ŁUKASZ STANISZEWSKI, NR INDEKSU: 304098

METODY NUMERYCZNE

PROJEKT 2 – ZADANIA 2.8

1 WSTĘP DO ZADANIA

Celem tego zadania było wyznaczenie funkcji wielomianowej $\boldsymbol{y = f(x)}$ najlepiej aproksymującą podane dane. Do rozwiązania zadania najmniejszych kwadratów w tym przypadku koniecznym było skorzystanie z układu równań normalnych oraz układu równań liniowych z macierzą R wynikającą z rozkładu QR macierzy układu równań problemu.

Istotą zadania było przetestowanie wielomianów różnych stopni jako funkcji aproksymujących podane próbki pomiarowe. Po wyliczeniu takich wielomianów, koniecznym było obliczyć błąd aproksymacji w dwóch normach: euklidesowej oraz maksimum.

Podany w zadaniu zestaw próbek danych wyglądał następująco:

$\boldsymbol{x_i}$	$\boldsymbol{y_i}$
-5	$2{,}0081$
-4	$-3{,}6689$
-3	$-4{,}9164$
-2	$-1{,}8700$
-1	$-0{,}0454$
0	$0{,}5504$
1	$-0{,}8392$
2	$-1{,}0113$
3	$2{,}6133$
4	$14{,}6156$
5	$39{,}6554$

Tabela 1.1 – aproksymowane próbki danych.

2 ZASTOSOWANE ALGORYTMY

Na początku, koniecznym jest zdefiniowanie problemu – poszukujemy funkcji $F(x)$, będącej kombinacją liniową funkcji bazowych $\boldsymbol{\Phi_0(x), \dots, \Phi_n(x)}$ (w postaci wielomianów, gdzie $\boldsymbol{\Phi_i(x) = x^i}$) ze współczynnikami $\boldsymbol{a_0, a_1, \dots, a_n}$. Tak więc, szukamy współczynników $\boldsymbol{a_0, a_1, \dots, a_n}$ funkcji F minimalizujące normę średniokwadratową dyskretną $\boldsymbol{||F - f||_2 = \sqrt{\sum_{j=0}^{N}[F(x_j) - f(x_j)]^2\, dx}}$ – taka aproksymacja jest metodą najmniejszych kwadratów.

Następnie, definiując, że funkcja aproksymowana $\boldsymbol{f(x)}$ na zbiorze punktów $\boldsymbol{x_0, \dots, x_N}$ przyjmuje znane nam wartości $\boldsymbol{y_j = f(x_j), j = 0, 1 \dots, N}$, możemy oznaczyć zadanie aproksymacji jako szukanie minimum funkcji:

$$\boldsymbol{H(a_0, \dots, a_n) = \sum_{j=0}^{N}\left[y_j - \sum_{i=0}^{n} a_i \Phi_i(x_j)\right]^2}$$

Gdy zdefiniujemy wektor $\boldsymbol{y} = [\boldsymbol{y_0}\ \boldsymbol{y_1}\ ..\boldsymbol{y_N}]^{\boldsymbol{T}}$ oraz $\boldsymbol{a} = [\boldsymbol{a_0}\ \boldsymbol{a_1}\ ..\boldsymbol{a_n}]^{\boldsymbol{T}}$, a także macierz

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{\Phi_0(x_0)} & \boldsymbol{\Phi_1(x_0)} & \dots & \boldsymbol{\Phi_n(x_0)} \\ \boldsymbol{\Phi_0(x_1)} & \boldsymbol{\Phi_1(x_1)} & \dots & \boldsymbol{\Phi_n(x_1)} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\Phi_0(x_N)} & \boldsymbol{\Phi_1(x_N)} & \dots & \boldsymbol{\Phi_n(x_N)} \end{bmatrix} \overset{\boldsymbol{APROKSYMACJA\ WIELOMIANEM\ STOPNIA\ n}}{\Longleftrightarrow} \boldsymbol{A} = \begin{bmatrix} \boldsymbol{x_0^0} & \boldsymbol{x_0^1} & \dots & \boldsymbol{x_0^n} \\ \boldsymbol{x_1^0} & \boldsymbol{x_1^1} & \dots & \boldsymbol{x_1^n} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x_N^0} & \boldsymbol{x_N^1} & \dots & \boldsymbol{x_N^n} \end{bmatrix}$$

to możemy przedstawić zapis funkcji $\boldsymbol{H}$ jako $\boldsymbol{H(a)} = \left(\left|\left|\boldsymbol{y} - \boldsymbol{Aa}\right|\right|_{\boldsymbol{2}}\right)^{\boldsymbol{2}}$ i rozwiązać to zadanie jak **Liniowe Zadanie Najmniejszych Kwadratów (LZNK)**.

Pierwszy sposób rozwiązywania zadania LZNK to zauważenie, że funkcja $\boldsymbol{H(a)}$ jest ściśle wypukła i ma jednoznaczne minimum w punkcie $\boldsymbol{H'(a)} = \boldsymbol{0}$, stąd otrzymujemy równanie $\boldsymbol{H'(a)^T} = \boldsymbol{2A^TAa - 2A^Ty}$ i mamy w ten sposób układ równań normalnych $\boldsymbol{A^TAx} = \boldsymbol{A^Tb}$, który można rozwiązać przy użyciu algorytmu eliminacji Gaussa z częściowym wyborem elementu głównego (zaimplementowanym w poprzednim projekcie), w wyniku czego otrzymujemy wektor $\boldsymbol{a}$ będący rozwiązaniem Liniowego Zadania Najmniejszych Kwadratów.

Drugi sposób rozwiązywania zadania LZNK wynika bezpośrednio z pierwszego i jest zalecany szczególnie dla słabo uwarunkowanych macierzy $\boldsymbol{A}$ – możemy w nim przekształcić nasz układ równań normalnych poprzez zastąpienie macierzy $\boldsymbol{A}$ jej rozkładem QR wąskim unormowanym. Wtedy otrzymujemy dobrze określony układ równań liniowych postaci $\boldsymbol{Ra} = \boldsymbol{Q^Ty}$. Dodatkowo zauważmy, że macierz $\boldsymbol{R}$ w tym przypadku będzie macierzą trójkątną górną i wtedy otrzymamy poszukiwany wektor $\boldsymbol{a}$ poprzez wykonanie jedynie drugiego etapu eliminacji Gaussa (algorytmu rozwiązywania układu z macierzą trójkątną). Zastosowany w tym zadaniu został rozkład QR ze zmodyfikowanym algorytmem Grama-Schmidta, który ortogonalizuje wobec każdej kolejnej kolumny wszystkie następne kolumny, przez co proces tworzenia rozkładu QR to iteracyjny proces polegający na tworzeniu ciągu macierzy, czego wynikiem jest nieunormowana macierz $\overline{\boldsymbol{Q}}$ oraz macierz $\overline{\boldsymbol{R}}$. Algorytm ten, wraz z kodem w MATLAB został zaczerpnięty bezpośrednio ze slajdów wykładowych.

3 OPIS IMPLEMENTACJI

Poza funkcją zwracającą dane wejściowe *getData.m* oraz zaczerpniętą z poprzedniego projektu funkcją rozwiązywania układów równań liniowych eliminacją Gaussa z częściowym wyborem elementu podstawowego *solveGaussPartial.m*, w wyniku projektu powstała między innymi funkcja *getA.m* tworząca macierz $\boldsymbol{A}$ na podstawie stopnia wielomianu aproksymującego oraz wektora $\boldsymbol{x}$:

```
function [A] = getA(n,x)
%GETA Creates A matrix from given polynomial degree and x
    x_size = size(x, 1);
    A = zeros(x_size,n+1);
    for i=1:(x_size)
        for j=1:(n+1)
            A(i,j)=x(i)^(j-1);
        end
    end
end
```

Kolejną funkcją, która została w projekcie użyta, jest funkcja *qrgsm.m*, zaczerpnięta prosto z wykładu, która oblicza rozkład QR macierzy wejściowej ze zmodyfikowanym algorytmem Grama-Schmidta:

```
function [Q,R]=qrgsm(A)
    %QR decomposition with modified Gram-Schmidt method
    %From dr Tatjewski MNUM slides
    [m, n]=size(A);
    Q=zeros(m,n);
    R=zeros(n,n);
    d=zeros(1,n);
    %Q columns orthogonal
    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
    %Q columns orthonormal
    for i=1:n
        dd=norm(Q(:,i));
        Q(:,i)=Q(:,i)/dd;
        R(i,i:n)=R(i,i:n)*dd;
    end
```

Jako pomocniczą funkcję, do obliczania wartości funkcji będącej złożeniem wielomianów kolejnych stopni z parametrami $\boldsymbol{a_i, i = 1 \dots n}$, zaimplementowałem funkcję *getPolynVal.m*:

```
function [val] = getPolynVal(x, a) % Returns value of function being polynomial.
    val = 0;
    for i = 0:size(a) - 1
       val = val + a(i+1,1) * x^i;
    end
end
```

Następnie koniecznym było zaimplementowanie funkcji, która oblicza współczynniki funkcji F metodą rozwiązania LNZK jako układu równań normalnych, w tym celu powstała funkcja solveNormal.m wykonująca te rozwiązanie:

```
function [a] = solveNormal(A, y_data)
%SOLVENORMAL makes approximation by using system of normal equations
    G = A'*A;
    rho = A'*y_data;
    a = solveGaussPartial(G, rho);
end
```

A także koniecznym było zaimplementowanie funkcji *solveQR.m* wykonującej analogiczną rzecz, ale wykorzystującą rozkład QR układu równań problemu:

```
function [a] = solveQR(n, A, y) %SOLVEQR makes approximation by using qr factorization
    [Q, R] = qrgsm(A);
    qtb = Q'*y;
    % solving stepped matrix
    a = zeros(n+1,1);
    for k = n+1:-1:1
        sum_x = 0;
        for j = k+1:n+1
            sum_x = sum_x + a(j)*R(k,j);
        end
        a(k) = (qtb(k,1) - sum_x)/R(k,k);
    end
 end
```

Ostatnim elementem projektu jest skrypt *testApproximation.m*, realizujący obliczanie współczynników funkcji aproksymujących metodami układu równań normalnych i faktoryzacji QR dla różnych stopni wielomianów, rysujący poszczególne funkcje na tle danych w formie wykresów i obliczający błędy aproksymacji w normie euklidesowej oraz maksimum. Można zauważyć, że wywołanie skryptu dla poszczególnej metody to odpowiednie odkomentowanie i skomentowanie linijek w kodzie.

```
[x, y] = getData();
max_degrees = 10;

normes_2=zeros(max_degrees+1,1);
normes_max=zeros(max_degrees+1, 1);

for degree=0:max_degrees
    A = getA(degree, x);
    % a=solveQR(degree, A,y);
    a=solveNormal(A,y);
    x_f=(-5:0.1:5.0)';
    y_f=zeros(size(x_f,1), 1);
    for ind = 1:101
       y_f(ind, 1) = getPolynVal(x_f(ind,1),a);
    end
    % approximated y for given x
    aproxY=zeros(size(x,1),1);
    for ind = 1:size(aproxY,1)
        aproxY(ind, 1) = getPolynVal(x(ind, 1), a);
    end
    % counting norms
    norm_2 = norm(aproxY-y,2);
    norm_max = norm(aproxY-y, "inf");
    normes_2(degree+1) = norm_2;
    normes_max(degree+1) = norm_max;
    %drawing plot and saving to file
    scatter(x,y);
    hold on
    plot(x, y, 'o');
    plot(x_f, y_f, '-');
    % title(['Approximation with QR for polynomial of degree=' int2str(degree)]);
    title(['Approximation with System of Normal Equations for polynomial of degree='
int2str(degree)]);
    xlabel('x');
    ylabel('y');
    hold off
    % savefig(['fig/QR_w' int2str(degree) '.fig']);
    savefig(['fig/Normal_w' int2str(degree) '.fig']);
end

% disp('Second norm of errors of approximations with next polynomials using QR');
disp('Second norm of errors of approximations with next polynomials using System of Normal
Equations');

for i=0:max_degrees
    disp(normes_2(i+1));
end
% disp('Max norm of errors of approximations with next polynomials using QR');
disp('Max norm of errors of approximations with next polynomials using System of Normal
Equations');
for i=0:max_degrees
    disp(normes_max(i+1));
end
```

4 WYNIKI I INTERPRETACJA

Zdecydowałem się na przetestowanie aproksymacji próbek danych przy użyciu wielomianów zaczynając od stopnia 0 i na stopniu 10 kończąc – w takim przypadku jest wciąż spełnione założenie $\boldsymbol{liczba_{probek} \geq stopien_{wielomianu}}$.

W wyniku uruchomienia skryptu *testApproximation*.m otrzymuje się następujące rezultaty dla **metody QR** (wyniki przepisane bezpośrednio z MATLABA):

STOPIEŃ WIELOMIANU	BŁĄD APROKSYMACJI W NORMIE EUKLIDESOWEJ	BŁĄD APROKSYMACJI W NORMIE MAKSIMUM
0	40.4795	35.3743
1	30.0127	22.4249
2	17.5724	9.9655
3	11.9240	4.9327
4	1.1184	0.7208
5	1.0707	0.6950
6	0.7356	0.4820
7	0.5142	0.3278
8	0.4045	0.2576
9	0.3493	0.2048
10	4.3055e-12	2.4996e-12

Tabela 3.1 – błędy aproksymacji przy metodzie QR.

Natomiast w wyniku uruchomienia skryptu *testApproximation*.m, dla metody z **układem równań normalnych**, rezultaty są takie:

STOPIEŃ WIELOMIANU	BŁĄD APROKSYMACJI W NORMIE EUKLIDESOWEJ	BŁĄD APROKSYMACJI W NORMIE MAKSIMUM
0	40.4795	35.3743
1	30.0127	22.4249
2	17.5724	9.9655
3	11.9240	4.9327
4	1.1184	0.7208
5	1.0707	0.6950
6	0.7356	0.4820
7	0.5142	0.3278
8	0.4045	0.2576
9	0.3493	0.2048
10	7.3998e-11	3.6595e-11

Tabela 3.2 – błędy aproksymacji przy metodzie z układem równań normalnych.

Różnice w błędach pomiędzy metodami przy użyciu różnicy:

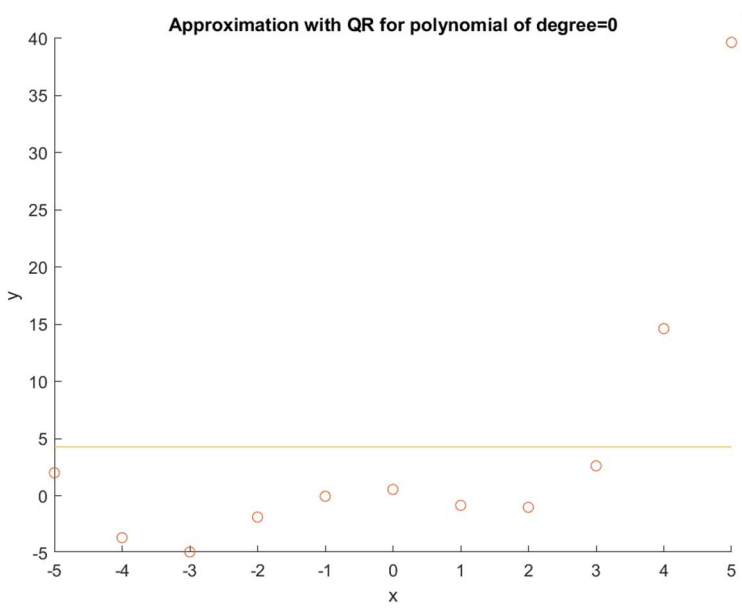
STOPIEŃ WIELOMIANU	BŁĄD EUKLIDESOWY QR-NORMALNY	BŁĄD MAKSIMUM QR-NORMALNY
0	7.1054e-15	0
1	-3.5527e-15	-7.1054e-15
2	3.5527e-15	-3.5527e-15
3	1.7764e-15	-8.8818e-16
4	-1.3323e-15	-1.5099e-14
5	2.2204e-16	-6.2172e-15

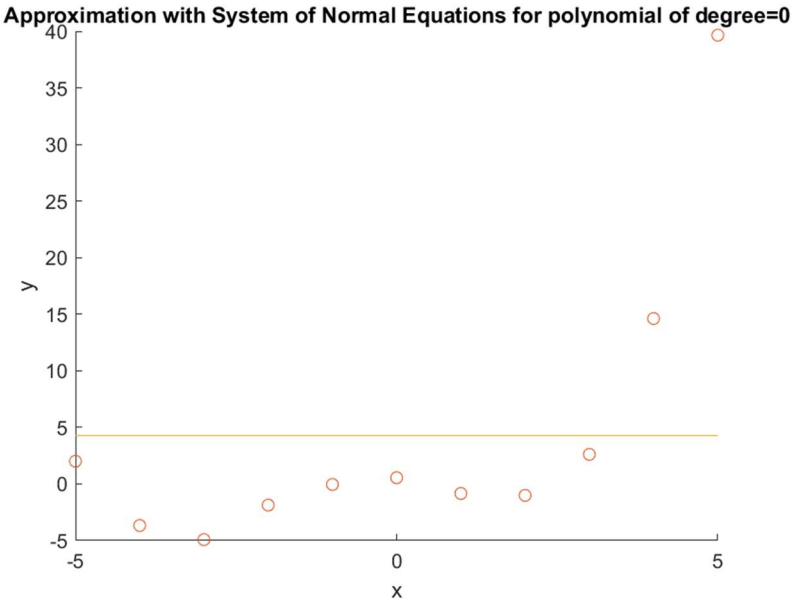
STOPIEŃ WIELOMIANU	BŁĄD EUKLIDESOWY QR-NORMALNY	BŁĄD MAKSIMUM QR-NORMALNY
6	0	-8.8818e-15
7	-3.3307e-16	3.0198e-14
8	-8.8818e-16	4.4409e-14
9	-2.2204e-16	3.8952e-13
10	-6.9693e-11	-3.4095e-11

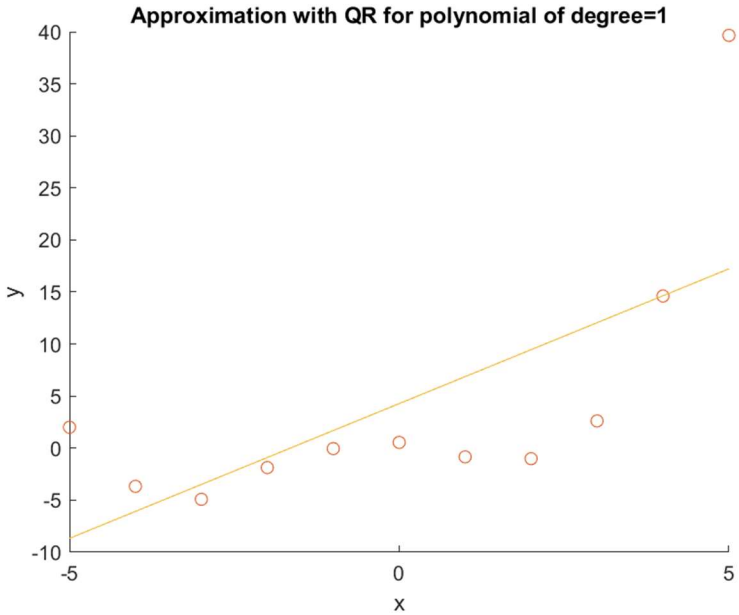
Tabela 3.3 – różnica w błędach euklidesowym i maksimum między metodą z rozkładem QR a metodą z układem równań normalnych

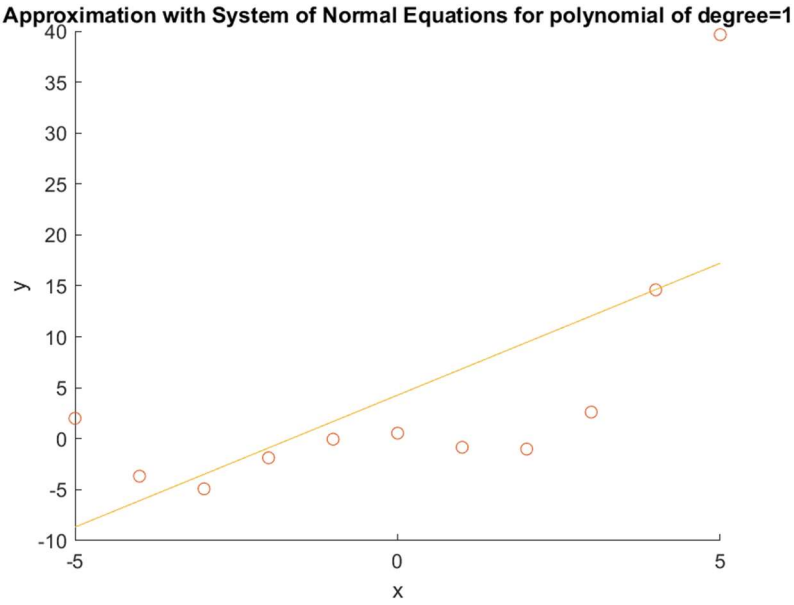
Można zauważyć, że spełniona jest tu zarówno zależność norm $||\cdot||_\infty \leq ||\cdot||_2$, dodatkowo nie widać żeby błędy popełniane przy rozwiązywaniu zadania metodami faktoryzacji QR oraz układu równań normalnych znacząco różniły się od siebie – różnice te są bardzo małe (rzędu $\mathbf{10^{-14}}$). Najbardziej widoczna różnica ma miejsce przy aproksymacji z użyciem wielomianu stopnia **10**, czyli największego stopnia. Jest wtedy rzędu $\mathbf{10^{-10}}$. Można również zauważyć, że nie ma tu zależności mówiącej, która metoda daje mniejsze błędy – oznacza to, że najprawdopodobniej błędy poszczególnych metod wynikają nie tylko ze stopnia wielomianu, jakim się aproksymuje funkcję, ale też z błędów numerycznych zaimplementowanych metod rozwiązywania układów równań.

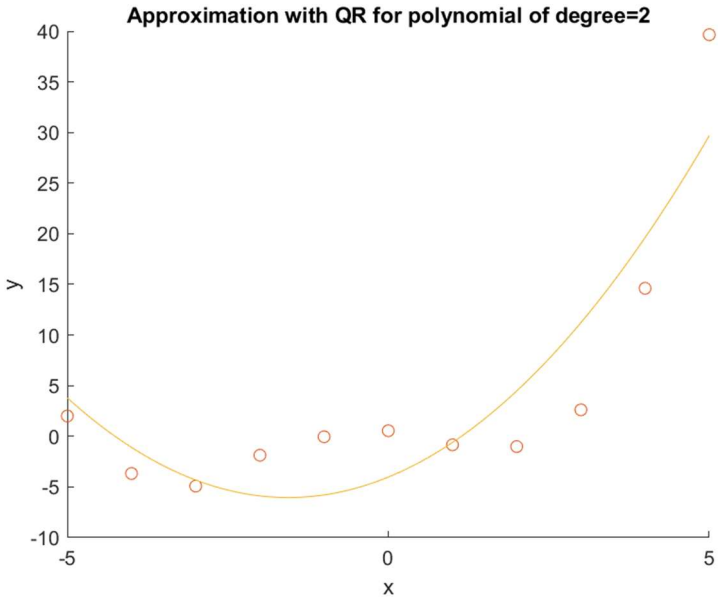
Na koniec postanowiłem zamieścić wykresy, które będą obrazowały jak wygląda funkcja aproksymująca dane na tle próbek, które są podawane na wejściu dla metody z układem równań wynikającym z rozkładu QR (na lewo) oraz metodą z układem równań normalnych (na prawo), jednak postanowiłem ograniczyć się do wykresów, które pokazują istotne zmiany między kolejnymi wielomianami.

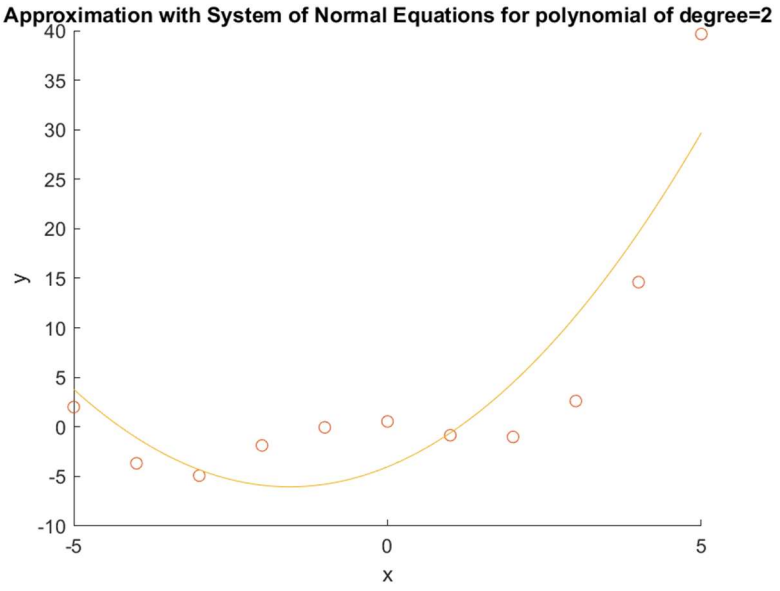


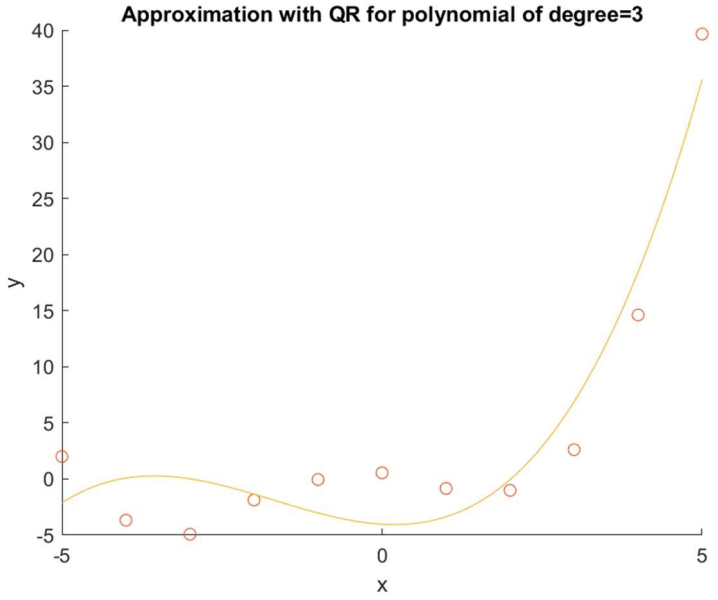


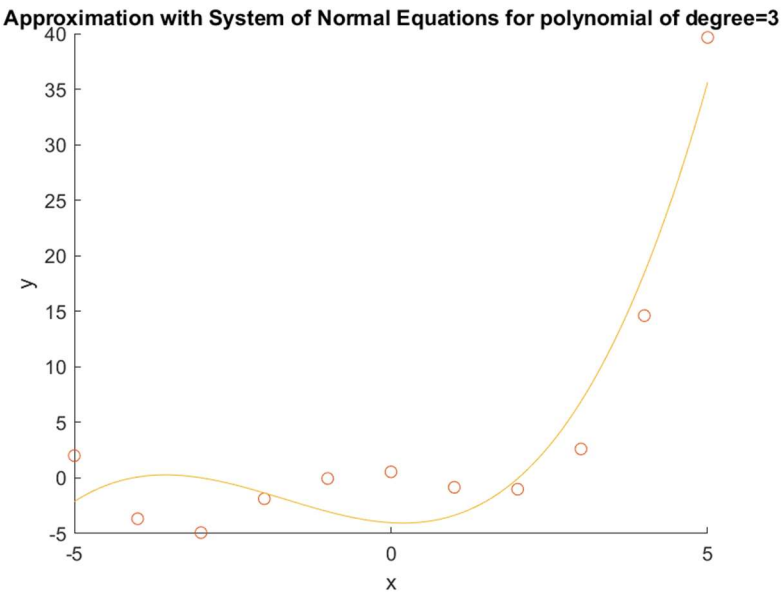


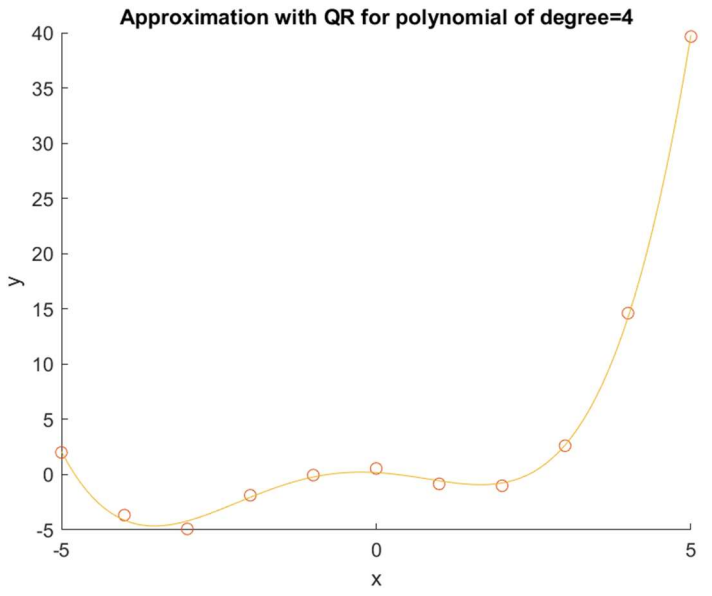


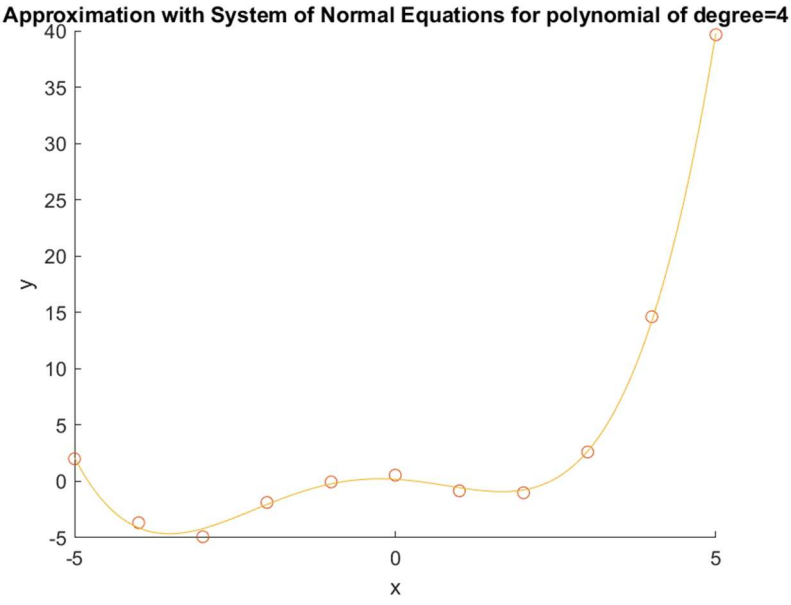

Approximation with QR for polynomial of degree=2
y
x

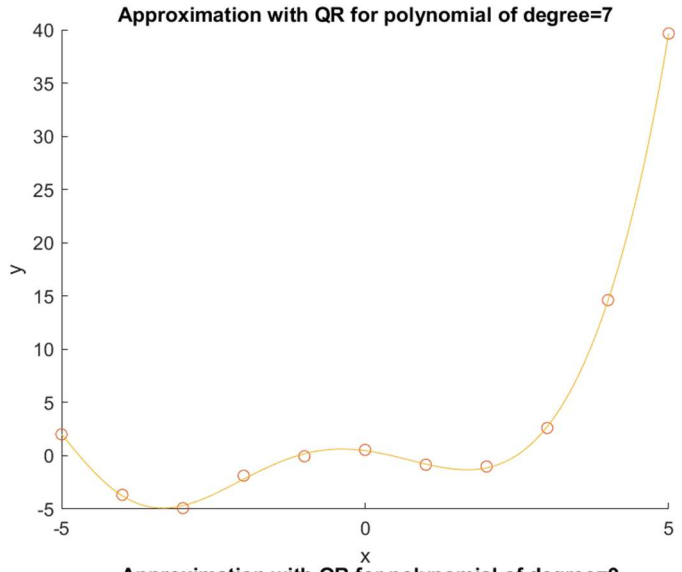

Approximation with System of Normal Equations for polynomial of degree=2
y
x

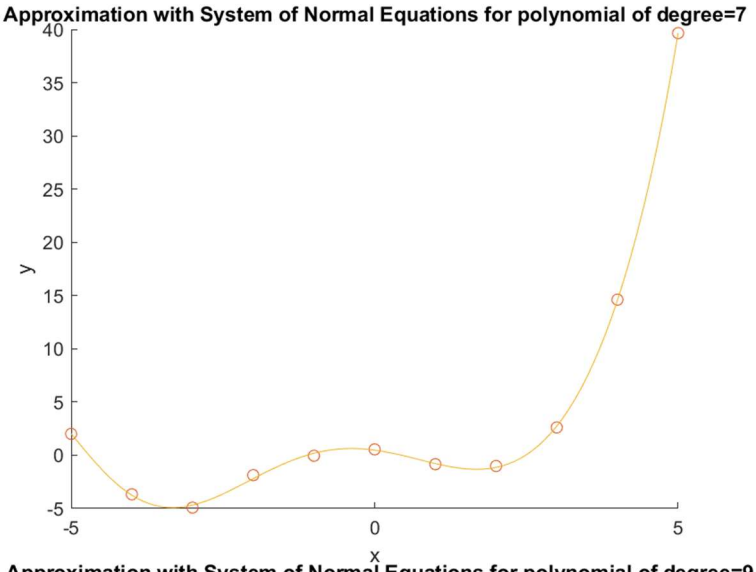

Approximation with QR for polynomial of degree=3
y
x

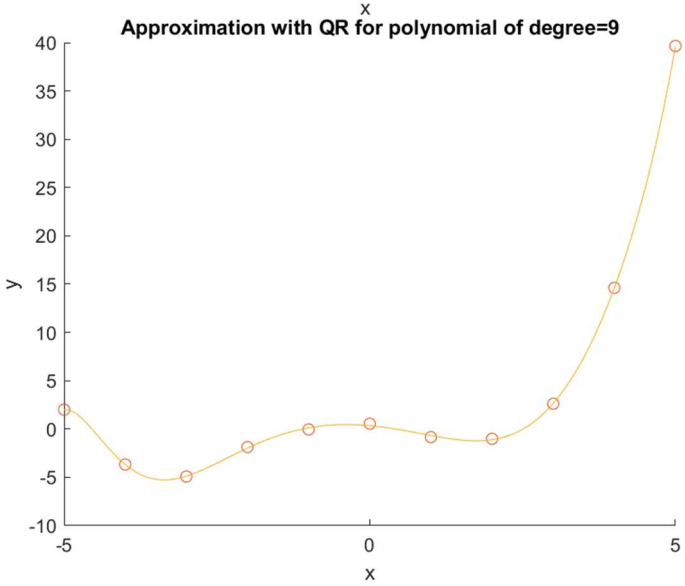

Approximation with System of Normal Equations for polynomial of degree=3
y
x

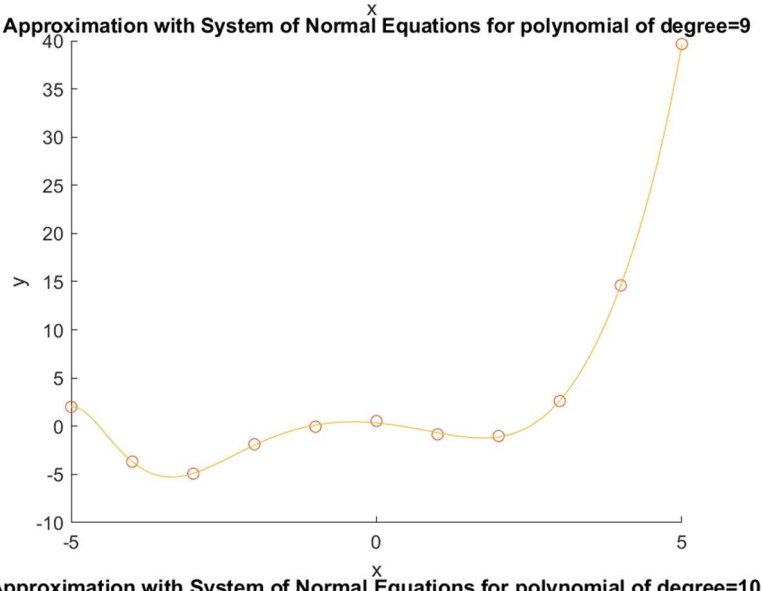

Approximation with QR for polynomial of degree=4
y
x

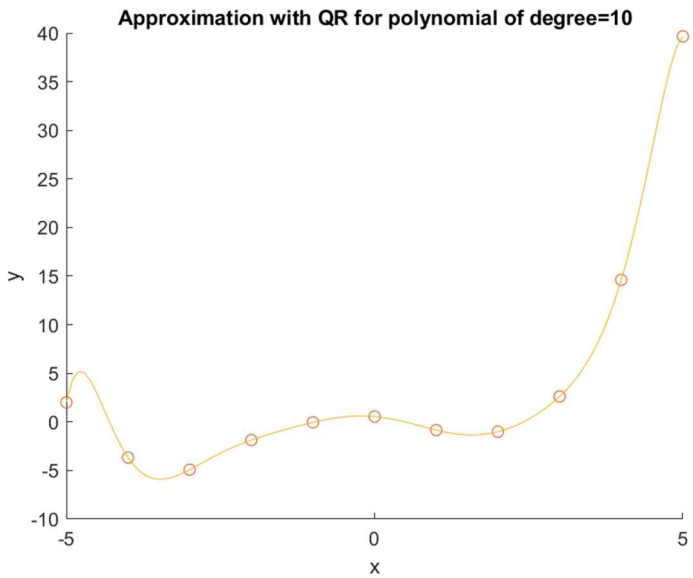

Approximation with System of Normal Equations for polynomial of degree=4
y
x

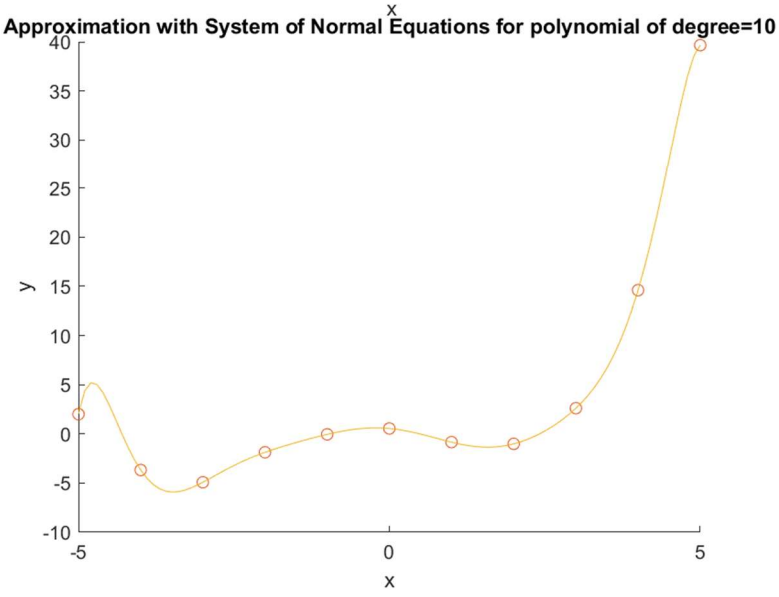












Na podstawie powyższych wykresów, można zauważyć, że już wielomian stopnia czwartego dobrze aproksymuje nasze próbki danych. Oczywiście, wielomiany stopnia większego robią to samo z coraz mniejszymi błędami, jednak w aproksymacji istotne jest to, aby funkcji bazowych, jakimi się ją wykonuje było jak najmniej.

Istotnym jest również zwrócenie uwagi na rozróżnienie między obiema metodami – zauważmy, że wraz ze wzrostem stopnia wielomianu aproksymującego funkcję, wskaźnik uwarunkowania w normie drugiej macierzy $\boldsymbol{A^T A}$ wzrasta w bardzo dużym tempie, co można sprawdzić uruchamiając skrypt *checkCond.m*, gdzie wynik jest następujący:

```
For polynomal of 0 cond of A^TA equals 1
For polynomal of 1 cond of A^TA equals 10
For polynomal of 2 cond of A^TA equals 409
For polynomal of 3 cond of A^TA equals 8558
For polynomal of 4 cond of A^TA equals 317981
For polynomal of 5 cond of A^TA equals 7467496
For polynomal of 6 cond of A^TA equals 283155896
For polynomal of 7 cond of A^TA equals 7646220978
For polynomal of 8 cond of A^TA equals 330546434323
For polynomal of 9 cond of A^TA equals 15167089011215
For polynomal of 10 cond of A^TA equals 929296536519140
```

Wykorzystanie wąskiego rozkładu QR unormowanego macierzy $\boldsymbol{A}$, w celu rozwiązania układu równań zadnia jest zalecane dla słabo uwarunkowanych macierzy $\boldsymbol{A}$, gdyż uwarunkowanie w normie drugiej macierzy $\boldsymbol{A^T A}$ jest kwadratem uwarunkowania w normie drugiej macierzy $\boldsymbol{A}$. W naszym przypadku widać to w różnicy w błędzie aproksymacji dla wielomianu stopnia 10 – różnica w błędzie wtedy jest już zdecydowanie bardziej widoczna i błąd jest mniejszy gdy skorzystamy z rozkładu QR.