Metody numeryczne – Projekt 1

Autor: Aleksandra Jamróz

Zadanie 1.20

1.

Treść:
Napisać uniwersalną procedurę w Matlabie o odpowiednich parametrach wejścia i wyjścia (solwer), rozwiązującą układ n równań liniowych Ax = b, gdzie x, b ∈ Rn, wykorzystując podaną metodę. Nie sprawdzać w procedurze, czy dana macierz A spełnia wymagania stosowalności metody. Zakazane jest użycie jakichkolwiek solwerów w środku. Obliczyć błąd rozwiązania ε = ‖A˜x −b‖2 (skorzystać z funkcji norm Matlaba).
Metoda: faktoryzacji LDLT
Następnie proszę zastosować swoją procedurę w programie do rozwiązania układu równań dla macierzy A i wektorów b danych wzorami:

$$a_{ij}=2(i+j)+1,\ j\neq i;\ a_{ij}=4n^2+(3i+2)n;$$
$$b_i=2.5+0.6i;$$

przyjmując n = 5, 25, 50, 100, 250, 500, 1000.
Proszę wykonać wykresy zależności czasu obliczeń i błędu ε od liczby równań n. Skomentować wyniki.

Zadanie rozbiłam na podproblemy, implemetując funkcję główną oraz 3 pomniejsze. W ten sposób powstały:

1. *my_ldl* – funkcja do faktoryzacji macierzy A na macierz L oraz D. Nazwa funkcji została skonstruowana, aby nie myliła się z funkcją matlabową. W ogólności, rozkład LDLT istnieje dla dowolnych macierzy symetrycznych.

Obliczenie składowych macierzy D oraz L nastąpiło zgodnie z algorytmem:

$$d_{ii}=a_{ii}-\sum_{k=1}^{i-1} l_{ik}^2 d_{kk}$$

$$l_{ji}=(a_{ji}-\sum_{k=1}^{i-1} l_{jk} d_{kk} l_{ik})/d_{ii}, i=1,\dots,n, j=i+1,\dots,n$$

Otrzymałam w ten sposób trójkątną macierz L, zawierającą na diagonali jedynki i zera ponad nią, oraz macierz D zawierającą zera wszędzie poza diagonalą.

```
function [L, D] = my_ldl(A)

    n = size(A, 1); % pobranie rozmiaru macierzy A
    L = eye(n);     % inicjalizacja macierzy dolnej trójkątnej L jako macierzy jednostkowej
    D = zeros(n);   % inicjalizacja macierzy diagonalnej D jako macierzy zerowej

    for i = 1:n
        % obliczanie składowych macierzy D
        d = A(i,i);
        for step = 1:i-1
            d = d - L(i,step)^2 * D(step, step);
        end
        D(i,i) = d;
        % obliczanie składowych macierzy L
        for j = i+1:n
            l = A(j, i);
            for step = 1:i-1
                l = l - L(j,step) * D(step, step) * L(i, step);
            end
            L(j, i) = l / D(i, i);
        end
    end
end

```

Obraz 1: Kod funkcji my_ldl

2. *Lyb* – funkcja rozwiązująca układ równań z macierzą trójkątną dolną

Rozwiązanie układu równań liniowych za pomocą metody faktoryzacji LDLT jest możliwe dzieki wykorzystaniu zależności I podstawień. Mając macierze L I D możemy zapisać następujące równości:

$$A = LDLT$$
$$Ax = b \rightarrow LDLT\ x = b$$
$$DLT\ x = y$$
$$L\ y\ = b$$

W ten sposób policzymy wektor y, którego potem użyjemy do wyliczenia wektora x. Zastosujemy w tym celu poniższy algorytm. Jest on wzorem na rozwiązanie układu równań z macierzą trójkątną dolną (w naszym przypadku macierzą L)

$$x_1 = \frac{b_1}{a_{11}}$$

$$x_k = \frac{\left(b_k - \sum_{j=1}^{k-1} a_{kj} x_j\right)}{a_{kk}}, k = 2,3,\ldots,n$$

3. *xDLTy* – funkcja rozwiązująca układ równań z macierzą trójkątną górną

Analogicznie do równania dla macierzy trójkątnej dolnej, zapisuję wzór I wykonuję obliczenia dla macierzy trójkątnej górnej.

$$x_n = \frac{b_n}{a_{nn}}$$

$$x_k = \frac{\left(b_k - \sum_{j=k+1}^{n} a_{kj} x_j\right)}{a_{kk}}, k = n-1, n-2, \ldots, 1$$

```
function y = Lyb(L, b)
    n = size(L, 1);
    y = zeros(n,1);
    for k = 1:n
        elem = b(k,1);
        for j = 1:k-1
            elem = elem - L(k,j)*y(j,1);
        end
        y(k,1) = elem / L(k,k);
    end
end
```

Obraz 2: Kod funkcji Lyb

```
function x = xDLTy(DLT, y)
    n = size(DLT, 1);
    x = zeros(n,1);
    for k = n:-1:1
        elem = y(k,1);
        for j = k+1:n
            elem = elem - DLT(k,j)*x(j,1);
        end
        x(k,1) = elem / DLT(k,k);
    end
end
```

Obraz 3: Kod funkcji xDLTy

4. LDLT – funkcja przyjmująca macierz A oraz wektor b, zwracająca rozwiązanie układu równań, wykorzystująca 3 powyższe funkcje.

```
function [x, epsilon] = LDLT(A, b)
    [L, D] = my_ldl(A);
    % Rozwiązanie Ly = b
    % L - macierz dolna trójkątna
    y = Lyb(L,b);
    % Rozwiązanie DLTx = y
    % DLT - macierz górna trójkątna
    x = xDLTy(D*L', y);
    % Obliczenie błędu rozwiązania epsilon
    epsilon = norm(A*x-b, 2);
end
```

Obraz 4: Kod funkcji LDLT

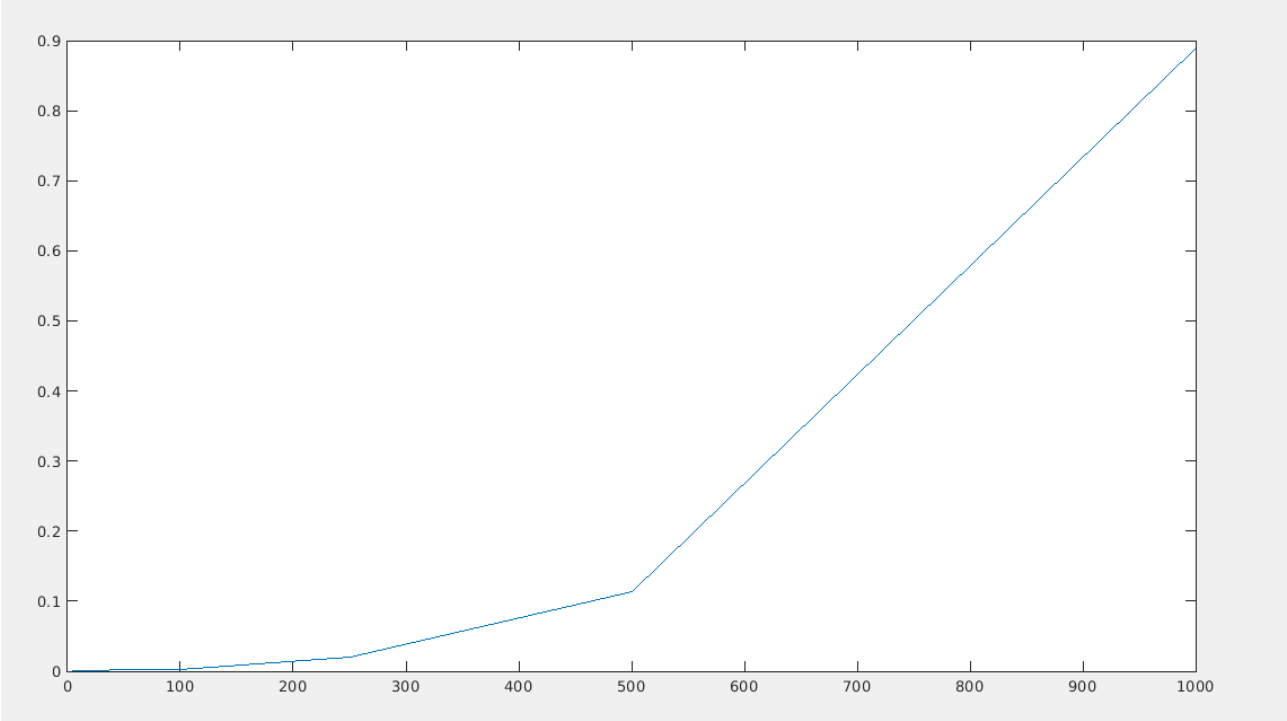
Eksperymenty

Po napisaniu funkcji przeszłam do wykonywania eksperymentów. Macierze A o wymiarach nxn oraz wektory b generowałam na bieżąco I od razu obliczałam dla nich rozwiązanie.

Obserwacje:

1. Czas wykonania

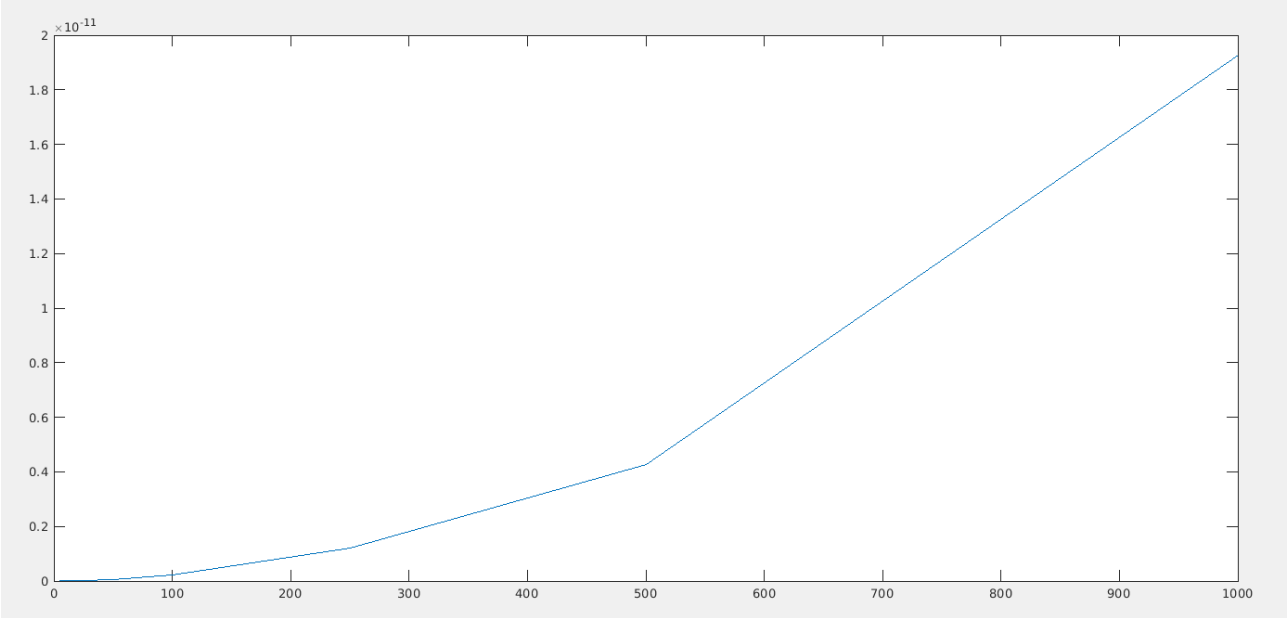
Można zauważyć, że czas wykonania nie rośnie liniowo. Im mniejszy rozmiar macierzy A, tym wolniej wzrasta czas wykonania.



Obraz 5. Wykres czasu wykonania do rozmaru macierzy A

2. Błąd rozwiązania

To samo dzieje się w przypadku błędu. Im większy rozmiar macierzy A, tym większy błąd, jednak wzrost nie jest liniowy.



Obraz 6: Wykres błędu rozwiązania do rozmiaru macierzy A

2. Porównanie z metodą Gaussa-Seidela

Treść:
Wykonać eksperymenty takie jak w p. 1 dla macierzy A i wektorów b danych wzorami:

$$a_{ij} = 12\,;\, a_{ij} = 3.8\,,\, j = i \pm 3\,;\, a_{ij} = 0 \; dla\; pozostałych$$

$$b_i = 4.5 - 0.5i$$

używając swojego solwera z p.1 oraz solwera GS.m ze strony przedmiotu, będącego implementacją metody Gaussa-Seidela. Przyjąć itmax=1000 · n, delta=10−9 ≙ 1e − 9. Przedstawić wyniki (dokładności i czasy, liczbę iteracji w metodzie Gaussa-Seidela) w tabelach i wykresach, porównać je i skomentować.

Tabela 1: Czas wykonania do rozmiaru macierzy A w przypadku 2 metod obliczeniowych

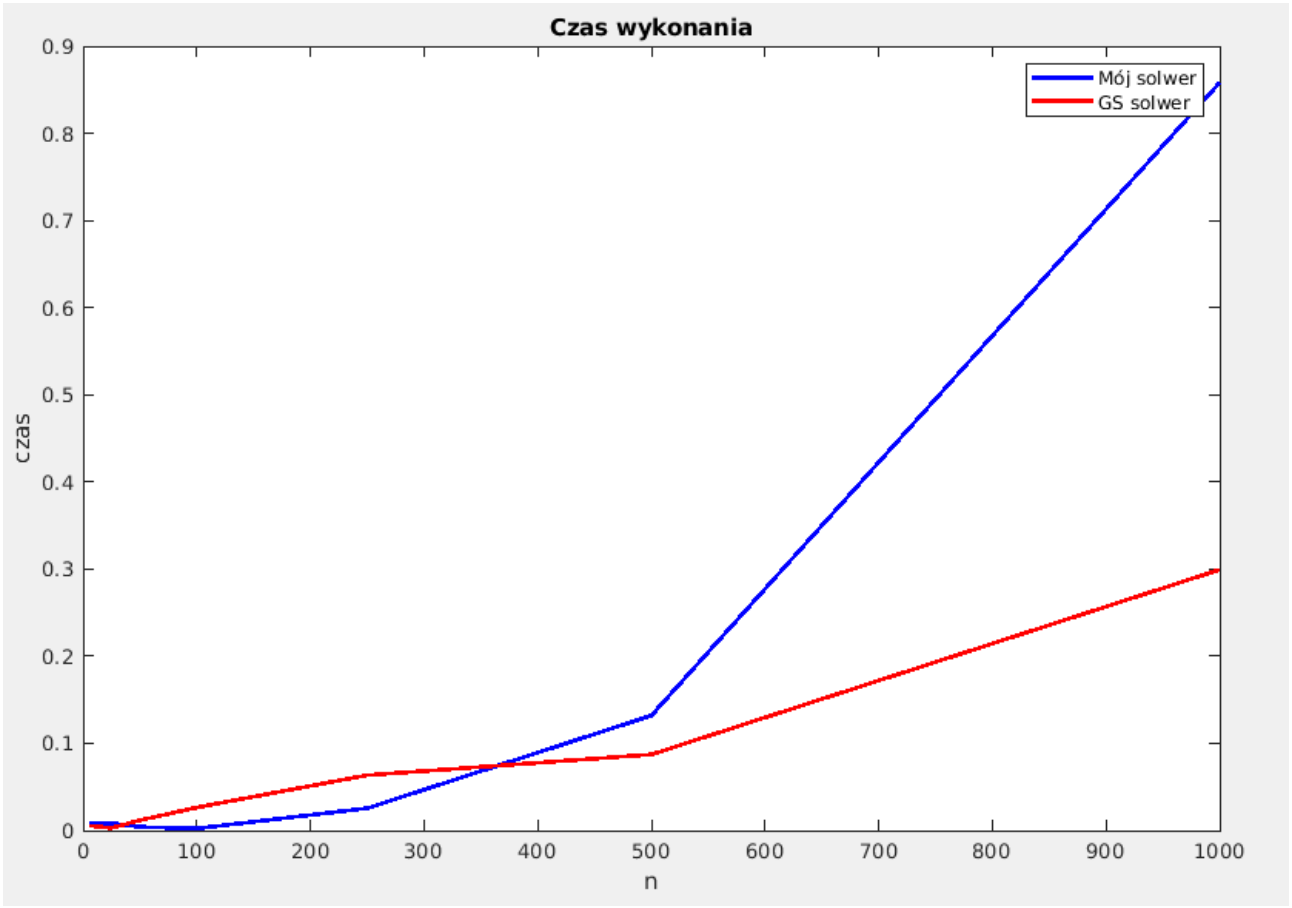
N	5	25	50	100	250	500	1000
Mój Solver	2.71e-04	2.16e-04	0.006232	0.001844	0.022774	0.14326	1.138043
GS	0.00258	0.002954	0.01	0.029286	0.062059	0.059974	0.327127

Tabela 2: Błąd rozwiązania do rozmiaru macierzy A w przypadku 2 metod obliczeniowych

N	5	25	50	100	250	500	1000
Mój Solver	1.2560e-15	7.5364e-15	1.4594e-14	3.9770e-14	1.4028e-13	3.8955e-13	1.195e-12
GS	22	66	78	84	87	90	92

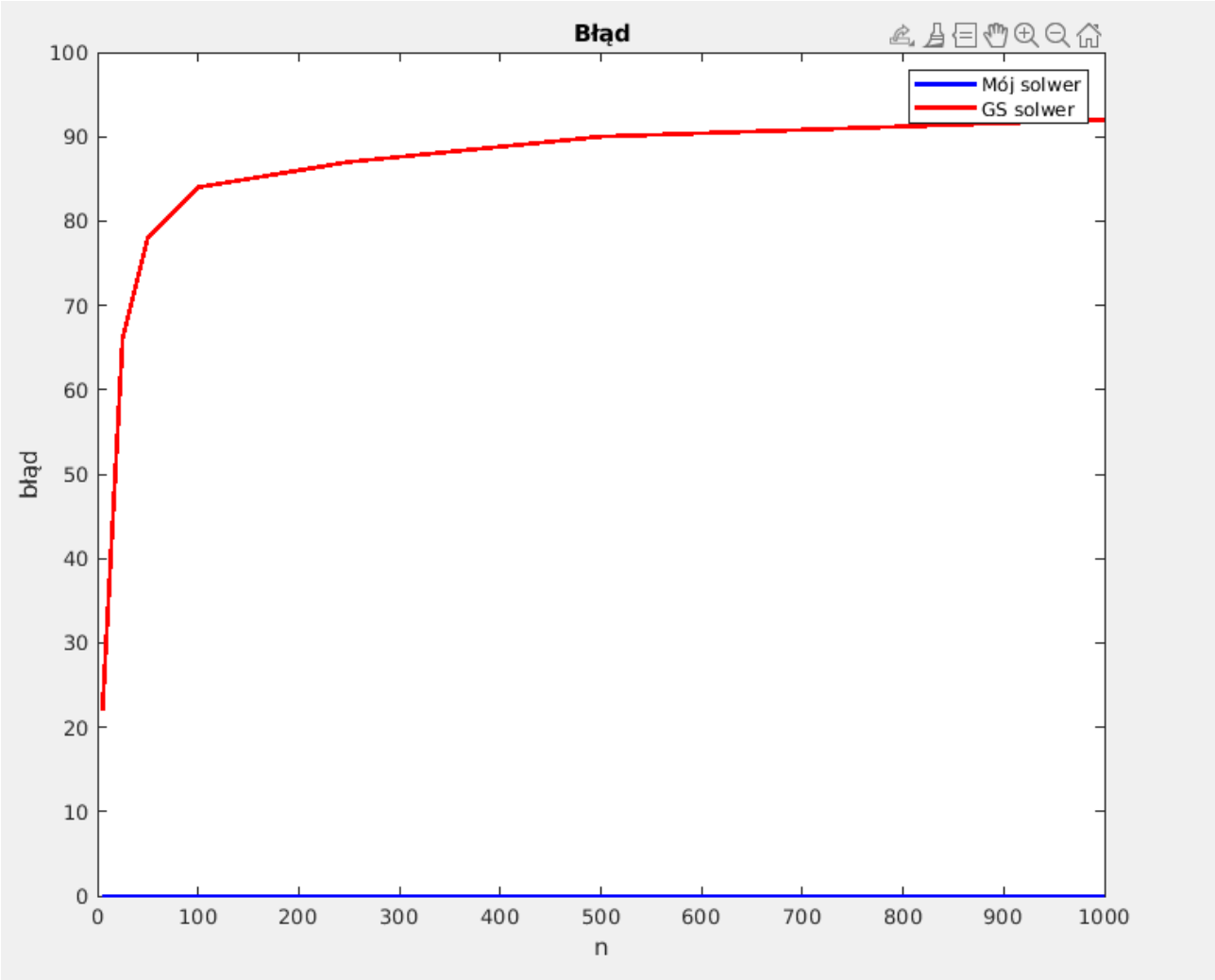
1. Czas wykonania

Rozwiązanie liczone za pomocą metody GS zajmowało więcej czasu niż za pomocą faktoryzacji LDLT w przypadku niewielkich macierzy. Jednak po przekroczeniu progu ok. 370 x 370, zajmowała ona mniej czasu. Możemy zaobserwować, że wzrost czasu do rozmiaru macierzy jest wolniejszy. Wybraną metodę możemy w ten sposób dopasować do naszych potrzeb.



Obraz 7: Czas wykonania

2. Błąd rozwiązania



Obraz 8: Błąd rozwiązania

Niestety metoda GS jest obarczona znacznie większym błędem niż metoda faktoryzacji. Podczas gdy za pomocą tej drugiej dokładnie wyliczamy rozwiązanie, tutaj przybliżamy się do rozwiązania, co zostawia większe pole do pojawienia się błędów.

3. Metoda najmniejszych kwadratów

Treść:

Dla podanych w tabeli danych pomiarowych (próbek) metodą najmniejszych kwadratów należy wyznaczyć funkcję wielomianową y = f (x) (tzn. wektor współczynników) najlepiej aproksymującą te dane. Proszę przetestować wielomiany stopni: 3, 5, 7, 9, 10. Kod aproksymujący powinien być uniwersalną procedurą w Matlabie o odpowiednich parametrach wejścia i wyjścia. Do rozwiązania zadania najmniejszych kwadratów proszę wykorzystać:

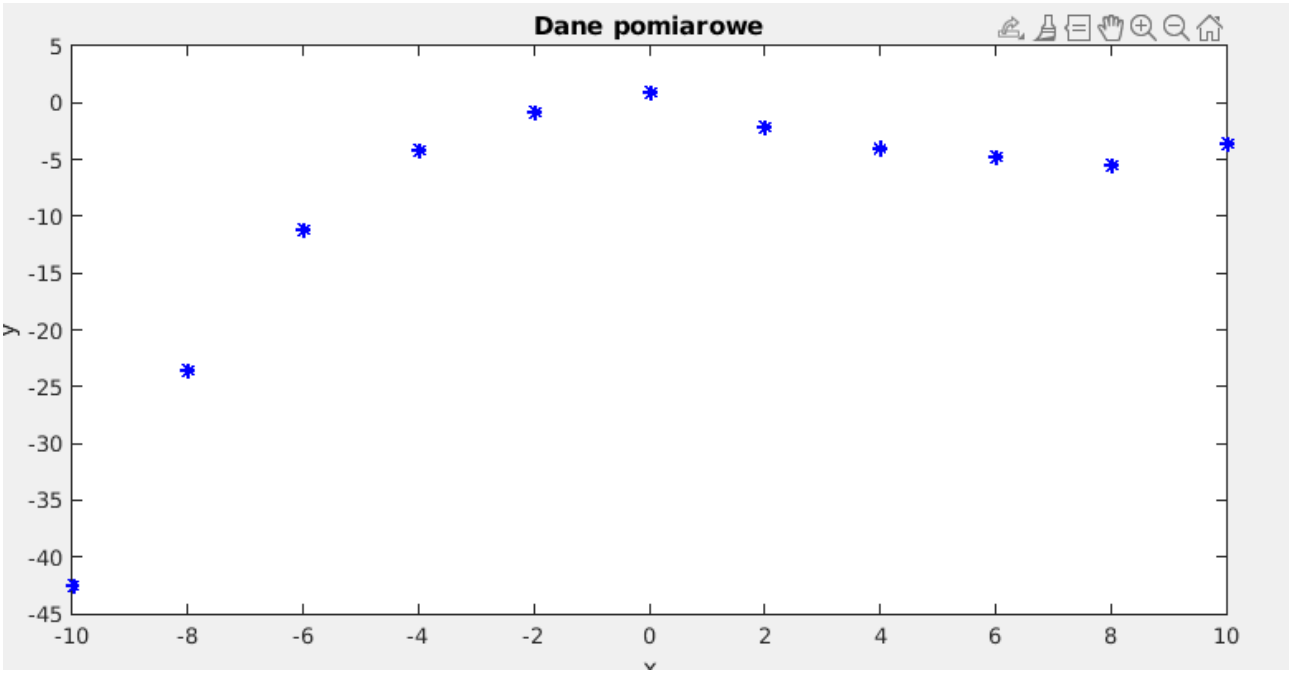
a) Układ równań normalnych i swój solwer z p. 1,

b) Rozkład SVD; do faktoryzacji użyć odpowiedniego solwera Matlaba.

Obliczać błąd aproksymacji w normach euklidesowej $\|.\|2$ oraz maksimum $\|.\|\infty$ (proszę użyć funkcji norm(.,inf)), porównać efektywność trzech podejść. Przedstawić na rysunku otrzymane funkcje na tle danych (funkcję aproksymującą proszę próbkować przynajmniej 10 razy częściej niż dane). Do liczenia wartości wielomianu użyć funkcji polyval.

Dane pomiarowe przedstawiają się następująco:

xi	-10	-8	-6	-4	-2	0	2	4	6	8	10
yi	-42.417	-23.440	-11.160	-4.128	-0.725	0.942	-2.069	-3.908	-4.705	-5.438	-3.578



Obraz 9: Dane wyjściowe

Aby dla zbioru próbek wyznaczyć aproksymującą funkcję wielomianową o stopniu z, należy stworzyć macierz wyjściową A o liczbie wierszy odpowiadającej liczbie próbek (w naszym przypadku 11) oraz liczbie kolumn odpowiadającej stopniu wielomianu + 1 (ostatnia kolumna składa się z samych 1). W celu generowania macierzy stworzyłam funkcję *generate_matrix*.

```
function [A] = generate_matrix(x, degree)
    [~, siz] = size(x);
    A = zeros(siz, degree+1);
    for i = 1:siz
        for j = 1:degree+1
            A(i,j) = x(i)^(degree-j+1);
        end
    end
end
```

Obraz 10: Kod funkcji generate_matrix

W zadaniu mowa o metodzie najmniejszych kwadratów, czyli innymi słowy o aproksymacji średniokwadratowej dyskretnej. Naszym, zadaniem jest wyznaczenie wartości współczynników a1,

a2, … an określających funkcję aproksymującą, aby zminimalizować błąd średniokwadratowy, liczony za pomocą wzoru:

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

a) Rozwiązanie z układem równań normalnych

Układ równań normalnych możemy zapisać w postaci:

AT A a = AT y.

Macierz Grama w naszym przypadku to będzie AT * A. Wektor prawej strony to będzie z kolei AT * y. Korzystając z solwera z punktu 1. w łatwy sposób możemy obliczyć wektor a.

```
% aproksymacja z układem równań normalnych
Gram = A'*A;              % macierz Grama
wekt = A' * y';           % wektor wynikowy
a = LDLT(Gram, wekt);     % wektor współczynników a
```

Obraz 11: Fragment kodu odpowiadający za wyliczenie wektora a za pomocą układu równań

b) Rozwiązanie z rozkładem SVD:

W tym przypadku minimalizujemy błąd pod postacią:

$$\|b-Ax\|_2=\|\tilde{b}-\sum\tilde{x}\|$$

gdzie $\tilde{b}=U^T b, \tilde{x}=V^T x$. Skorzystamy z tych 2 równości oraz zależności $\tilde{x}=\sum^{+¿¿}\tilde{b}$. Po przekształceniach otrzymamy algorytm zaimplementowany poniżej.

```
% aproksymacja z rozkładem SVD
[U,S,V] = svd(A);             % rozkład SVD
s_plus = nonzeros(S);         % wybranie niezerowych wartości szczególnych
y_tilde = U' * y';            % obliczenie wektora y z daszkiem
a_tilde = zeros(3+1);         % inicjalizacja wektora a z daszkiem
for n = 1:3+1;                % obliczenie wartości wektora a z daszkiem
    a_tilde(n) = y_tilde(n) / s_plus(n);
end
a = LDLT(V', a_tilde)         % wektor współczynników a
```

Obraz 12: Fragment kodu odpowiadający za wyliczenie wektora a za pomocą rozkładu SVD – przykład dla wielomianu stopnia 3

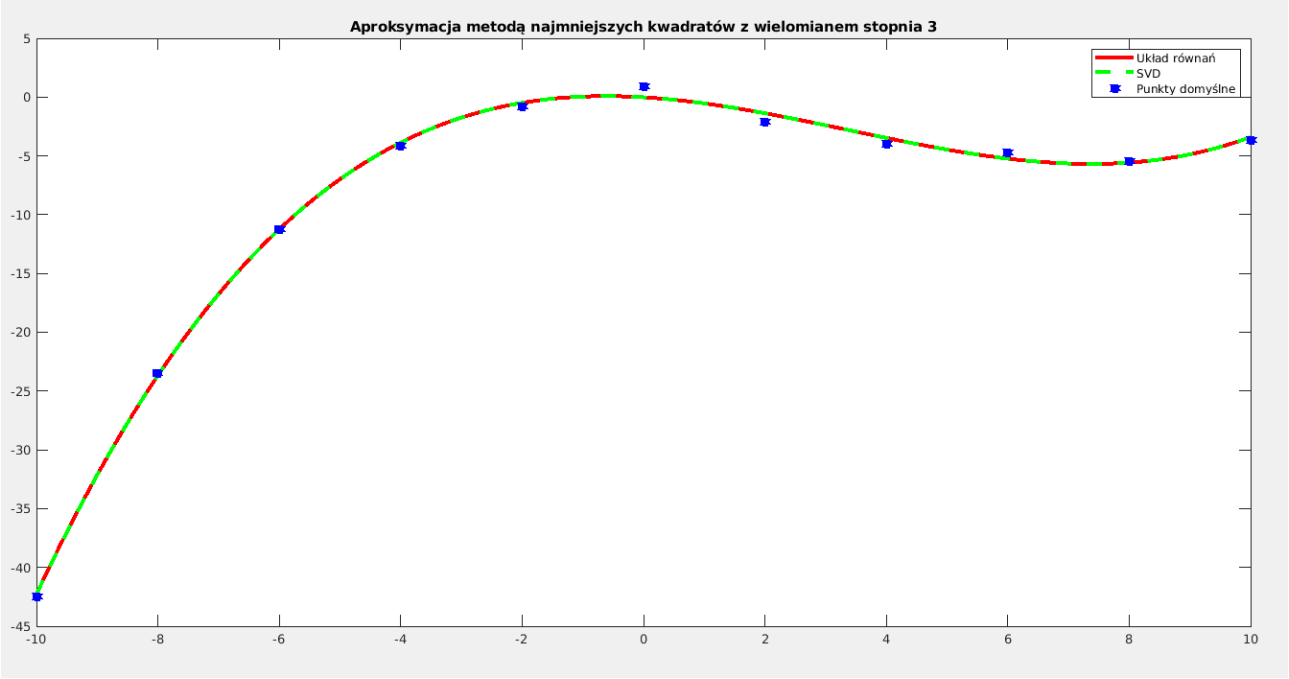
.

Obserwacje I wnioski

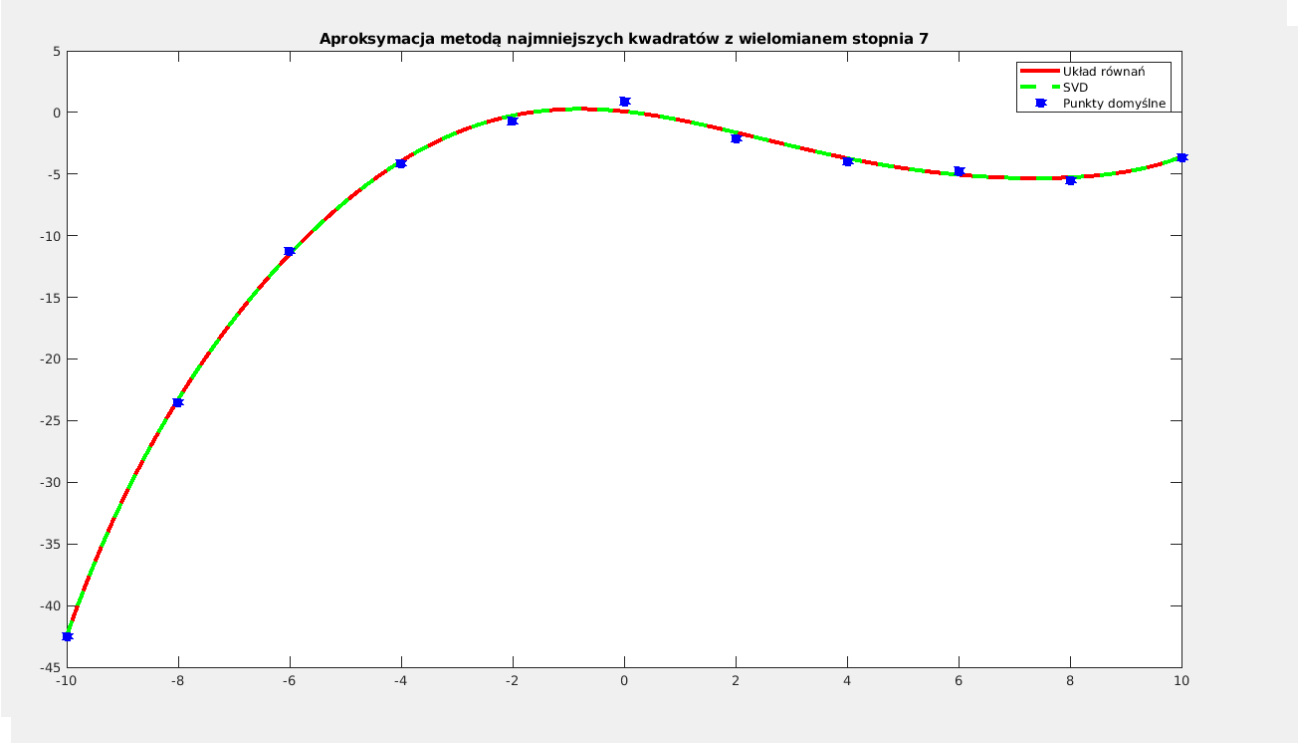
Tabela 3: Błędy obliczeniowe

Błąd	Układ równań normalnych		Rozkład SVD	
Stopień wielomianu	Norm (…, 2)	Norm (…, inf)	Norm (…, 2)	Norm (…, inf)
3	1.458472939305119	0.959445221445218	1.458472939305120	0.959445221445221
5	1.414700923033452	1.064340326340311	1.414700923033452	1.064340326340326
7	1.232509236550306	0.860012340600572	1.232509236550306	0.860012340602922
9	0.769937080948135	0.451394747668274	0.769937080948135	0.451394747669481
10	1.066134375037333e-09	6.536161611947477e-10	1.714279762435301e-10	1.123439119510294e-10

Możemy zaobserować, że zgodnie z oczekiwaniami, wraz ze zwiększaniem stopnia wielomianu, błąd obliczeniowy maleje. Podczas gdy wielomian 3. stopnia nie był w stanie idealnie dopasować się do danych, tak błąd przy wielomianie 10. stopnia jest znikomy. Między testowanymi metodami nie ma znaczących różnic. Oba sprawdzają się na równym poziomie, często błąd obu metod jest identyczny.

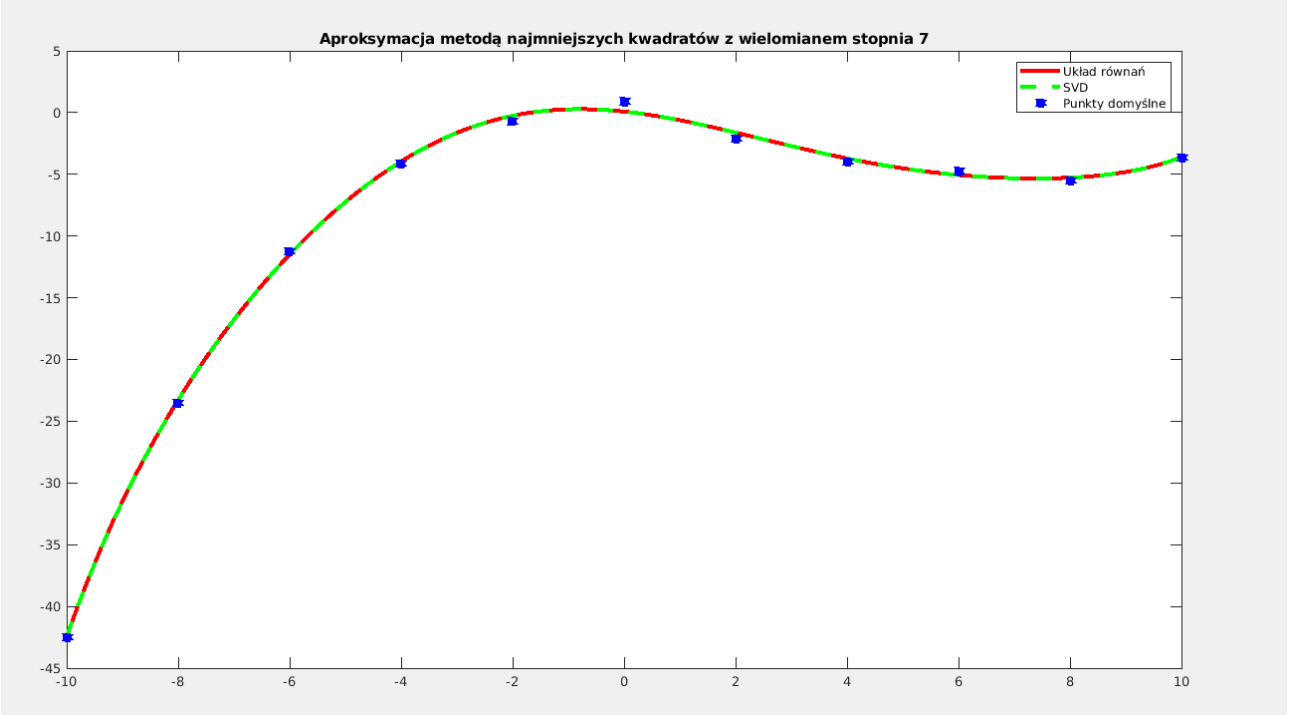


Obraz 13: Aproksymacja wielomianem stopnia 3



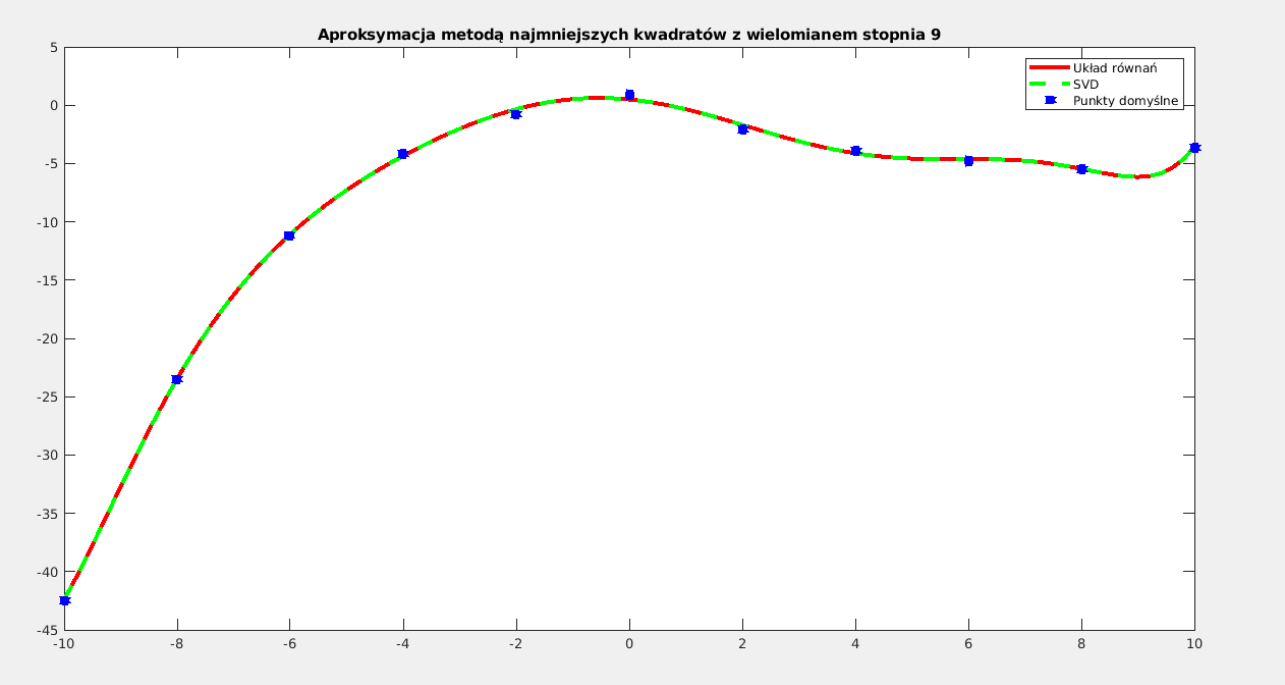
Obraz 14: Aproksymacja wielomianem stopnia 5

.



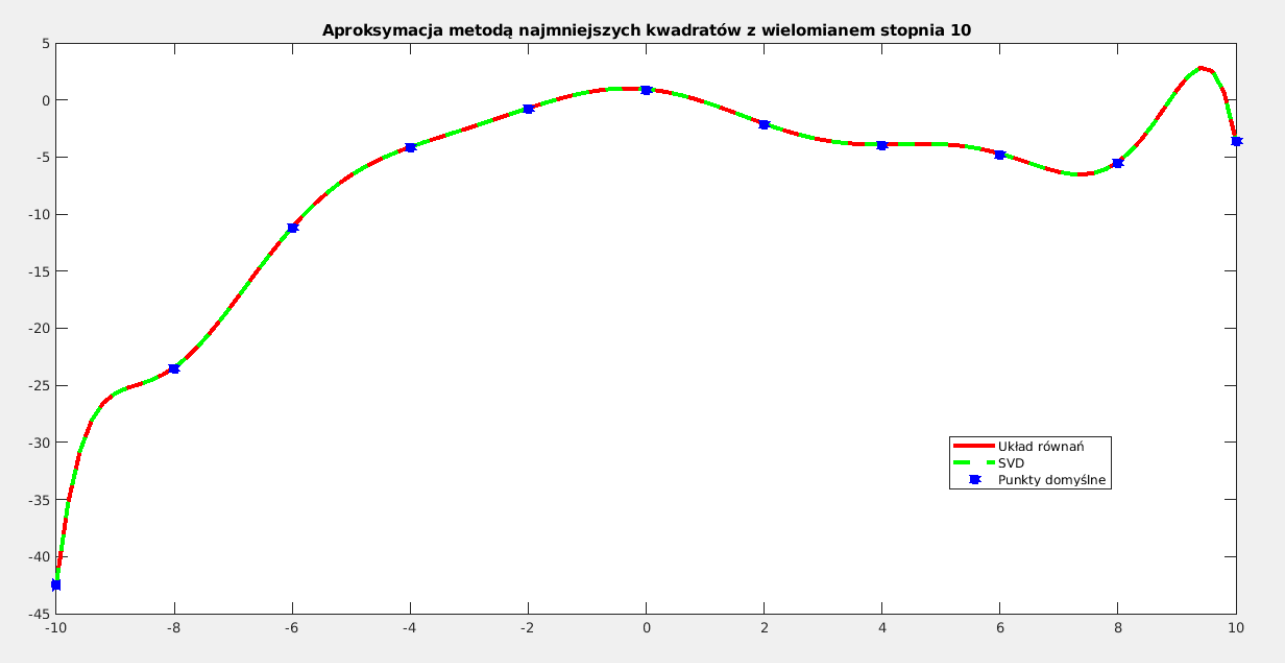
Obraz 15: Aproksymacja wielomianem stopnia 7

.



Obraz 16: Aproksymacja wielomianem stopnia 9

.



Obraz 17: Aproksymacja wielomianem stopnia 10

.

Przyglądając się wykresom, można zauważyć, że wykresy dla wielomianów o stopniach 3-7 nie różnią się od siebie zbyt wiele. Najlepszym odwzorowaniem funkcji wydaje się być wielomian stopnia 9, najdokładniej pokrywa się z wyjściowymi punktami. Wielomian 10. stopnia, mimo najmniejszego błędu, dopasował się do danych gorzej, wykonując na samym końcu “wywijas”. Zarówno wielomiany 9 I 10 stopnia są bardziej złożone niż 3-7. Widać to po ukształtowaniu wykresu.