MNUM Projekt 2.

Autor:	Numer projektu:	Prowadzący:	Liczba punktów:
Mariusz Słapek	2.42	dr inż. Adam Krzemienowski	

Kody do poszczególnych programów zostały umieszczone na końcu sprawozdania.

Spis treści

Zadanie 1

Treść zadania:

Proszę napisać program służący do obliczania **wartości własnych macierzy** nieosobliwych metodą rozkładu QR w dwóch wersjach: bez przesunięć i z przesunięciami dla macierzy symetrycznej, oraz w wersji z przesunięciami dla macierzy niesymetrycznej.

a) następnie proszę przetestować skuteczność (zbieżność) obu wersji algorytmu dla 30 różnych macierzy losowych o wymiarach: *5×5, 10×10* i *20×20*.

b) proszę podać średnią liczbę iteracji dla metody bez przesunięć i z przesunięciami.

c) dla wybranych macierzy proszę porównać otrzymane wyniki z wartościami własnymi obliczonymi poleceniem *eig*.

Ogólny zarys:

a) rozkład QR (ogólnie)

Zdefiniowanie pojęć:

i. macierz ortogonalna – jest to taka macierz **Q**, która: $\mathbf{Q} * \mathbf{Q^T} = \mathbf{I}$ (jej kolumny są wektorami ortonormalnymi, **I** – macierz jednostkowa).
ii. macierz ortonormalne – macierz ortogonalna oraz długości jednostkowej.

Rozkład macierzy **A** do postaci iloczynu dwóch macierzy **Q** i **R**, gdzie **Q** jest macierzą ortonormalną (lub ogólniej ortogonalną) a **R** jest macierzą trójkątną górną.

Możemy rozłożyć dowolną macierzy na iloczyn macierzy **Q** i **R**.

b) sposoby numeryczne rozkładu QR:

i. metoda ortogonalizacji Grama-Schmidta (ew. zmodyfikowany algorytm Grama-Schmidta);
ii. metoda odbić Householdera;
iii. metoda obrotów Givensa (szczególnie macierze rzadkie tj.: macierz w której większość elementów ma wartość 0)

c) wektory i wartości własne macierzy

Wartości własne macierzy mają zastosowanie w dzisiejszej nauce i technice.
Wektor **v** jest wektorem własnym macierzy **A**, gdy istnieje taka liczba λ, że **A v** = λ **v**, gdzie *λ* to wartość własna macierzy A (warto dodać, że ogólnie λ może być dowolną liczbą zespoloną, a więc λ ϵ **C**, ale **wartości i wektory własne macierzy symetrycznej są rzeczywiste).**
Ponadto wiemy, iż λ jest wartością własną macierzy **A** wtedy i tylko wtedy, gdy jest pierwiastkiem wielomianu charakterystycznego, tzn.: $det(\mathbf{A} - \lambda \mathbf{I}) = 0$ macierzy **A**, gdzie: **I** – macierz jednostkowa (o rozmiarze takim samym jak macierz A).

Możemy zauważyć, że macierzy kwadratowa *n* – wymiarowa ma dokładnie *n* wartości własnych i odpowiadających im wektorów własnych.

Z faktu, iż wektorem własnych jest każdy wektor pomnożony przez pewną stałą α, wprowadza się pojęcie wektorów unormowanych. Są to takie wektory własne, których długość jest równa 1.

W algebrze wprowadza się również pojęcie widma macierzy. Jest to zbiór wszystkich wartości własnych.

d) macierz trójdiagonalnie symetryczna

Macierz postaci:

$$\begin{bmatrix} b_1 & a_2 & 0 & 0 & 0 \\ a_2 & b_2 & \dots & 0 & 0 \\ 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & \dots & b_{n-1} & a_n \\ 0 & 0 & 0 & a_n & b_n \end{bmatrix}$$

Metoda QR dla macierzy nieprzekształconych do tej postaci też działa, lecz w tej postaci liczba iteracji jest mniejsza. W zadaniu będę korzystał z tego faktu, w celu zoptymalizowania nakładu obliczeń.

e) wybrane metody wyznaczania wartości własnych

Istnieje wiele metod wyznaczania wartości własnych, m.in.:

i. metoda QR (wykorzystana przeze mnie w tym zadaniu, opisana w następnym punkcie)
ii. metoda Jacobiego:

Służy ona do wyznaczania wartości własnych tylko macierzy symetrycznej **A** polegająca na przekształceniu macierzy do postaci diagonalnej **P** ciągiem obrotów *Givensa*. W macierzy diagonalnej na przekątnej głównej znajdą się wartości własne macierzy **A**, natomiast wektory własne odpowiadające tym wartościom własnym będą zapisane w kolumnach macierzy **P**.

iii. metoda wyznacznikowa:

Metoda korzysta z faktu, iż wartości własne są zerami wielomianu charakterystycznego obliczając wartości własne wprost z definicji, czyli $det(\mathbf{A} - \lambda \mathbf{I}) = 0$.

Metoda QR znajdowania wartości własnych macierzy symetrycznej

Opiera się ona na sukcesywnym znajdowaniu rozkładów QR.

Istnieją dwie metody algorytmu QR bez przesunięć i z przesunięciami.

a) algorytm podstawowy:

Pojedynczy krok algorytmu QR (w $k+i$ kroku):

$$\boldsymbol{A^{(k)} = Q^{(k)} \cdot R^{(k)}}$$
$$\boldsymbol{A^{(k+1)} = R^{(k)} \cdot Q^{(k)}}$$

Czyli macierz $A^{(k+1)}$ jest przekształconą przez podobieństwo macierzą $A^{(k)}$. Mają te same wartości własne.

Podsumowując dla macierzy symetrycznej $\boldsymbol{A}$, $\boldsymbol{A^{(k)}}$zbiega do macierzy diagonalnej.

b) algorytm z przesunięciami:

Gdy wartości własne λ leżą blisko siebie to metoda z algorytmem podstawowym jest wolno zbieżna. Wówczas stosuje się algorytm metody QR z przesunięciami (w celu poprawienia szybkości zbieżności).

Pojedynczy krok algorytmu QR z przesunięciami (w $k+i$ kroku):

$$\boldsymbol{A^{(k)} - p_k * I = Q^{(k)} \cdot R^{(k)}}$$
$$\boldsymbol{A^{(k+1)} = R^{(k)} \cdot Q^{(k)} + p_k * I}$$

Metoda QR wyznaczania wartości własnych dla macierzy niesymetrycznej

Podobnie jak w macierzy symetrycznej macierz wyjściową zaleca się przekształcić do *postaci Hessenberga*.
Konstrukcja algorytmu QR dla macierzy dowolnych pozostaje w istocie taka sama jak dla macierzy symetrycznych, musimy wziąć pod uwagę możliwość wystąpienia wartości własnych zespolonych. Dlatego ważne jest wybieranie przesunięcia zawsze jako wartości własnej podmacierzy 2x2 z prawego dolnego rogu aktualnej macierzy. Macierz przekształcona zbiega na ogół do macierzy trójkątnej górnej.
Metoda QR dla macierzy niesymetrycznych nie zawsze jest zbieżna.

Prezentacja wyników:

W algorytmie QR z przesunięciami i bez przesunięć zawarłem warunki max. liczby iteracji (*maxit*) oraz górnej granicy elementów zerowanych (*up0*).
UWAGA 1: wszystkich iteracji było 30 (tyle ile macierzy)
UWAGA 2: biały kolor – wyniki dla metody bez przesunięć, niebieski - z przesunięciami

a) średnia ilość iteracji - macierze symetryczne

Dla macierzy w zwykłej postaci:

Dla *maxit* = 1000 i *up0* = 0.00001 mamy takie wyniki:

Rozmiar macierzy:	5	5	10	10	20	20
Udane iteracje:	30	30	25	30	18	30
Średnia ilość iteracji:	65.8	8.5	196.6	15.3	531.7	29.7

Dla *maxit* = 1500 i *up0* = 0.0000001 mamy takie wyniki:

Rozmiar macierzy:	5	5	10	10	20	20
Udane iteracje:	30	30	25	30	20	30
Średnia ilość iteracji:	124.9	9.3	295.9	16.8	825.4	32.4

Dla *maxit* = 2000 i *up0* = 0.000000001 mamy takie wyniki:

Rozmiar macierzy:	5	5	10	10	20	20
Udane iteracje:	27	30	30	30	19	30
Średnia ilość iteracji:	146.6	9.8	451	18.1	801.5	35.1

Dla macierzy trójdiagonalnej:

b) średnia ilość iteracji - macierze niesymetryczne

Dla *maxit* = 1000 i *up0* = 0.00001 mamy takie wyniki:

Rozmiar macierzy:	5	10	20
Udane iteracje:	30	30	30
Średnia ilość iteracji:	10.7	22.4	50.0

Dla *maxit* = 1500 i *up0* = 0.0000001 mamy takie wyniki:

Rozmiar macierzy:	5	10	20
Udane iteracje:	30	30	30
Średnia ilość iteracji:	11.3	24.6	50.8

Dla *maxit* = 2000 i *up0* = 0.000000001 mamy takie wyniki:

Rozmiar macierzy:	5	10	20
Udane iteracje:	30	30	30
Średnia ilość iteracji:	13.6	25.9	55.6

c) porównanie wyników z wartościami własnymi obliczonymi poleceniem *eig*

Dla macierzy niesymetrycznej 5x5:

0,242642	0,065371	0,036122	0,358549	0,578669
0,628819	0,816841	0,126167	0,207167	0,295455
0,15391	0,747382	0,685629	0,401966	0,004955
0,811663	0,587144	0,567019	0,842753	0,131829
0,830686	0,033185	0,260106	0,28643	0,091482

Algorytm z przesunięciem:		*eig( )*	
Re	**Im**	**Re**	**Im**
1,962976	-5,30E-14	1,962976	0
-0,56635	-7,43E-13	-0,56635	0
0,542652	0,363987	0,542652	0,363988
0,542652	-0,36399	0,542652	-0,36399
0,197416	0	0,197417	0

Dla macierzy symetrycznej 5x5:

Algorytm z przesunięciem:		*eig( )*	
Re	**Im**	**Re**	**Im**
5,064631	0	5,064631	0
1,237736	0	1,237736	0
-0,33539	0	-0,33539	0
-0,98736	0	-0,98736	0
-1,12134	0	-1,12134	0

Komentarz i wnioski do otrzymanych wyników

Wniosek dotyczący wydajności danych algorytmów, średniej ilości iteracji
Istotnym wnioskiem, że algorytm z przesunięciami jest znacznie wydajniejszy od algorytmu bez przesunięć.
Ponadto, działa on dla macierzy symetrycznych jak i niesymetrycznych. Dla obu macierzy symetrycznej jest zdecydowanie szybciej zbieżny – liczba iteracji jest parę razy mniejsza (jak nie parenaście w niektórych przypadkach).
Warto zauważyć, że algorytm bez przesunięć przy danej liczbie iteracji maksymalnych nie radził sobie ze wszystkimi macierzami.

Wniosek podsumowujący algorytm z przesunięciami i wewnętrznie wbudowany w Matlaba:
Na podstawie porównania wyników uzyskanych za pomocą algorytmu z przesunięciami i funkcji wewnętrznej *eig* możemy zauważyć, że wartości własne macierzy są właściwie takie same.

Zadanie 2

Treść:

Dla przedstawionych danych, metodą najmniejszych kwadratów należy wyznaczyć funkcję wielomianową *y=f(x)* najlepiej aproksymującą te dane (proszę przetestować wielomiany różnych rzędów).

a) w sprawozdaniu proszę przedstawić na rysunku otrzymaną funkcję na tle danych.
b) do rozwiązania zadania najmniejszych kwadratów proszę wykorzystać: a) układ równań normalnych, b) układ równań liniowych z macierzą R wynikającą z rozkładu QR macierzy układu równań problemu.
c) dla każdego układu równań proszę obliczyć błąd rozwiązania jako normę residuum.

Ogólny zarys:

a) metoda najmniejszych kwadratów

Jest to metoda przybliżania rozwiązań układów równań nadokreślonych, tzn. równań w których jest więcej równań niż niewiadomych. Określenie najmniejszych kwadratów mówi, że końcowe rozwiązanie ma zminimalizowaną sumę kwadratów błędów przy rozwiązaniu każdego z równań.

Metoda ta daje wynik o najmniejszej sumie kwadratów błędów. Nie ma jednak gwarancji, iż wynik ten ma praktyczny sens. Jeśli jedna z danych wystąpią pewne zakłócenia, które są bardzo oddalone od reszty, wówczas dane te przyciągną do siebie linię trendu. W przypadku stosowania należy więc usunąć ewentualne elementy odstające.

Mamy następujące implementacje metody najmniejszych kwadratów:

i. metoda układu równań normalnych, czyli takich, w którym rząd $k = n$ (macierz pełnego rzędu). W takim przypadku algorytm rozwiązywania zadania najmniejszych kwadratów przyjmuje postać: $A^T Ax = A^T b$
ii. metoda rozkładu **QR** macierzy **A**, aby rozwiązać równoważne układowi równań normalnych, równanie: (gdy macierz **A**, źle uwarunkowana) $Rx = Q^T b$

b) aproksymacja

Aproksymacja ma za zadanie przybliżenie funkcji *f(x)* (określona w dokładnie zadanym przedziale lub w przybliżeniu) inną prostszą funkcją *F(x)* należącą do wybranej klasy funkcji aproksymujących.

Funkcja aproksymująca może być przedstawiana w różnej postaci:

i. wielomianu (wówczas wielomianowa) (nasz przypadek)
ii. funkcji sklejanych (*splajn*, stopnia *s* to dowolna funkcja która w każdym przedziale jest wielomianem stopnia co najwyżej *s* oraz jej pochodne rzędu *1, 2, … s-1* są ciągłe dla wszystkich argumentów)
iii. sztucznych sieci neuronowych

Aproksymacje można wykorzystać np. w sytuacji, gdy nie istnieje **funkcja analityczna** pozwalająca na wyznaczenie wartości dla dowolnego z jej argumentów, a jednocześnie wartości tej nieznanej funkcji są dla pewnego zbioru jej argumentów znane.

Innym przykładem wykorzystanie (mam to zrobić w tym zadaniu) mam wyznaczyć funkcję najlepiej aproksymującą dane.

c) rodzaje aproksymacji

Istnieje wiele metod aproksymujących:

i. aproksymacja jednostajna ciągła

ii. aproksymacja średniokwadratowa

iii. aproksymacja liniowa

iv. aproksymacja jednostajna dyskretna (punktowa)

v. aproksymacja średniokwadratowa dyskretna (metoda najmniejszych kwadratów)

d) opis aproksymacji średniokwadratowej dyskretnej

Niech f(x) przyjmuje na pewnym zbiorze punktów znane wartości. (tak jak w naszym zadaniu)

Zadanie:

Wyznaczyć wartości współczynników określających funkcję aproksymującą tak, aby zminimalizować błąd średniokwadratowy:

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

gdzie:

x_j – dany *j-ty* punkt

$f(x_j)$ – wartość funkcji w danym *j-tym* punkcie

$\phi_i(x)$ – *i-ta* funkcja bazowa w przestrzeni funkcji aproksymujących

a_i – wartość *i-tego* współczynnika

Wyznaczamy poszczególne współczynniki z warunków koniecznych minimum (dostatecznych i jednoznacznych). Powstaje nam wówczas układ równań liniowych względem współczynników a_i tzw. układ równań normalnych. Macierz tego układu to tzw. *macierz Grama.*

Wówczas funkcja celu wygląda w sposób następujący:

$$H(a) = (\|y - Aa\|_2)^2$$

Zadnie aproksymacji średniokwadratowej jest więc liniowym zadanie najmniejszych kwadratów. (*LZNK*)

Prezentacja otrzymanych wyników w postaci wykresów i tabel:

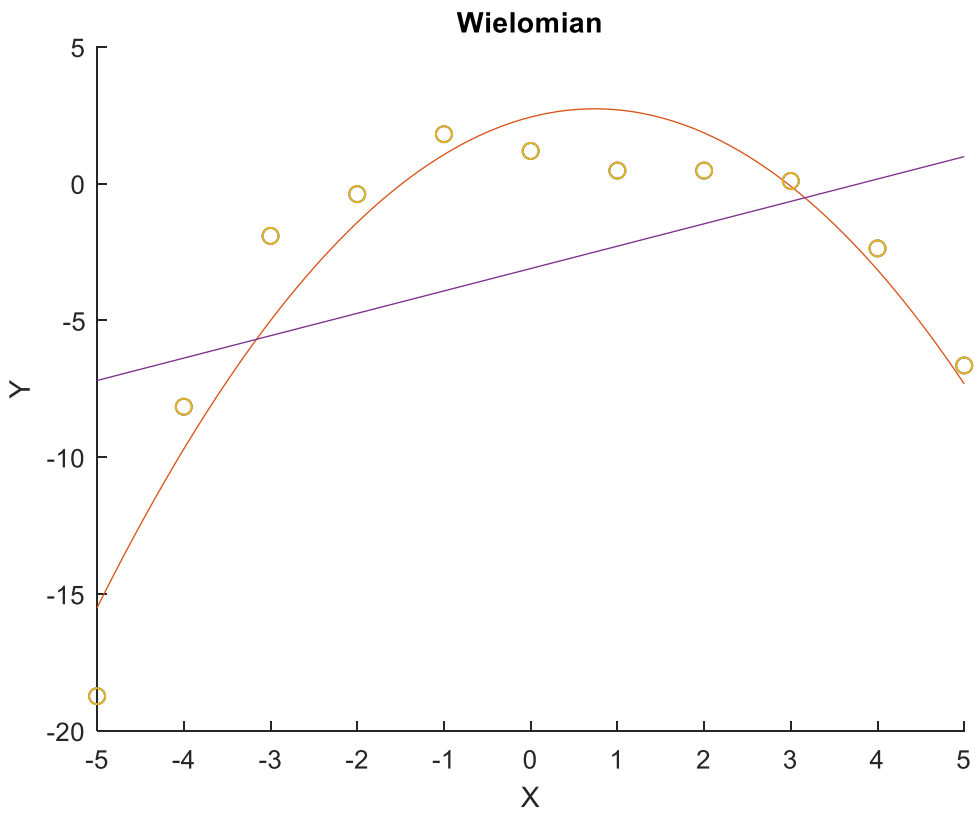
Poniższa tabela prezentuje błędy w zależności od metody oraz stopnia wielomianu.

stopień wielomianu	norma dla podpunktu a)	norma dla podpunktu b)
0	19,23985656	19,23985656
1	17,22043575	17,22043575
2	5,788226947	5,788226947
3	3,860804608	3,860804608
4	1,197957855	1,197957855
5	1,183177907	1,183177907
6	1,183177836	1,183177836
7	1,107039823	1,107039823
8	0,87285679	0,87285679
9	0,387174749	0,387174749
10	7,63E-10	1,36E-11
11	7,63E-10	15,2341121
12	4,25E-10	89,68611693
13	7,95E-10	99,24743155
14	4,34E-10	162,4885047
15	4,53E-10	28,03487698
16	2,06E-10	120,5797022
17	1,10E-10	710,0575041
18	4,91E-10	1118,443785
19	2,59E-11	695354,5939

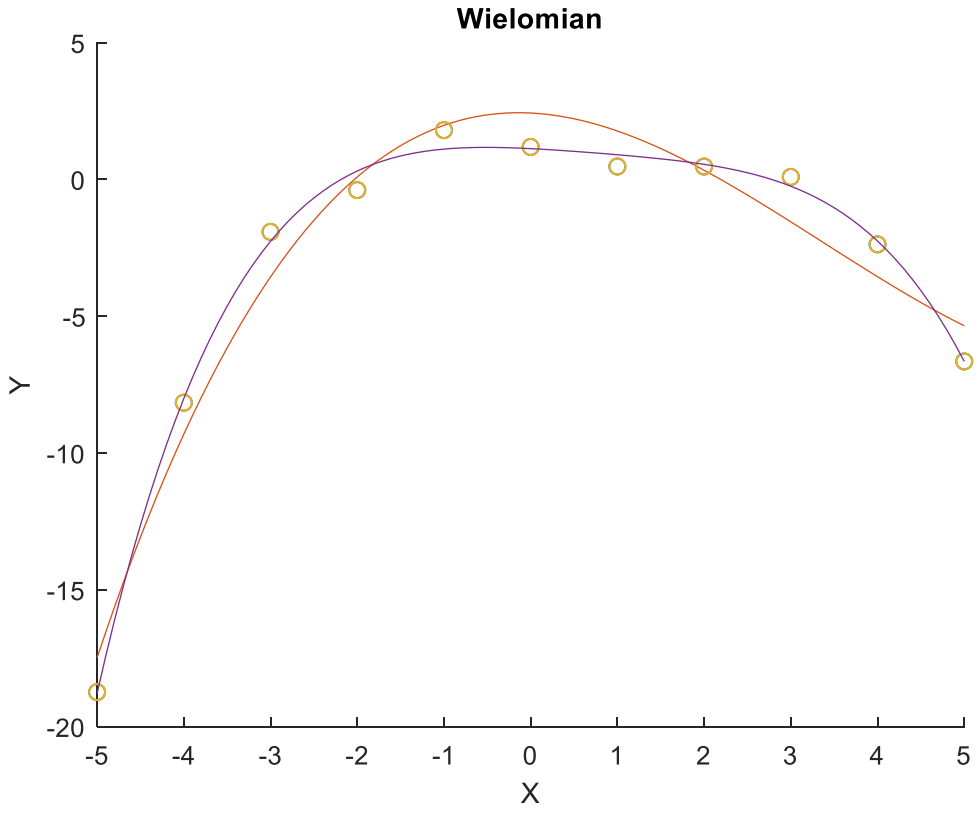
UWAGA: rysowana funkcja jest próbkowana 10 razy częściej niż dane.

Przedstawienie funkcji na tle danych dla:

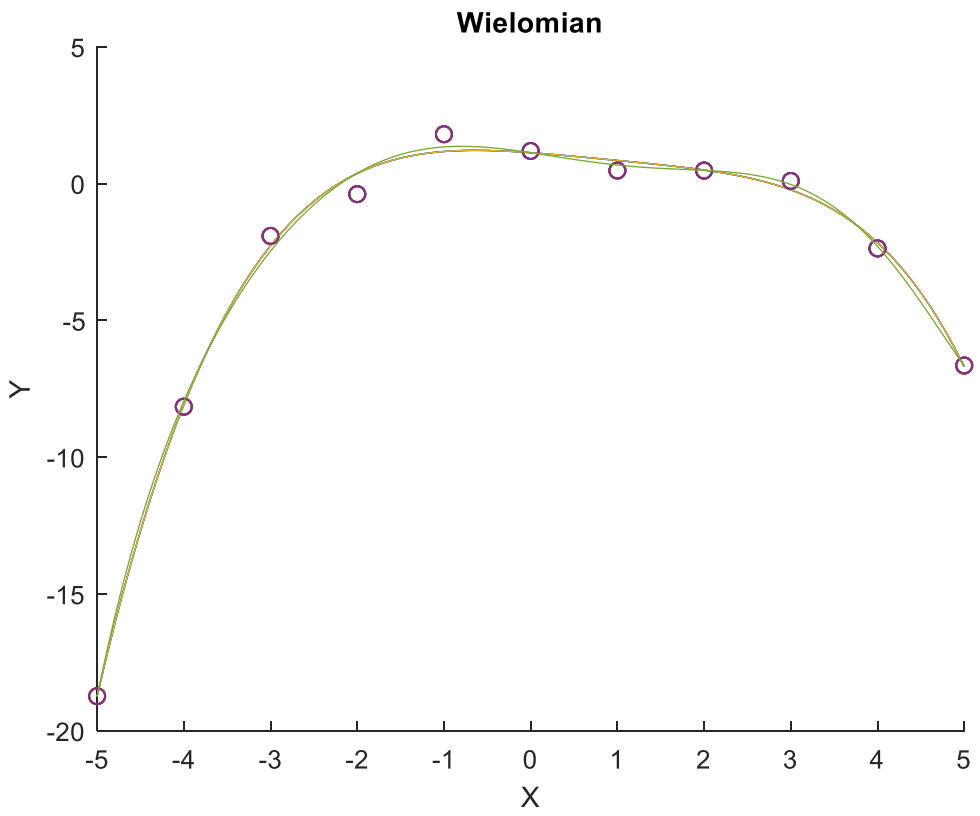
i. wielomianów stopnia 1 i 2:



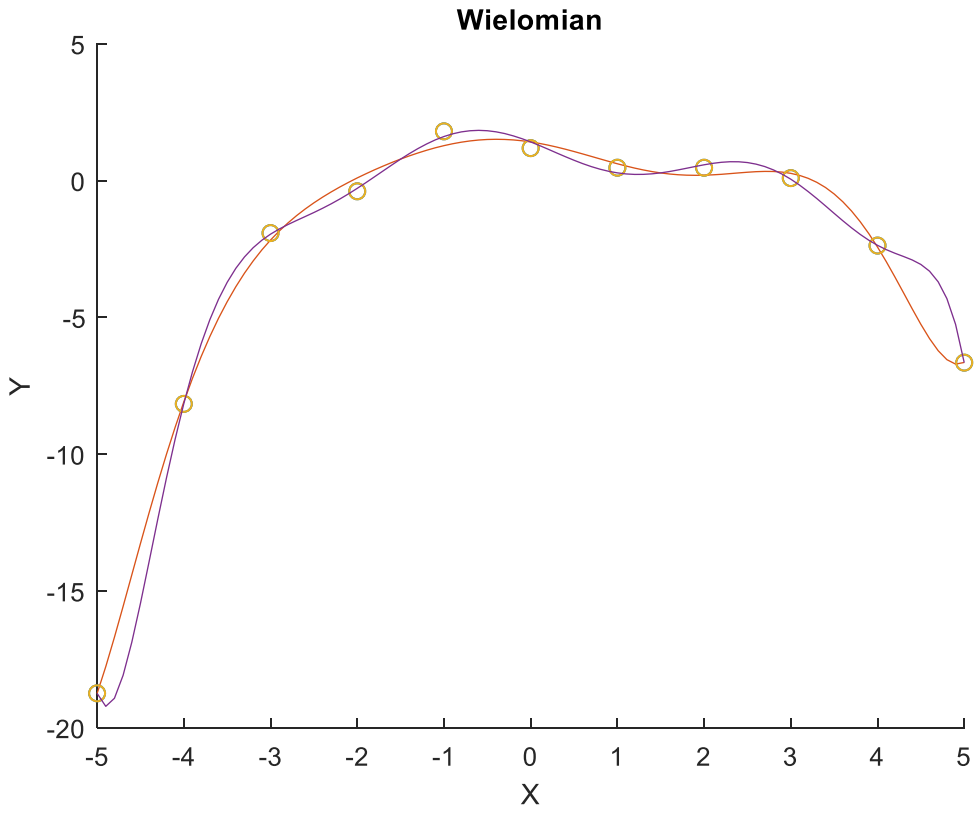
ii. wielomianów stopnia 3 i 4:



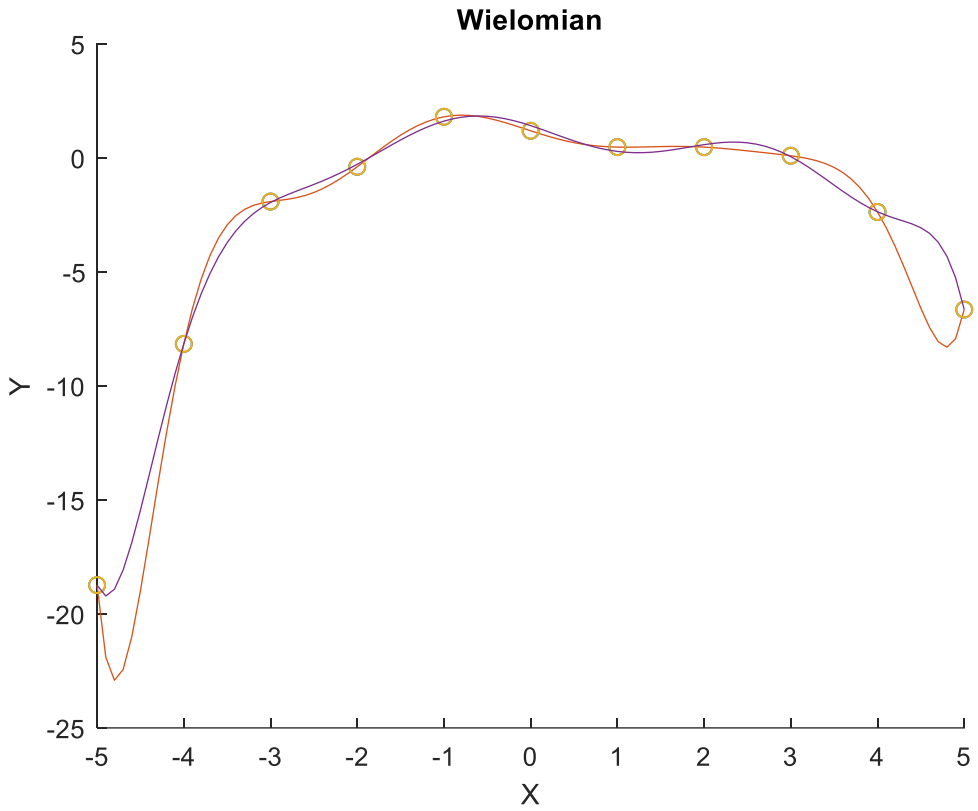
iii. wielomianów stopnia 6 i 7:



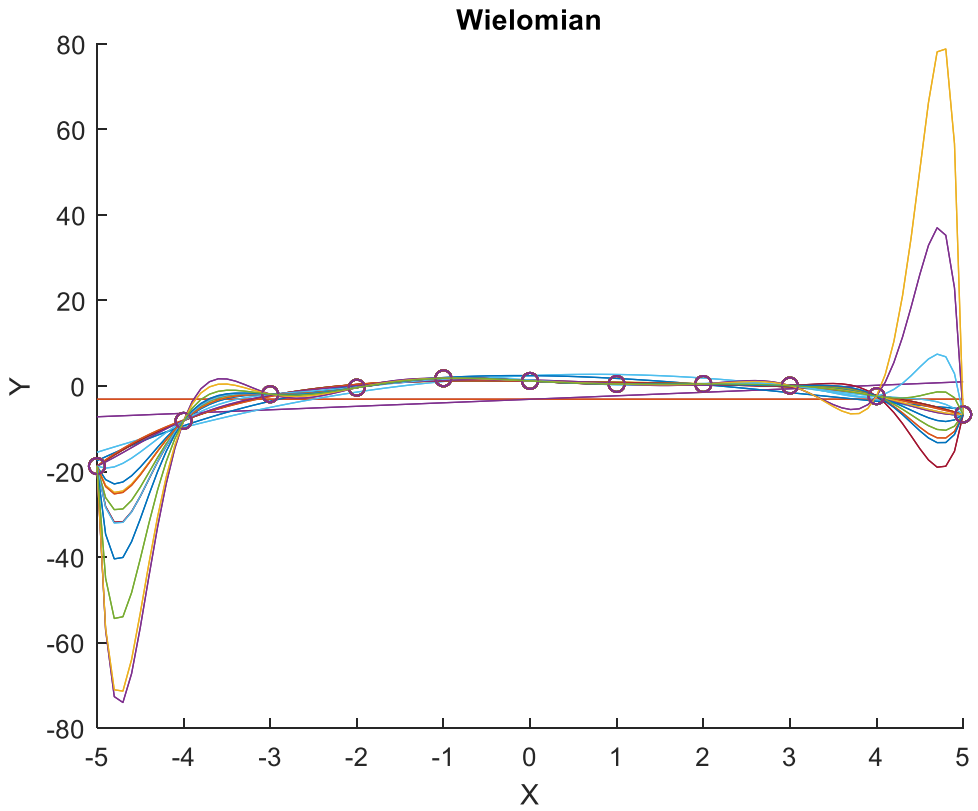
iv. wielomianów stopnia 8 i 9:



v. wielomianów stopnia 9 i 10:



vi. wszystkich wielomianów stopnia od 0 do 19 na jednym rysunku:



Funkcją najlepiej aproksymującą dane wykorzystując **układ równań liniowych** jest:

n, gdzie n wykładnik potęgi *x*	współczynniki dla URN
10	4,59E-05
9	-0,000106291
8	-0,002315499
7	0,005609511
6	0,036853344
5	-0,096585477
4	-0,218260252
3	0,667470709
2	0,133076546
1	-1,240988452
0	1,189

Wykorzystując **układ równań normalnych** widzimy, iż w miarę wzrostu stopnia błąd maleje, lecz dane są obarczone pewnym błędem pomiarowym, więc wstawię współczynniki dla *n =15*, gdyż różnica pomiędzy błędem dla stopnia *15* i *19* nie są duże (biorąc pod uwagę szumy pomiarowe):

n, gdzie n to wykładnik potęgi przy x	Współczynniki dla URN
15	-2,78E-08
14	-1,68E-08
13	-7,58E-07
12	2,96E-06
11	4,51E-05
10	-8,33E-05
9	0,000635496
8	-0,000104351
7	-0,030835203
6	0,020936356
5	0,254186254
4	-0,175115707
3	-0,398309372
2	0,10376408
1	-0,490321536
0	1,189

Komentarz do otrzymanych wyników:

Podczas przygotowywania projektu oraz uzyskiwania wyników możemy zaobserwować jak duży wpływ mają zastosowane metody. Metoda QR generuje znacznie większe błędy niż metoda układu równań normalnych. Zalecanym sposobem rozwiązywania aproksymacji przy źle uwarunkowanym układzie równań normalnych jest wykorzystanie rozkładu QR rozkładu A.

Kod do programów:

Zadanie 1:

a) program dla macierzy symetrycznej

```
function fun = zad1symetryczna_macierz()

    n = 5; %%inicjalizajca macierzy
    rozmiar = 30;
    up0 = 0.00001; %%ustawienie wartość up0 - ustawienie górnej granicy elemen-
tów zerowych, maxit - maksymalnej ilości iteracji
    maxit = 1000;

    while n<=20
        A = zeros(n, n); %%alokacja pamięci dla macierzy A
        goodit = 0;
        gooditbp = 0; %%bez przesunięc
        gooditzp = 0; %%z przesunięciami
        itbp = 0; %%bez przesunięc
        itzp = 0; %%z przesunięciami
        iterbp = 0;
        iterzp = 0;
        wek = zeros(n, 1);
        wekbp = zeros(n, 1); %%macierz przechowująca wektory własne metodą bez
przesunięć
        wekzp = zeros(n, 1); %%macierz przechowująca wektory własne metodą z
przeunięciami
        EIG = zeros(n, 1);


        for i = 1:rozmiar
           while(det(A) == 0)
               A = rand(n, n); %%wygenerowanie macierzy losowej
               A = A + A'; %%macierz symetryczna
           end
           [wekbp, itbp, goodit] = EigvalQRNoShift(A,up0,maxit);
           if goodit == 1
               gooditbp = gooditbp + 1 ;
               iterbp = iterbp + itbp;
           end
           [wekzp, itzp , goodit] = EigvalQRshifts(A,up0,maxit);
           if goodit == 1
               gooditzp = gooditzp +1 ;
               iterzp = iterzp + itzp;
           end
           wek = eig(A);
           EIT = eig(A);
           A = 0;

        end

        sredniabp = 0;
        sredniazp = 0;
        if gooditbp > 0
            sredniabp = iterbp/gooditbp;
        end
        if gooditzp > 0
```

```
            sredniazp = iterzp/gooditzp;
        end

        fprintf(1, "Wielkość macierzy: %d \n", n);
        fprintf(1, "(Średnia ilość iteracji dla metody bez przesunięć, ilość
udanych) = (%g, %d) \n", sredniabp, gooditbp);
        fprintf(1, "(Średnia ilość itracji dla metody z przeunięciami, ilość
udanych) = (%g, %d) \n", sredniazp, gooditzp);
        n = 2 * n; %%wystąpi generacja macierzy 5, 10, 20 - tak jak w poleceniu
    end
end
```

b) program dla macierzy niesymetrycznej

```
function fun = zad1niesymetryczna_macierz()

    n = 5; %%inicjalizajca macierzy
    rozmiar = 30;
    up0 = 0.00001; %%ustawienie wartość up0 - ustawienie górnej granicy elemen-
tów zerowych, maxit - maksymalnej ilości iteracji
    maxit = 1000;

    while n<=20
        A = zeros(n, n); %%alokacja pamięci dla macierzy A

        goodit = 0;

        gooditzp = 0; %%z przesunięciami
        itzp = 0; %%z przesunięciami
        iterzp = 0;


        wek = zeros(n, 1);
        wekzp = zeros(n, 1); %%macierz przechowująca wektory własne metodą z
przeunięciami
        EIG = zeros(n, 1);


        for i = 1:rozmiar
           while(det(A) == 0)
               A = rand(n, n); %%wygenerowanie macierzy losowej
               M = hess(A);
           end

           [wekzp, itzp , goodit] = EigvalQRshifts(A,up0,maxit);
           if goodit == 1
               gooditzp = gooditzp +1 ;
               iterzp = iterzp + itzp;
           end
           wek = eig(A);
           EIT = eig(A);
           A = 0;

        end

        sredniazp = 0;

        if gooditzp > 0
```

```
            sredniazp = iterzp/gooditzp;
        end


        fprintf(1, "Wielkość macierzy: %d \n", n);
        fprintf(1, "(Średnia ilość itracji dla metody z przeunięciami, ilość
udanych) = (%g, %d) \n", sredniazp, gooditzp);
        n = 2 * n; %%wystąpi generacja macierzy 5, 10, 20 - tak jak w poleceniu
    end
end
```

c) algorytm rozkładu QR

```
function [Q,R] = qrGS(A)
    %%rozkład macierzy mxn, gdzie m >= n, rzeczywistej lub zespolonej
    [m n]=size(A);      %%pobranie wymiarów macierzy

    %%alokacja pamięci macierzy Q i R
    Q=zeros(m,n);
    R=zeros(n,n);

    %%alokacja macierzy przechowująca daną kolumnę
    kolumna = zeros(1,n);

    for (i = 1:n)
        Q(:,i)=A(:,i);  %%kolumna i-ta macierzy A staje się i-tą kolumną macie-
rzy Q
        R(i,i)=1;   %%element na diagonali macierzy R = 1
        kolumna(i)=Q(:,i)'*Q(:,i);
        for (j = (i+1):n) %% dla kolejnej kolumny
            R(i,j)=(Q(:,i)'*A(:,j))/kolumna(i); %%element R jest obliczany zgod-
nie z algorytmem
            A(:,j)=A(:,j)-R(i,j)*Q(:,i); %%każda kolejna kolumna macierzy A jest
zmniejszana o iloczyn danego elementu z macierzy R i odpowiedniej kolumn Q
        end
    end

    %%normalizujemy kolumny macierzy Q
    for (i = 1:n)
        norma = norm(Q(:,i));
        Q(:,i)= Q(:,i)/norma;
        R(i,i:n)= R(i,i:n)*norma;
    end
end
```

d) algorytm QR bez przesunięć:

```
function [eigenvalues, it, goodit] = EigvalQRNoShift(A, up0, maxit)
%%up0-górna granica elementów zerowanych
%%maxit-maksymalna liczba iteracji


    it=1; %%inicjalizacja iteratora
    goodit = 1; %%sprawdzenie czy liczba iteracji nie przekroczyła maxit
    n=size(A,1); %%pobranie rozmiaru macierzy
    eigenvalues=diag(zeros(n)); %%alokacja pamięci dla macierzy diagonalnej
przechowującej wartości własne macierzy A
```

```
    Ak = A; %%alukuję pamięć w celu nie zmienienia macierzy A

    while (it <= maxit && max(max(Ak-diag(diag(Ak)))) > up0) %%wykonuj aż do na-
stępujących warunków - liczba it przekroczy itmax lub
        [Q,R]=qrGS(Ak); %%rozkład QR metodą Grama-Schmidta
        Ak=R*Q; %%zgodnie z metodą A w następnej iteracji będzie obliczone w ten
sposób
        it = it + 1; %%zwiększamy wartość iteratora
    end

    if it>maxit
       goodit = 0; %%jeśli nie udało się osiągnąć wyniku w mniej niż itmax ite-
racji wtedy oznacza, że się nie udało obliczyć wartości własnej z daną dokładno-
ścią
    end

    eigenvalues=diag(Ak);
end
```

e) algorytm QR z przesunięciami:

```
function [eigenvalues, it, goodit] = EigvalQRshifts(A,up0,maxit)
%%up0-górna granica elementów zerowanych
%%maxit-maksymalna liczba iteracji
    n=size(A,1);    %%pobieramy rozmiar macierzy
    eigenvalues=diag(zeros(n)); %%alokacja pamięci dla macierzy diagonalnej
przechowującej wartości własne macierzy A
    Ak = A;     %%macierz początkowa (oryginalna)
    it=0;           %%iterator
    goodit=1;           %%jesli został przekroczony maxit wówczas uznajemy że
algorytm nie dał rady
    for k=n:-1:2
        DK=Ak;  %%DK - macierz startowa dla jednej wartości własnej
        it=0;
        while (it <= maxit && max(abs(DK(k,1:k-1))) > up0)
       %%wykonuj aż do następujących warunków - liczba it przekroczy itmax lub
            DD=DK(k-1:k,k-1:k); %%podmacierz dolnego prawego rogu
            e=[1,-(DD(1,1)+DD(2,2)),DD(2,2)*DD(1,1)-DD(1,2)*DD(2,1)];
            r=roots(e); %%obliczamy pierwiastki wielomianu e
            if abs(r(1,1)-DD(2,2)) < abs(r(2,1)-DD(2,2))    %%wartość własna
podmacierzy DD
                shift=r(1,1);
            else
                shift=r(2,1);
            end
            DK=DK-eye(k)*shift;     %%macierz przesunięta
            [Q,R]=qrGS(DK);     %%faktoryzacja QR
            DK=R*Q+eye(k)*shift;        %%dodajemy przesunięcia
            it=it+1;                %%zwiększamy iterator
        end
        if it > maxit
            goodit = 0;     %%%%jeśli nie udało się osiągnąć wyniku w mniej niż
itmax iteracji wtedy oznacza, że się nie udało obliczyć wartości własnej z daną
dokładnością
        end
        eigenvalues(k)=DK(k,k);     %%zapisujemy jako wartość własna element z
rogu macierzy
        if k>2
            Ak=DK(1:k-1,1:k-1); %%macierz - deflacja
        else
            eigenvalues(1)=DK(1,1);         %%jeżeli to ostatnia macierz własna
```

```
        end
    end
end
```

Zadanie 2:

a) rozkład QR:

```
function [Q,R] = qrGS(A)
    %%rozkład macierzy mxn, gdzie m >= n, rzeczywistej lub zespolonej
    [m n]=size(A);      %%pobranie wymiarów macierzy

    %%alokacja pamięci macierzy Q i R
    Q=zeros(m,n);
    R=zeros(n,n);

    %%alokacja macierzy przechowująca daną kolumnę
    kolumna = zeros(1,n);

    for (i = 1:n)
        Q(:,i)=A(:,i);  %%kolumna i-ta macierzy A staje się i-tą kolumną macie-
rzy Q
        R(i,i)=1;   %%element na diagonali macierzy R = 1
        kolumna(i)=Q(:,i)'*Q(:,i);
        for (j = (i+1):n) %% dla kolejnej kolumny
            R(i,j)=(Q(:,i)'*A(:,j))/kolumna(i); %%element R jest obliczany zgod-
nie z algorytmem
            A(:,j)=A(:,j)-R(i,j)*Q(:,i); %%każda kolejna kolumna macierzy A jest
zmniejszana o iloczyn danego elementu z macierzy R i odpowiedniej kolumn Q
        end
    end

    %%normalizujemy kolumny macierzy Q
    for (i = 1:n)
        norma = norm(Q(:,i));
        Q(:,i)= Q(:,i)/norma;
        R(i,i:n)= R(i,i:n)*norma;
    end
end
```

b) funkcja rysująca

```
function [norma1, norma2] = zad2(stopien)

X = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]';
Y = [-18.7370, -8.1583, -1.9146, -0.3887, 1.8030, 1.1890, 0.4738, 0.4726,
0.0941,  -2.3716, -6.6512]';
%obliczenie macierzy
[a1,norma1,a2,norma2] = obliczenia(X,Y,stopien);
disp(norma1); %wyswietlenie wartości normy
disp(norma2); %wyświetlenie wartości normy
% Dane probkowane 10 razy czesciej
t=-5:0.1:5;
wsp1 = polyval(a1,t);
wsp2 = polyval(a2,t);


%Rysowanie wykresu
polyfit(X,Y,stopien);
```

```
%plot(t, wsp1);
title('Wielomian stopnia 1 i 2');
xlabel('X')
ylabel('Y')
hold on
scatter(X,Y)
plot(t, wsp1)
hold off
```

c) funkcja obliczająca dane wartości współczynników:

```
function [a1,norma1,a2,norma2]=obliczenia(X,Y,stopien)
    [rozmiar, ~]  = size(X); %macierz A
    A = zeros(rozmiar, stopien+1);
    %Wypelniamy macierz A potegami elementow x %
    for i=1:rozmiar
        for j=0:stopien
            A(i,stopien + 1 - j) = X(i)^(j);
        end
    end
    %układ równań normalnych
    a1 = (A'*A)\(A'*Y);
    %układ równań liniowych z macierzą R wynikającą z rozkładu QR
    [Q, R] = qrGS(A);
    a2 = R\(Q'*Y);
    % Wyliczanie normy euklidesowej
    norma1 = norm(A*a1-Y);
    norma2 = norm(A*a2-Y);
end
```

d) menu programu

```
n = 20;
norm1 = zeros(n, 1);
norm2 = zeros(n,1);

for i=0:n-1
    [norm1(i+1, 1), norm2(i+1, 1)] = zad2(i);

end

test1 = 0;
```