Sprawozdanie MNUM Projekt 02

Autor: **TOMASZ SACHANOWSKI**

Grupa: **czwartek 8-10**

Nr. Indexu: **276467**

Nr. Zadania**: 2.55**

Spis treści

Treść zadań

MNUM–PROJEKT, zadanie 2.55

1. 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. Następnie proszę przetestować skuteczność (zbieżność) obu wersji algorytmu dla 30 różnych macierzy losowych o wymiarach: 5×5, 10×10 i 20×20. Proszę podać średnią liczbę iteracji dla metody bez przesunięć i z przesunięciami. Dla wybranych macierzy proszę porównać otrzymane wyniki z wartościami własnymi obliczonymi poleceniem *eig*.

   Uwagi:
   – W programie nie można wykorzystać dostępnych w Matlabie poleceń *qr* i *eig*.
   – Macierz $\mathbf{B}=(\mathbf{A}+\mathbf{A}^T)$ jest macierzą symetryczną dla dowolnej macierzy **A**.

2. Dla następujących danych pomiarowych (próbek):

x_i	y_i
–5	23,4523
–4	11,9631
–3	4,4428
–2	1,1010
–1	–1,6826
0	–1,2630
1	–0,0357
2	–1,3156
3	–3,4584
4	–8,4294
5	–18,4654

   metodą najmniejszych kwadratów należy wyznaczyć funkcję wielomianową $y=f(x)$ najlepiej aproksymującą te dane (proszę przetestować wielomiany różnych rzędów). W sprawozdaniu proszę przedstawić na rysunku otrzymaną funkcję na tle danych. 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.
   Dla każdego układu równań proszę obliczyć błąd rozwiązania jako normę residuum (wektor residuum $\mathbf{r} = \mathbf{Ax}–\mathbf{b}$).

   Uwagi:
   – Rysowaną funkcję proszę próbkować 10 razy częściej niż dane.
   – Dane są obarczone pewnym błędem (szumem pomiarowym).

Programy muszą być napisane w Matlabie.
Sprawozdanie powinno zawierać:
- krótki opis zastosowanych algorytmów,
- wydruki dobrze skomentowanych programów z implementacją użytych algorytmów,
- prezentację otrzymanych wyników,
- komentarz do otrzymanych wyników oraz wnioski z eksperymentów (ocena poprawności wyników, dokładności, efektywności algorytmów itd.).

Sprawozdanie powinno być wysłane na adres prowadzącego:
a.krzemienowski@elka.pw.edu.pl.

Zadanie 1

Cel:

Celem jest napisanie programu do obliczenia wartości własnych macierzy nieosobliwych metodą rozkładu QR w wersjach z przesunięciem i bez przesunięcia dla macierzy symetrycznych oraz w wersji z przesunięciem dla macierzy niesymetrycznej.

Teoria:

Zdefiniowanie pojęć:

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

Sposoby numeryczne rozkładu QR:

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

Wartości i wektory własne macierzy kwadratowej rzeczywistej A są to takie liczby λ i odpowiadające im wektory v, że

$$Av= \lambda v,$$

gdzie λ – wartość własna, v – odpowiadający jej wektor własny.
Wartości i wektory własne spełniają więc równanie:

$$(A - \lambda I)\, v = 0.$$

Macierz kwadratowa n-wymiarowa ma dokładnie n wartości własnych i odpowiadających im wartości własnych. Zbiór wszystkich wartości własnych macierzy nazywany jest widmem macierzy. Wartości własne odgrywają ważną rolę w wielu dziedzinach nauki i techniki.

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.

Wybrane metody wyznaczania wartości własnych

- metoda QR (wykorzystana przeze mnie w tym zadaniu, opisana w następnym punkcie)
- 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.
- metoda wyznacznikowa:
  Metoda korzysta z faktu, iż wartości własne są zerami wielomianu charakterystycznego obliczając wartości własne wprost z definicji ***det(A− λI) = 0.***
- metoda QR – najbardziej ogólna, efektywna

Rozkład QR

Każdą macierz rzeczywistą $A_{m\,x\,n}$ ($m \geq n$) o liniowo niezależnych kolumnach można przedstawić w postaci:

$$\boldsymbol{A_{m\,x\,n} = Q_{m\,x\,n}\, R_{m\,x\,n}}$$

gdzie $\boldsymbol{Q_{m\,x\,n}}$ jest macierzą o kolumnach ortonormalnych, a $\boldsymbol{R_{m\,x\,n}}$ jest macierzą trójkątną górną z dodatnimi elementami na diagonali.

algorytm podstawowy bez przesunięć:

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

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

$\boldsymbol{Q^{(k)}}$ jest ortogonalna, więc:

$$\boldsymbol{R^{(k)}=Q^{(k)^{-1}}A^{(k)}=Q^{(k)T}A^{(k)}}$$

skąd mamy

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

Widzimy, że macierz $\boldsymbol{A^{(k+1)}}$ jest przekształconą przez podobieństwo macierzą $\boldsymbol{A^{(k)}}$, więc ma te same wartości własne.

Czyli macierz $\boldsymbol{A^{(k+1)}}$ jest przekształconą przez podobieństwo macierzą $\boldsymbol{A^{(k)}}$. Mają te same wartości własne.
Podsumowując dla macierzy symetrycznej $\boldsymbol{A}$, $\boldsymbol{A^{(k)}}$ zbiega do macierzy diagonalnej.

Metoda QR bez przesunięć – algorytm podstawowy:

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

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

Dla macierzy $\boldsymbol{A}$ symetrycznej macierz $\boldsymbol{A^{(k)}}$ zbiega do macierzy diagonalnej $\boldsymbol{diag\{\lambda_i\}}$.Jeśli macierz A ma n wartości o własnych o różnych modułach,

$$|\lambda_1|>\ |\lambda_2|>\ |\lambda_3|>\cdots>\ |\lambda_n|>0,$$

to można pokazać, że:

$$a_{i,,i}^{(k)}\xrightarrow[k\to\infty]{}\lambda_i,\quad i=1,\dots,n$$

$$a_{i+1,i}^{(k)}\xrightarrow[k\to\infty]{}0,\quad i=1,\dots,n-1$$

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

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

Stąd jeśli wartości własne leżą blisko siebie, to metoda jest wolno zbieżna. Aby poprawić szybkość zbieżności stosuje się algorytm metody QR z przesunięciami.

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):

$$A^{(k)} - p_k I = Q^{(k)} R^{(k)}$$

$$A^{(k+1)} = R^{(k)} Q^{(k)} + p_k I = Q^{(k)T}\left(A^{(k)} - p_k I\right) Q^{(k)} + p_k I = Q^{(k)T} A^{(k)} Q^{(k)}.$$

Zbieżność jest wtedy liniowa z ilorazem $\left|\frac{\boldsymbol{\lambda_{i+1} - p_k}}{\boldsymbol{\lambda_i - p_k}}\right|$. Wynika z tego, że najlepszym przesunięciem $\boldsymbol{p_k}$ byłoby aktualne przybliżenie $\boldsymbol{\lambda_{i+1}}$.

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.

Rozwiązanie:

Program został podzielony na kilka funkcji. Kod znajdujący się w ***Generate_symetric_matrix*** i ***Generate_symetric_matrix*** generuje nam odpowiedniej wielkości macierze. Następnie funkcja ***Decompose_QR*** rozkłada macierze na dwie Q i R. Został tu wykorzystany algorytm z książki. Jest to zmodyfikowany algorytm Grama-Schmidta, który działa następująco:

$A^{(1)} = A;$
for $i = 1 : n,$
$\bar{q}_i = a_i^{(i)}; \ \bar{r}_{ii} = 1;$
$d_i = \bar{q}_i^T \bar{q}_i ;$
for $j = i + 1 : n$
$\bar{r}_{ij} = \frac{\bar{q}_i^T a_j^{(i)}}{d_i}$
$a_j^{(i+1)} = a_j^{(i)} - \bar{r}_{ij}\bar{q}_i$
end
end

Zmodyfikowany algorytm w innej kolejności przeprowadza proces ortogonalizacji i ma znacznie lepsze własności numeryczne w porównaniu z algorytmem standardowym.
Standardowy algorytm ortogonalizuje kolumny macierzy po kolei, natomiast algorytm zmodyfikowany po wyznaczeniu kolejnej kolumny ortogonalnej od razu ortogonalizuje wobec niej wszystkie następne.

Kolejnymi krokami są funkcje wyznaczające wartości własne w macierzach są to odpowiednio: ***EigvalQRNoShift, EigvalQRShift.*** Na samym końcu działa funkcja ***compose_task_1*** pozwalająca przeprowadzić analizę dla kolejnych rozmiarów macierzy jak i ilości iteracji i błędów.

Wynik:

ŚREDNIA LICZBA ITERACJI						
	err 0.0001, maxit = 1000		*err 0.00001, maxit = 1000*		*err 0.000001, maxit = 1000*	
ALG. Z PRZESUNIECIEM	**NIE**	**TAK**	**NIE**	**TAK**	**NIE**	**TAK**
Macierz symetryczna 5x5	72.9000	7.2000	42.5172	7.5333	65.3333	8.0333
Macierz symetryczna 10x10	187.3571	13.3667	183.0741	14.3333	225.5185	14.4667
Macierz symetryczna 20x20	408.5000	26.7667	509.6500	28.1000	531.4444	29.7667
Macierz asymetryczna 5x5		9.3000		9.9333		9.9333
Macierz asymetryczna 10x10		19.2667		21.5000		22.6000
Macierz asymetryczna 20x20		41.2333		45.8333		47.4667

PROCENT UDANCYH PRÓB						
	err 0.0001, maxit = 1000		*err 0.00001, maxit = 1000*		*err 0.000001, maxit = 1000*	
ALG. Z PRZESUNIECIEM	**NIE**	**TAK**	**NIE**	**TAK**	**NIE**	**TAK**
Macierz symetryczna 5x5	1.0000	1.0000	0.9667	1.0000	1.0000	1.0000
Macierz symetryczna 10x10	0.9333	1.0000	0.9000	1.0000	0.9000	1.0000
Macierz symetryczna 20x20	0.6667	1.0000	0.6667	1.0000	0.6000	1.0000
Macierz asymetryczna 5x5		1.0000		1.0000		1.0000
Macierz asymetryczna 10x10		1.0000		1.0000		1.0000
Macierz asymetryczna 20x20		1.0000		1.0000		1.0000

ŚREDNI BŁĄD WZGLĘDEM EIG						
	err 0.0001, maxit = 1000		*err 0.00001, maxit = 1000*		*err 0.000001, maxit = 1000*	
ALG. Z PRZESUNIECIEM	**NIE**	**TAK**	**NIE**	**TAK**	**NIE**	**TAK**
Macierz symetryczna 5x5	0.1015	0.0000	0.1523	0.0000	0.0812	0.0000
Macierz symetryczna 10x10	0.1199	0.0000	0.1026	0.0000	0.1309	0.0000
Macierz symetryczna 20x20	0.0648	0.0000	0.0891	0.0000	0.1621	0.0000
Macierz asymetryczna 5x5		0.5645		0.5029		0.5746
Macierz asymetryczna 10x10		1.3914		1.3731		1.6333
Macierz asymetryczna 20x20		3.8106		3.8004		3.5644

Podsumowanie:

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.
Ponadto algorytm z przesunięciem radzi sobie dużo lepiej.

Zadanie 2

Cel:

Celem zadania jest znalezienie funkcji wielomianowej najlepiej aproksymującej podane dane, przy pomocy metody najmniejszych kwadratów

Teoria:

Liniowe zadanie najmniejszych kwadratów (LZNK) to zadanie polegające na rozwiązaniu układu m równań liniowych z n zmiennymi, gdzie $n<m$, w sensie minimalizacji normy drugiej wektora niespełnienia równań (wektora błędu).

Liniowe zadanie najmniejszych kwadratów jest zadaniem równoważnym minimalizacji następującej funkcji kwadratowej:

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

Metoda najmniejszych kwadratów:

Jest to metoda przybliżania rozwiązań układów równań nad określonych, 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:

- metoda układu równań normalnych, czyli takich, w którym rząd $k = n$ (macierz pełnego rzędu). W takim przypadku algorytm

rozwiązywania zadania najmniejszych kwadratów przyjmuje postać:

$$\boldsymbol{A^T A x = A^T b}$$

Problematyczne może być złe uwarunkowanie macierzy $\boldsymbol{A^T A}$. Jej wskaźnik uwarunkowania jest kwadratem wskaźnika uwarunkowania macierzy A i jest równy kwadratowi stosunku największej i najmniejszej wartości szczególnej macierzy. W przypadku źle uwarunkowanego układu równań normalnych zalecane jest skorzystanie z rozkładu QR macierzy A.

Metoda rozkładu QR macierzy A, aby rozwiązać równoważne układowi równań normalnych, równanie:(gdy macierz A, źle uwarunkowana). Korzystając z rozkładu QR macierzy A układ równań normalnych

$$\boldsymbol{A^T A x = A^T b}$$

możemy zapisać jako:

$$\boldsymbol{R^T Q^T Q R x = R^T Q^T b}$$

Uwzględniając ortonormalność kolumn macierzy Q ($\boldsymbol{Q^T Q = I}$) oraz nieosobliwość macierzy R otrzymujemy następujący układ równań liniowych:

$$\boldsymbol{Rx = Q^T b}$$

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:

- wielomianu (wówczas wielomianowa)(nasz przypadek)
- 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-1są ciągłe dla wszystkich argumentów)
- 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.

Rodzaje aproksymacji

- aproksymacja jednostajna ciągła
- aproksymacja średniokwadratowa
- aproksymacja liniowa
- aproksymacja jednostajna dyskretna (punktowa)
- aproksymacja średniokwadratowa dyskretna (metoda najmniejszych kwadratów)

Aproksymacja średniokwadratowa dyskretna

W tym przypadku zadanie polega na wyznaczeniu wartości współczynników $a0$, $a1$, … , an określających funkcję aproksymującą, tak aby zminimalizować błąd średniokwadratowy:

$$H(a_0, a_1, \dots, a_n) \stackrel{\text{def}}{=} \sum_{j=0}^{N} \left[f(x_j) - \sum_{i=0}^{n} a_i \phi_i(x_j) \right]^2$$

gdzie:
xj –dany j-ty punkt
$f(xj)$ –wartość funkcji w danym j-tym punkcie
$\phi i(x)$ –i-ta funkcja bazowa w przestrzeni funkcji aproksymujących
ai –wartość i-tego współczynnika

Warunkiem koniecznym minimum jest:

$$\frac{\partial H}{\partial a_k} = -2 \sum_{j=0}^{N} \left[f(x_j) - \sum_{i=0}^{n} a_i \phi_i(x_j) \right] \cdot \phi_k(x_j) = 0, \qquad k = 0, \dots, n,$$

Z powyższego warunku wyznaczyć można układ równań normalnych oraz macierz tego układu – tzw. macierz Grama. Układ ten zapisać można jako:

$$\langle \phi_i, \phi_k \rangle \stackrel{\text{def}}{=} \sum_{j=0}^{N} \phi_i(x_j)\phi_k(x_j)$$

Wyznaczamy poszczególne współczynniki z warunków koniecznych minimum (dostatecznych i jednoznacznych). Powstaje nam wówczas układ równań liniowych względem współczynników *ai* 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).

Wynik

Dla wszystkich stopni wielomianu otrzymałem takie same wyniki obiema metodami (układ równań normalnych i układ równań liniowych z macierzą R z rozkładu QR), co przedstawia poniższa tabela z błędami rozwiązania obliczonymi jako norma residuum w zależności od stopnia wielomianu. Natomiast dla wielomianów stopnia wyższego niż 9 metoda z rozkładem QR generowała większe błędy, a metoda z układem równań normalnych zera co może świadczyć o idealnym dopasowaniu. Dodatkowo Matlab zwracał Warning:

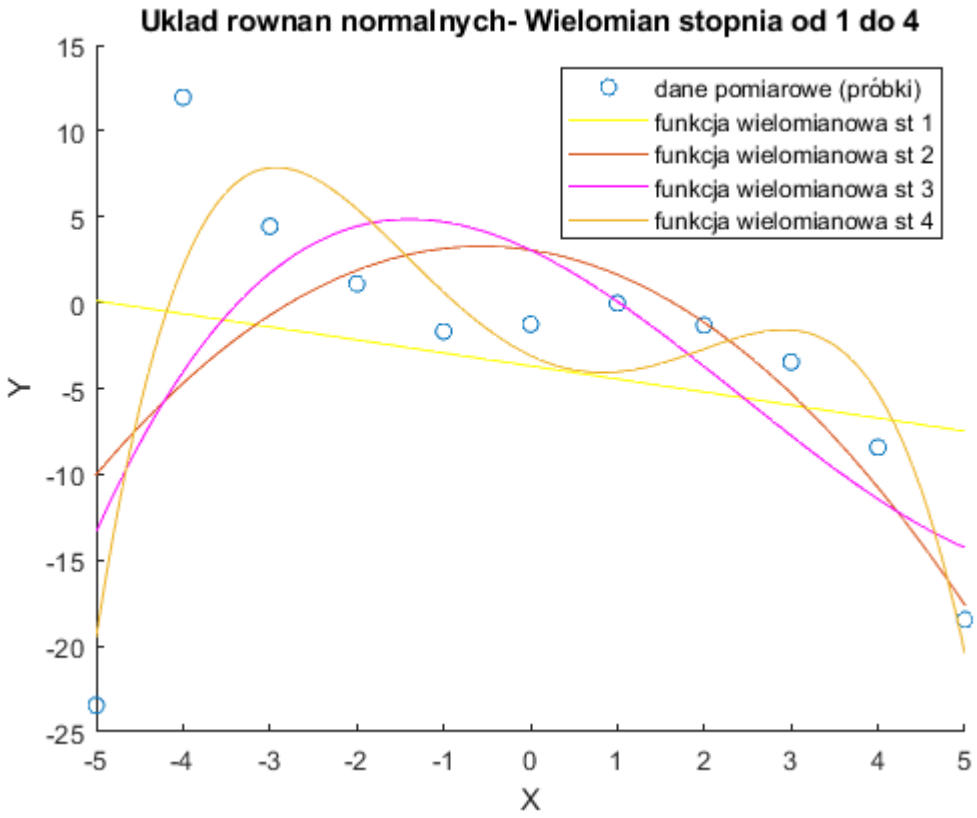
„Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 2.384828e-21.

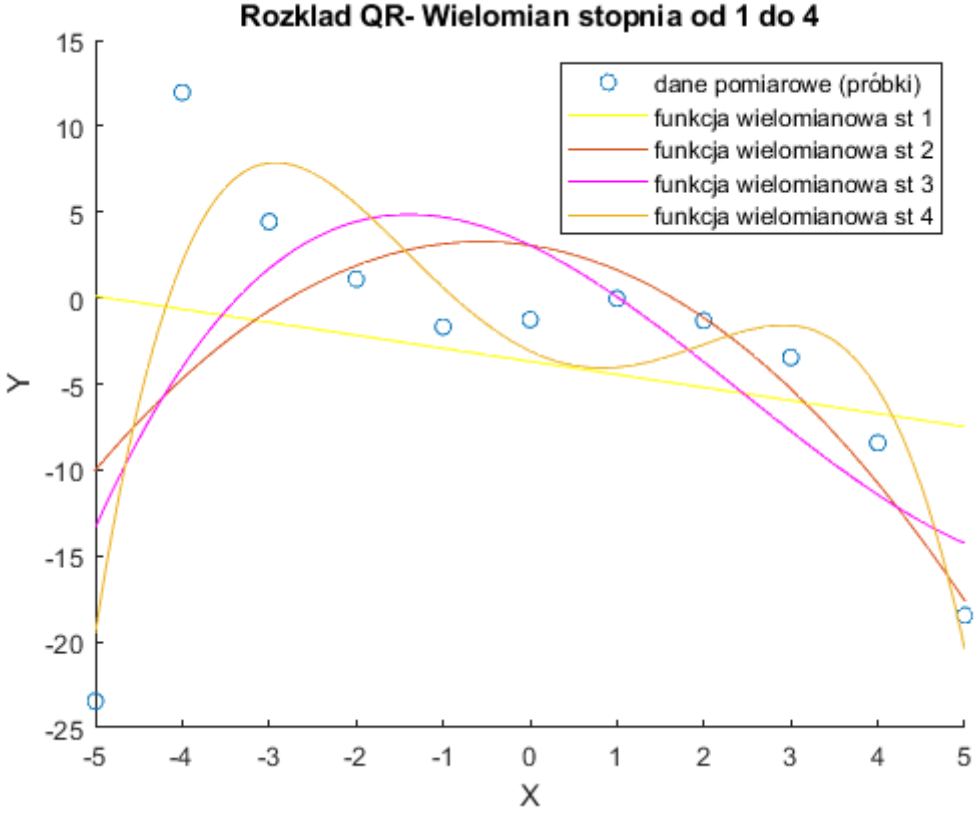
> In aproksymacja (line 15)

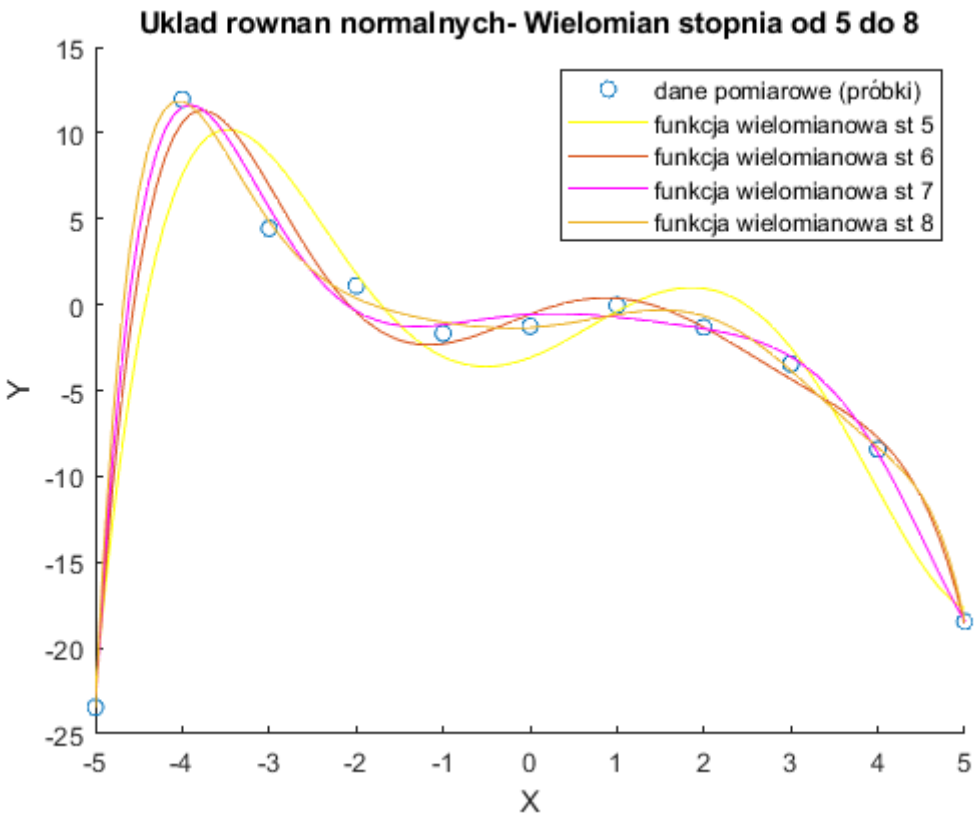
In compose_task_2 (line 33)”

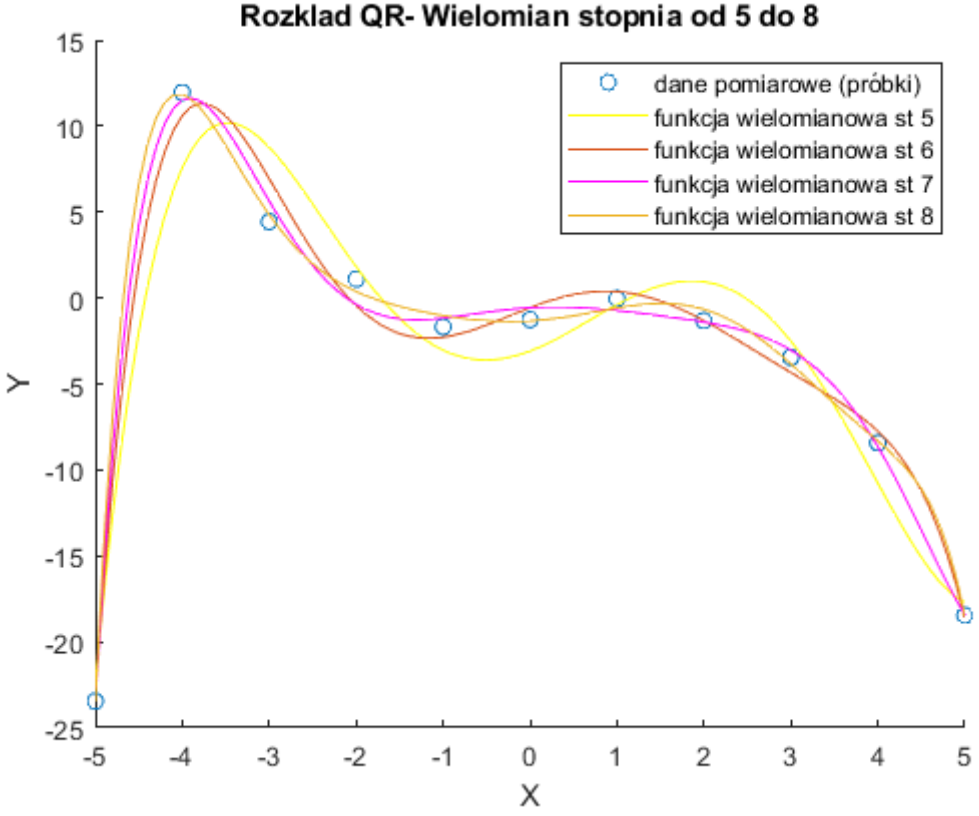
Stopień wielomianu	Błąd dla punktu a	Błąd dla punktu b
1	30.5120	30.5120

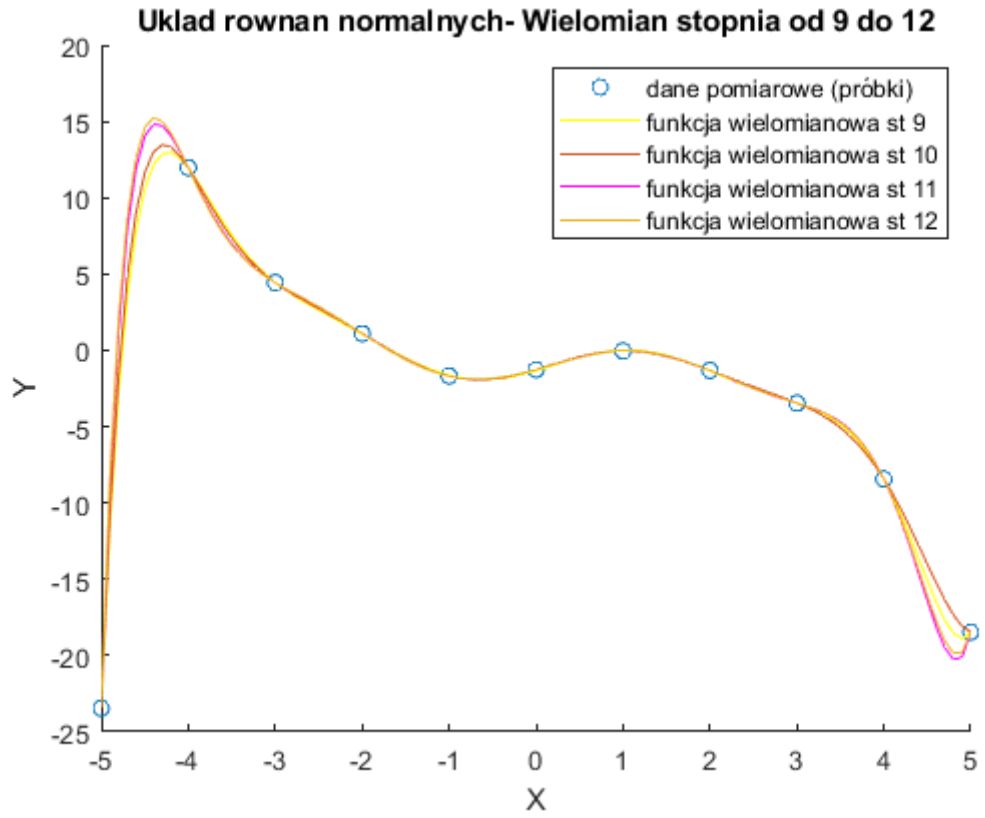
2	23.2386	23.2386
3	22.0614	22.0614
4	13.6689	13.6689
5	7.5716	7.5716
6	3.5851	3.5851
7	2.2968	2.2968
8	1.4241	1.4241
9	0.1158	0.1158
10	0.0000	0.0000
11	0.0000	12.7645
12	0.0000	135.7367

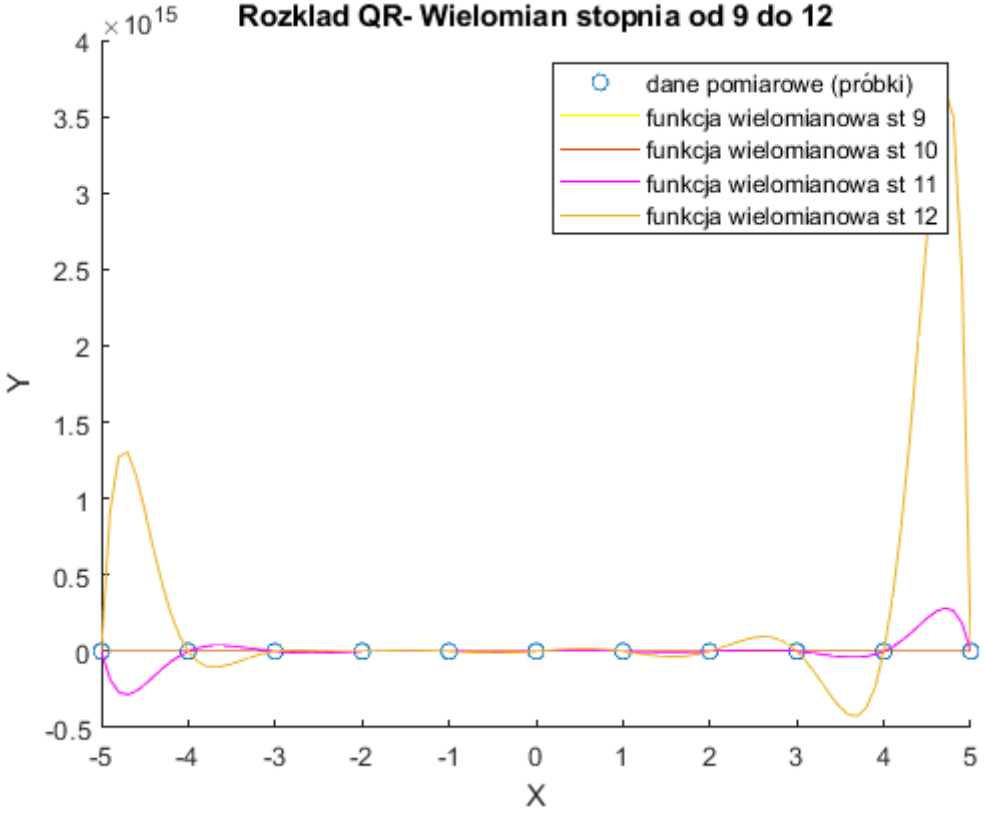

Uklad rownan normalnych- Wielomian stopnia od 1 do 4
dane pomiarowe (próbki)
funkcja wielomianowa st 1
funkcja wielomianowa st 2
funkcja wielomianowa st 3
funkcja wielomianowa st 4
Y
X
15
10
5
0
-5
-10
-15
-20
-25
-5
-4
-3
-2
-1
0
1
2
3
4
5


Rozklad QR- Wielomian stopnia od 1 do 4
dane pomiarowe (próbki)
funkcja wielomianowa st 1
funkcja wielomianowa st 2
funkcja wielomianowa st 3
funkcja wielomianowa st 4
Y
X
15
10
5
0
-5
-10
-15
-20
-25
-5
-4
-3
-2
-1
0
1
2
3
4
5


Uklad rownan normalnych- Wielomian stopnia od 5 do 8
dane pomiarowe (próbki)
funkcja wielomianowa st 5
funkcja wielomianowa st 6
funkcja wielomianowa st 7
funkcja wielomianowa st 8
Y
X
15
10
5
0
-5
-10
-15
-20
-25
-5
-4
-3
-2
-1
0
1
2
3
4
5


Rozklad QR- Wielomian stopnia od 5 do 8
dane pomiarowe (próbki)
funkcja wielomianowa st 5
funkcja wielomianowa st 6
funkcja wielomianowa st 7
funkcja wielomianowa st 8
Y
X
15
10
5
0
-5
-10
-15
-20
-25
-5
-4
-3
-2
-1
0
1
2
3
4
5


Uklad rownan normalnych- Wielomian stopnia od 9 do 12
dane pomiarowe (próbki)
funkcja wielomianowa st 9
funkcja wielomianowa st 10
funkcja wielomianowa st 11
funkcja wielomianowa st 12
Y
X


Rozklad QR- Wielomian stopnia od 9 do 12
×10^15
dane pomiarowe (próbki)
funkcja wielomianowa st 9
funkcja wielomianowa st 10
funkcja wielomianowa st 11
funkcja wielomianowa st 12
Y
X

Podsumowanie:

Dla wyższych stopni wielomianów obie metody zwracały mniejsze błędy, co jest rezultatem spodziewanym. Najlepszą aproksymacja okazał się wielomian 8 stopnia biorąc pod uwagę że dane obarczone są błędem. Wielomiany od stopnia pierwszego do czwartego nie są najlepszymi aproksymacjami, natomiast wykresy funkcji wielomianowych dla stopnia od 5 do 8 są bardzo podobne do siebie i błąd dla tych rozwiązań jest akceptowalny. Znaczna różnica w kształcie widoczna jest dla wielomianów stopnia >8. W tym przypadku kształt został dokładnie dopasowany do posiadanych danych (próbek), co nie jest najlepszym rozwiązaniem wiedzą, że dane obarczone są szumem. Ponadto metoda z rozkładem QR stworzyła duży pik dla wielomianu stopnia 9. W tym przypadku lepszą okazała się metoda z układem równań normalnych. Natomiast dla niższych stopni obie metody są tak samo skuteczne.

Dodatek zadanie 1

Generate_symetric_matrix

```
function [M] = Generate_symetric_matrix(size)
% M- wyjsciowa macierz symetryczna
    A = rand(size); % loswa macierz
    M = A + transpose(A); % utworzenie macierzy symetrycznej
    return;
end
```

Generate_symetric_matrix

```
function [M] = Generate_asymetric_matrix(size)
% M- wyjsciowa macierz asymetryczna

    M = rand(size); % loswa macierz
    while issymmetric(M)
        M = rand(size); % loswa macierz
    end
    return;
end
```

EigvalQRShift

```
function [ eigenvalues, iteracje,success] = EigvalQRShift( A, tol, imax )
    %Oblicznie wartosci wlasnych metoda rozkladu QR z przesunieciami
    % tol - tolerancja
    % imax - maksymalna liczba iteracji
    success = 1;
    n=size(A,1);
    eigenvalues = diag(zeros(n));
    INITIALsubmatrix = A; %macierz początkowa (oryginalna)
    iteracje = 0;
    for k = n:-1:2
        DK = INITIALsubmatrix; %macierz startowa dla jednej wart. własnej
        i = 0;
        while i<= imax && max(abs(DK(k,1:k-1)))>tol
            DD = DK(k-1:k,k-1:k); %macierz 2x2 prawego dolnego rogu
            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
                shift = r(1,1);
            else
                shift = r(2,1);
            end
            DK = DK - eye(k)*shift; %macierz przesunięta
            [Q1,R1] = Decompose_QR(DK); %faktoryzajca QR
            DK = R1 *Q1 +eye(k)*shift; %macierz przekształcona
            i = i+1;
            iteracje = iteracje + 1;
        end
        if i >imax
            success = 0;
            disp('imax exceeded program terminated')
        end
        eigenvalues(k) = DK(k,k);
        if k>2
            INITIALsubmatrix = DK(1:k-1,1:k-1); %deflacja macierzy
        else
            eigenvalues(1) = DK(1,1); %ostatnia wartość własna
        end
    end
end
```

EigvalQRNoShift

```
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
    %%alokacja pamięci dla macierzy diagonalnej
    eigenvalues = diag(zeros(n));
    %przechowującej wartości własne macierzy A
    Ak = A; %%alukuję pamięć w celu nie zmienienia macierzy A
    %%wykonuj aż do na-stępujących warunków - liczba it przekroczy itmax lub
    while (it <= maxit && max(max(Ak-diag(diag(Ak)))) > up0)
        [Q,R]=Decompose_QR(Ak); %%rozkład QR metodą Grama-Schmidta
        %%zgodnie z metodą A w następnej iteracji będzie obliczone w ten sposób
        Ak=R*Q;
        it = it + 1; %%zwiększamy wartość iteratora
    end
    if it>maxit
        %%jeśli nie udało się osiągnąć wyniku w mniej niż itmax ite-racji
        %%wtedy oznacza, że się nie udało obliczyć wartości
        %%własnej z daną dokładnością
        goodit = 0;
    end
    eigenvalues = diag(Ak);
end
```

Decompose_QR

```
function [Q,R] = Decompose_QR(A)
    %rozkład QR (wąski) zmodyfikowanym algorytmem Grama-Schmidta dla macierzy m x n (m>=n) o
rzędzie n, rzeczywistej lub zespolonej
    [m n] = size(A);
    Q=zeros(m,n);
    R=zeros(n,n);
    d=zeros(1,n);
    %rozkład z kolumnami Q ortogonalnymi
    for i=1:n
        Q(:,i) = A(:,i);
        R(i,i) = 1;
        d(i) = Q(:,i)'*Q(:,i);
        for j=i+1:n
            R(i,j)=(Q(:,i)'*A(:,j))/d(i);
            A(:,j)=A(:,j)-R(i,j)*Q(:,i);
        end
    end
    %normowanie rozkładu (kolumny Q ortonormalne)
    for i=1:n
        dd=norm(Q(:,i));
        Q(:,i)=Q(:,i)/dd;
        R(i,i:n)=R(i,i:n)*dd;
    end
end
```

compose_task_1

```
function [itterations, successes, errors, srednie] = compose_task_1()
    sizes = [5 10 20];
    maxit = 1000;
    up0 = 0.0001;%%up0 = 0.00001; up0 = 0.000001;
    % wektory do przechowywania statystyk
    itterations = zeros(9, 1);
    successes = zeros(9, 1);
    errors = zeros(9, 1);
    srednie = zeros(9, 3);

    for i=1:3
        %%petla dla macierzy symetrycznej i algorytmu bez przesuniecia.
        for j=1:30
            Sym = Generate_symetric_matrix(sizes(i));
            [eigValues, iteration, succes] = EigvalQRNoShift(Sym, up0, maxit);
            if succes
               %zliczam liczbe udanych prob dla pierwszej kombiancji
               successes(i, 1) = successes(i, 1) + 1;
               itterations(i, 1) = itterations(i, 1) + iteration;
               %norma rďż˝nicy
               errors(i,1) = errors(i,1) + norm(sort(eigValues) - sort(eig(Sym)));
            end
        end
        %%petla dla macierzy symetrycznej i algorymtu z przeunieciem.
        for j=1:30
            Sym = Generate_symetric_matrix(sizes(i));
            [eigValues, iteration, succes] = EigvalQRShift(Sym, up0, maxit);
            if succes
               %zliczam liczbe udanych prob dla drugiej kombiancji
               successes(i+3, 1) = successes(i+3, 1) + 1;
               itterations(i+3, 1) = itterations(i+3, 1) + iteration;
               %norma rďż˝nicy
               errors(i+3,1) = errors(i+3,1) + norm(sort(eigValues) - sort(eig(Sym)));
            end
        end
%         %%petla dla macierzy asymetrycznej i algorymtu z przeunieciem.
        for j=1:30
            Asym = Generate_asymetric_matrix(sizes(i));
            [eigValues, iteration, succes] = EigvalQRShift(Asym, up0, maxit);
            if succes
               %zliczam liczbe udanych prob dla drugiej kombiancji
               successes(i+6, 1) = successes(i+6, 1) + 1;
               itterations(i+6, 1) = itterations(i+6, 1) + iteration;
               %norma rďż˝nicy
               errors(i+6,1) = errors(i+6,1) + norm(sort(eigValues) - sort(eig(Asym)));
            end
        end
    end
    srednie(:,1)= itterations./successes;
    srednie(:,2) = successes./30;
    srednie(:,3) = errors./successes;
end
```

Dodatek zadanie 2

aproksymacja

```
function [a1, norma1, a2, norma2]=aproksymacja(X,Y,stopien)
    rozmiar = size(X, 1); %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;
    approximation1 = polyval(a1,X);
    %układ równań liniowych z macierzą R wynikającą z rozkładu QR
    [Q, R] = Decompose_QR(A);
    a2 = R\(Q'*Y);
    approximation2 = polyval(a2,X);
    % Wyliczanie normy euklidesowej
    norma1 = norm(approximation1-Y);
    norma2 = norm(approximation2-Y);
end
```

compose_task_2

```
function [norma] = compose_task_2()
    Xdane = [ -5; -4; -3; -2; -1; 0; 1; 2; 3; 4; 5];
    Ydane = [-23.4523; 11.9631; 4.4428; 1.1010; -1.6826; -1.2630; -0.0357; -1.3156; -3.4584; -
8.4294; -18.4654];
    colors = ['y-','m--', 'c:', 'r-.'];
    norma = zeros(4,2);
    przedzial_x = -5:0.1:5;
    figure()
    title('Uklad rownan normalnych- Wielomian stopnia od 9 do 12');
    xlabel('X')
    ylabel('Y')
    hold on
    scatter(Xdane, Ydane)
    for stopien=9:12
        [a1, norma1, a2, norma2] = aproksymacja(Xdane,Ydane, stopien);
        norma(stopien-8, 1) = norma1;
        wsp1 = polyval(a1,przedzial_x);
        %wsp2 = polyval(a2,przedzial_x);
        plot(przedzial_x, wsp1,colors(stopien-8));
        %polyfit(Xdane,Ydane,stopien);

    end
    legend('dane pomiarowe (próbki)','funkcja wielomianowa st 9','funkcja wielomianowa st
10','funkcja wielomianowa st 11','funkcja wielomianowa st 12')
    hold off


    figure()
    title('Rozklad QR- Wielomian stopnia od 9 do 12');
    xlabel('X')
    ylabel('Y')
    hold on
    scatter(Xdane, Ydane)
    for stopien=9:12
        [a1, norma1, a2, norma2] = aproksymacja(Xdane,Ydane, stopien);
        %wsp1 = polyval(a1,przedzial_x);
        norma(stopien-8, 2) = norma2;
        wsp2 = polyval(a2,przedzial_x);
        plot(przedzial_x, wsp2,colors(stopien-8));
        %polyfit(Xdane,Ydane,stopien);

    end
    legend('dane pomiarowe (próbki)','funkcja wielomianowa st 9','funkcja wielomianowa st
10','funkcja wielomianowa st 11','funkcja wielomianowa st 12')
    hold off
end
```