MNUM Projekt - 1

Bartosz Latosek 310790

Listopad 2022

1 Rozwiązywanie układu równań liniowych za pomocą metody eliminacji Gaussa z częściowym wyborem elementu głównego

1.1 Macierze wykorzystywane do obliczeń

W opisie projektu dostarczone zostały wzory matematyczne generujące macierze używane do obliczeń.

- Przypadek 1. - pliki generateA_1.m, generateB_1.m

$$a_{ij} = \begin{cases} -7 & \text{dla } j = i \\ 2 - \frac{i}{n} & \text{dla } j = i-2 \text{ lub } j = i+2, \quad b_i = 2.5 + 0.5i \\ 0 & \text{dla pozostałych} \end{cases}$$

Rysunek 1: Wzory generujące macierze

- Przypadek 2. - pliki generateA_2.m, generateB_2.m

B) $a_{ij} = 2(i-j) + 2, j \neq i; \;\; a_{ii} = \frac{1}{6}; \;\; b_i = 2.5 + 0.4i$

Rysunek 2: Wzory generujące macierze

1.2 Pseudokod algorytmu

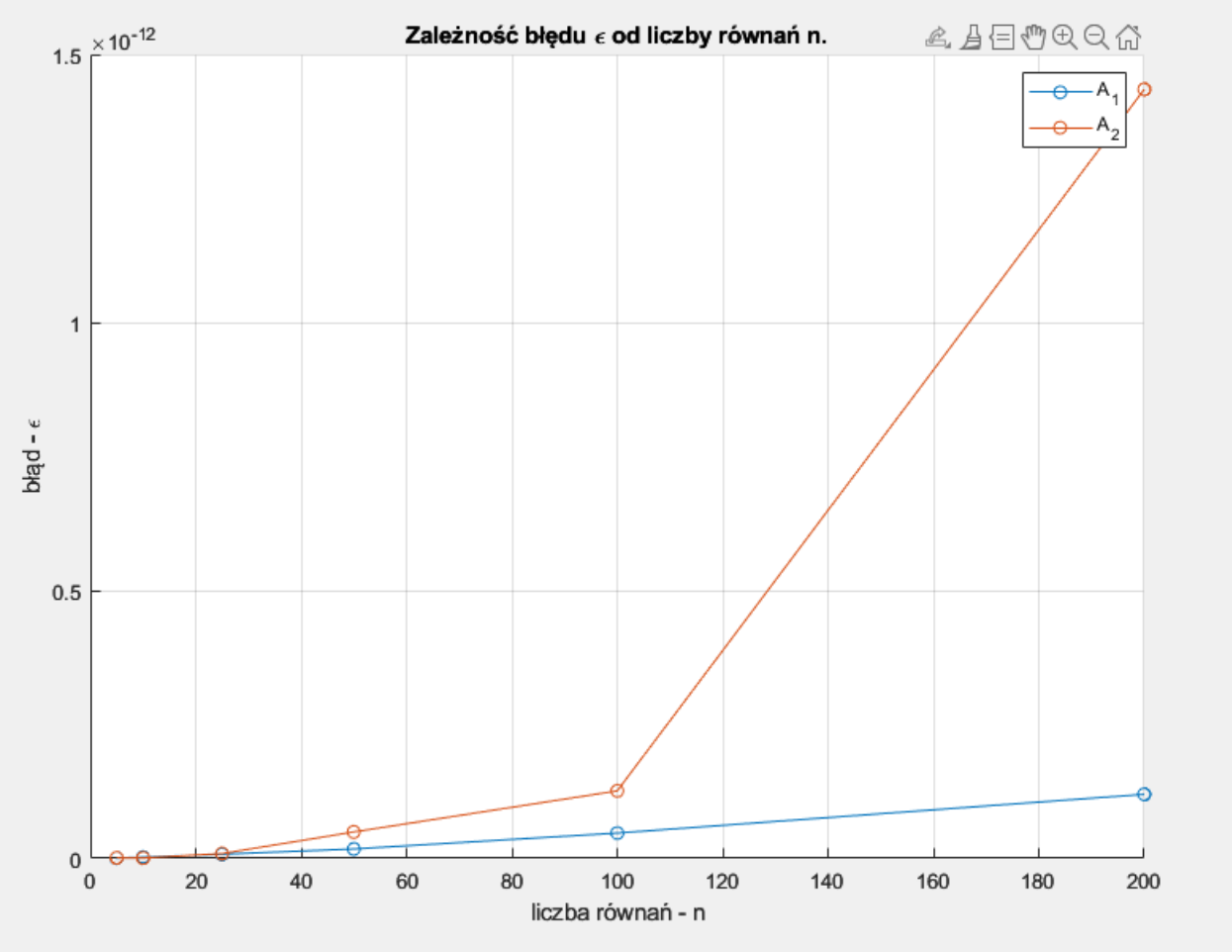
Algorithm 1 Eliminacja Gaussa z częściowym wyborem elementu głównego

function GAUSSIANWITHPARTIALPIVOTING(A, b) ▷ Where A - matrix A, b - matrix b
  $CMB = [A, b]$
  for $i = 1$ to $n_1 - 1$ **do** ▷ Where n_1 - number of linear equations in matrix CMB
    $[max, index] = max(abs(CMB(i : n_1, i)))$ ▷ Finding the index of row with highest abs value of el.
    $tmp = CMB(index + i - 1, :)$ ▷ Switching the found 'max' row with upper most row
    $CMB(index + i - 1, :) = CMB(i, :)$
    $CMB(i, :) = tmp$
    for $j = i + 1$ to n_1 **do**
      $m = CMB(j, i)/CMB(i, i)$ ▷ Gaussian Elimination
      $CMB(j, :) = CMB(j, :) - m * CMB(i, :)$
    end for
  end for
  $Solve for x, where Ax = b$ ▷ Where A, b - matrices changed accordingly to CMB
end function

Funkcja ta jest zaimplementowana w pliku *gaussianWithPartialPivoting.m*. Parametry, które przyjmuje to funkcja generujaca macierz A, funkcja generująca macierz B oraz liczba równań liniowych do rozwiązania.

1.3 Analiza działania algorytmu

Analiza działania algorytmu zostanie przeprowadzona na podstawie skryptu *zadanie1.m*. Funkcja wykonana zostanie dla n równań, gdzie n należy do zbioru [5, 10, 25, 50, 100, 200], oraz dla obydwu macierzy wymienionych w punkcie 1.1.



Rysunek 3: Zależność błędu od liczby równań dla obydwu macierzy

Na wykresie wyraźnie widać, że (dla liczby równań większej od 20) dla macierzy opisanej za pomocą drugiego wzoru, generowany błąd jest znacząco większy niż dla macierzy opisanej za pomocą pierwszego wzoru.

2 Rozwiązywanie układów równań z wykorzystaniem iteracyjnej metody Jackobiego

2.1 Macierze wykorzystywane do obliczeń

W zadaniu wykorzystane zostaną dokładnie te same macierze co w zadaniu 1, generowane za pomocą odpowiednich funkcji.

2.2 Pseudokod algorytmu

Algorithm 2 Iteracyjna metoda Jackobiego

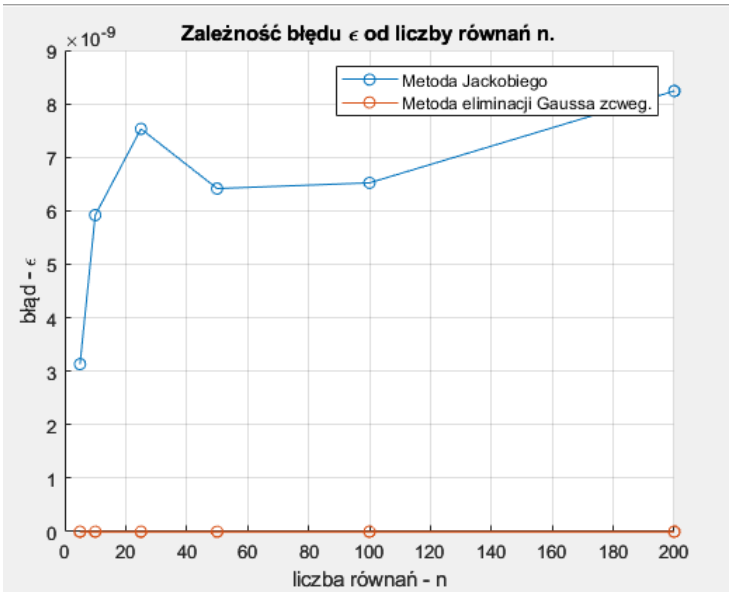
function JACOBIMETHOD(δ_0, A, b) ▷ Where δ_0 - stop criterium, A - matrix A, b - matrix b
  $X = zeros(n_1, 1)$ ▷ Where n_1 - size of output vector
  $\delta = inf$
  while $\delta > \delta_0$ **do**
    $X_i = zeros(n_1, 1)$ ▷ Snapshot of current X (so that we wont change the X until iteration completes)
    for $i = 1$ to n_1 **do**
      $X_{temp} = 0$
      for $j = 1$ to n_1 **do**
        if $j! = i$ **do**
          $X_{temp} = X_{temp} + A(i, j) * X(j)$
        endif
      end for
      $X_i = (-1/A(i, i)) * (X_{temp} - b(i))$
    end for
    $X = X_i$
    $\delta = norm(A * X - b)$
  end while
end function

2.3 Analiza działania algorytmu

Analiza działania algorytmu zostanie przeprowadzona na podstawie skryptów *zadanie2_eps.m i zadanie2_times.m*. Funkcja wykonana zostanie dla n równań, gdzie n należy do zbioru [5, 10, 25, 50, 100, 200], oraz dla obydwu macierzy wymienionych w punkcie 1.1.

Porównanie błędów w stosunku do ilości równań dla metody eliminacji Gaussa z częściowym wyborem elementu głównego i iteracyjnej metody Jackobiego

2.3.1 Macierz A1, B1



Rysunek 4: Zależność błędu od liczby równań dla obydwu algorytmów

Na rysunku wyraźnie widać, że iteracyjna metoda Jackobiego generuje większe błędy od metody Gaussa. Wiąże się to z tym, że w metodzie Jackobiego wykonywane jest znacznie więcej operacji mnożenia, przez co generowany błąd rośnie wraz z każdą iteracją.

2.3.2 Macierz A2, B2

W drugim przypadku algorytm Jackobiego nie wykonał się poprawnie. Po analizie kodu okazało się, że przewidywane X dążą do nieskończoności.

```
x_1 =

   Inf
   Inf
   Inf
   Inf
   Inf
```

Rysunek 5: Znalezione rozwiązanie X dla n = 5 równań

Po dalszej analizie doszedłem do wniosku, że problem nie leży w implementacji algorytmu, a ma związek z macierzą A2 generowaną za pomocą funkcji *generateA_2.m*. Poniżej przykładowo wygenerowana macierz dla n = 5 równań.

```
ans =

    0.1667         0   -2.0000   -4.0000   -6.0000
    4.0000    0.1667         0   -2.0000   -4.0000
    6.0000    4.0000    0.1667         0   -2.0000
    8.0000    6.0000    4.0000    0.1667         0
   10.0000    8.0000    6.0000    4.0000    0.1667
```

Rysunek 6: Wygenerowana macierz A dla n = 5 równań

W tym momencie ważne jest zaznaczenie, że iteracyjna metoda Jackobiego działa dobrze dla macierzy, które posiadają cechę dominacji diagonalnej.

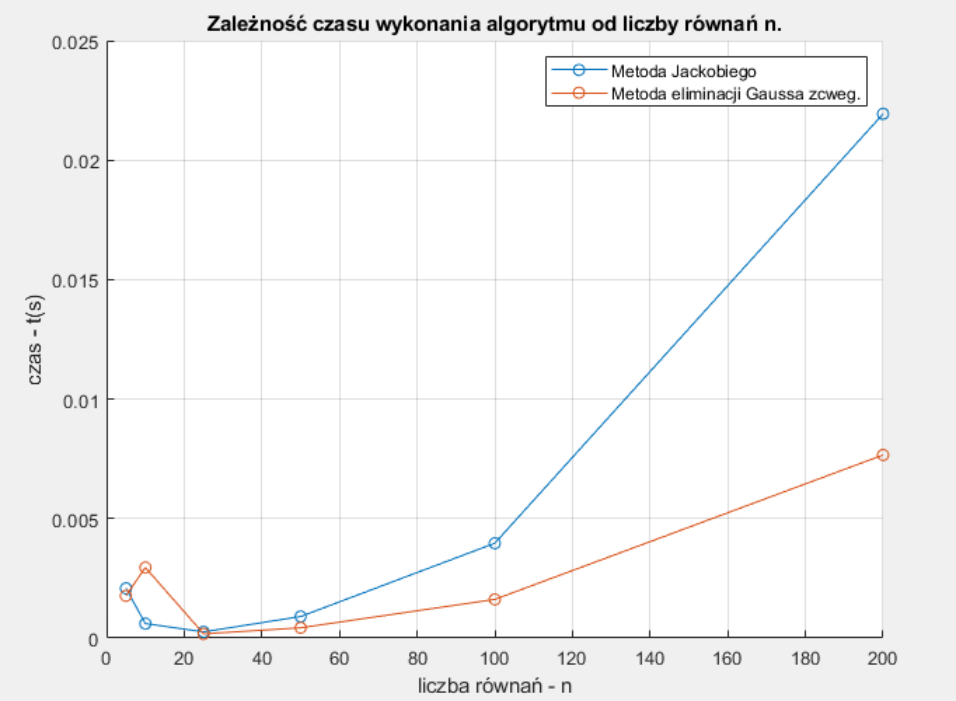
$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \quad \text{for all } i$$

Rysunek 7: Warunek diagonalnej dominacji

Wyraźnie widać, ze dla macierzy na *Rys 6.* powyższa zależność nie jest spełniona. Metoda Jackobiego "gubi się", przy obliczaniu kolejnych przybliżeń rozwiązania układu równań, dla którego to rozwiązania wartości bezwzględne poszczególnych składowych zbiegają do nieskończoności.

Porównanie czasów wykonania algorytmu w stosunku do ilości równań dla metody eliminacji Gaussa z częściowym wyborem elementu głównego i iteracyjnej metody Jackobiego

2.3.3 Macierz A1, B1



Rysunek 8: Wykres zależności czasu od liczby równań dla obydwu algorytmów

Z powyższego rysunku wynika, że metoda Jackobiego jest nieznacznie szybsza niż metoda eliminacji Gaussa dla małej ilości równań (n ¡ 20) i wolniejsza niż niż metoda eliminacji Gaussa wraz ze wzrostem liczby równań.

2.3.4 Macierz A2, B2

Przypadek analogiczny do punktu 2.3.2 .

3 Wyznaczenie funkcji wielomianowej aproksymującej dane metodą najmniejszych kwadratów

3.1 Dane do aproksymacji

x_i	y_i
-10	-32.959
-8	-20.701
-6	-12.698
-4	-5.150
-2	-1.689
0	0.126
2	0.074
4	-0.870
6	-1.737
8	-3.995
10	-4.898

Rysunek 9: Tabela x i y do aproksymacji

3.2 Pseudokod algorytmu

Algorithm 3 Metoda najmniejszych kwadratów

function LEASTSQUARESMETHOD(X, Y, n) ▷ Where X - input vector, Y - output vector, n - degree of aprox. polynomial
  $A = zeros(n, size(X, dim = 1))$ ▷ size(X, 1) - number of given x in X
  for $i = 1$ to n **do**
    for $j = 1$ to $size(X, dim = 1)$ **do**
      $A(j, i) = X(j)^{(n-1)}$
    end for
  end for
    if $NormalEquations$ **do**
      $A_n = A^T * A$
      $n_n = A^T * Y$
      $x_n = A_n / b_n$
    endif
    if $QRfactorisation$ **do**
      $[Q, R] = qr(A)$
      $x_n = linsolve(R, Q^T * Y)$ ▷ Solve linear equations using matlab function
    endif
    $x_1 = linspace(min(X), max(X))$ ▷ Create linear space from X vector
    $y_1 = 0$
  for $j = 1$ to n **do**
      $y_1 = y_1 + x_n(i) .* x_1.^{(n-i)}$
  end for
end function

W powyższym pseudokodzie, w sekcji z rozkładem QR rozwiązywany jest układ równań liniowych: $R * X = Q^T * Y$. Wynika to z poniższej własności.

$$Ax \approx b \quad \text{"best" solution:} \quad x = (A^T A)^{-1} A^T b$$

$$A = QR \quad \text{where} \quad Q^T Q = I$$

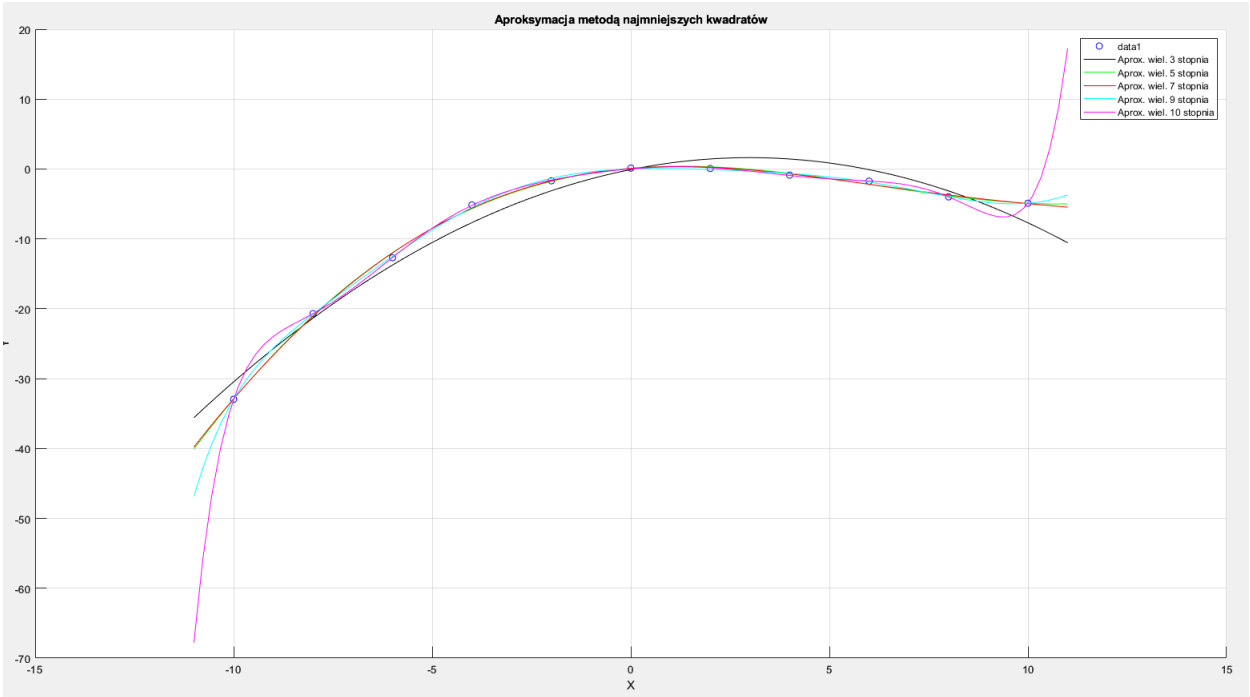
$$\begin{aligned} x &= (A^T A)^{-1} A^T b = ((\underbrace{QR}_{A})^T (\underbrace{QR}_{A}))^{-1} (\underbrace{QR}_{A})^T b \\ &= (R^T \underbrace{Q^T Q}_{I} R)^{-1} R^T Q^T b = (R^T R)^{-1} R^T Q^T b \\ &= R^{-1} \underbrace{R^{-T} R^T}_{I} Q^T b = R^{-1} Q^T b. \end{aligned}$$

$$Ax \approx b \quad \text{"best" solution solves:} \quad R\hat{x} = Q^T b$$

Rysunek 10: Wyjaśnienie algorytmu

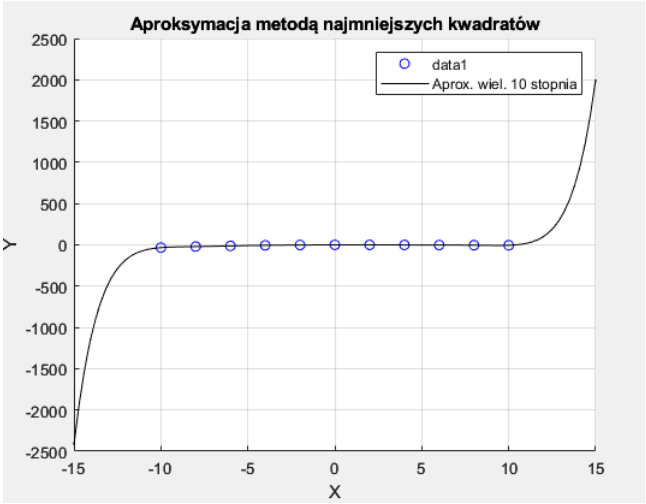
3.3 Analiza algorytmu

Analiza wykonana zostanie za pomocą skryptu *zadanie3_main.m*. Poniżej wykres funkcji aproksymujących dla podanego zbioru danych dla wielomianów stopnia n, gdzie n należy do [3, 5, 7, 9, 10].



Rysunek 11: Wykresy funkcji aproksymujących

Na podanym wykresie widać, że najlepszą aproksymacje uzyskujemy dla stopni wielomianu n = 5 i n = 7. Dla n = 3, kształt funkcji aproksymującej słabo odzwierciedla faktyczne rozłożenie punktów, natomiast dla n = 9 i n = 10, następuje zjawisko przeuczenia (overfitting). Stopień wielomianu jest zbyt wysoki, przez co funkcja dobrze aproksymuje jedynie obszar występowania punktów, na których była ona budowana, co dobrze obrazuje poniższy obraz.



Rysunek 12: Zjawisko przeuczenia

3.4 Porównanie działania algorytmu wykorzystując układ równań normalnych i rozkład QR

3.4.1 Norma euklidesowa

Analiza wykonana zostanie za pomocą skryptu *zadanie3_norms.m*.
Po wykonaniu algorytmu wyniki norm euklidesowych uzyskanych dla algorytmu wykorzystującego układ równań normalnych były identyczne, co wyniki dla algorytmu wykorzystującego rozkład QR.

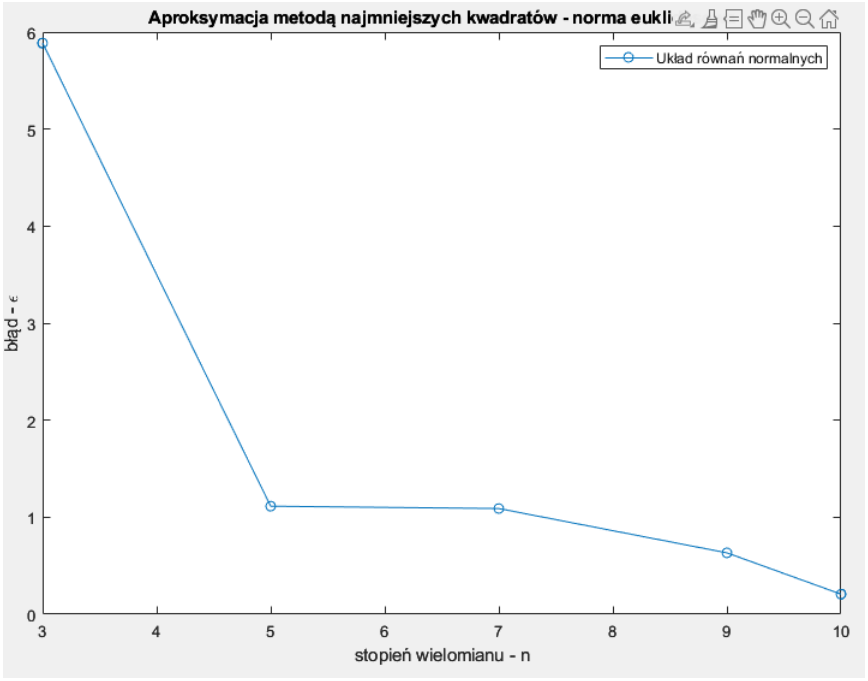
```
err2_n_all =

    5.8864    1.1134    1.0894    0.6328    0.2070


err2_qr_all =

    5.8864    1.1134    1.0894    0.6328    0.2070
```

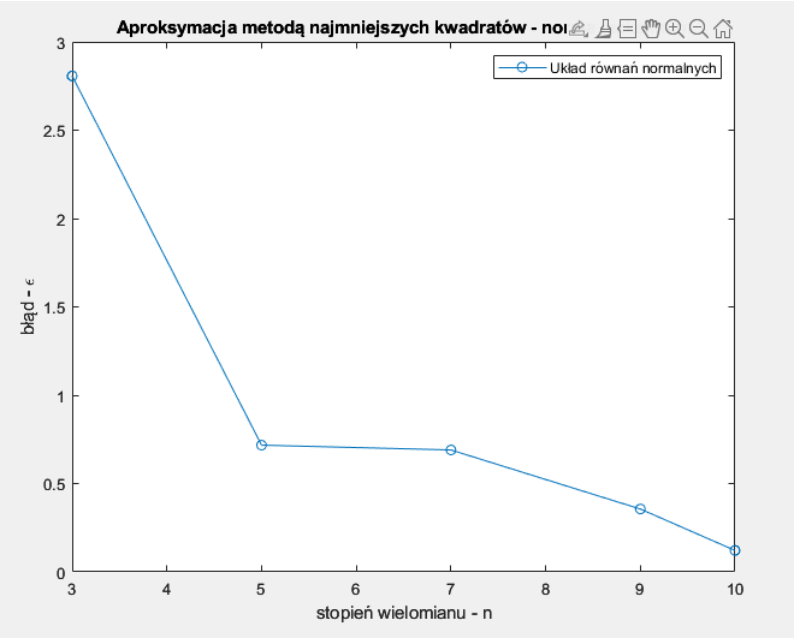
Rysunek 13: Wyniki obliczenia norm euklidesowych dla układu równań normalnych



Rysunek 14: Wyniki obliczenia norm euklidesowych dla układu równań normalnych i rozkładu QR

3.4.2 Norma maksimum

W tym przypadku nastąpiła analogiczna sytuacja. Wartości obliczonych norm pokrywały się.



Rysunek 15: Wyniki obliczenia norm maksimum dla układu równań normalnych i rozkładu QR

Z powyższych wykresów wynika, że obydwa podejścia mają zbliżoną efektywność.