MNUM-PROJEKT, zadanie 1.25

Wojciech Grunwald (311566)

11.04.2023

1 Wstęp

Celem projektu jest napisanie solwera w Matlabie rozwiązującego układ n równań liniowych Ax=b, gdzie x, b $\in \mathbb{R}^n$ za pomocą metody eliminacji Gaussa-Jordana z częściowym wyborem elementu głównego oraz zaimplementowanie aproksymacji wielomianowej metodą najmniejszych kwadratów przy użyciu układu równań normalnych i napisanego wcześniej solwera. Uzyte w zadaniu są również: gotowy solwer Gaussa-Seidla i procedura dekompozycji macierzy QR.

Algorytmy zaimplementowałem w *Matlabie* w wersji R2022b. Obliczenia były wykonywane na procesorze Intel Core i5-9300H CPU 2.40Ghz.

W folderze *Kod źródłowy* załączonym do sprawozdania znajdują się odpowiednie skrypty rozwiązujące zadania:

1. ZAD1.m - skrypt korzystający z własnego solwera Gaussa-Jordana
2. ZAD2.m - skrypt, w którym porównywane są: własny solwer i algorytm Gaussa-Seidla
3. ZAD3.m - skrypt, w którym porównywane są trzy metody aproksymacji wielomianowej:
   (a) - z wykorzystaniem układu równań normalnych i własnego solwera
   (b) z wykorzystaniem układu równań normalnych i solwera Gaussa-Seidla
   (c) z wykorzystaniem dekompozycji QR
4. create_matrixes.m - kreator macierzy do zadania 1
5. create_matrixes_2.m - kreator macierzy do zadania 2
6. solver.m - własny solwer Gaussa-Jordana z częściowym wyborem elementu głównego
7. GS.m - gotowy solwer Gaussa-Seidla
8. approx.m - w.w. pierwsza metoda aproksymacji wielomianowej
9. approxGS.m - w.w. druga metoda aproksymacji wielomianowej
10. approxQR.m - w.w. trzecia metoda aproksymacji wielomianowej

2 Metoda eliminacji Gaussa-Jordana z częściowym wyborem elementu głównego

2.1 Wprowadzenie teoretyczne

Rozważamy układ równań liniowych postaci $Ax = B$ (n - rozmiar macierzy, która jest kwadratowa i nieosobliwa), gdzie:

$$\text{A}=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & ... & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & ... & a_{2n} \\ ... & ... & ... & ... & ... & ... \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & ... & a_{nn} \end{bmatrix} \text{x}=\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \text{B}=\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

Metoda eliminacji Gaussa-Jordana jest modyfikacją metody eliminacji Gaussa. Układ n równań liniowych jest przekształcany za pomocą operacji wierszowych tak, aby zarówno poniżej, jak i powyżej głównej przekątnej znalazły się współczynniki zerowe.

Wykonujemy następujące kroki:

1. Pierwsze równanie dzielimy obustronnie przez a_{11}, a następnie od i-tego wiersza (i=1,2,3,...,n) odejmujemy wiersz pierwszy pomnożony przez $a_{i1}^{(1)}$, otrzymując układ równań:

$$A^{(2)}x = B^{(2)} \tag{1}$$

$$\begin{cases} x_1 + a_{12}^{(2)}x_2 + ... + a_{1n}^{(2)}x_n = b_1^{(2)} \\ \quad a_{22}^{(2)}x_2 + ... + a_{2n}^{(2)}x_n = b_2^{(2)} \\ ..... \\ \quad a_{n2}^{(2)}x_2 + ... + a_{nn}^{(2)}x_n = b_n^{(2)} \end{cases}$$

2. Następnie drugie równanie dzielimy obustronnie przez $a_{22}^{(2}$ i od i-tego wiersza (i=1,3,4,...n) odejmujemy drugi wiersz pomnożony przez a_{i2}^{2}, otrzymując układ:

$$A^{(3)}x = B^{(3)} \tag{2}$$

$$\begin{cases} x_1 & & a_{13}^{(3)}x_3 + ... + a_{1n}^{(3)}x_n & = b_1^{(3)} \\ & x_2 & a_{23}^{(3)}x_3 + ... + a_{2n}^{(3)}x_n & = b_1^{(3)} \\ ... & .... & ..... & ..... \\ & & a_{n3}^{(3)}x_3 + ... + a_{nn}^{(3)}x_n & = b_n^{(3)} \end{cases}$$

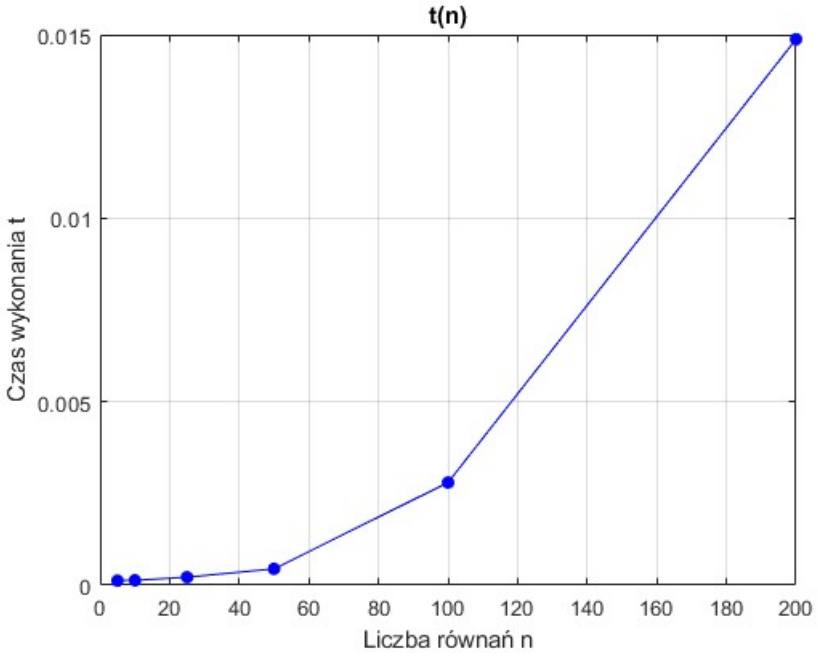
3. Po n-1 eliminacjach otrzymuje się układ o macierzy diagonalnej: $\begin{cases} x_1 & & & & = b_1^{(n)} \\ & x_2 & & & = b_2^{(n)} \\ ... & ... & ... & ... & ... \\ & & & x_n & = b_n^{(n)} \end{cases}$

   Stąd mamy już gotowe rozwiązanie.

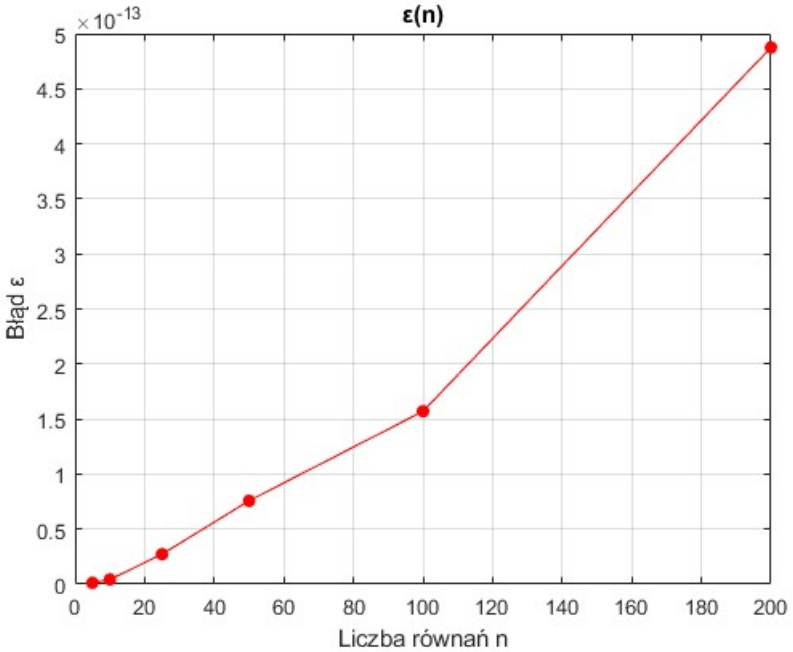
Metoda wymaga wykonania około półtora raza więcej działań niż w przypadku metody eliminacji Gaussa. Liczba zajętych komórek pamięci jest taka sama. Aby nie wystąpiło dzielenie przez zerowy element a_{ii}^{i-1}, stosuje się tzw. wybór elementu głównego (w tym wypadku stosujemy wybór częściowy), który polega na tym, że w danym kroku wybieramy element główny, czyli poszukujemy elementu o największym module w odpowiadającej krokowi kolumnie:

$$\left|a_{ik}^{(k)}\right| = \max_j \{\left|a_{ik}^{(k)}\right|, j = k, k+1, ..., n\} \tag{3}$$

2.2 Testowanie własnego solwera dla kilku układów równań



Rysunek 1: Zależność czasu obliczeń t od liczby równań n

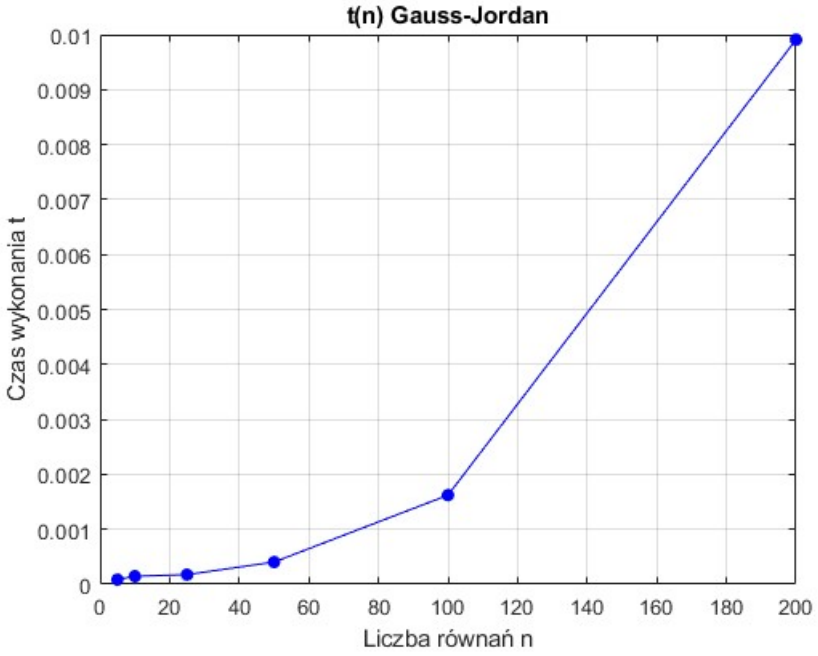


Rysunek 2: Zależność błędu ϵ od liczby równań n

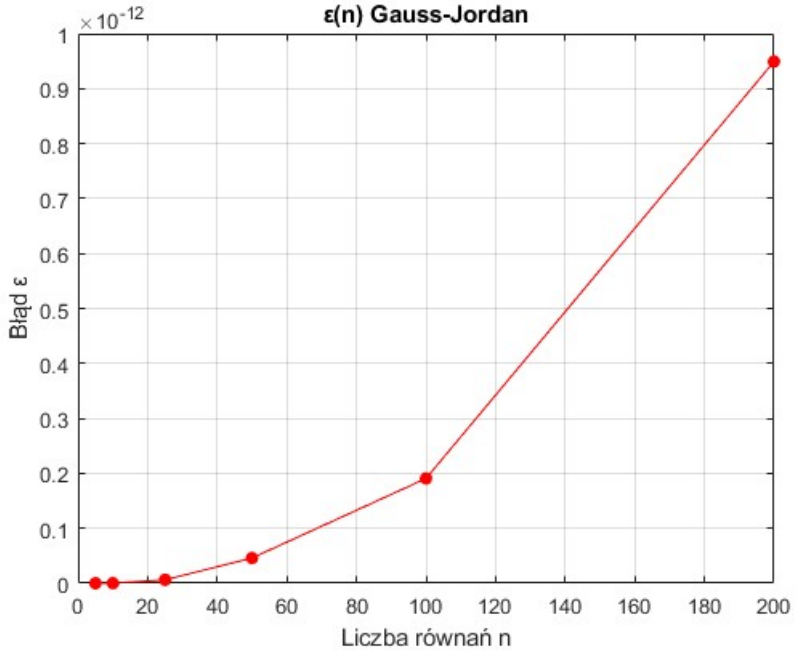
Jak widać na wykresach otrzymaliśmy zależności, których się spodziewano - zarówno czas obliczeń, jak i błąd rosną szybko. Zależność między czasem obliczeń a liczbą równań w metodzie eliminacji Gaussa-Jordana z częściowym wyborem elementu głównego jest funkcją trzeciego stopnia n^3.

Zależność między błędem rozwiązania a liczbą równań w układzie liniowym może być opisana za pomocą współczynnika uwarunkowania macierzy. W przypadku metody eliminacji Gaussa-Jordana z częściowym wyborem elementu głównego współczynnik uwarunkowania macierzy A może wzrastać wraz z liczbą równań, co widać na wykresie.

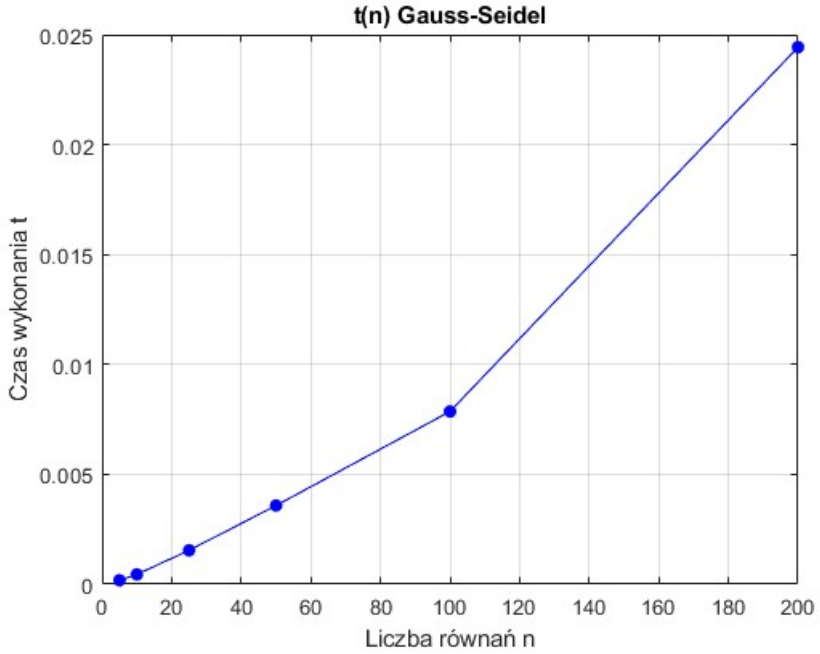
3 Porównanie stworzonego algorytmu z gotowym algorytmem Gaussa-Seidela na innych macierzach



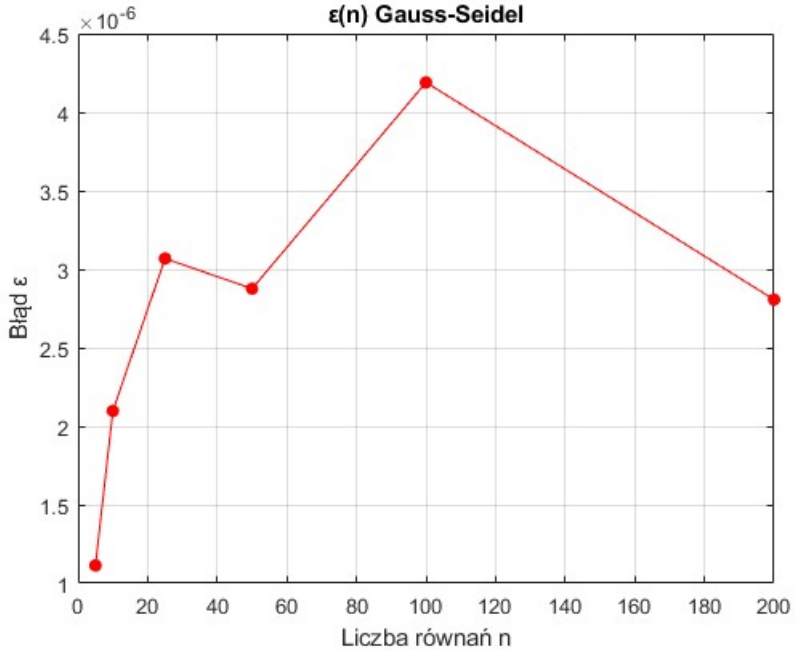
Rysunek 3: Zależność czasu obliczeń t od liczby równań n dla algorytmu Gaussa-Jordana



Rysunek 4: Zależność błędu ϵ od liczby równań n dla algorytmu Gaussa-Seidla



Rysunek 5: Zależność czasu obliczeń t od liczby równań n dla algorytmu Gaussa-Seidla



Rysunek 6: Zależność błędu ϵ od liczby równań n dla algorytmu Gaussa-Seidla

Z wykresów można zauważyć, że algorytm Gaussa-Jordana jest algorytmem szybszym niż algorytm iteracyjny Gaussa-Seidla. Algorytm Gaussa-Seidla ma złożoność mniejszą ($O(n^2)$) i może być szybszy od metody Gaussa-Jordana w przypadku rzadkich macierzy, ale jeśli bierzemy pod uwagę dokładność i szerokie zastosowanie, to metoda Gaussa-Jordana jest lepsza. Ponadto metoda Seidla nie jest dokładna i może zbiegać się wolno lub wcale dla niektórych macierzy.

N	Gauss-Jordan		Gauss-Seidel	
	ϵ $[10^{-12}]$	t[s]	ϵ $[10^{-6}]$	t[s]
5	0	0.0001	1.114	0.0002
10	0.0003	0.0001	2.098	0.0004
25	0.0063	0.0002	3.068	0.0015
50	0.0460	0.0004	2.876	0.0036
100	0.1907	0.0016	4.189	0.0079
200	0.9493	0.0099	2.807	0.0244

Tabela 1: Błędy i czasy obliczeń algorytmów Gaussa-Jordana i Seidla

4 Metoda najmniejszych kwadratów do funkcji wielomianowej

4.1 Wprowadzenie teoretyczne

4.1.1 Aproksymacja średniokwadratowa dyskretna (wielomianowa) i układ równań normalnych

Niech f(x) przyjmuje na pewnym zbiorze punktów $x_0, x_1, ..., x_N$ znane wartości $y_j = f(x_j), j = 0, 1, 2, ..., N$ (N - liczba punktów). Wtedy funkcję f(x) aproksymującą będzie reprezentować układ funkcji bazowych przestrzeni funkcji aproksymujących, tzn.:

$$F(x) = \sum_{i=0}^{n} a_i \phi_i(x) \tag{4}$$

Zadaniem aproksymacji będzie wtedy wyznaczenie wartości współczynników $a_0, a_1, ..., a_n$ określających funkcję aproksymującą tak, aby zminimalizować błąd średniokwadratowy zdefinioway zależnością:

$$H(a_0, ..., a_n) = \sum_{N}^{j=0} \left[f(x_j) - \sum_{i=0}^{n} a_i \phi(x_j) \right]^2 \tag{5}$$

Jest to UKŁAD RÓWNAŃ LINIOWYCH WZGLĘDEM WSPÓŁCZYNNIKÓW $a_0, ..., a_n$, który nazywany jest układem równań normalnych, a macierz tego układu to tzw. macierz Grama.
Zaznaczmy teraz, że n będzie naszym stopniem wielomianu. Dalej zadanie aproksymacji możemy zapisać także w postaci:

$$H(a) = (\|y - Aa\|_2)^2 \tag{6}$$

gdzie A=$\begin{bmatrix} \phi_0(x_0) & \phi_1(x_0) & ... & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & ... & \phi_n(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & ... & \phi_n(x_2) \\ ... & ... & ... & ... \\ \phi_0(x_N) & \phi_1(x_N) & ... & \phi_n(x_N) \end{bmatrix}$, $a = a = \begin{bmatrix} a_0 & a_1 & ... & a_n \end{bmatrix}^T$, y=$\begin{bmatrix} f(x_0) & f(x_1) & ... & f(x_j) \end{bmatrix}^T$ (j=0, 1, ..., N)

UWAGA: W środowisku Matlab indeksy macierzy są numerowane od 1, a nie od 0, więc trzeba wszystko siłą rzeczy przesunąć o 1 indeks.

Wykorzystując definicję macierzy A układ równań normalnych możemy zapisać w postaci:

$$A^T Aa = A^T y \tag{7}$$

Dalej przyjmujemy jako bazę definiującą naszą funkcję aproksymującą - bazę wielomianów, tzn. $\phi_0(x) = 1, \phi_1(x) = x, ..., \phi_n(x) = x^n$, co implikuje:

$$F(x) = a_0 + a_1 x + ... + a_n x^n \tag{8}$$

Wprowadzamy oznaczenia pomocnicze:

$$g_{ik} = \sum_{j=0}^{N} (x_j)^i x_j^k = \sum_{j=0}^{N} (x_j)^{(i+k)} \tag{9}$$

$$\rho_{ik} = \sum_{j=0}^{N} f(x_j)(x_j)^k \tag{10}$$

Uzyskujemy układ równań normalnych w postaci:

$$G \cdot a = \rho \tag{11}$$

z którego wyznaczamy wektor współczynników w celu uzyskania wielomianu aproksymującego.

$$\begin{bmatrix} a_0 g_{00} + a_1 g_{10} + ... + a_n g_{n0} & = \rho_0 \\ a_0 g_{01} + a_1 g_{11} + ... + a_n g_{n1} & = \rho_1 \\ ..... & ... \\ a_0 g_{0n} + a_1 g_{1n} + ... + a_n g_{nn} & = \rho_n \end{bmatrix} = G \cdot a* = \rho \tag{12}$$

4.1.2 Faktoryzacja QR

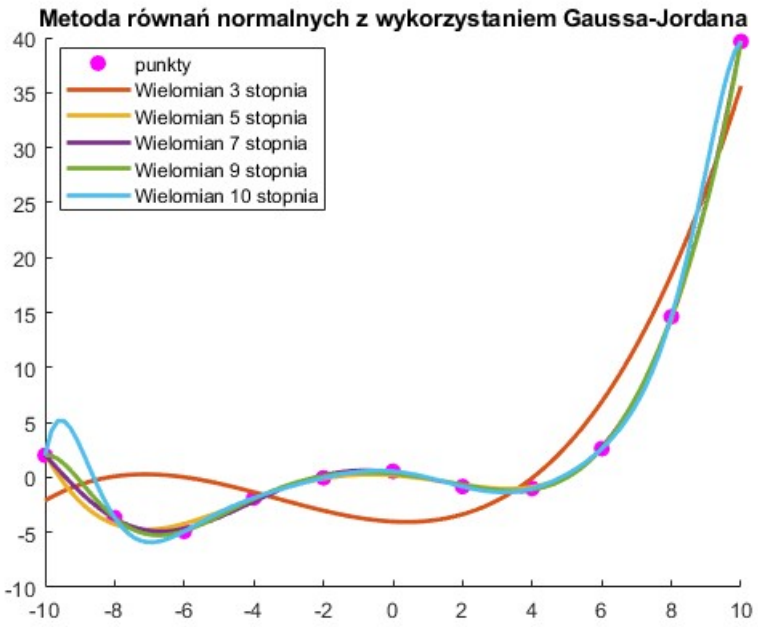
Faktoryzacja QR oznacza rozkład macierzy do postaci iloczynu A=QR, gdzie Q jest macierzą ortogonalną($Q^T Q = I$, a R macierzą trójkątną górną. Faktoryzacja umożliwia rozwiązanie układu równań normalnych w inny sposób:

$$A^T A a = A^T y \tag{13}$$

Otrzymujemy:

$$a = R^{-1} Q^T y \tag{14}$$

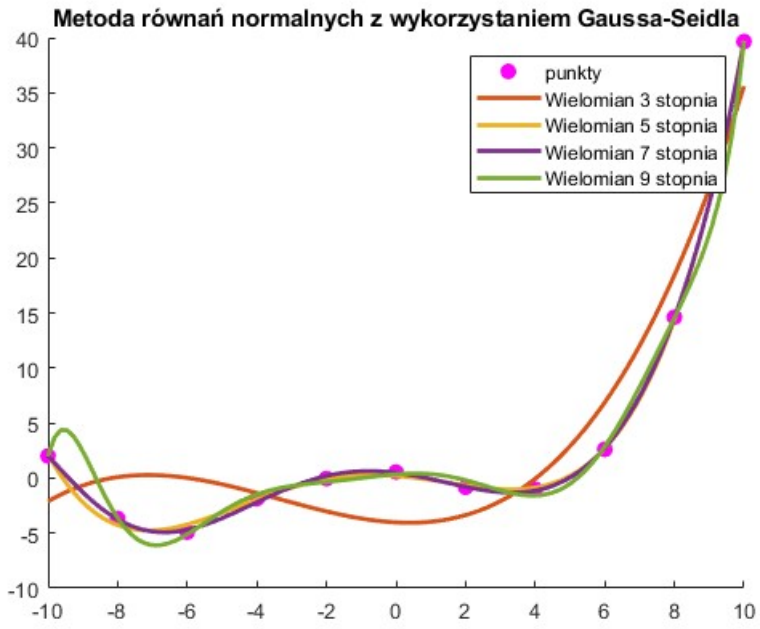
4.2 Układ równań normalnych i własny solwer



Rysunek 7: Aproksymacja pierwszą metodą

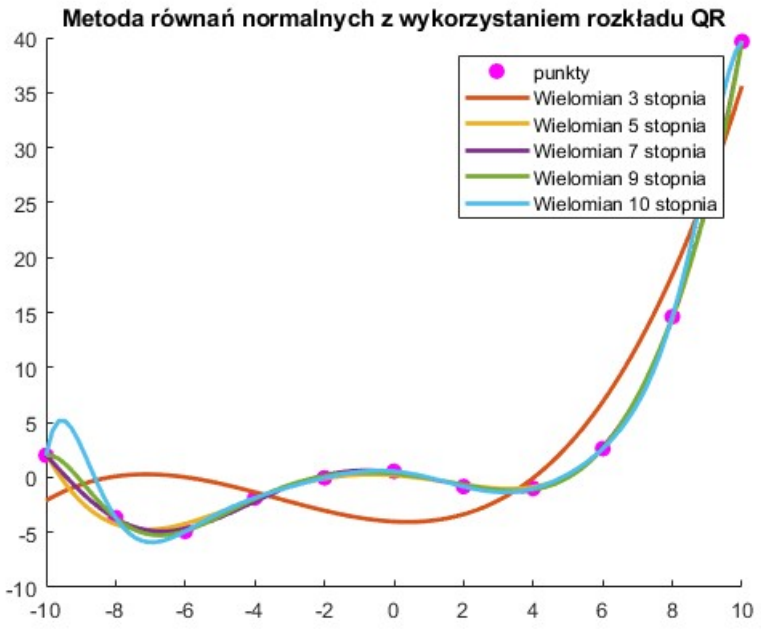
4.3 Układ równań normalnych i solwer GS

Przy obliczeniach tą metodą pojawiły się pewne problemy: dla 10 stopnia wielomianu algorytm generuje nieadekwatny wielomian do danych. Prawdpodobnie dzieje się tak przez liczbę iteracji algorytmu Gaussa-Seidla, ponieważ algorytm ten ze względu na swoją naturę potrzebuje już w tym przypadku wielu iteracji, aby uzyskać zadowalającą aproksymację. Dla iteracji rzędu 10^6 algorytm wciąż nie działa tak, jakby tego oczekiwano, a dla 10^7 wykonuje się bardzo długi czas.



Rysunek 8: Aproksymacja drugą metodą

4.4 Rozkład QR



Rysunek 9: Aproksymacja trzecią metodą

P	**Gauss-Jordan**			**Gauss-Seidel**		
	ϵ_2	ϵ_∞	t $[s \cdot 10^{-3}]$	ϵ_2	ϵ_∞	t [s]
3	11.9237	4.9326	0.1300	11.9237	4.9326	0.0008
5	1.0707	0.6951	0.1926	1.0707	0.6951	0.0219
7	0.5140	0.3276	0.1172	0.5142	0.3261	1.0227
9	0.3489	0.2045	0.1061	1.0718	0.6461	113.2482
10	0.0000	0.0000	0.1964	———	———	———

Tabela 2: Pierwsza część zbiorczej tabeli błędów i czasu obliczeń dla różnych metod aproksymacji

P	**Rozkład QR**		
	ϵ_2 $[10^{-12}]$	ϵ_∞	t $[s \cdot 10^{-3}]$
3	11.9237	4.9326	0.2568
5	1.0707	0.6951	0.1959
7	0.5140	0.3276	0.1288
9	0.3489	0.2045	0.0952
10	0.0000	0.0000	0.0814

Tabela 3: Druga część zbiorczej tabeli błędów i czasu obliczeń dla różnych metod aproksymacji

Z obliczeń można wywnioskować, że błędy tych trzech algorytmów były podobnego rzędu. Ponadto przy metodach dokładniejszych (Gaussa-Jordana i rozkładu QR) przy stopniu wielomianu równemu 10 błąd zbliżył się niemalże do 0, jednka wielomian ten jest niewygodny do wykorzystywania w obliczeniach ze względu na swój duży stopień i wynikajaće z tego problemy obliczeniowe. W praktyce przy aproksymacji wielomianem jego stopień powinien być znacznie mniejszy od liczby punktów, na podstawie których dokonujemy aproksymacji - uzyskujemy wtedy prostsze funkcje.
Dodatkowo widoczna jest zauważalna przewaga algorytmów bezpośrednich ponad iteracyjnymi (w tym przypadku), ponieważ czas wykonania algorytmu Gaussa-Seidla był znacznie większy od pozostałych.