Maciej Suchocki I.SID 277377

MNUM PROJEKT 2 - zestaw 2.4

Zadanie 1

Wartości własne macierzy są to pierwiastki jej równania charakterystycznego. Dla danej macierzy A jej równanie charakterystyczne przedstawia się w następujący sposób:

$$det(A - \lambda E) = 0 ,$$

gdzie E jest macierzą jednostkową, a λ jest wartością własną

Warto nadmienić, że wartości własne występują tylko w macierzach kwadratowych, ponieważ tylko te posiadają elementy diagonalne, więc przykładowa macierz o wymiarze n ma dokładnie n wartości diagonalnych, więc ma również n wartości własnych. Warto również powiedzieć, iż zbiór wszystkich wartości własnych nazywany jest widmem macierzy A i oznacza się go jako $Sp(a)$. Najbardziej efektywną metodą w aspekcie numerycznym do znajdowania wartości własnych macierzy A jest metoda QR, która opiera się na rozkładzie QR macierzy.

Rozkład QR

Rozkład QR macierzy polega na tym, iż każdą macierz A_{mxn} o liniowo niezależnych kolumnach można przedstawić za pomocą iloczynu dwóch macierzy Q_{mxn} oraz R_{nxn}. Gdzie należy nadmienić, że macierz Q **jest macierzą ortogonalną** utworzoną z macierzy A za pomocą elementarnych działań na kolumnach i wierszach macierzy A. Macierz ortogonalna jest to macierz w której iloczyn skalarny wszystkich par kolumn wierszy macierzy Q równa się zero, czyli spełnia następujący warunek:

$$Q \cdot Q^T = I$$

Za to macierz R **jest macierzą trójkątną górna z dodatnimi elementami na diagonali** budowaną za pomocą różnych algorytmów. Jeżeli założymy, że macierz A jest nieosobliwa i że na przekątnej macierzy R są wyrazy dodatnie, to rozkład jest jednoznaczny, a więc nie zależy od wyboru algorytmu.

Metoda Grama-Schmidta

Metoda ta zakłada zbudowanie macierzy ortogonalnej Q w oparciu o podaną macierz A. Skoro ma to być macierz ortogonalna to wszystkie wartości oprócz przekątnej wynoszą zero. Pierwszy wektor zatem macierzy Q jest pierwszym wektorem macierzy A, a kolejne budujemy według wzoru, gdzie każdy z następnych wektorów musi być ortogonalny do wszystkich poprzednich.

$$q_1 = a_1$$

$$q_2 = a_2 - r_{12} \cdot q_1$$

pamiętajmy jednak o tym, że q_1 oraz q_2 muszą być ortogonalne, więc:

$$(q_1, q_2) = (q_1, a_2 - r_{12} \cdot q_1) = (q_1, a_2) - r_{12} \cdot (q_1, q_2) = 0$$

gdzie r_{12} jest przedstawiane wzorem:

$$r_{12} = \frac{(q_1,a_1)}{(q_1,q_1)} = \frac{q_1^T a_1}{q_1^T q_1} q_2 = a_2 - r_{12} \cdot q_1$$

I tak dalej zgodnie z zasadą, że następny następny wektor musi być ortogonalny do poprzednich wektorów. Dla macierzy dowolnych rozmiarów mamy:

$$q_i = a_i - \Sigma_{j=1}^{i-1} r_{ji} q_j$$

Korzystając z definicji iloczynu macierzy należy dalej przekształcając każde następne równanie na i-tą kolumnę macierzy Q w taki sposób aby wynikiem była i-ta kolumna macierzy A :

$$A_{mxn} = [a_1, a_2, ..., a_n] = [q_1, q_2, ..., q_n] \begin{bmatrix} 1 & r_{12} & \dots & r_{1n} \\ 0 & 1 & \dots & r_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} = QR$$

Ta metoda otrzymujemy macierz Q o kolumnach ortogonalnych i macierz R o diagonali złożonej z samych jedynek. Aby utworzyć macierz Q ortonormalna trzeba każdy wyraz kolumny macierzy Q podzielić przez jej normę. Następnie macierz R pomnożyć prawostronnie przez macierz diagonalna N, która na diagonali ma normy poszczególnych kolumn macierzy Q. Bardzo ważnym jest aby pamiętać, że powyższa metoda jest prawidłowa dla macierzy o liniowo niezależnych kolumnach czyli o pełnym rzędzie. W przypadku kiedy mamy do czynienia z macierzą o niepełnym rzędzie lepiej jest skorzystać z innych metod np. metody odbić Householdera. Algorytm opisany jest prawidłowy, jednak należy zauważyć, że ma niekorzystne własności numeryczne i dlatego też w zadaniu użyję zmodyfikowanego algorytmu Grama-Schmidta poznanego na wykładzie. Różnica polega na tym, iż wykonywane są te same operacje, jednak kolumny ortogonalne wyznaczane są etapami, gdzie po ortogonalizacji wszystkie są ortogonalne wobec niej.

Metoda ta została zaimplementowana w następujący sposób:

```
%rozkład QR (waski) macierzy zmodyfikowanym alg. Grama-Schmidta dla
%macierzy mxn (m>=n) o pelnym rzedzie, rzeczywistej lub zespolonej
%na podstawie książki prof. Tatjewskiego
function [Q,R] = qrZmodGS(A)
    [m n] = size(A);
    Q = zeros(m,n);
    R = zeros(n,n);
    d = zeros(1,n);
    %rozkład A z kolumnami Q ortogonalnymi
    for i=1:n
        R(i,i)=1;
        Q(:,i)=A(:,i);
        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 ortogonalne)
    for i=1:n
        dd = norm(Q(:,i));
        Q(:,i) = Q(:,i)/dd;
        R(i,i:n) = R(i,i:n)*dd;
    end
end
```

Metoda QR

Metoda QR polega na kolejnym wykorzystywaniu wcześniej już opisanego rozkładu QR. Metoda ta trochę różni się w zależności od tego czy rozpatrujemy macierz symetryczną czy też niesymetryczną. Dla macierzy symetrycznych, czyli takich gdzie wszystkie wartości parami leżące naprzeciwko siebie, względem diagonali, są równe. Dla jakiejś macierzy A tworzy się następujące macierze iteracyjnie - najpierw rozkład na Q oraz R, a następnie obliczenie kolejnego ciągu:

$$A^{(1)} = Q^{(1)}R^{(1)}$$
$$A^{(2)} = R^{(1)}Q^{(1)} \ (= Q^{(1)T}A^{(1)}Q^{(1)})$$
$$A^{(2)} = Q^{(2)}R^{(2)}$$
$$A^{(3)} = R^{(2)}Q^{(2)}(= Q^{(2)T}A^{(2)}Q^{(2)}) = Q^{(2)T}Q^{(1)T}A^{(1)}Q^{(1)}Q^{(2)}$$

Itd.

$$A^{(k)} = V^{-1}AV \ = diag\{\lambda_i\}$$

W ogólnym przypadku, czyli sama idea pojedynczego kroku metody wygląda następująco:

$$A^{(k)} = Q^{(k)}R^{(k)} \text{ (faktoryzacja)},$$
$$A^{(k+1)} = R^{(k)}Q^{(k)}$$

Ponieważ $Q^{(k)}$ jest ortogonalna, więc

$$R^{(k)} = (Q^{(k)})^{-1}A^{(k)} = (Q^{(k)})^{T}A^{(k)}, \text{ skąd mamy}$$
$$A^{(k+1)} = Q^{(k)T}A^{(k)}Q^{(k)}$$

Z tego mamy, iż macierz $A^{(k+1)}$ jest przekształconą przez podobieństwo macierzą $A^{(k)}$, więc ma te same wartości własne. Powyższy algorytm jest to **metoda bez przesunięć**.

Powyższy algorytm został osiągnięty następującą implementacją:

```
function [wartWlasne, iteracje, time, ok] = bezPrzesun(D, tolerance, imax)
    %imax - maksymalna liczba iteracji
    % ok - czy funkcja wykonala sie z przekroczeniem imax
    start = tic;
    ok = 1;
    i = 1;

    while i <= imax && max(max(D-diag(diag(D)))) > tolerance
        [Q1, R1] = qrZmodGS(D);
        D = R1*Q1; %macierz po przeksztalceniu
        i = i + 1;
    end

    if i > imax
        ok = 0;
    end
    iteracje = i;
    wartWlasne = diag(D); %wykstraktowanie wektora wartości własnych
    time = toc(start);
end
```

Dla macierzy A symetrycznej, macierz $A^{(k)}$ zbiega do macierzy diagonalnej $diag\{\lambda_i\}$.
Biorąc pod uwagę, że macierz posiada n wartości własnych o różnych modułach,

$$\left|\lambda_1\right| > \left|\lambda_2\right| > ... > |\lambda_n| > 0$$, wtedy można pokazać że:

$$a_{i,i}^{(k)} \rightarrow {}_{k\rightarrow\infty}\lambda_i,\ i-1,...,n$$

$$a_{i+1,i}^{(k)} \rightarrow {}_{k\rightarrow\infty}0,\ i=1,...,n-1$$

widzimy , że zbieżność elementu poddiagonalnego $a_{i+1,i}^{(k)}$ do zera jest liniowa z ilorazem zbieżności $\left|\frac{\lambda_{i+1}}{\lambda_i}\right|$, czyli

$$\frac{\left|a_{i+1,i}^{(k+1)}\right|}{\left|a_{i+1,i}^{(k)}\right|} \approx \left|\frac{\lambda_{i+1}}{\lambda_i}\right|,$$

Jeżeli wartości własne leżą blisko siebie, to metoda jest wolno zbieżna, aby poprawić szybkość zbieżności, stosuje się algorytm **metody QR z przesunięciami**. W metodzie QR z przesunięciami wyliczane są kolejne wartości własne od końca i można ją przybliżyć poniższymi krokami:

1) Znajdź wartość własną λ_n jako najbliższą wartość własną podmacierzy $2x2$ z prawego dolnego rogu macierzy $A^{(k)}$ wyznaczając wartości własne. W tym przypadku wartości własne są pierwiastkami wielomianu drugiego stopnia o współczynnikach wielomianu charakterystycznego.
2) Należy opuścić ostatni i wiersz i ostatnią kolumnę aktualnej macierzy $A^{(k)}$ - proces deflacji.
3) Znajdź następną wartość własną λ_{n-1}. W celu osiągnięcia tego kroku należy przekształcić macierz $A_{n-1}^{(k)}$, aż do momentu uzyskania $e_{n-2}^{(k)} = 0$. A literujemy powyższy schemat dopóki nie uzyskamy wyzerowanych wszystkich elementów poza elementem diagonalnym. Realizowany przez pętle while w dalszej części, gdzie przedstawiony zostanie kod.
4) Dopóki nie uzyskamy wszystkich wartości własnych (uwzględniając precyzję obliczeń) powtarza się kroki 2) oraz 3).

Biorąc pod uwagę uwarunkowanie zadania należy zauważyć, że w zadaniu macierze symetryczne i niesymetryczne generowane są losowo. Dla Symetrycznych i rzeczywistych macierzy trzeba powiedzieć, że dla X będącym symetryczną macierzą o wartościach rzeczywistych chodzi warunek $cond(X) = 1$. Wynikający z twierdzenia Bauera-Frike'a, które również mówi o tym, że dla macierzy niediagonalizowalnych uwarunkowanie zadania, a zatem uwarunkowanie wartości może być dowolnie duże.

Powyższy algorytm osiągnąłem poniższą implementacja:

```
function [wartWlasne, iteracje, time, ok] = zPrzesun(A,tolerance,imax)
    % ok - czy funkcja wykonala sie z przekroczeniem imax
    % tolerance - toleracnaj jako górna granica wartości elementów zerowanych
    start = tic;
    ok = 1;
    n = size(A,1);
    wartWlasne = diag(zeros(n));
    iteracje = 0;
    INITIALsubmatrix = A;
    for k=n:-1:2
        DK = INITIALsubmatrix(1:k, 1:k); %macierz potrzebna do wyznaczenia wartosci wl
        i = 0;

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

            ev = roots([1, -(DK(k-1,k-1)+DK(k,k)), DK(k,k)*DK(k-1,k-1)-DK(k,k-1)*DK(k-1,k)]);
            if abs(ev(1)-DK(k,k)) < abs(ev(2)-DK(k,k))
                shift = ev(1); % nasze przesuniecie jako najbliższa DK(k,k) wartosc
                               % wlasna analizowanej macierzy 2x2
            else
                shift = ev(2);
            end
            DK = DK - eye(k)*shift; %nasza macierz przesunięta
            [Q1, R1] = qrZmodGS(DK); %faktoryzacja QR
            DK = R1*Q1 + eye(k)*shift; %macierz przekształcona
            i = i+1;

            iteracje = iteracje + 1;
        end

        if i > imax
            ok = 0;
            break;
        end
        wartWlasne(k) = DK(k,k);

        if k>2
            INITIALsubmatrix = DK(1:k-1,1:k-1); %definicja macierzy
        else
            wartWlasne(1) = DK(1,1); %ostatnia wartosc wlasna
        end
    end
    time = toc(start);
end
```

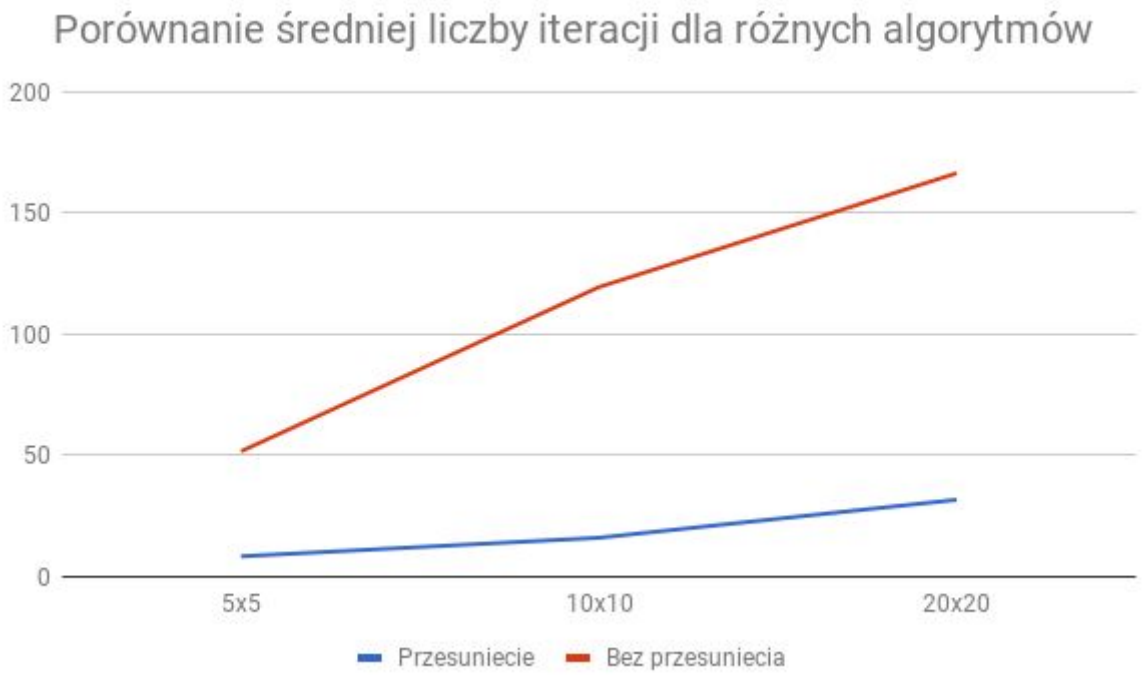
Poniższa implementacja przedstawia główne ciało programu:

```
function [] = Zadaniel(czyPrzesun,czySym)
    % czyPrzesun - wartosc 1 jesli wersja z przesunieciami
    % czySym - wartosc 1 jestli macierz symetryczna

    iterQR = 0; %liczba wartosci wlasnej QR bez przesuniec
    iterQRS = 0; %liczba wartosci wlasnej QR z przesunieciami

    timeQR = 0; %czas wykonania alg QR bez przesuniec
    timeQRS = 0; %czas wykonania alg QR z przesunieciami
    timeE = 0; %czas wykonania alg eig() wbudowanym w MATLAB

    SIZE = 20; %rozmiar maciery
    matrixNumberR = 30; %ilosc losowych macierzy

    ILEMACQR = 0;
    ILEMACQRS = 0;

    %Generowanie danych
    for i=1:matrixNumberR

        A = genA(SIZE,czySym);
        tolerance = 0.0000001;
        imax = 200;

        start = tic;
        [~,D] = eig(A);
        timeEig = toc(start);
        timeE = timeE + timeEig;

        if czyPrzesun == 1 %QR z przesunieciami (eignes - wartWlasne)
            [eigens, iteracje, time, ok] = zPrzesun(A, tolerance, imax);
            if ok == 1
                ILEMACQRS = ILEMACQRS + 1;
                iterQRS = iterQRS + iteracje;
                timeQRS = timeQRS + time;
            end
        else %QR bez przesuniec  (eigens - wartWlasne)
            [eigens, iteracje, time, ok] = bezPrzesun(A, tolerance, imax);
            if ok == 1
                ILEMACQR = ILEMACQR + 1;
                iterQR = iterQR + iteracje;
                timeQR = timeQR + time;
            end
        end
    end
```

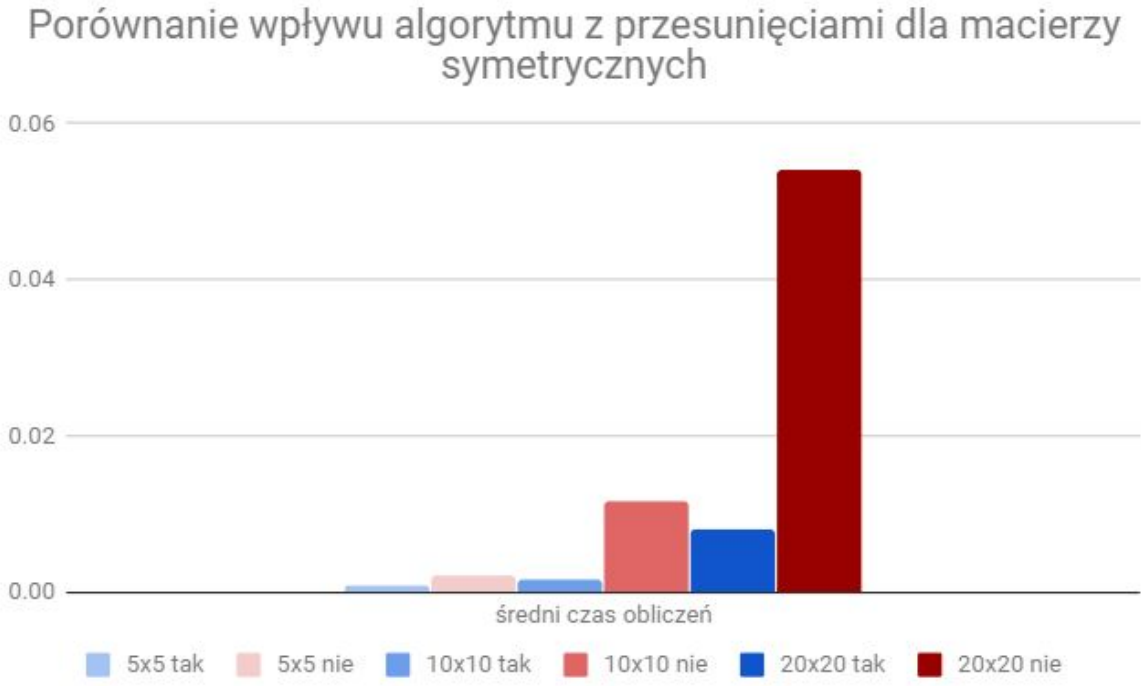
```
    fprintf('Ilosc macierzy: %d\n',matrixNumberR);
    fprintf('Wielkosc macierzy: %d\n',SIZE);
    SREDNIAczasEig = timeE / matrixNumberR;

    if czyPrzesun == 1
        %Wyniki z przesunieciami:
        SREDNIAQRSI = iterQRS / ILEMACQRS;
        SREDNIAczasQRS = timeQRS / ILEMACQRS;
        fprintf('Z przesunieciami:\n');
        fprintf('Ilosc zkonczonych sukcesem: %d\n', ILEMACQRS);
        fprintf('Srednia ilosci iteracji %d\n',SREDNIAQRSI);
        fprintf('Sredni czas obliczen %d\n',SREDNIAczasQRS);
    else
        %Wyniki bez przesuniec:
        SREDNIAQRI = iterQR / ILEMACQR;
        SREDNIAczasQR = timeQR / ILEMACQR;
        fprintf('Bez przesuniec:\n');
        fprintf('Ilosc zkonczonych sukcesem: %d\n', ILEMACQR);
        fprintf('Srednia ilosci iteracji %d\n',SREDNIAQRI);
        fprintf('Sredni czas obliczen %d\n',SREDNIAczasQR);
    end

    fprintf('Sredni czas obliczen eig %d\n',SREDNIAczasEig);
    d = diag(D);
    disp(sort(d));
    disp(sort(eigens));
end
```

Oto moje wyniki dla macierzy symetrycznych:

Wymiary macierzy	Przesunięcie	średnia ilość iteracji	ilość zakończonych sukcesem	średni czas obliczeń	różnica z funkcją eig
5x5	tak	8.166667	30	0.0007530231	0.0007530231
5x5	nie	51.55556	27	0.001982526	0.001982526
10x10	tak	15.83333	30	0.001647260	0.001647260
10x10	nie	119.4167	12	0.01154099	0.01154099
20x20	tak	31.53333	30	0.007959675	0.007959675
20x20	nie	166.5	2	0.0541243	0.0541243


Porównanie średniej liczby iteracji dla różnych algorytmów
200
150
100
50
0
5x5
10x10
20x20
Przesuniecie
Bez przesuniecia


Porównanie średniego czasu obliczeń dla różnych algorytmów
0.06
0.04
0.02
0.00
5x5
10x10
20x20
Przesuniecie
Bez przesuniecia



Oto moje wyniki dla macierzy niesymetrycznych:

Wymiary macierzy	Przesunięcie	średnia ilość iteracji	ilość zakończonych sukcesem	średni czas obliczeń	różnica z funkcją eig
5x5	tak	10.86667	30	0.001348438	0.001348438
10x10	tak	23.83333	30	0.01455645	0.01455645
20x20	tak	50	30	0.0319175	0.0319175

Porównanie wartości własnych osiągniętych w zadaniu z metodą eig():

Symetryczna macierz 10x10 metoda bez przesunięć	
moje rozwiązanie	eig()
-2.1223	-2.1223
-1.2218	-1.2218
-1.0534	-1.0534
-0.3551	-0.3551
-0.0163	-0.0163
0.5927	0.5927
1.0885	1.0885
1.5954	1.5954
3.0941	3.0941
10.2983	10.2983

Symetryczna macierz 20x20 metoda z przesunięciami	
moje rozwiązanie	eig()
-3.369	-3.369
-3.1753	-3.1753
-1.8081	-1.8081
-1.5051	-1.5051
-1.2962	-1.2962
-1.0394	-1.0394
-0.7083	-0.7083
-0.4368	-0.4368
-0.2239	-0.2239
0.112	0.112
0.2798	0.2798
0.6185	0.6185
0.9226	0.9226
1.474	1.474
1.6065	1.6065
2.3632	2.3632
2.5532	2.5532
2.9707	2.9707
3.2614	3.2614
20.3448	20.3448

Symetryczna macierz 20x20 metoda bez przesunięć	
moje rozwiązanie	eig()
0.1559 - 0.0000i	0.1559 - 0.0000i
-0.0593 + 0.3326i	-0.0593 + 0.3326i
-0.0593 - 0.3326i	-0.0593 - 0.3326i
-0.2996 - 0.3263i	-0.2996 - 0.3263i
-0.2996 + 0.3263i	-0.2996 + 0.3263i
0.4973 - 0.5502i	0.4973 - 0.5502i
0.4973 + 0.5502i	0.4973 + 0.5502i
0.9077 - 0.0775i	0.9077 - 0.0775i
0.9077 + 0.0775i	0.9077 + 0.0775i
0.1438 + 0.9289i	0.1438 + 0.9289i
0.1438 - 0.9289i	0.1438 - 0.9289i

-0.3732 - 0.9192i	-0.3732 - 0.9192i
-0.3732 + 0.9192i	-0.3732 + 0.9192i
-1.0272 - 0.2441i	-1.0272 - 0.2441i
-1.0272 + 0.2441i	-1.0272 + 0.2441i
-0.9396 - 0.5960i	-0.9396 - 0.5960i
-0.9396 + 0.5960i	-0.9396 + 0.5960i
0.9070 + 0.8741i	0.9070 + 0.8741i
0.9070 - 0.8741i	0.9070 - 0.8741i
9.8429 - 0.0000i	9.8429 - 0.0000i

Symetryczna macierz 5x5 metoda z przesunięciami	
moje rozwiązanie	eig()
-1.0256	-1.0256
0.0074	0.0074
0.3628	0.3628
1.3012	1.3012
4.9575	4.9575

Jak możemy zauważyć wartości własne we wszystkich przypadkach wyszły identyczne. Co potwierdza poprawność zaimplementowanego rozwiązania, a jedyną różnicą jest kolejność w jakiej wartości własne znajdują się w obu wektorach. Co wynika z zaimplementowanego algorytmu.

Wnioski:
Jak możemy wyczytać z przedstawionych wyników, tak jak się spodziewaliśmy - dla macierzy symetrycznych funkcje z przesunięciami otrzymują o wiele lepsze wyniki. Na pewno możemy stwierdzić, że są dokładniejsze, osiągane w krótszym czasie, a co się z tym wiążę w mniejszej liczbie iteracji od algorytmu z przesunięciami. Niestety metoda ta jest o wiele trudniejsza w implementacji. Biorąc pod uwagę porównanie z funkcją eig() to różnica w wyznaczonych wartościach między nim a algorytmem z przesunięciami rośnie wraz z rozmiarem macierzy, ale nieznacznie. Widać również że dla algorytmu bez przesunięć przy coraz większych macierzach bardzo rzadko udaje się otrzymać wynik w ograniczonej liczbie iteracji. Biorąc pod uwagę macierze niesymetryczne to funkcja z przesunięciami poradziła sobie z wyznaczeniem we wszystkich przypadkach, a w porównaniu z funkcja eig() różnica wartości była znaczna. Wynika to ze złego uwarunkowania macierzy niesymetrycznej względem algorytmu z przesunięciami. Jeżeli mamy do czynienia z macierzą niezdiagnozowana to uwarunkowanie drastycznie się pogarsza.

Zadanie 2

Zadanie to polega na wyznaczeniu funkcji wielomianowej najlepiej aproksymującej podane dane przy użyciu metody najmniejszych kwadratów. Aby rozwiązać to zadanie należy wykorzystać układ równań normalnych oraz układ równań liniowych wynikającą z rozkładu QR macierzy układu równań problemu.

Biorąc na warsztat przypadek gdy macierz $A \in R^{mxn}$, $k \leq n$, gdzie k oznaczamy jako rząd macierzy. W podprzypadku gdzie $k = n$, czyli macierz A jest pełnego rzędu. Wtedy możemy zauważyć, że macierz $A^T A$ jest symetryczna i kwadratowa, a dodatkowo ma wszystkie wartości własne dodatnie (dodatnie określona). Dane te implikują nam to, iż funkcja $J(x)$ minimalizująca funkcję kwadratową jest zadaniem równoważnym do liniowego zadania najmniejszych kwadratów:

$$J(x) = (b - Ax)^T (b - Ax) = x^T A^T Ax - 2x^T A^T b + b^T b$$

Jest ściśle wypukła i ma jednoznaczne minimum w punkcie zerowania gradientu funkcji (spełnienie warunku koniecznego minimum):

$$J'(x) = 2A^T A\, x - 2A^T b = 0$$

A w konsekwencji wynika z tego dany układ równań liniowych nazywamy **układem równań normalnych z jednoznacznym rozwiązaniem**:

$$A^T Ax = A^T b$$

Dla słabo uwarunkowanych macierzy stosuje się metodę, wykorzystującą rozkład QR . W tym zadaniu, w przeciwieństwie do poprzedniego możemy korzystać z funkcji $[Q, R] = qr(A)$ = qr, więc w tym zadaniu pominę objaśnienie mechanizmu tego rozkładu. Po skorzystaniu z **rozkładu QR**, nasze powyższe równanie przybiera następującą postać :

$$R^T Q^T QRx = R^T Q^T b$$

Równanie to, ze względu na ortonormalność kolumn macierzy Q i nieosobliwość macierzy R , możemy sprowadzić do postaci znacznie “przyjemniejszej”:

$$Rx = Q^T b$$

Przy tak zwany rozkładzie wąskim:

$$A_{mxn} = Q_{mxn} R_{nxn}$$

Ponownie, rozwiązanie tego układu równań jest rozwiązaniem liniowego zadania najmniejszych kwadratów. W języku matlab, do rozwiązania tego typu równań stosuje się operator “\”, który zastosowałem również w moim programie.
Implementacja metody QR znajduje się w poprzednim zadaniu, więc posłużę się nią do obliczenia wyniku.

```
function [] = Zadanie2(STOPIEN)
    x = -5:1:5;
    x2 = -5:0.1:5;
    y = [-7.7743 -0.2235 1.9026 0.6572 0.1165 -1.8144 -1.0968 -0.8261 1.3327 6.1857 8.2891];
    x=x';
    y=y';
    a = najmnKwadratow(STOPIEN,x,y);

    eukNorm = euklidesNorm(a,y,x);
    czebNorm = czebyszewNorm(a,y,x);

    fprintf('Błąd aproksymacji w normie Czebyszewa dla równania rzędu %d wynosi %d\n',STOPIEN,czebNorm);
    fprintf('Błąd aproksymacji w normie Euklidesowej dla równania rzędu %d wynosi %d\n',STOPIEN,eukNorm);

    %Rysowanie wykresu
    scatter(x,y,'filled')
    hold on
    p = polyval(a,x2);
    plot(x2,p);
    grid;
    title('Aproksymacja przy stopniu równym 10 - równania normalne');
    xlabel('x');
    ylabel('y');
    legend('Punkty oryginalne','funkcja aproksymująca')
end
```

```
function [a] = najmnKwadratow(n, x, y)
    % n - zadany stopien wielomianu
    % x - wektor argumentow
    % y - wektor wartosci

    % a - wektor wyznaczonych wspolczynnikow


    [m, ~] = size(x); %pobranie rozmiarow
    A = zeros(m, n+1);

    for i=1:m %wiersze
        for j=0:n %kolumny
            A(i,n+1-j) = x(i)^(j); %wypełniamy odpowiednimi wartooeciami potegi x
        end
    end

    %Rozkład QR
    [Q, R] = qrZmodGS(A); %rozklad qr metoda g-s z poprzedniego zadania
    a = R \ (Q'*y);

    %Uklad równań normalnych
    %a = (A'*A)\(A'*y); %rozwiazanie ukaadu równan
end
```

Błędy aproksymacji i ich normy:
Jak widzimy w niektórych przypadkach (szczególnie początkowych) aproksymacja nie była idealna. Błędy aproksymacji są to wartości, które wynikają z niedokładności aproksymowania. Oblicza się je jako moduł z różnicy pomiędzy wartością oryginalną, a wartością wynikającą z aproksymowanej funkcji, biorąc pod uwagę, że rozpatrujemy aproksymacje dyskretną.

$$p(x_i) - \text{wartość funkcji aproksymującej w punkcie } x_i$$
$$f_i - \text{wartość oryginalna aproksymowana funkcją}$$
$$\left|p(x_i) - f_i\right| - \text{błąd aproksymacji}$$

Można również przedstawić wyniki jako norma błędu aproksymacji. Moim zadaniem było przedstawienie ich w dwóch normach. Pierwsza z nich to **norma euklidesowa** przedstawiająca się w następujący sposób:

$$\varepsilon = \left\|p(x_i) - f_i\right\|_{i=1,2,\ldots,n} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p(x_i) - f_i)^2}$$

Powyższą metodę osiągnąłem poniższą implementacją:

```
function [err] = euklidesNorm(a,f,x)
    p = polyval(a,x);
    [s,~] = size(f);
    i = 1;
    sum=0;
    while i<=s
        sum = sum + (p(i)-f(i))^2;
        i = i + 1;
    end
    err = sqrt((1/s)*sum);
end
```

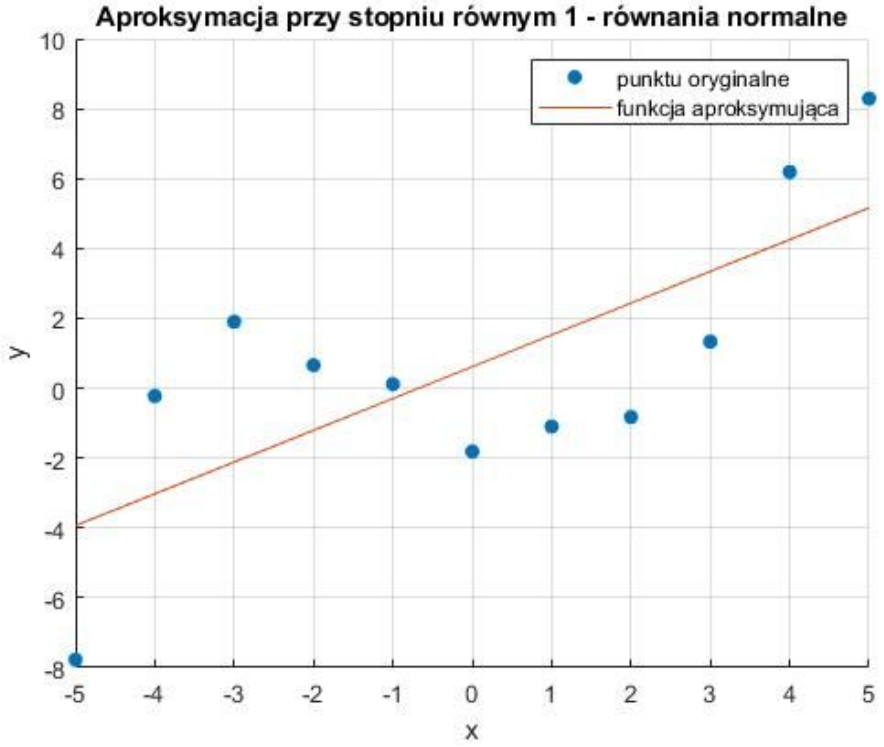
Druga z nich jest to tak zwane **norma Czebyszewa**, wyliczana jako wartość największa ze wszystkich błędów aproksymacji w punktach dyskretnych aproksymowanego zbioru.

$$\varepsilon = \left\|p(x_i) - f_i\right\|_{i=1,2,\ldots,n} = max_i \left|p(x_i) - f_i\right|$$

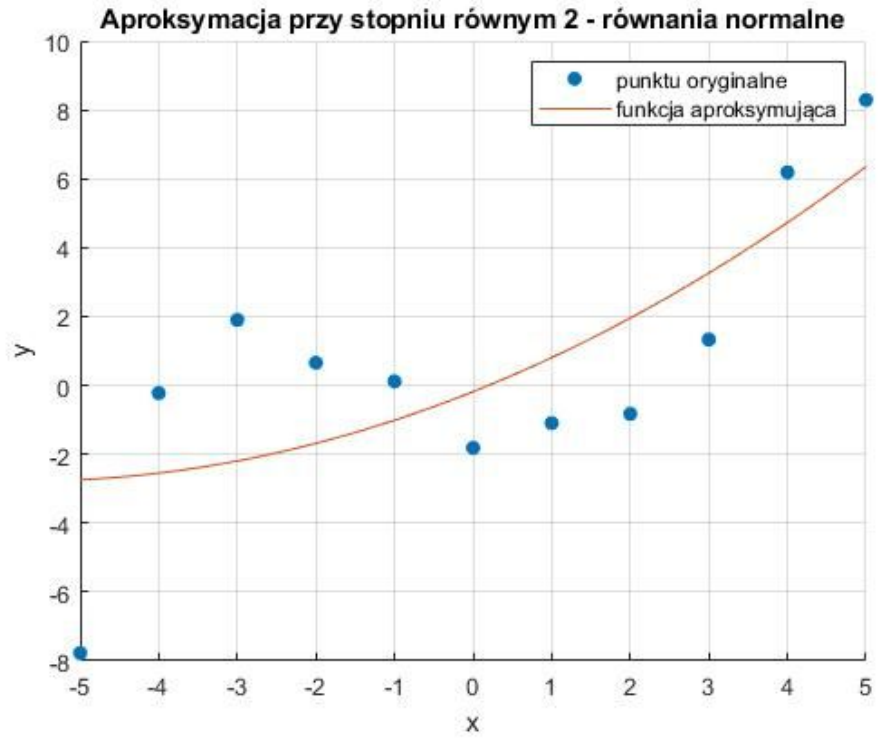
Powyższą metodę osiągnąłem poniższą implementacją:

```
function [maxError] = czebyszewNorm(a,f,x)
    p = polyval(a,x);
    [s,~] = size(f);
    i = 1;
    maxError = 0;
    while i<=s
        err = abs(p(i)-f(i));
        if err > maxError
            maxError = err;
        end
        i = i + 1;
    end
end
```

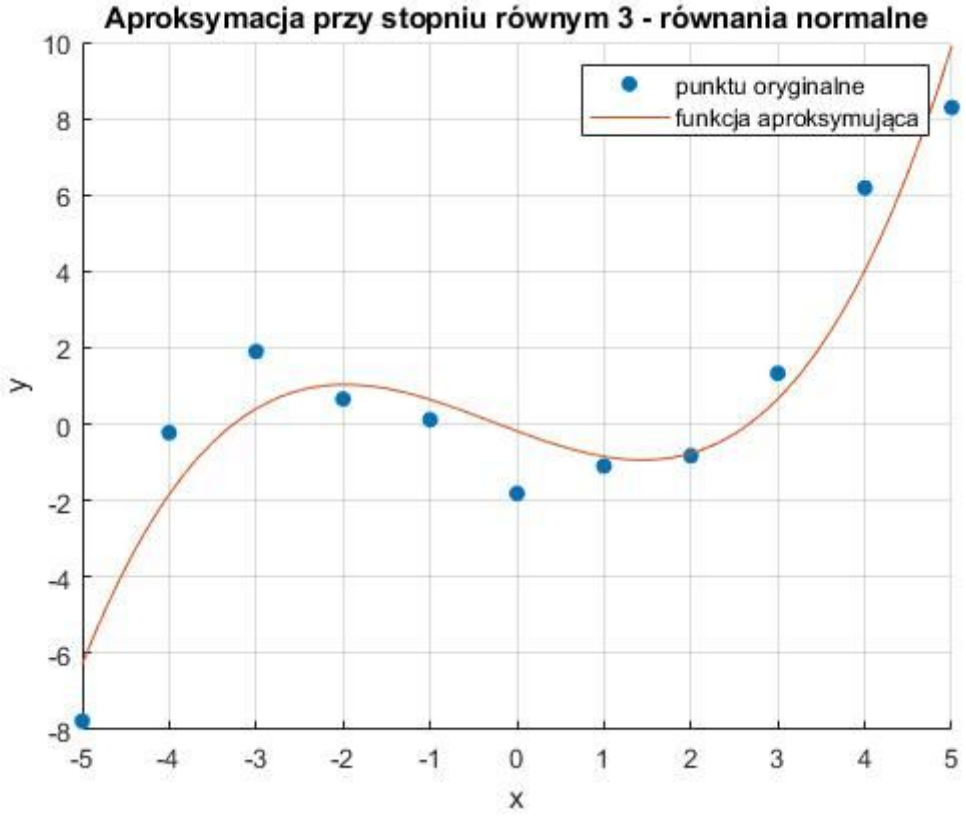
Wykresy z rozwiązaniem metodą równań normalnych:



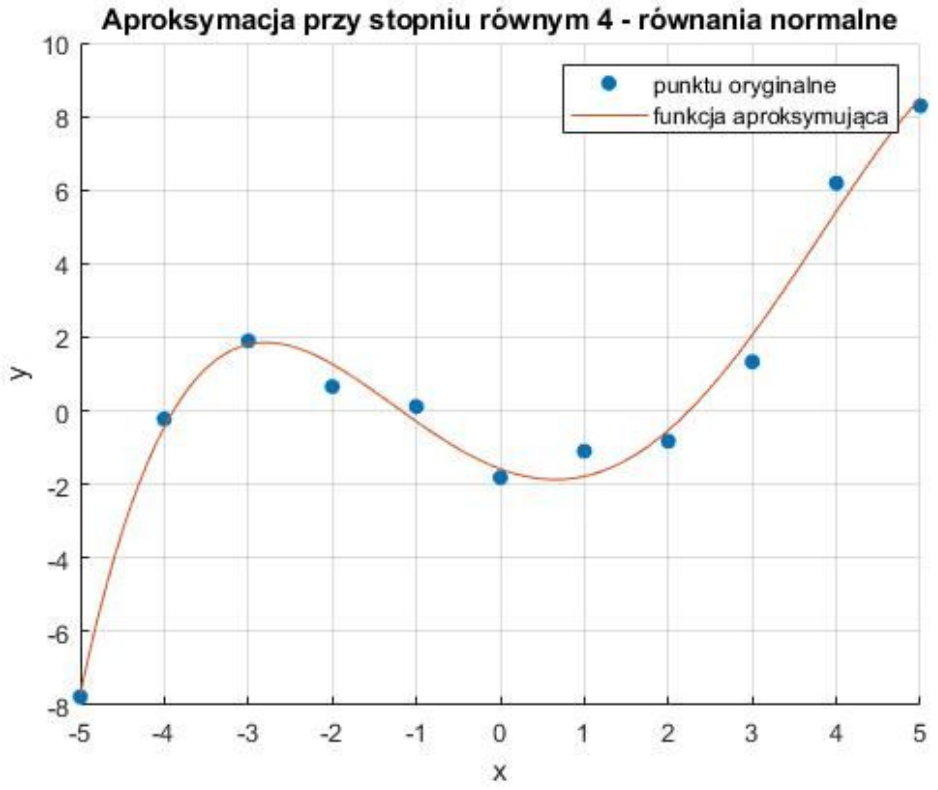
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 1 wynosi 4.018105e+00
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 1 wynosi 2.752096e+00



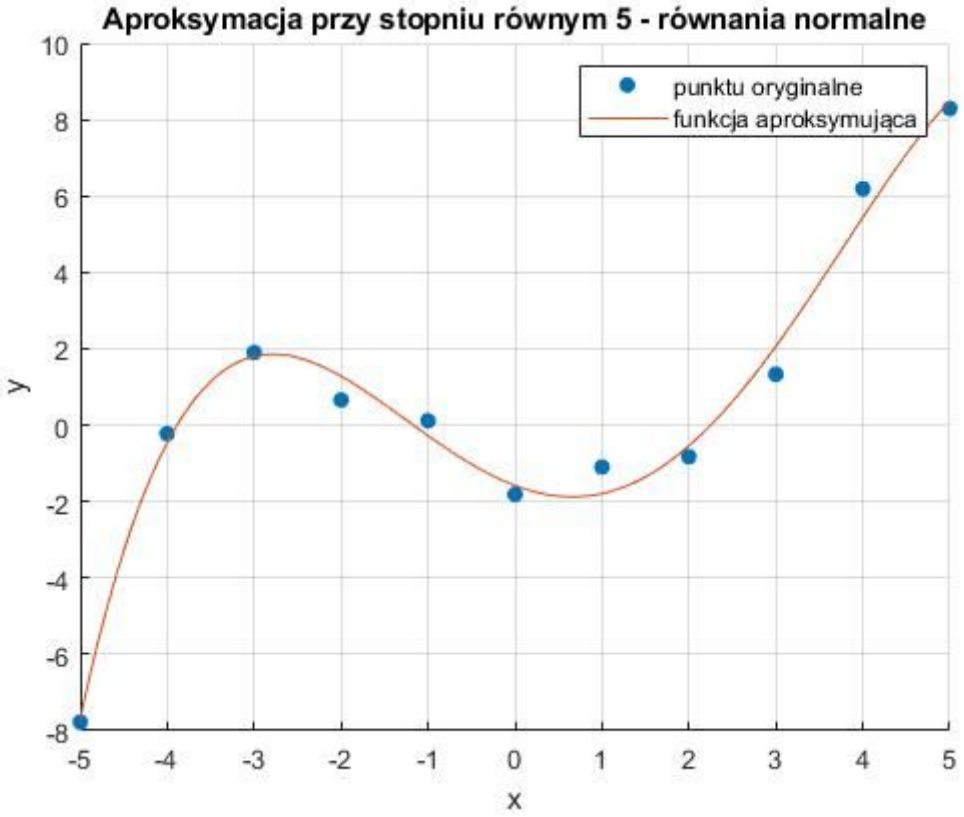
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 2 wynosi 5.032452e+00
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 2 wynosi 2.660946e+00



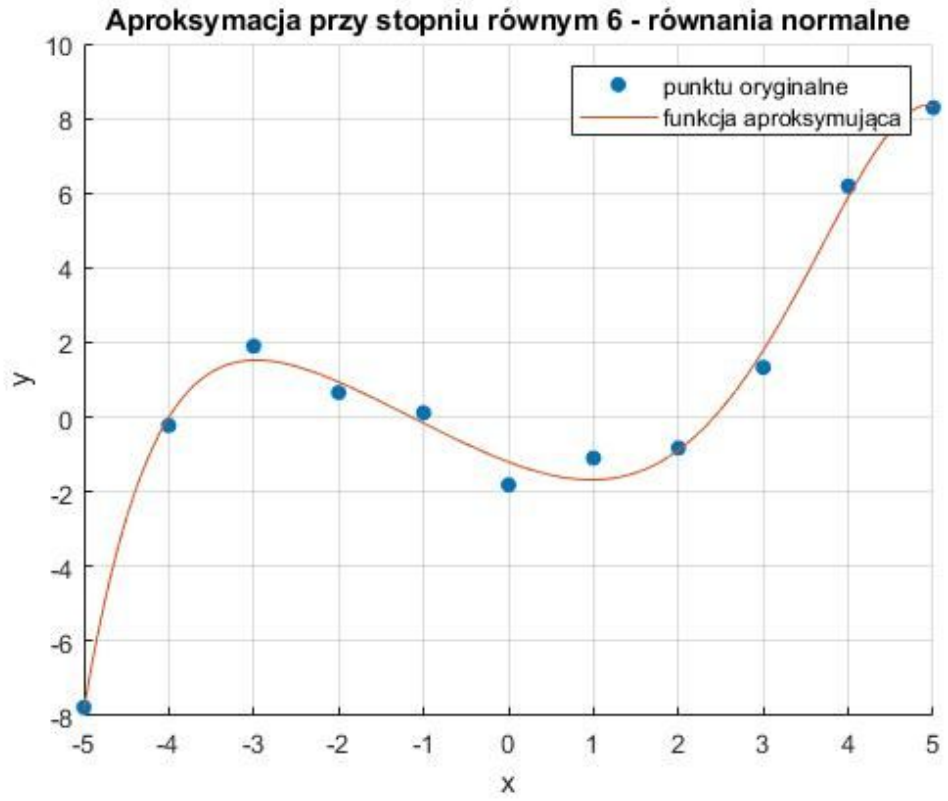
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 3 wynosi 2.165493e+00
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 3 wynosi 1.277366e+00



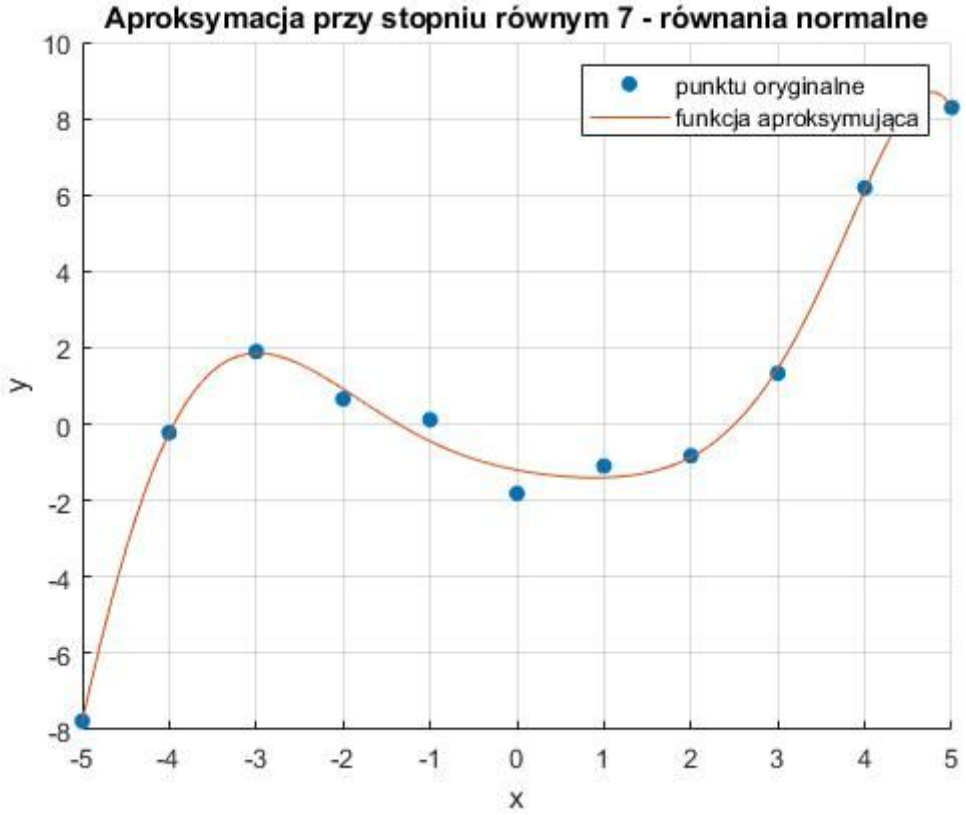
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 4 wynosi 7.654601e-01
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 4 wynosi 4.648034e-01



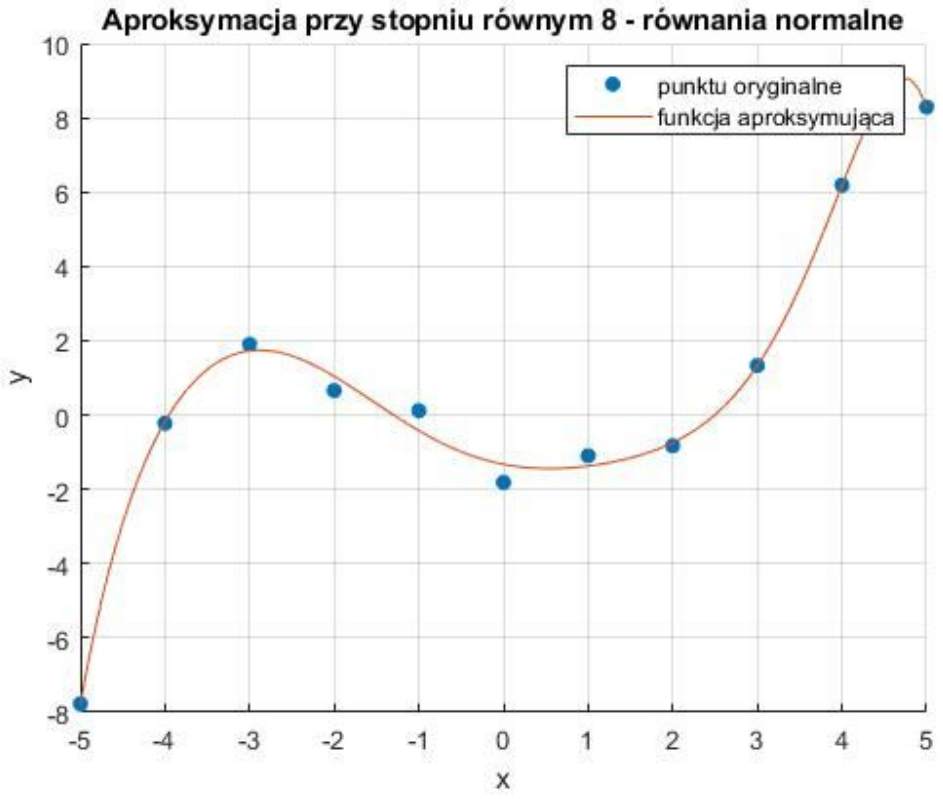
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 5 wynosi 7.469409e-01
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 5 wynosi 4.646580e-01



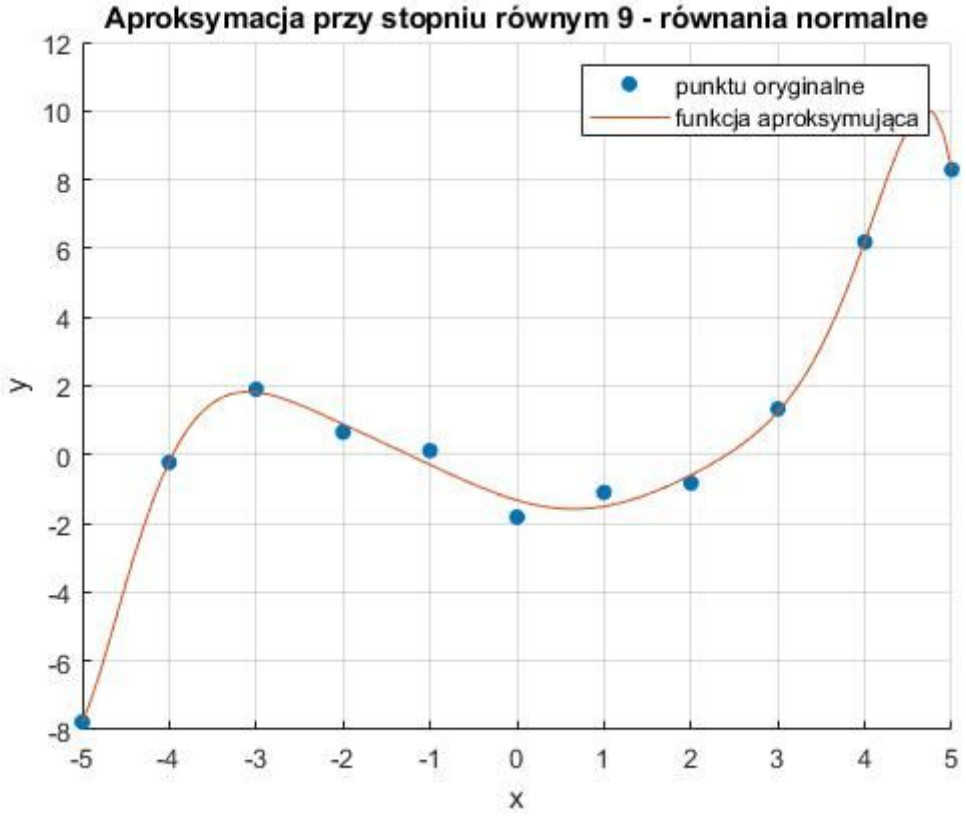
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 6 wynosi 6.118012e-01
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 6 wynosi 3.524390e-01



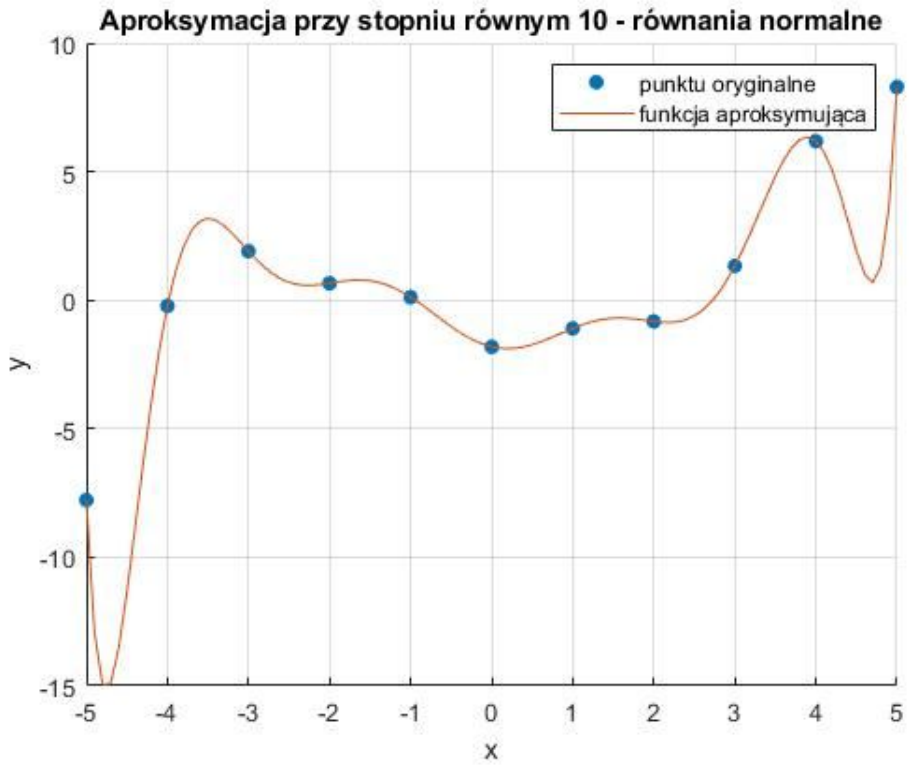
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 7 wynosi 6.118012e-01
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 7 wynosi 2.817600e-01



Błąd aproksymacji w normie Czebyszewa dla równania rzędu 8 wynosi 5.381672e-01
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 8 wynosi 2.678049e-01



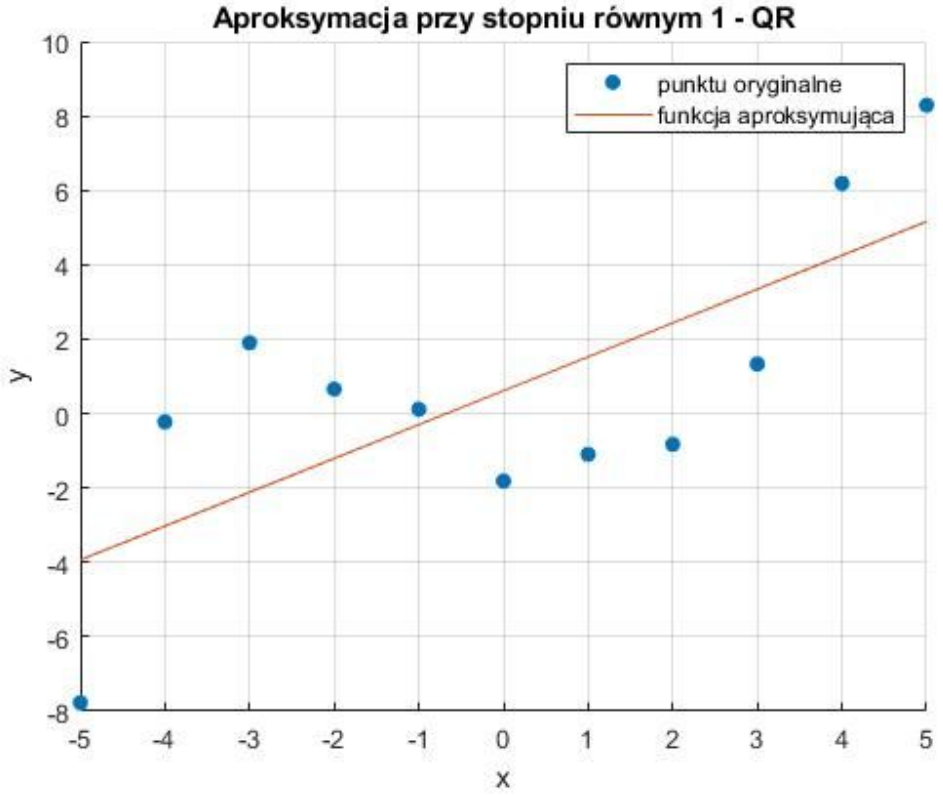
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 9 wynosi 4.884551e-01
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 9 wynosi 2.512043e-01



Błąd aproksymacji w normie Czebyszewa dla równania rzędu 10 wynosi 1.139605e-09

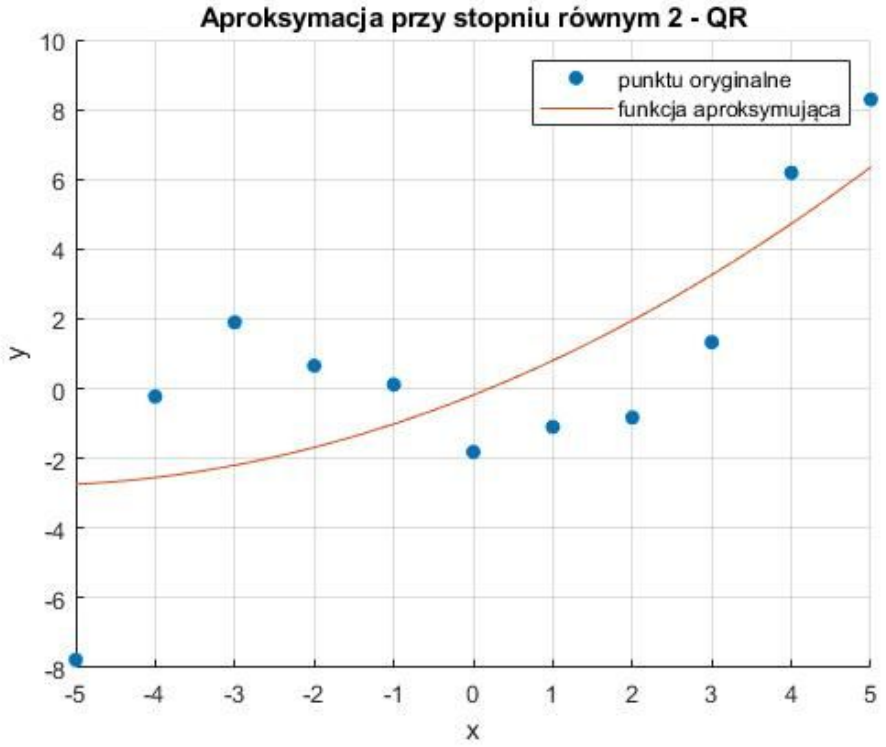
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 10 wynosi 5.609315e-10

Wykresy z rozwiązaniem metodą rozkładu QR:

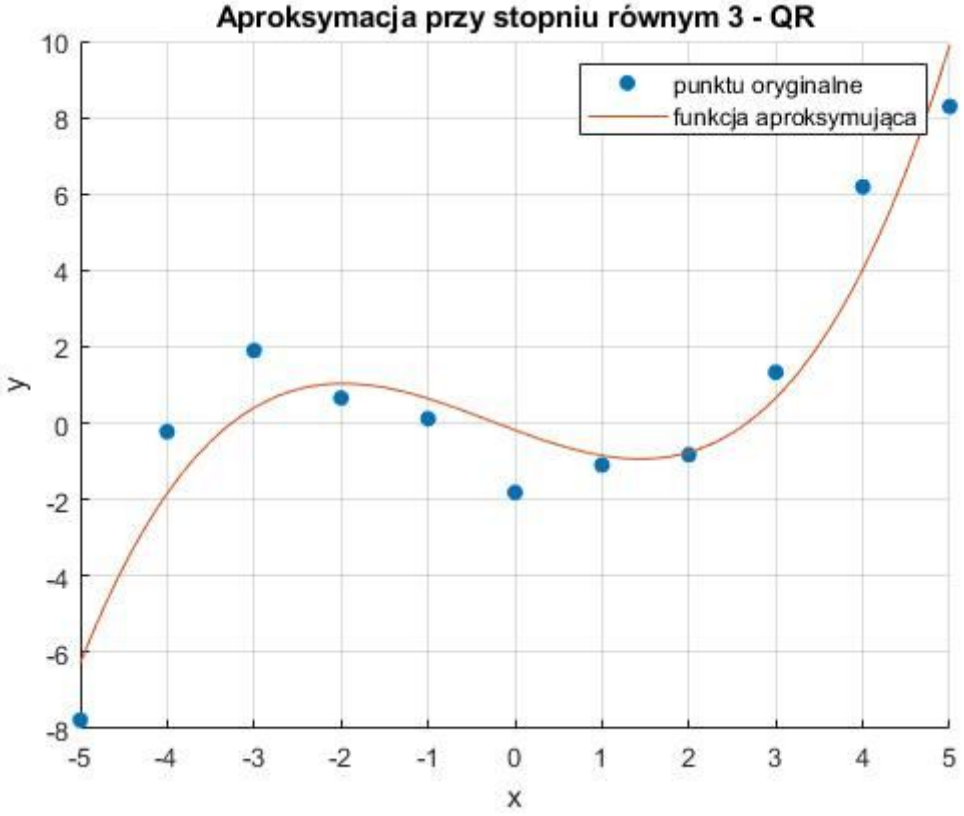


Błąd aproksymacji w normie Czebyszewa dla równania rzędu 1 wynosi 4.018105e+00

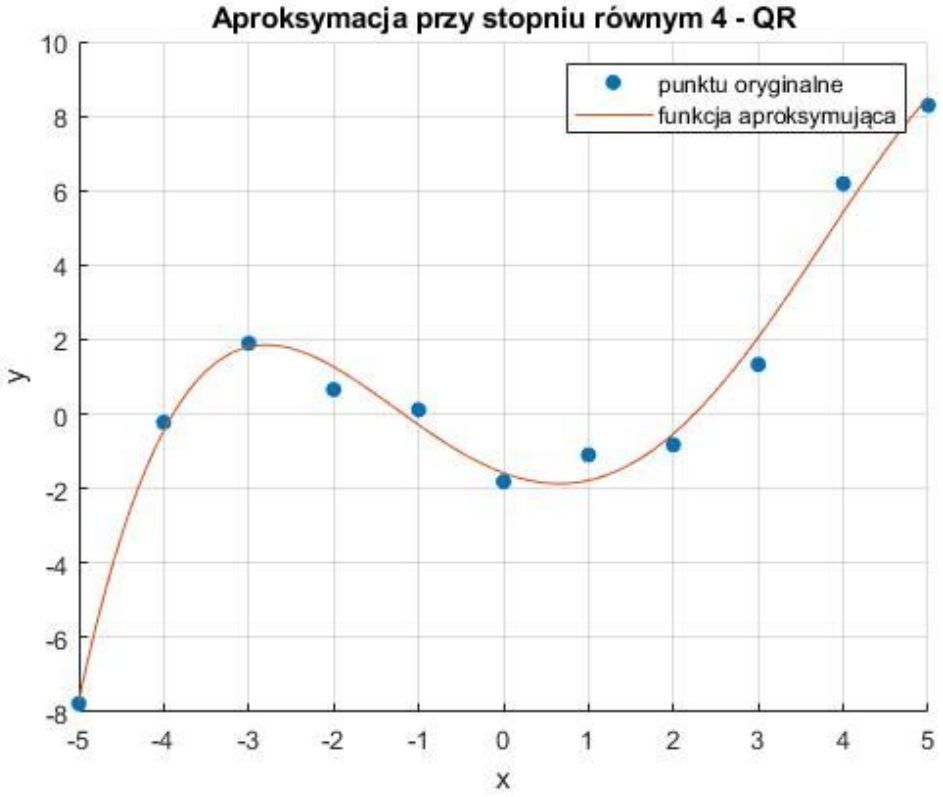
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 1 wynosi 2.752096e+00



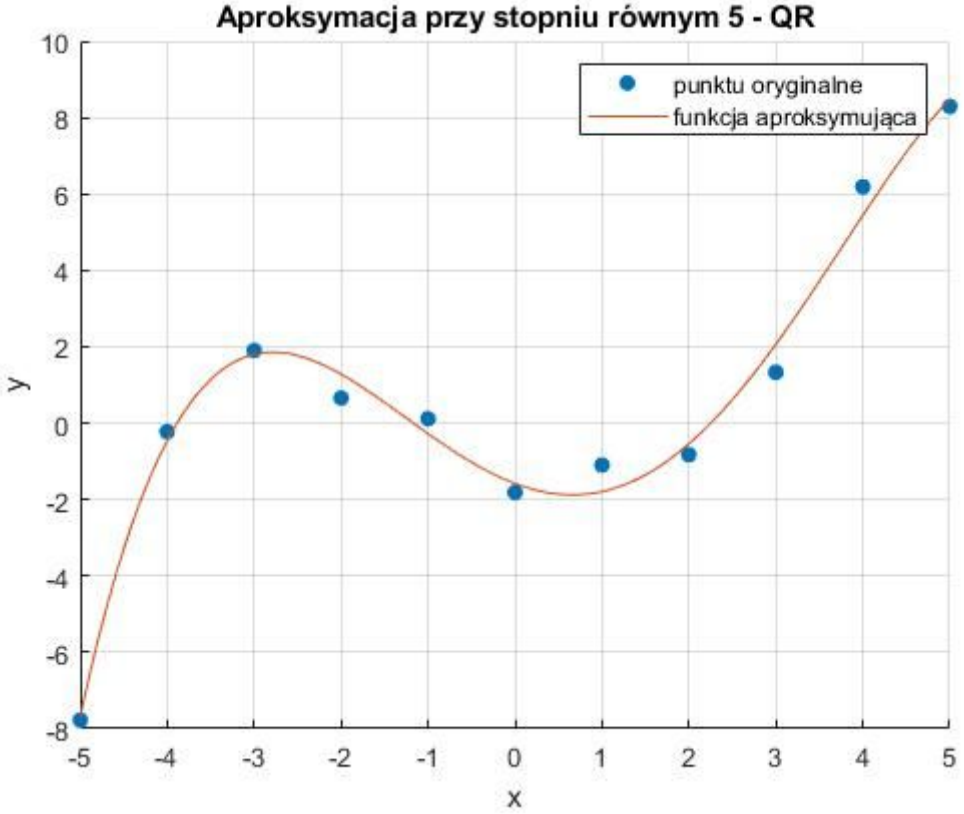
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 2 wynosi 5.032452e+00
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 2 wynosi 2.660946e+00



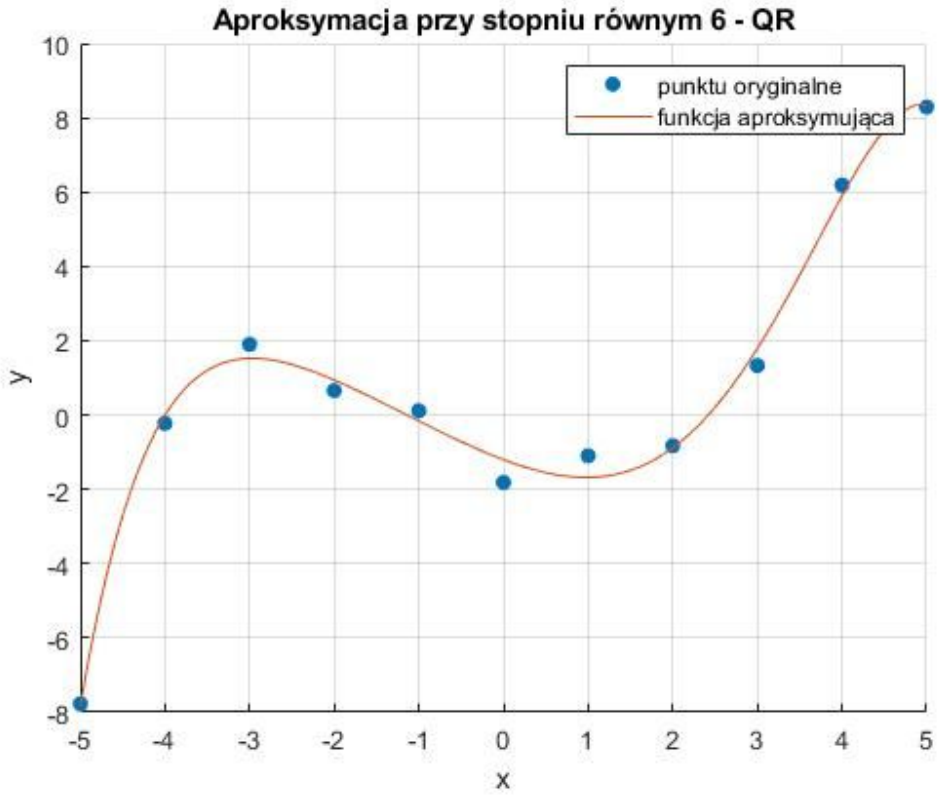
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 3 wynosi 2.165493e+00
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 3 wynosi 1.277366e+00



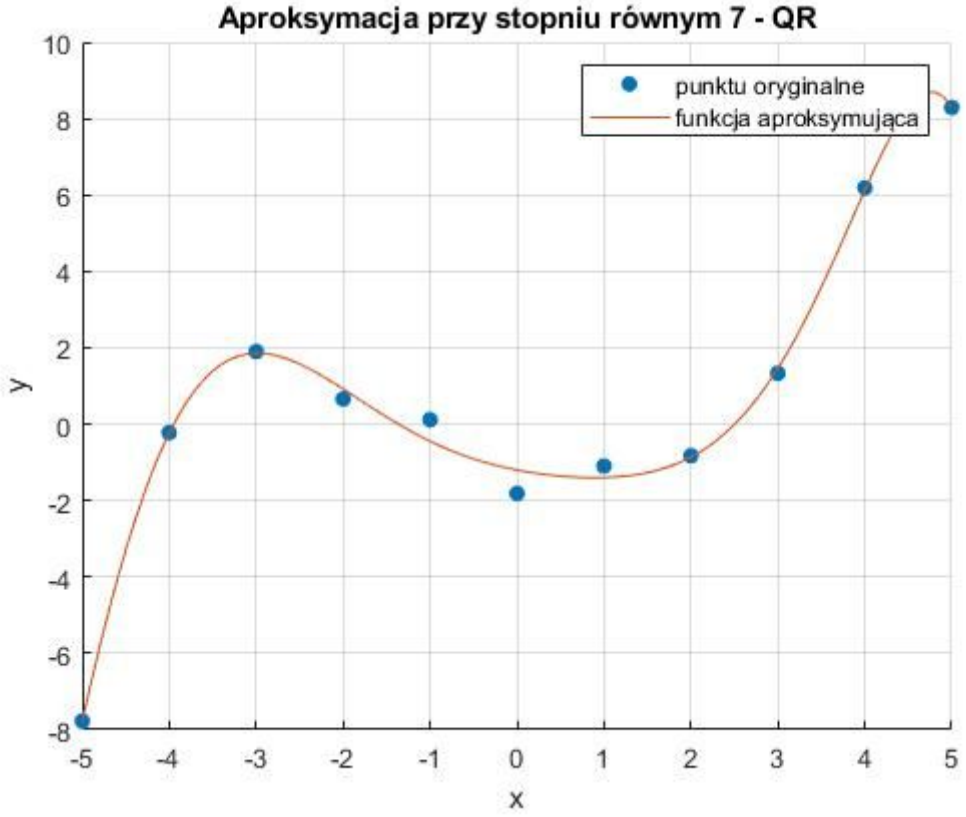
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 4 wynosi 7.654601e-01
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 4 wynosi 4.648034e-01



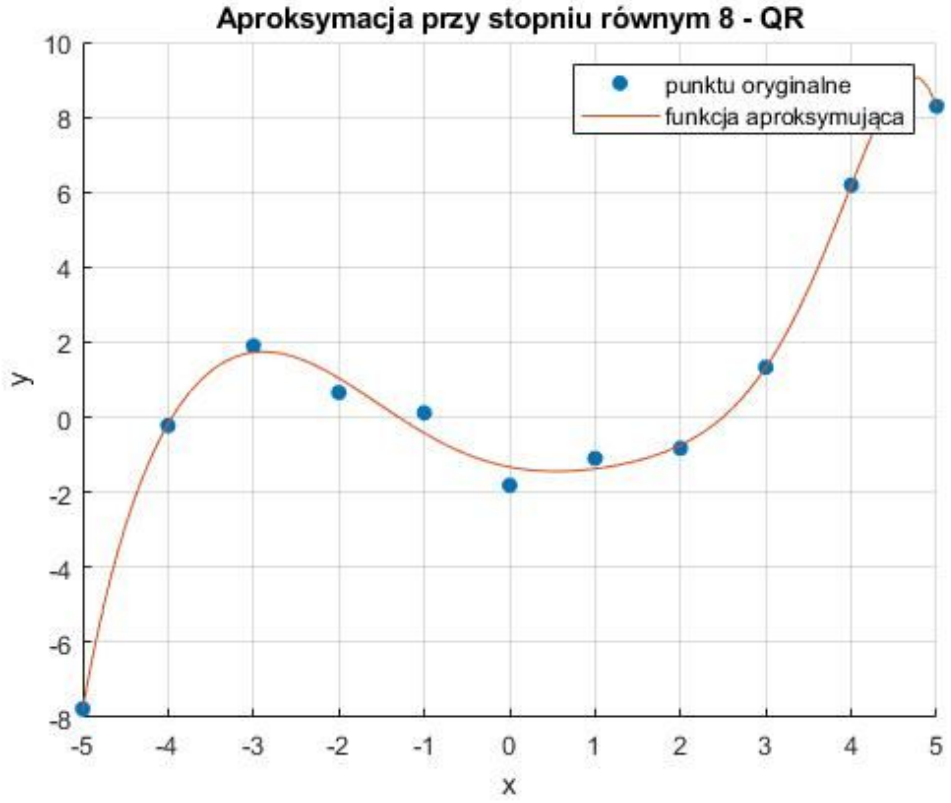
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 5 wynosi 7.469409e-01
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 5 wynosi 4.646580e-01



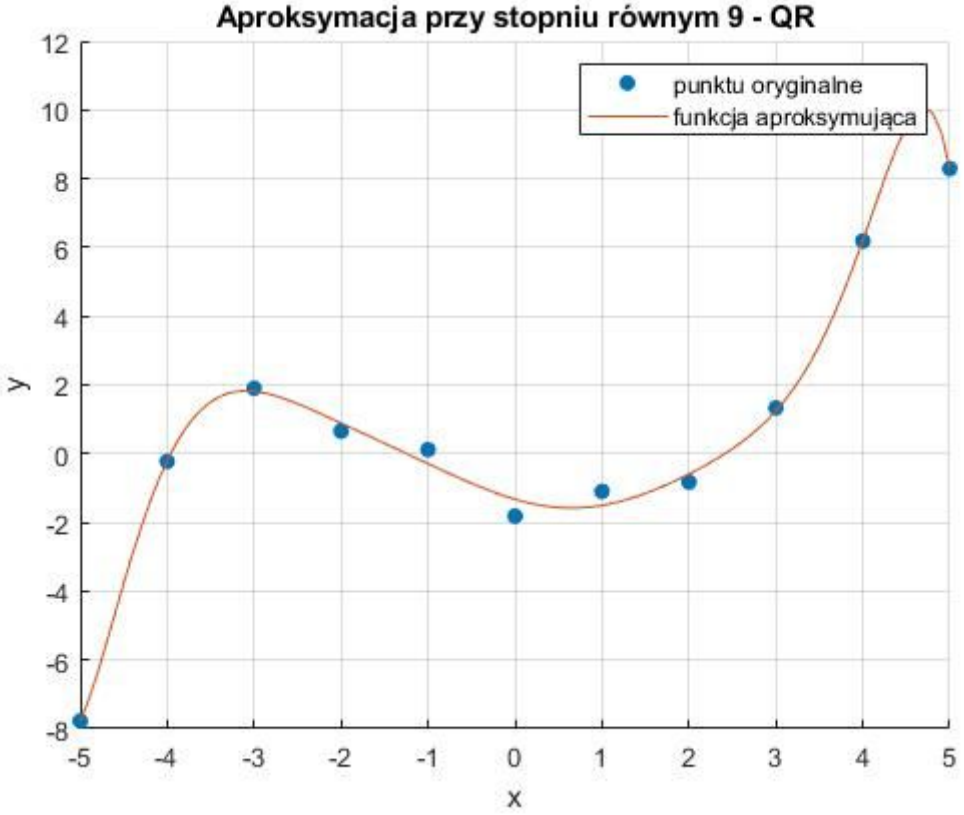
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 6 wynosi 6.118012e-01
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 6 wynosi 3.524390e-01



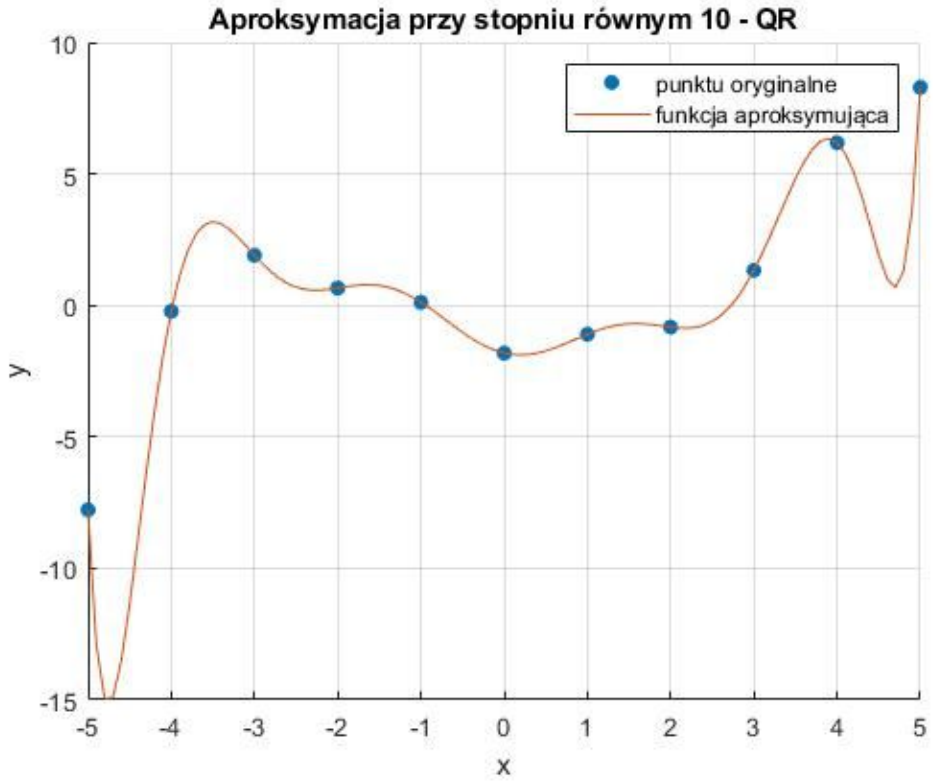
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 7 wynosi 6.118012e-01
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 7 wynosi 2.817600e-01



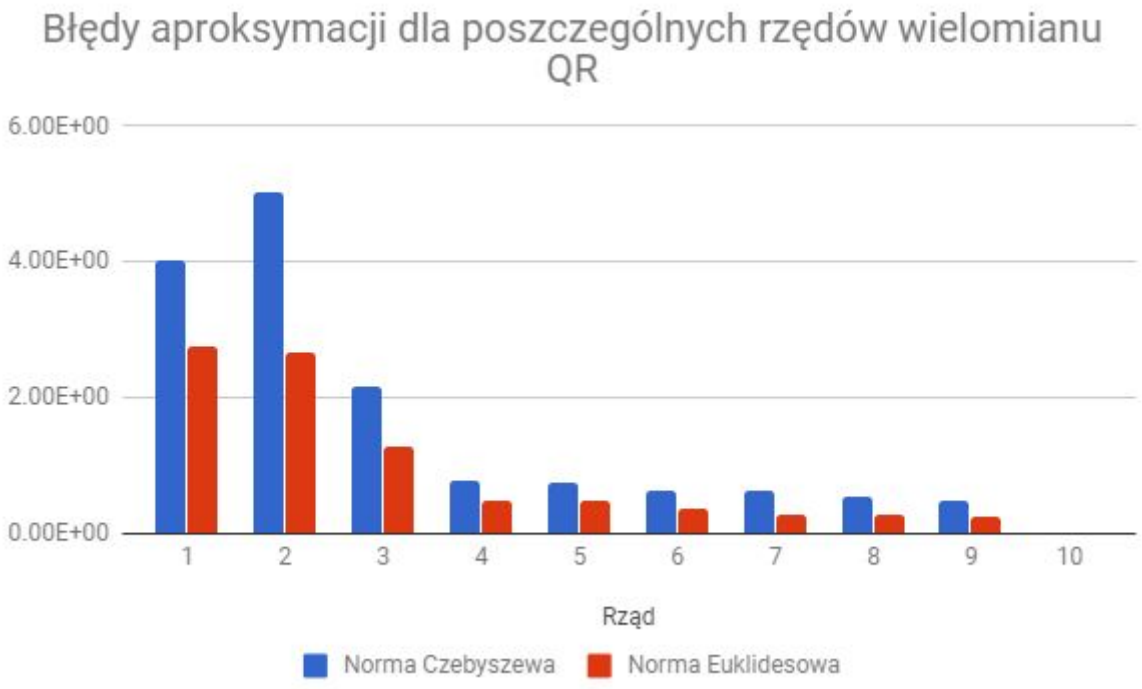
Błąd aproksymacji w normie Czebyszewa dla równania rzędu 8 wynosi 5.381672e-01
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 8 wynosi 2.678049e-01



Błąd aproksymacji w normie Czebyszewa dla równania rzędu 9 wynosi 4.884551e-01
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 9 wynosi 2.512043e-01



Błąd aproksymacji w normie Czebyszewa dla równania rzędu 10 wynosi 8.020251e-13
Błąd aproksymacji w normie Euklidesowej dla równania rzędu 10 wynosi 3.917945e-13



Wnioski:

Zarówno metoda równań normalnych jak i metoda rozkładu QR dobrze odwzorowuje wielomian znany na podstawie próbek. Jednak razem ze wzrostem rzędu wielomianu powiększają się błędy numeryczne (macierz A szybko traci dobre uwarunkowanie), co jest widoczne w przypadku rozkładu QR dla n=10. Biorąc pod uwagę, że dane wejściowe zostały obarczone pewnym błędem, trzeba zauważyć, że zwiększając rząd wielomiany w pewnym momencie przestaje się aproksymować oryginalną funkcję, a zaczyna tę, która najbardziej pasuje do zebranych danych. Biorąc pod uwagę wykresy i wartości błędów aproksymacji możemy zauważyć, że wraz ze zwiększaniem rzędu wielomianu błąd aproksymacji maleje co świadczy o zbliżaniu się funkcji aproksymującej do wartości aproksymowanej. Można też zauważyć, iż w każdym z przypadków dla metody z rozkładem QR oraz z układem równań liniowych.norma czebyszewa jest zawsze większa - co wynika z jej charakteru dlatego bo bierze zawsze największą wartość błędu aproksymacji, w przeciwieństwie do normy euklidesowej która opiera się na wyżej wymienionym wzorze.