ŁUKASZ STANISZEWSKI, NR INDEKSU: 304098

METODY NUMERYCZNE

PROJEKT 2



# Wstęp do zadania

Celem tego zadania było wyznaczenie funkcji wielomianowej 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 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:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Tabela 1.1 – aproksymowane próbki danych.

# Zastosowane algorytmy

Na początku, koniecznym jest zdefiniowanie problemu – poszukujemy funkcji , będącej kombinacją liniową funkcji bazowych (w postaci wielomianów, gdzie ) ze współczynnikami . Tak więc, szukamy współczynników funkcji minimalizujące normę średniokwadratową dyskretną – taka aproksymacja jest metodą najmniejszych kwadratów.

Następnie, definiując, że funkcja aproksymowana na zbiorze punktów przyjmuje znane nam wartości , możemy oznaczyć zadanie aproksymacji jako szukanie minimum funkcji:

Gdy zdefiniujemy wektor oraz , a także macierz

to możemy przedstawić zapis funkcji jako i rozwiązać to zadanie jak **Liniowe Zadanie Najmniejszych Kwadratów (LZNK)**.

Pierwszy sposób rozwiązywania zadania LZNK to zauważenie, że funkcja jest ściśle wypukła i ma jednoznaczne minimum w punkcie , stąd otrzymujemy równanie i mamy w ten sposób układ równań normalnych , który można rozwiązać przy użyciu algorytmu eliminacji Gaussa z częściowym wyborem elementu głównego (zaimplementowanym w poprzednim projekcie), w wyniku czego otrzymujemy wektor będący rozwiązaniem Liniowego Zadania Najmniejszych Kwadratów.

Drugi sposób rozwiązywania zadania LZNK wynika bezpośrednio z pierwszego i jest zalecany szczególnie dla słabo uwarunkowanych macierzy – możemy w nim przekształcić nasz układ równań normalnych poprzez zastąpienie macierzy jej rozkładem QR wąskim unormowanym. Wtedy otrzymujemy dobrze określony układ równań liniowych postaci . Dodatkowo zauważmy, że macierz w tym przypadku będzie macierzą trójkątną górną i wtedy otrzymamy poszukiwany wektor 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 oraz macierz . Algorytm ten, wraz z kodem w MATLAB został zaczerpnięty bezpośrednio ze slajdów wykładowych.

# 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 na podstawie stopnia wielomianu aproksymującego oraz wektora :

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 , 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

# 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 .

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 |

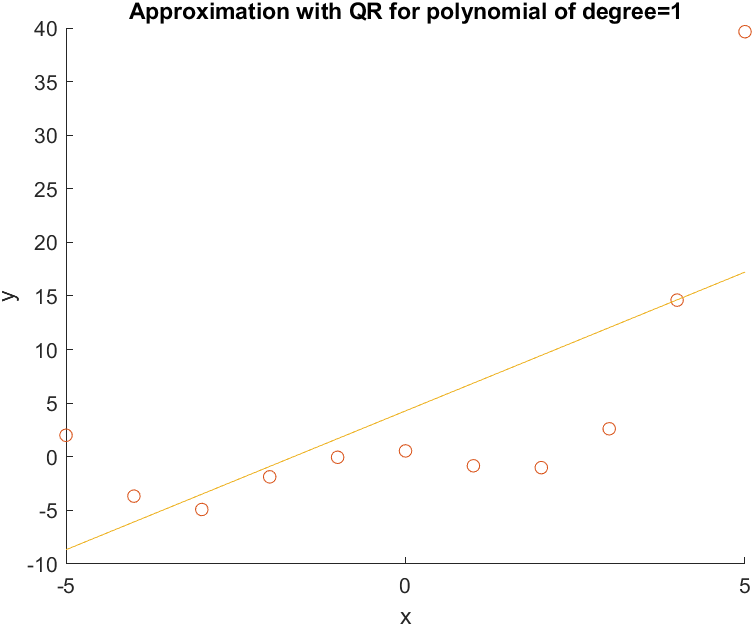
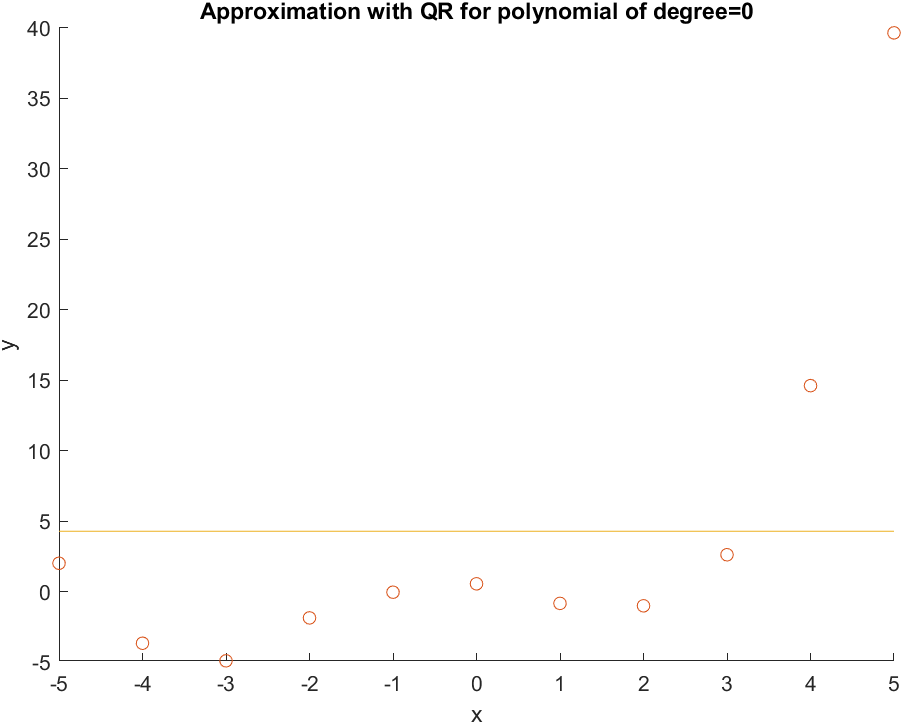
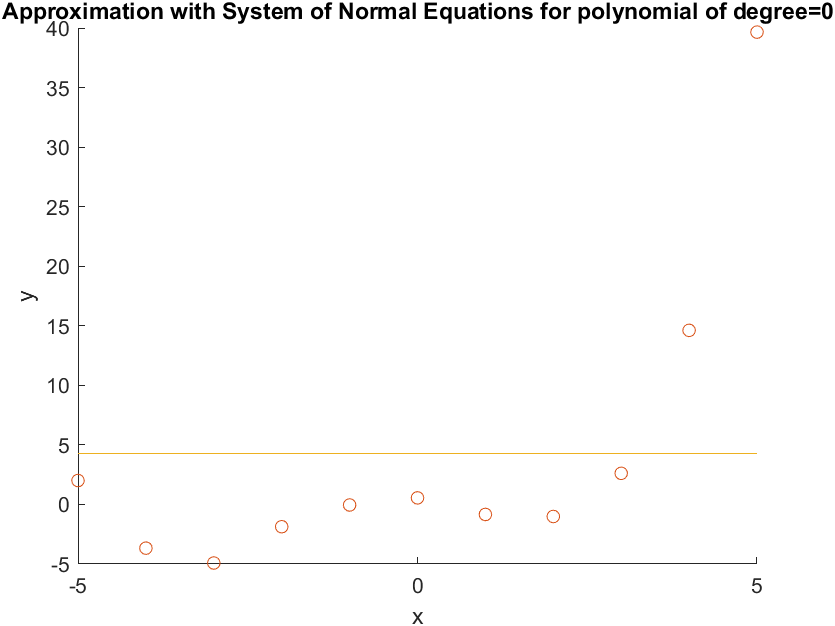
Tabela 3.2 – błędy aproksymacji przy metodzie z układem równań normalnych.

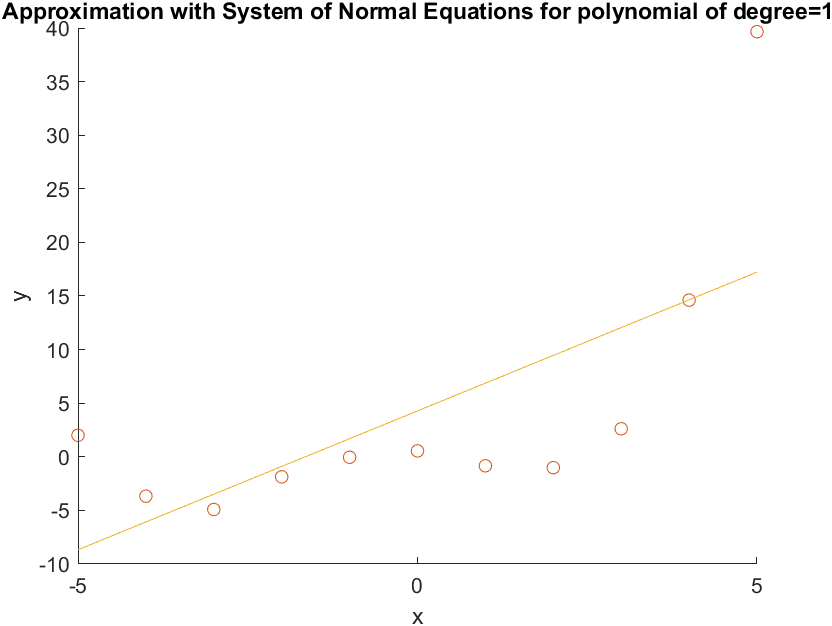
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **STOPIEŃ WIELOMIANU** | **BŁĄD EUKLIDESOWY**  **QR-NORMALNY** | **BŁĄD MAKSIMUM**  **QR-NORMALNY** |  | **STOPIEŃ WIELOMIANU** | **BŁĄD EUKLIDESOWY**  **QR-NORMALNY** | **BŁĄD MAKSIMUM**  **QR-NORMALNY** |
| 0 | 7.1054e-15 | 0 |  | 6 | 0 | -8.8818e-15 |
| 1 | -3.5527e-15 | -7.1054e-15 |  | 7 | -3.3307e-16 | 3.0198e-14 |
| 2 | 3.5527e-15 | -3.5527e-15 |  | 8 | -8.8818e-16 | 4.4409e-14 |
| 3 | 1.7764e-15 | -8.8818e-16 |  | 9 | -2.2204e-16 | 3.8952e-13 |
| 4 | -1.3323e-15 | -1.5099e-14 |  | 10 | -6.9693e-11 | -3.4095e-11 |
| 5 | 2.2204e-16 | -6.2172e-15 |  |  |  |  |

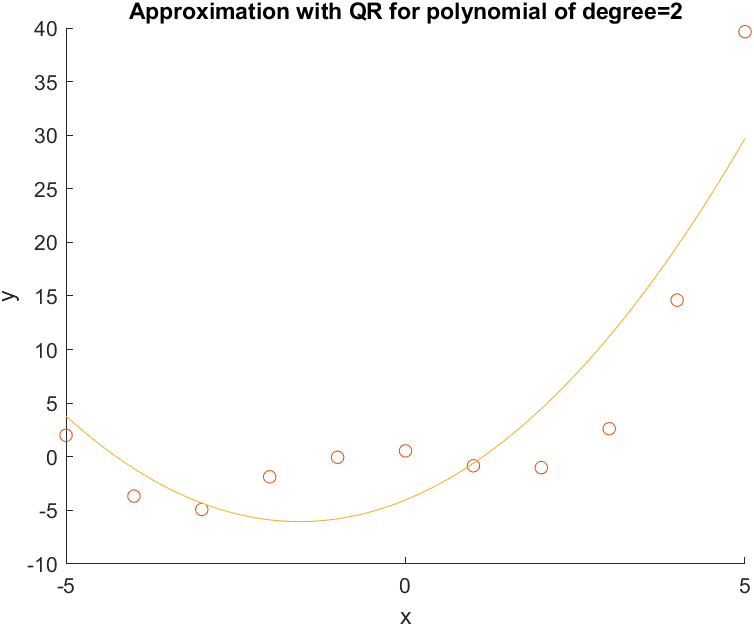
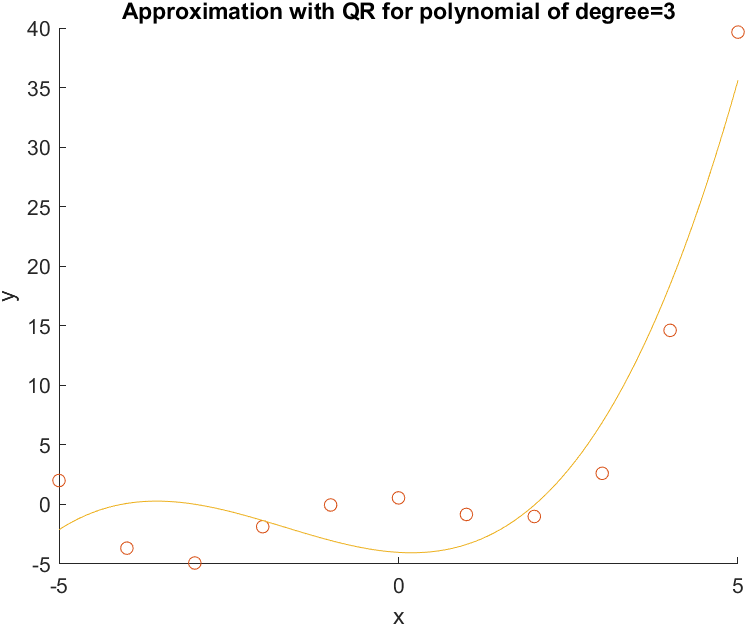
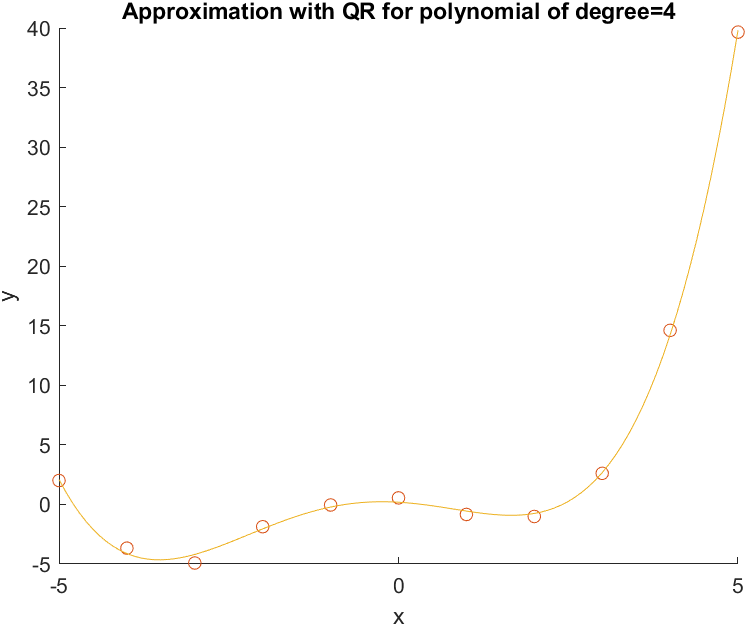
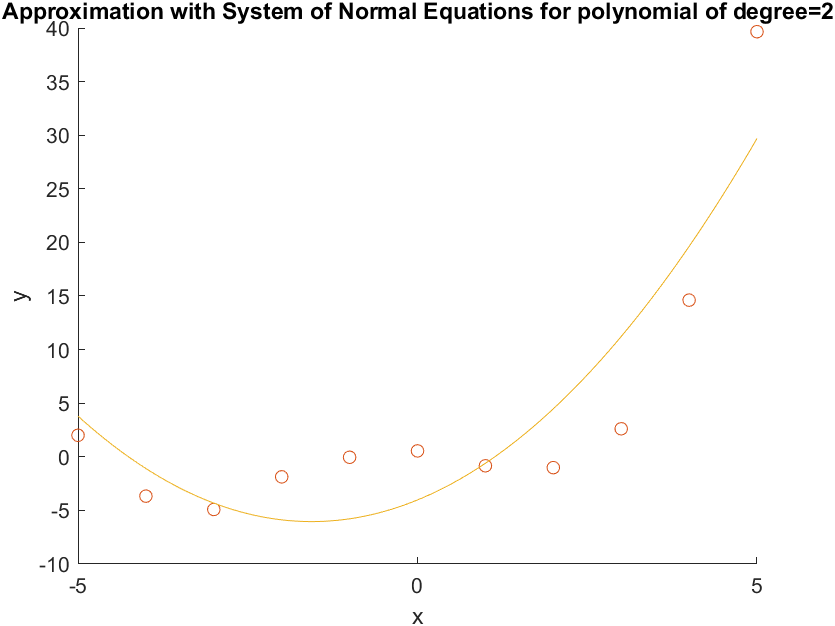
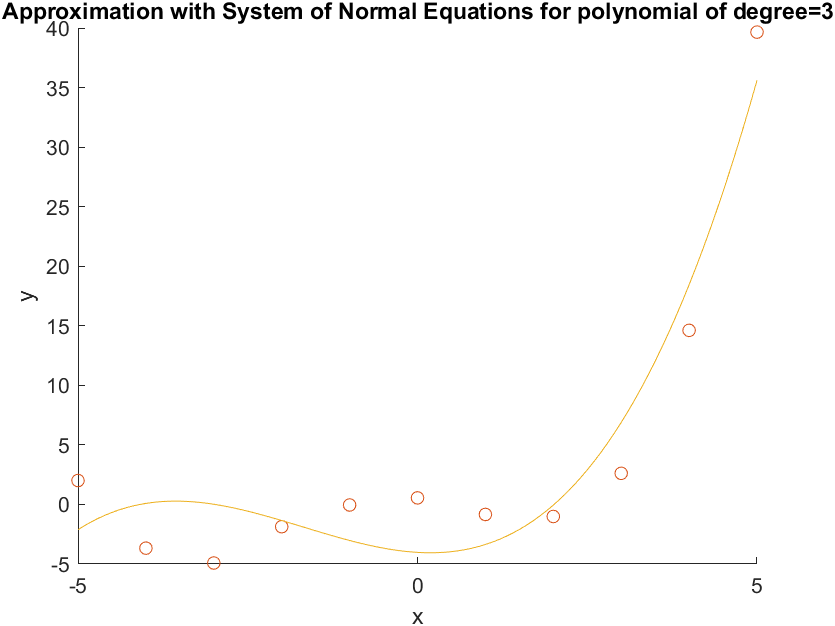
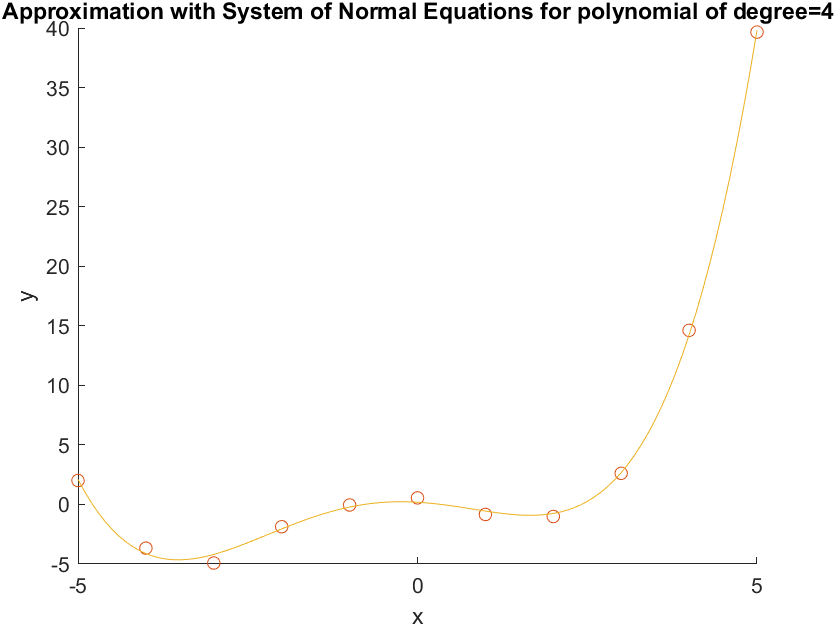
Różnice w błędach pomiędzy metodami przy użyciu różnicy:

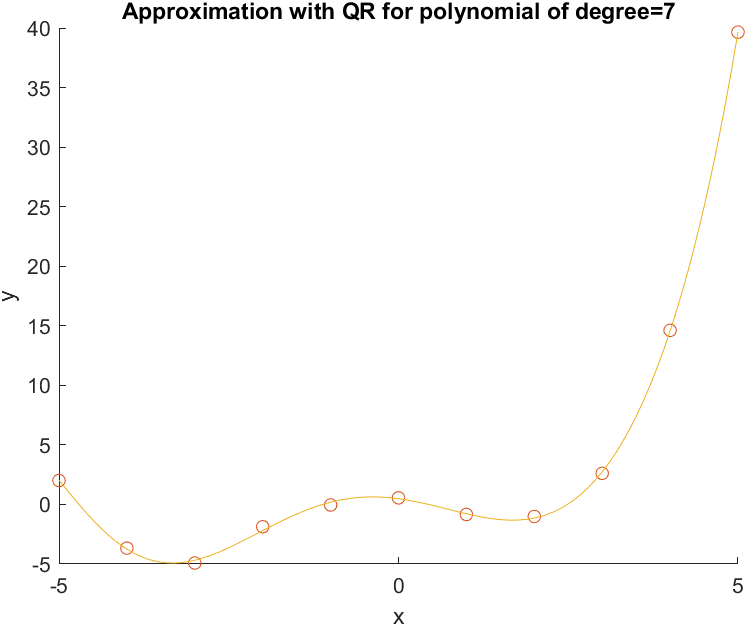
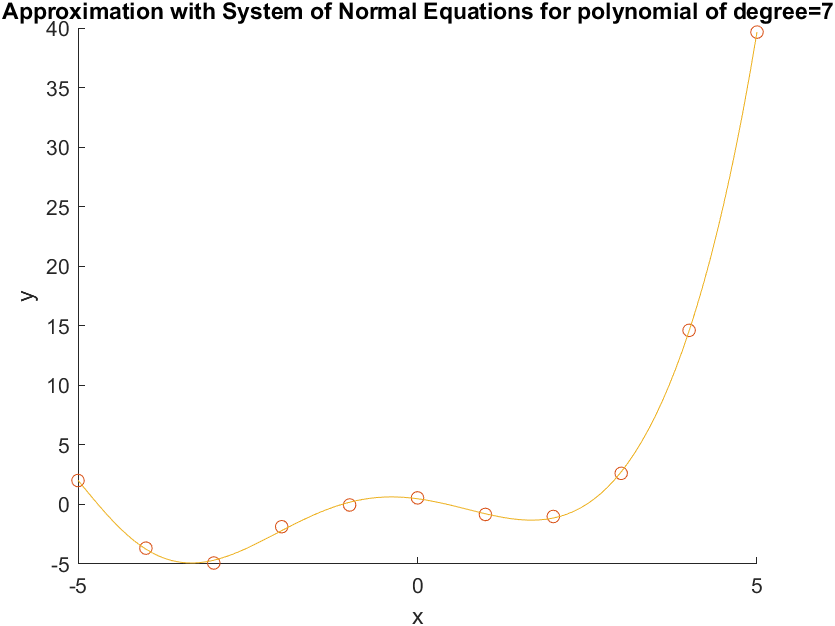
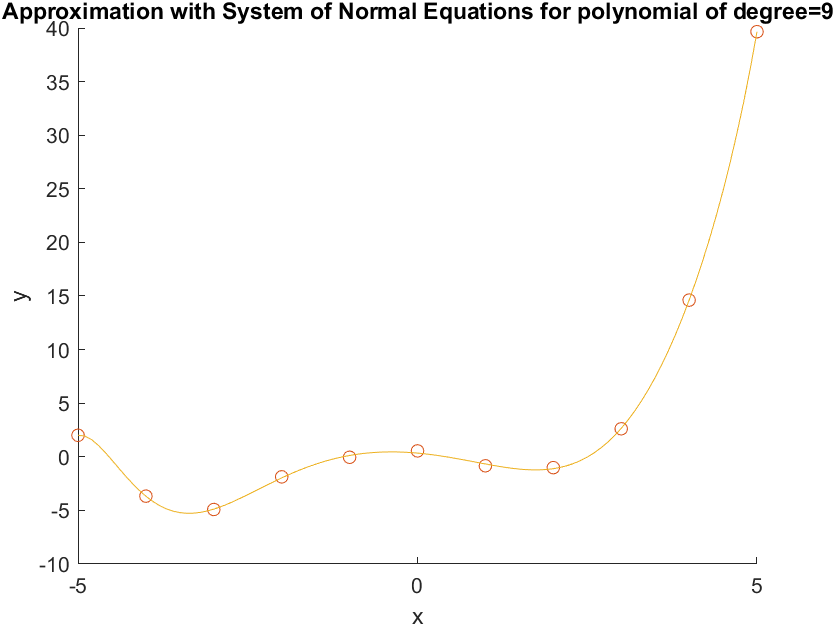
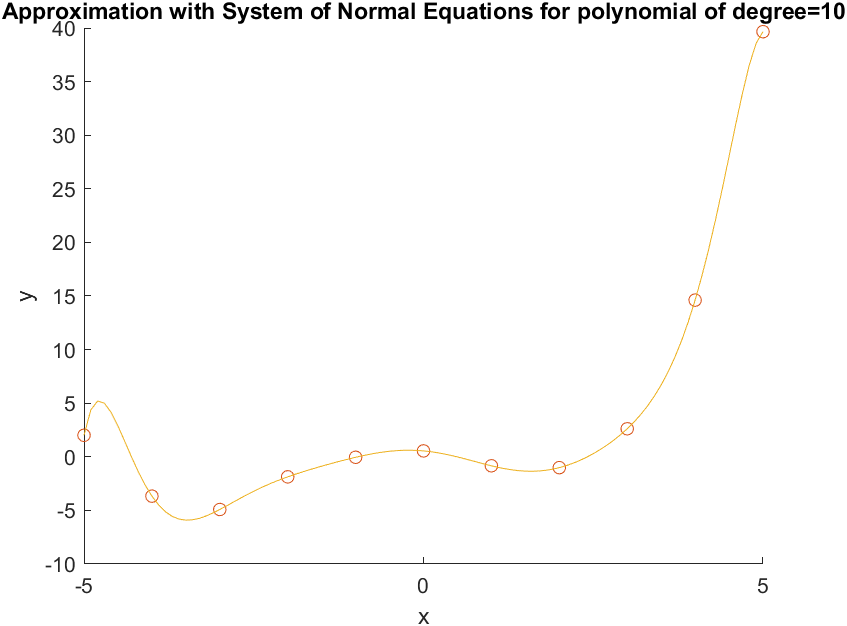
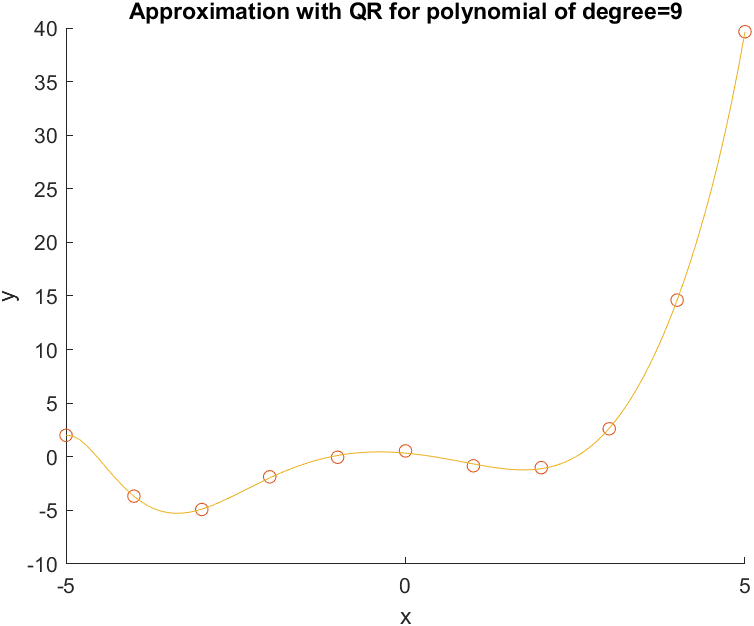
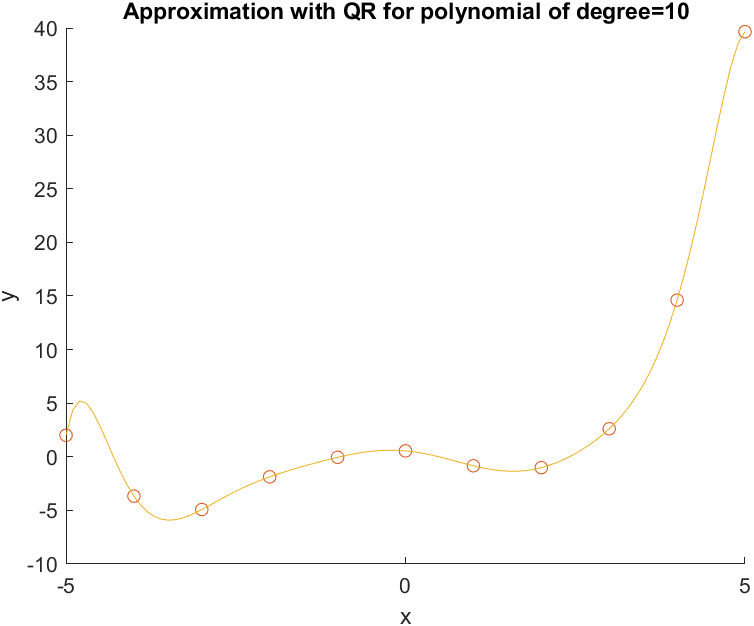
*Tabela 3.3 – różnica w błędach euklidesowym i maksimum między metodą z rozkładem QR a metodą z układem równań normalnych*

Można zauważyć, że spełniona jest tu zarówno zależność norm , 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 ). Najbardziej widoczna różnica ma miejsce przy aproksymacji z użyciem wielomianu stopnia **10**, czyli największego stopnia. Jest wtedy rzędu . 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.





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 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 , w celu rozwiązania układu równań zadnia jest zalecane dla słabo uwarunkowanych macierzy , gdyż uwarunkowanie w normie drugiej macierzy jest kwadratem uwarunkowania w normie drugiej macierzy . 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.