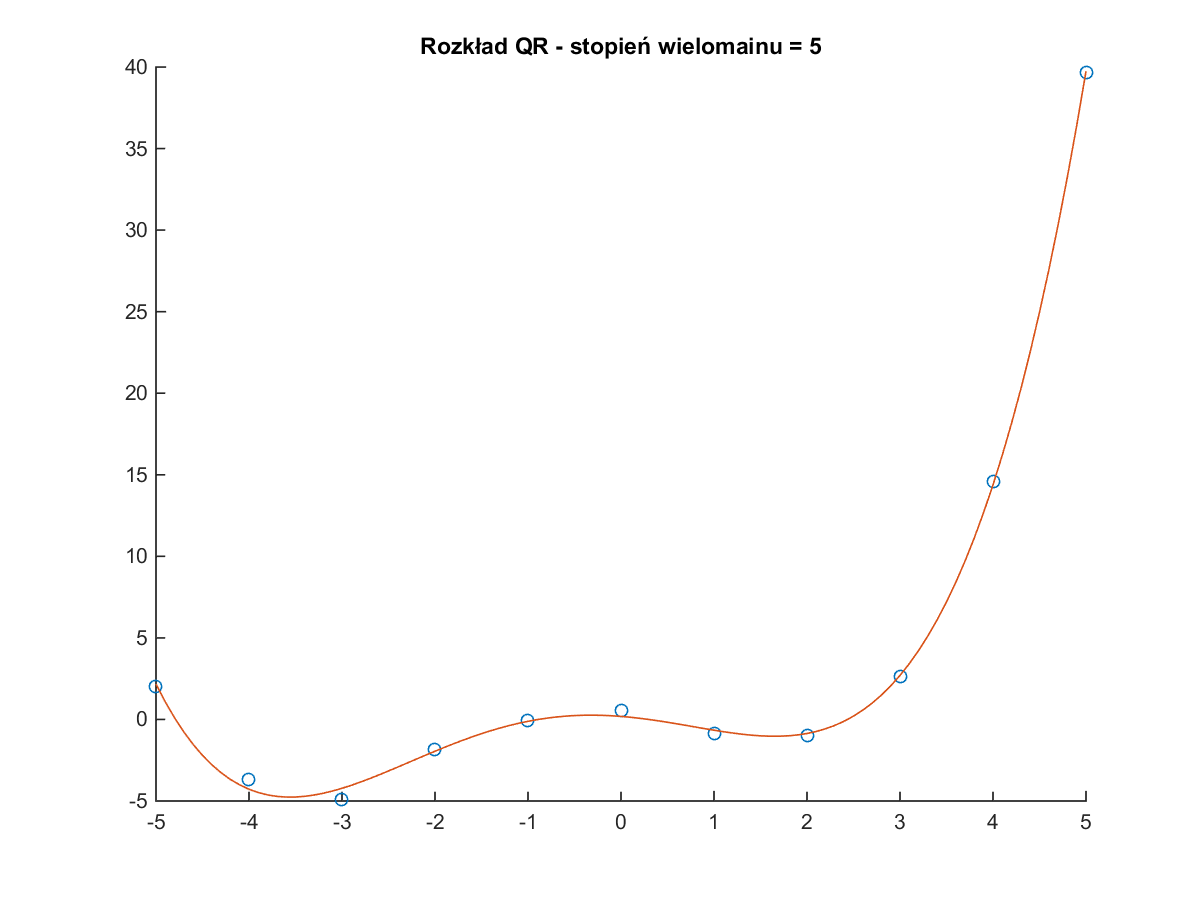
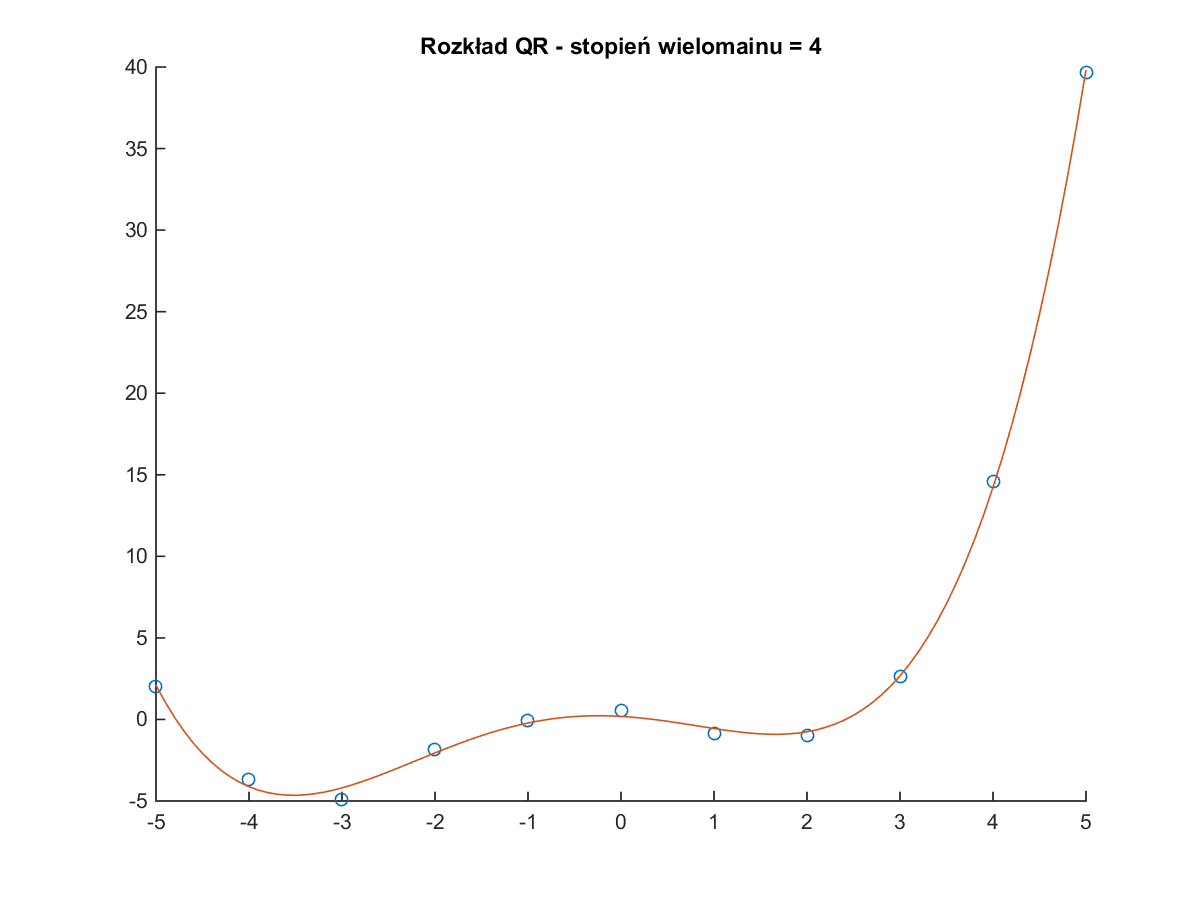
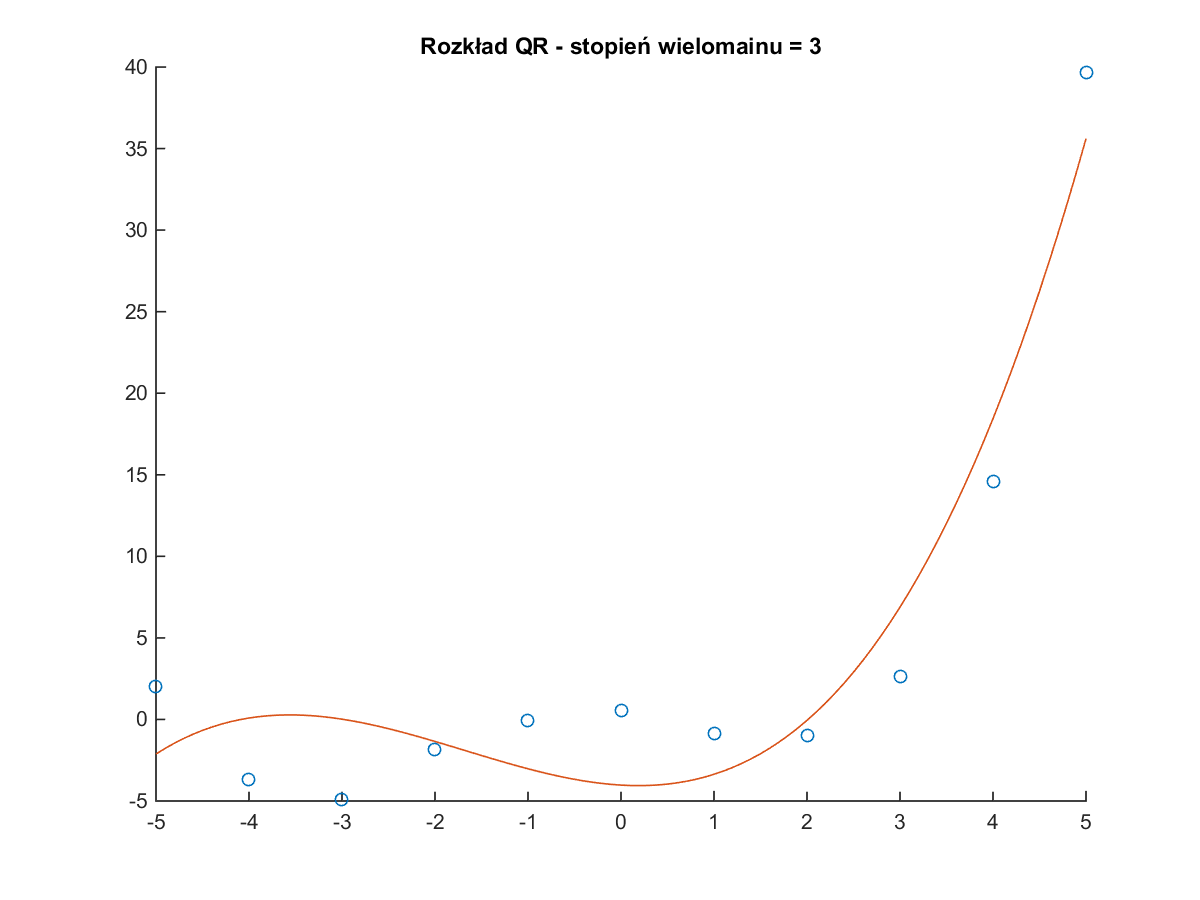
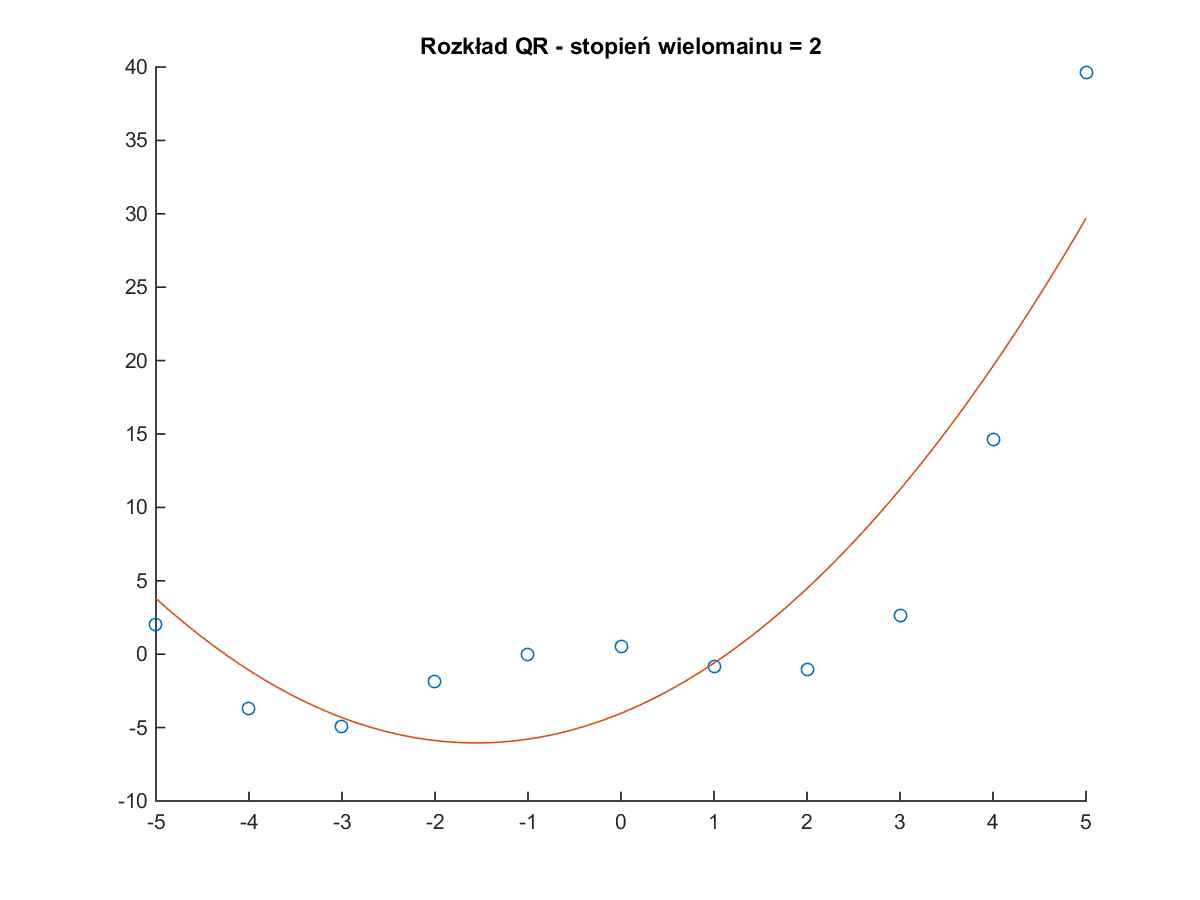
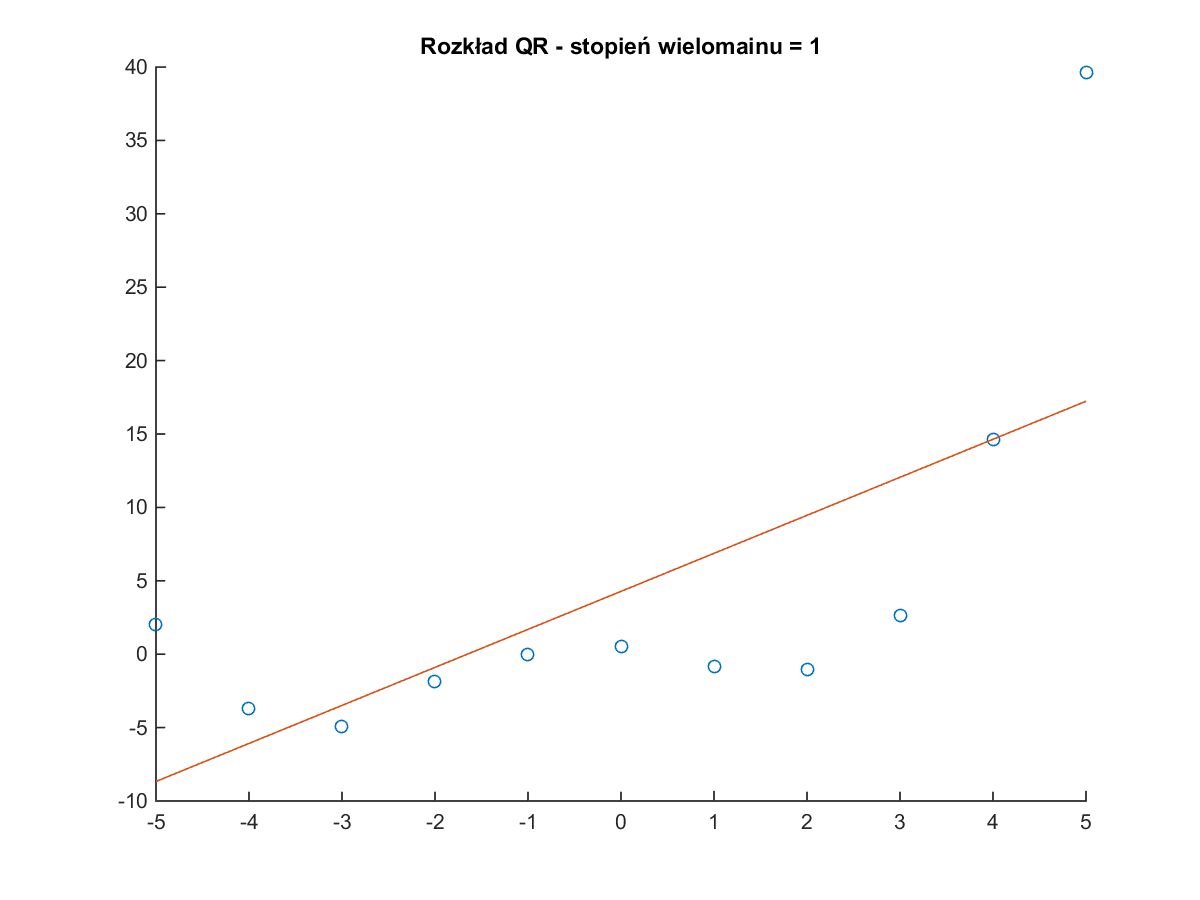
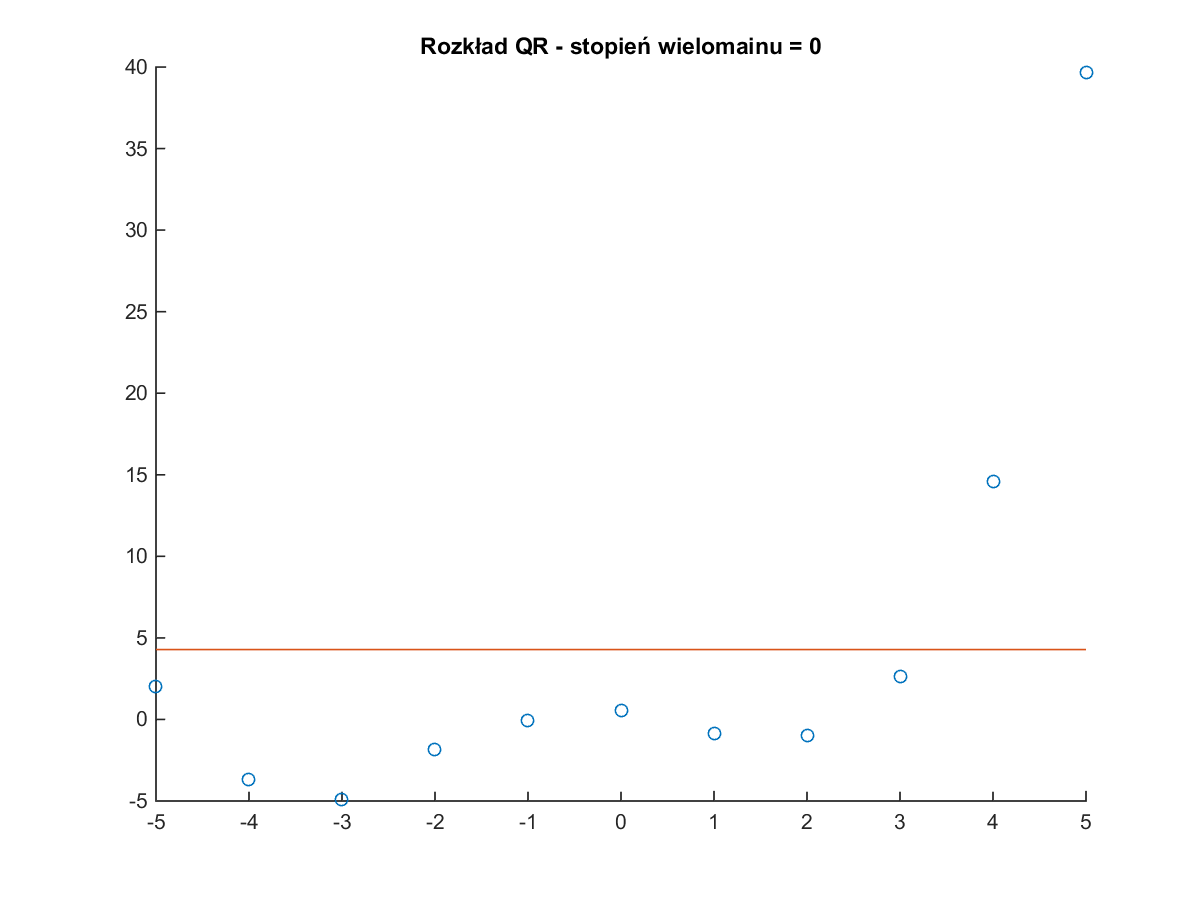
end

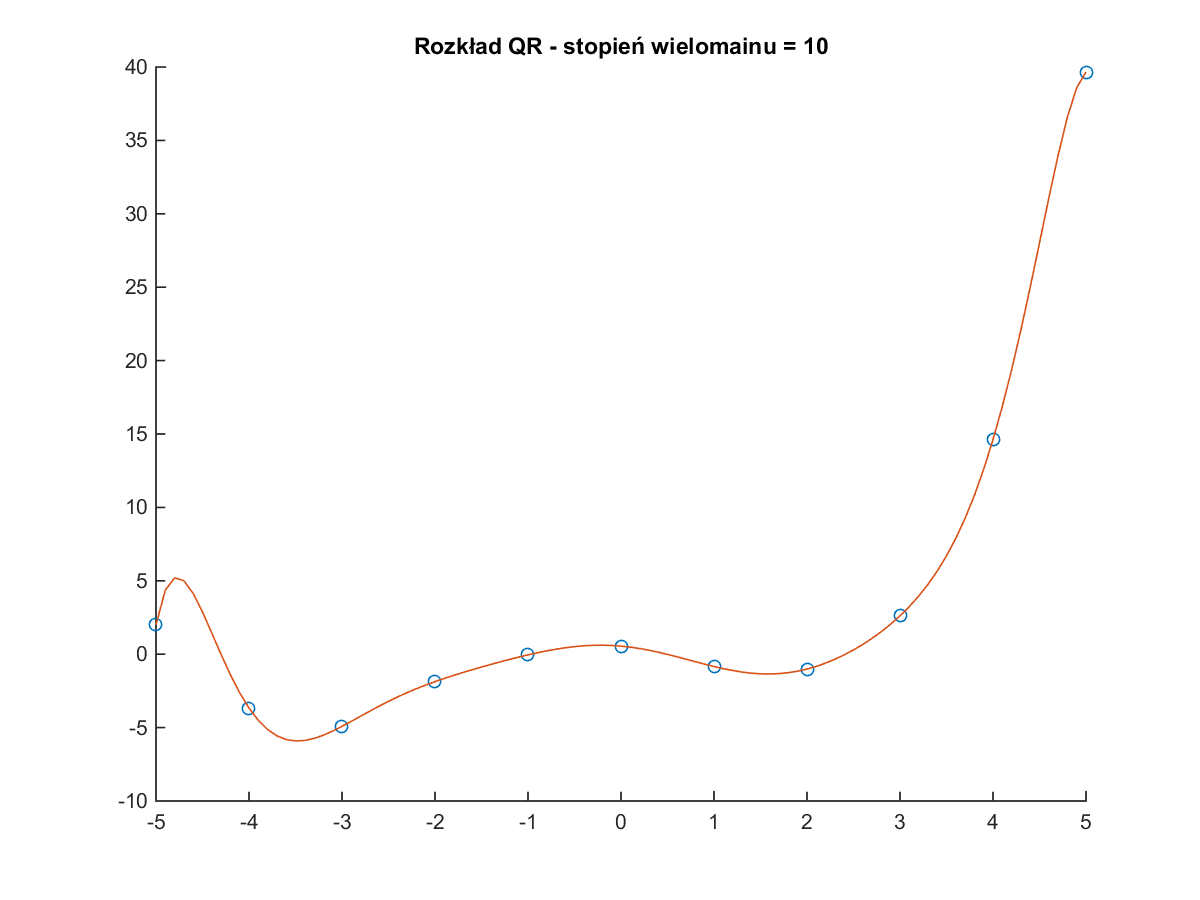
**Wyniki:**











Błędy rozwiązania obliczone jako norma residuum:

|  |  |  |
| --- | --- | --- |
| Stopień wielomianu | Błąd rozwiązania  Układ równań normalnych | Błąd rozwiązania  Rozkład QR |
| 0 | 0 | 0 |
| 1 | 0 | 3,55271367880050e-15 |
| 2 | 7,10542735760100e-15 | 8,14028967780416e-15 |
| 3 | 7,10542735760100e-15 | 7,32410687763558e-15 |
| 4 | 3,63800656256145e-12 | 3,55271367880050e-15 |
| 5 | 2,91038339261805e-11 | 5,33167196845001e-15 |
| 6 | 1,19998167083181e-10 | 5,77742823859925e-15 |
| 7 | 4,80930315469239e-10 | 5,09051482459403e-14 |
| 8 | 4,66571281122666e-10 | 1,02447329786068e-13 |
| 9 | 1,86499304351714e-08 | 2,73369781880071e-13 |
| 10 | 1,37415774943358e-06 | 5,99787696540388e-13 |

Błędy aproksymacji:

|  |  |  |
| --- | --- | --- |
| Stopień wielomianu | Maksymalny błąd aproksymacji  Układ równań normalnych | Maksymalny błąd aproksymacji  Rozkład QR |
| 0 | 9,19745454545455 | 9,19745454545455 |
| 1 | 10,4721218181818 | 10,4721218181818 |
| 2 | 8,60677491841492 | 8,60677491841492 |
| 3 | 4,93266899766900 | 4,93266899766900 |
| 4 | 0,720822144522122 | 0,720822144522149 |
| 5 | 0,694955477855475 | 0,694955477855482 |
| 6 | 0,481961591937421 | 0,481961591937473 |
| 7 | 0,232988062526106 | 0,232988062525711 |
| 8 | 0,257555186299342 | 0,257555186299766 |
| 9 | 0,170669953885007 | 0,170669953885225 |
| 10 | 3,26312421528030e-10 | 2,59603449848100e-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.