Dokumentacja projektu laboratoryjnego numer 2 przedmiot MNUM

Kamil Foryszewski

25 kwietnia 2016

Spis treści

1 Zadanie 1 **1**
1.1 Polecenie . . . 1
1.2 Rozkład QR . . . 1
1.2.1 Opis teoretyczny . . . 1
1.2.2 Metoda Gramma-Schmidta . . . 1
1.2.3 Realizacja w programie Matlab . . . 2
1.3 Znajdowanie wartości własnych metodą QR bez przesunięć . . . 2
1.3.1 Opis teoretyczny . . . 2
1.3.2 Realizacja w programie Matlab . . . 2
1.3.3 Zbieżność metory QR . . . 3
1.4 Metoda QR z przesunięciami . . . 3
1.4.1 Opis teoretyczny . . . 3
1.4.2 Realizacja w programie Matlab . . . 3
1.5 Uwarunkowanie danych z zadania . . . 4
1.6 Skrypt generujący rozwiązanie zadania w programie Matlab . . . 4
1.7 Wyniki . . . 6
1.8 Wnioski . . . 6

2 Zadanie 2 **7**
2.1 Polecenie . . . 7
2.2 Metoda równań normalnych . . . 7
2.3 Metoda QR . . . 7
2.4 Uwarunkowanie . . . 8
2.5 Realizacja w programie Matlab . . . 8
2.6 Wyniki działania programu . . . 9
2.7 Wnioski . . . 10

1 Zadanie 1

1.1 Polecenie

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 z przesunięciami dla macierzy niesymetrycznej. Następnie proszę przetestować skutecznoność (zbieżność obu wersji algorytmu dla 30 różnych macierzy losowych o wymiarach 5x5, 10x10 i 20x20. Proszę podać średnią liczbę iteracji dla obu metod. Dla wybranych macierzy proszę porównać otrzymane wyniki z wartościami własnymi obliczonymi poleceniem eig().

1.2 Rozkład QR

1.2.1 Opis teoretyczny

Rozkład QR macierzy kwadratowej A polega na tym aby macierz A zapisać w postaci iloczynu QR, gdzie macierz Q jest macierzą ortogonalną, a R jest macierzą trójkątną górną. Macierz Q o wyrazach rzeczywistych nazywamy ortogonalną, jeżeli spełnia warunek $QQ^T = I$. Rozkład QR można uzyskać stosując różne algorytmy zależne od wyboru przekształceń. Jeżeli założymy, że macierz A jest nieosobliwa i że na przekątnej macierzy R są wyrazy dodatnie, to rozkład jest jednoznaczny, a więc nie zależy od wyboru algorytmu.

1.2.2 Metoda Gramma-Schmidta

$A \in \mathbb{R}^{m,n}$ jest macierzą o liniowo niezależnych kolumnach $\vec{a}_1, \dots, \vec{a}_n \in \mathbb{R}^n$. Przeprowadzając ortogonalizację Grama-Schmidta tych kolumn, otrzymujemy ortonormalny układ wektorów $\vec{q}_1, \dots, \vec{q}_n$

$$\vec{b}_k = \vec{a}_k - \sum_{j=1}^{k-1} (\vec{q}_j^T \vec{a}_k)\vec{q}_j, \quad \vec{q}_k = \|\vec{b}_k\|_2^{-1} \vec{b}_k, \qquad k = 1, 2, \dots, n.$$

Niech

$$r_{j,k} = \vec{q}_j^T \vec{a}_k, \quad (j = 1, \dots, k-1), \qquad r_{k,k} = \|\vec{b}_k\|_2, \qquad r_{j,k} = 0, \quad (j > k)$$

i

$$Q = [\vec{q}_1, \dots, \vec{q}_n] \in \mathbb{R}^{m,n}, \qquad R = [r_{i,j}] \in \mathbb{R}^{n,n}.$$

Wtedy Q jest macierzą o ortogonalnych kolumnach, R jest macierzą trójkątną górną i $A = QR$. Ponieważ wyżej przedstawiona medota ma gorsze własności numeryczne od tzw. zmodyfikowanej metody Gramma-Schmidta, to na potrzeby realizacji zadania zostanie użyta metoda o lepszych własnościach numerycznych. Modyfikacja polega na zmianie kolejności ortogonalizacji. Zamiast ortogonalizować kolumny po kolei, algorytm najpierw wyznacza współczynnik dla pierwszej kolumny a następnie ortogonalizuje względem niego pozostałe.

1.2.3 Realizacja w programie Matlab

```
%funkcja rozkladu qr macierzy zmodyfikowanym algorytmem Grama-Schmidta
%Na podstawie ksiazki prof. Tatjewskiego
function [Q,R] = qrgsm(A)

  [m n] = size(A);
  Q = zeros(m,n);
  R = zeros(n,n);
  d = zeros(1,n);
  %rozklad A kolumnowo ortogonalny
  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
  for i=1:n
```

```
    dd = norm(Q(:,i));
    Q(:,i) = Q(:,i)/dd;
    R(i,i:n) = R(i,i:n)*dd;
  end

end
```

1.3 Znajdowanie wartości własnych metodą QR bez przesunięć

1.3.1 Opis teoretyczny

Jedną z metod wyznaczania wartości własnych macierzy jest metoda wykorzystująca rozkład QR macierzy. Metoda ta w najprostszym wariancie ma następującą postać:

$A_0 := A; Z_0 := I;$
dla $k = 1, 2, 3...$

$$\begin{cases} A_{k-1} := Q_k R_k; \\ A_k := R_k Q_k; \\ Z_k := Zk - 1Q_k; \end{cases}$$

1.3.2 Realizacja w programie Matlab

```
%funkcja wyznaczjaca wartosci wlasne metoda qr bez przesuniec
%na podstawie ksiazki prof. Tatjewskiego
function [D,t,i,v] = eignoshift (A, prec, it)

  s = tic;
  v=1;
  n = size(A,1);
  i = 1;
  while i <= it && max(max(A-diag(diag(A)))) > prec
    [Q1,R1] = qrgsm(A);
    A = R1*Q1;
    i = i + 1;
  end
  if i > it
    %error('przekrczono maksymalna liczbe iteracji');
    v=0;
  end
  D = diag(A);
  t = toc(s);

end
```

1.3.3 Zbieżność metory QR

Dla macierzy A symetrycznej macierz A_k zbiega do macierzy diagonalnej $diag(\lambda_i)$. Szybkość zbierzonośći przedstawia następujący wzór:

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

Z którego wynika że jeżeli wartości własne leżą blisko siebie to metoda powoli zbiega do rozwiązania. Aby poprawić jej zbierzność stosuje sięzmodywikowaną metodę, którą opiszę poniżej.

1.4 Metoda QR z przesunięciami

1.4.1 Opis teoretyczny

Metodę wyznaczania wartości własnych QR z przesunięiami można przybliżyć poniższym algorytmem:

1. Znajdujemy wartość własną λ_n jako najbliższą wartość własną podmacierzy 2x2 z prawego dolnego rogu macierzy $A^{(k)}$ wyznaczając wartości własne jako pierwiastki wielomianu drugiego stopnia o współczynnikach wielomianu charakterystycznego.

2. Opuszczamy ostatni wiersz i ostatnią kolumnę aktualnej macierzy $A^{(k)}$ (deflacja).

3. Znajdujemy następną wartość własną λ_{n-1}, przekształcając macierz $A^{(k)}_{n-1}$ aż do uzyskania $e^{(k)}_{n-2} = 0$. Iterujemy aż do wyzerowania wszystkich elementów poza elementem diagonalnym (pętla while).

4. Powtarzamy kroki 2 i 3 aż do uzyskania wszystkich wartości własnych uwzględniając oczekiwaną prezycję obliczeń.

1.4.2 Realizacja w programie Matlab

```
%funkcja zwracajaca diagonalna maciez z wartosciami wlasnymi metoda qr z przesunieciami
%A macierz wejsciowa , prec prezycja wyniku, it maksymalna liczba iteracji
%Na podstawie skryptu prof. Tatjewskiego
function [D,t,iteration,v] = eigshift (A, prec, it)

  n = size(A,1);
  D = diag(zeros(n));
  I = A; %macierz poczatkowa
  v = 1;
  iteration = 0;
  time = tic;
  for k=n:-1:2
    K = I(1:k, 1:k); % macierz poczatkowa dla pojedynczej wart. wlasnej
    i=0;
    while i <= it && max(abs(K(k,1:k-1))) > prec
      p = [1 -(K(k-1,k-1)+K(k,k)) K(k,k)*K(k-1,k-1)-K(k,k-1)*K(k-1,k)];
      ev = roots(p);
      % M = [a b,c d] rownanie dla M : 1*x^2 -(a+d)*x +a*d-c*b
      if abs(ev(1)-K(k,k)) < abs(ev(2)-K(k,k))
        shift = ev(1); %bajblizsza wartosc wlasna podmacierzy DD
      else
        shift = ev(2);
      end
      K = K-eye(k)*shift; %przesuniecie macierzy
      [Q,R] = qrgsm(K);
      K = R*Q+eye(k)*shift; % przeksztaucenie macierzy
      i = i+1;
      iteration = iteration +1;
    end
    if i>it
      %error('przekroczono maksymalna liczbe iteracji');
      v = 0;
```

```
      break;
    end
    D(k) = K(k,k);
    if k>2
      I = K(1:k-1,1:k-1); %deflacja macierzy
    else
      D(1) = K(1,1); %ostatnia wartosc wlasna
    end
  end
  t = toc(time);

end
```

1.5 Uwarunkowanie danych z zadania

W zadaniu dane są macierze zarówno symetryczne jak i niesymetryczne generowane losowo przez wbudowaną funkcję rand(). Dla macierzy rzeczywistych i symetrycznych zachodzi twierdzenie Bauera-Frike'a mówiące wprost że $cond(X) = 1$ gdzie X jest symetyczna o wartościach rzeczywistych. Z kolei dla macierzy niediagonalizowalnych uwarunkowanie wartości może być dowolnie duże, co również wynika z twierdzenia.

1.6 Skrypt generujący rozwiązanie zadania w programie Matlab

```
%realizacja zadania 1 z projektu 2.17
  clear;
  F = fopen('results.txt','w'); %wynik zapisany do pliku
  fprintf(F, 'Wyniki do zadania numer 1 \n');
  Z = [5 10 20];

  %macierz symetryczna
  fprintf(F, '<<<Macierz symetryczna>>>\n\n');
  for i=1:3;
    Gtimen = 0; %zmienne do wyliczania srednich wartosci
    Gitern = 0;
    Gprecn = 0;
    Gvalidn = 0;
    Gtimes = 0;
    Giters = 0;
    Gprecs = 0;
    Gvalids = 0;
    Gteig = 0;
    for j=1:30
      A = cmsim(Z(i));
      [R,timen,itn,validn] = eignoshift(A,0.00001,1000);% metoda bez
          przesuniec
      [S,times,its,valids] = eigshift(A,0.00001,1000); % metoda z
          przesunieciami
      eigstart = tic;
      E = eig(A);
      teig = toc(eigstart);
      Gteig = Gteig + teig;
      Gtimen = Gtimen + timen;
      Gitern = Gitern + itn;
      Gprecn = Gprecn + norm(R-E);
      if validn == 1
```

```
      Gvalidn = Gvalidn + 1; % zliczanie poprawnych rozwiazan
    end
    Gtimes = Gtimes + times;
    Giters = Giters + its;
    Gprecs = Gprecs + norm(S-E);
    if valids == 1
      Gvalids = Gvalids + 1;
    end
  end
  Gtimen = Gtimen/Gvalidn;
  Gitern = Gitern/Gvalidn;
  Gprecn = Gprecn/Gvalidn;
  Gtimes = Gtimes/Gvalids;
  Giters = Giters/Gvalids;
  Gprecs = Gprecs/Gvalids;
  Gteig = Gteig/30;
  fprintf(F, '<<<Liczba rownan %d >>>\n',Z(i));
  fprintf(F, '<<<Macierz symetryczna bez przesuniec>>>\n');
  fprintf(F, 'Sredni czas wykonania: %d\n',Gtimen);
  fprintf(F, 'Srednia ilosc iteracji: %d\n',Gitern);
  fprintf(F, 'Ilosc zakonczonych: %d\n',Gvalidn);
  fprintf(F, 'Srednia roznica miedzy eig(): %d\n\n',Gprecn);

  fprintf(F, '<<<Macierz symetryczna z przesunieciami>>>\n');
  fprintf(F, 'Sredni czas wykonania: %d\n',Gtimes);
  fprintf(F, 'Srednia ilosc iteracji: %d\n',Giters);
  fprintf(F, 'Ilosc zakonczonych: %d\n',Gvalids);
  fprintf(F, 'Srednia roznica miedzy eig(): %d\n\n',Gprecs);

  fprintf(F, '<<<Wyniki dla funkcji eig()>>>\n');
  fprintf(F, 'Sredni czas wykonania: %d\n',Gteig);
end
% macierz niesymetryczna
fprintf(F, '<<<Macierz niesymetryczna>>>\n\n');
for i=1:3;
  Gtimes = 0;
  Giters = 0;
  Gprecs = 0;
  Gvalids = 0;
  Gteig = 0;
  for j=1:30
    A = cmunsim(Z(i));
    [S,times,its,valids] = eigshift(A,0.00001,1000);
    eigstart = tic;
    E = eig(A);
    teig = toc(eigstart);
    Gteig = Gteig + teig;
    if valids == 1
      Gtimes = Gtimes + times;
      Giters = Giters + its;
      Gprecs = Gprecs + norm(S-E);
      Gvalids = Gvalids + 1;
    end
```

```
    end
    Gtimes = Gtimes/Gvalids;
    Giters = Giters/Gvalids;
    Gprecs = Gprecs/Gvalids;
    Gteig = Gteig/30;

    fprintf(F, '<<<Liczba rownan %d >>>\n',Z(i));
    fprintf(F, '<<<Macierz niesymetryczna z przesunieciami>>>\n');
    fprintf(F, 'Sredni czas wykonania: %d\n',Gtimes);
    fprintf(F, 'Srednia ilosc iteracji: %d\n',Giters);
    fprintf(F, 'Ilosc zakonczonych: %d\n',Gvalids);
    fprintf(F, 'Srednia roznica miedzy eig(): %d\n\n',Gprecs);

    fprintf(F, '<<<Wyniki dla funkcji eig()>>>\n');
    fprintf(F, 'Sredni czas wykonania: %d\n',Gteig);
  end

  fclose(F);
```

1.7 Wyniki

Wyniki w postaci tabularycznej wygenerowane przez powyższy skrypt:

Tablica 1: Wyniki dla macierzy symetrycznych

Rozmiar	Funkcja	Czas (s)	Ilość Iteracji	Ilość zakończonych	Różnica eig()
5x5	bez przesunięć	0.0619796	62.1333	30	7.95247
5x5	z przesunięciami	0.00865486	7.96667	30	7.48066
5x5	eig()	3.31163e-05	-	30	-
10x10	bez przesunięć	0.773586	256.321	28	17.7433
10x10	z przesunięciami	0.0323579	14.5333	30	15.7378
10x10	eig()	4.24306e-05	-	30	-
20x20	bez przesunięć	9.13961	865.773	22	44.0736
20x20	z przesunięciami	0.161214	28.2333	30	31.0682
20x20	eig()	-	-	30	-

Tablica 2: Wyniki dla macierzy niesymetrycznych

Rozmiar	Funkcja	Czas (s)	Ilość Iteracji	Ilość zakończonych	Różnica eig()
5x5	z przesunięciami	0.010867	9.73333	30	0.916182
5x5	eig()	3.57231e-05	-	30	-
10x10	z przesunięciami	0.046351	20.6	30	2.35018
10x10	eig()	5.66324e-05	-	30	-
20x20	z przesunięciami	0.251406	44.7333	30	4.95558
20x20	eig()	0.000143798	-	30	-

1.8 Wnioski

Dla macierzy symetrycznych funkcje z przesunięciami osiągają znacznie dokładniejsze wyniki w dużo krótszym czasie i mniejszej liczbie iteracji niż algorytm bez przesunięć. Dla macierzy o większych rozmiarach funkcja bez przesunięć nie jest w stanie wyznaczyć wartości w ograniczonej liczbie iteracji.

Jeżeli chodzi o porównanie wyników z funkcją eig() to algotytm z przesunięciami jest dużo wolniejszy, oraz różnica w wyznaczonych wartościach rośnie wraz rozmiarem macierzy. Więc dokładność maleje. Jeżeli chodzi o macierze niesymetryczne to funkcja z przesunięciami poradziła sobie z wyznaczeniem we wszystkich przypadkach, jednak zajęło to zdecydowanie więcej czasu i iteracji, oraz w porównaniu z funkcją eig() różnica wartości była znaczna. Wynika to ze złego uwarunkowania macierzy niesymetrycznej względem algorytmu z przesunięciami. Jeżeli mamy do czynienia z macierzą niediagonizowalną to uwarunkowanie drastycznie się pogarsza.

2 Zadanie 2

2.1 Polecenie

Proszę napisać program rozwiązujący układ n równań liniowych $Ax = b$ w sensie minimalizacji sumy kwadratów niespełnienia równań dla rosnącej liczby równań n = 10,20,40... .

2.2 Metoda równań normalnych

Rozwiązanie układu równań normalnych jest uogólnieniem rozwiązywania kwadratowych układów równań liniowych. Można je scharakteryzować w następujący sposób:

$A^T Ax = A^T b$

Złożoność obliczeniowa takiego algorytmu wynosi $n^2(m + n/3)$.

2.3 Metoda QR

Metoda QR rozwiązywania układów równań w sensie minimalizacji sumy kwadratów polega na rozwiązanu układu równań normalnych który wykorzystuje rozkład QR. Równanie można przedstawić w następujący sposób:

$Rx = Q^T b$

Złożoność obliczeniowa takiego algorytmu wynosi około $2n^2(m + n/3)$.

2.4 Uwarunkowanie

Warunkiem poprawności metody równań normalnych jest dodatinia określoność macierzy. Wskaźnik uwarunkowania dany jest wzorem $cond_2(A^T A) = (\frac{\delta_1}{\delta_n})^2$ gdzie δ_1 i δ_n to największa i najmniejsza wartość szczególna macierzy. Dlatego dla źle uwarunkowanych macierzy wskaźnik może osiągać dowolnie duże wartości. W takim przypadku lepiej jest stosować metodę QR z rozkładem wąskim. Powyższe metody działają tylko dla macierzy o pełnym rzędzie, lecz dane w zadaniu macierze spełniają ten warunek.

2.5 Realizacja w programie Matlab

```
%funkcja tworzaca macierze i wektory rozwiazan podane w zadaniu 2
function [A1,A2,b1,b2] = ctmx (n)

A1 = zeros(2*n,n);

   for i=1:2*n
      for j=1:n
         A1(i,j) = 2*(i-j)+1;
         A1(j,j) = 1/6;
         A2(i,j) = 8/(9*(i+j+1));
```

```
    end;
    b1(i,1) = 1 + 0.4*i;
    if (mod(i, 2) == 0)
    b2(i,1) = 4/(3*i);
    end
  end

end

%funkcja obliczajaca lnzk metoda rownan normalnych
function [a] = lznkn (M,b)

  a = (M'*M)\(M'*b);

end

%funkcja zwaracjaza rozwiazanie lnzk metoda QR
function [a] = lnzkqr (M,b)

  [Q,R] = qr(M);
  a = R\(Q'*b); %rozwiazanie ukladu rownan metoda QR

end

%skrypt rozwiazujacy zadanie numer 2
clear;

for i=0:8
  [A1,A2,b1,b2] = ctmx(10*(2^i)); %generowanie macierzy podanych w
     zadaniu
  rn1 = lznkn(A1,b1); %obliczanie wynikow dla obu podpunktow i metod
  rq1 = lnzkqr(A1,b1);
  rn2 = lznkn(A2,b2);
  rq2 = lnzkqr(A2,b2);

  error1(1,i+1) = norm(A1*rn1 - b1); %obliczenie normy residuum

  error1(2,i+1) = norm(A1*rq1 - b1);

  error1(3,i+1) = 10*(2^i); % uzupelnienie wektora rozwiazan o liczbe
     rownan

  error2(1,i+1) = norm(A2*rn2 - b2);

  error2(2,i+1) = norm(A2*rq2 - b2);

  error2(3,i+1) = 10*(2^i);
end


plot(error2(3,:),error1(1,:),"or;uklad rownan normalnych;",
error2(3,:),error1(2,:),"og;metoda QR;" ) % utworzenie wykresu 1
h1 = gcf ();
```

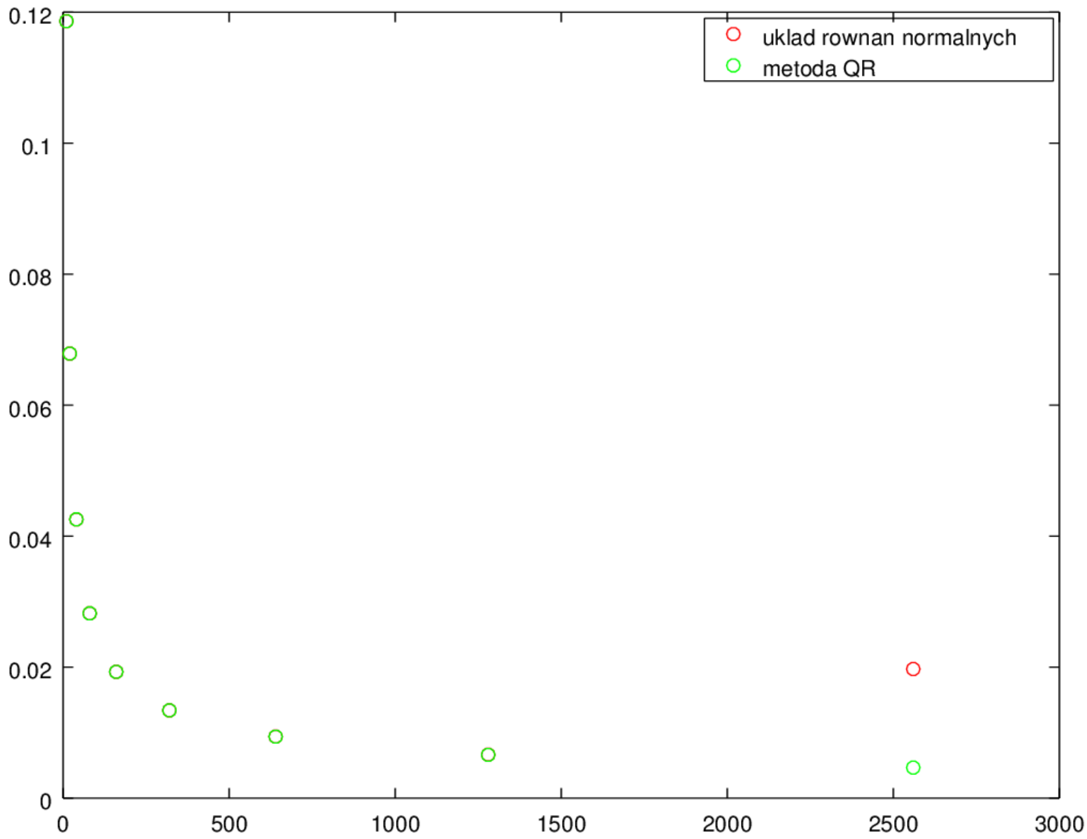
```
saveas(h1, "figure1.png");
clf ();

plot(error2(3,:),error2(1,:),"or;uklad rownan normalnych;",
error2(3,:),error2(2,:),"og;metoda QR;" ) % utworzenie wykresu 2
h2 = gcf();
saveas(h2, "figure2.png");
```

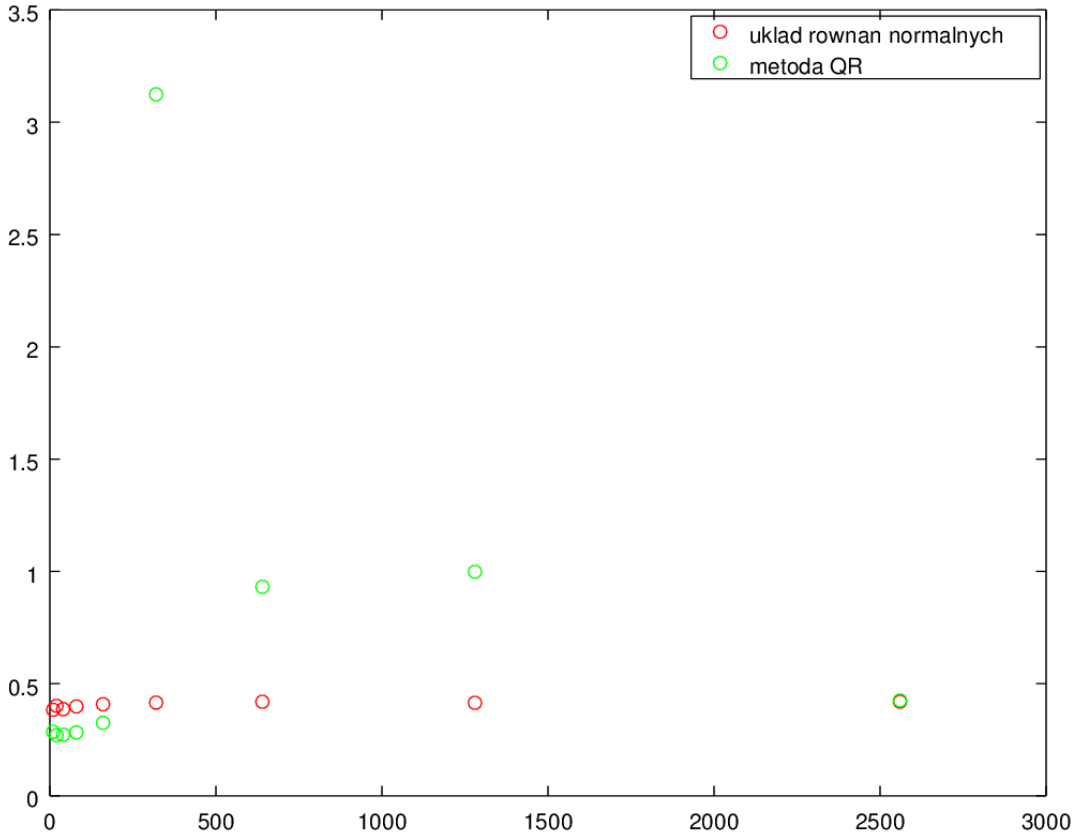
2.6 Wyniki działania programu

Wyniki w postaci wykresów zależności błędu od liczby równań odpowiednio dla podpunktu 1 oraz 2.

Rysunek 1: Podpunkt 1



Rysunek 2: Podpunkt 2



2.7 Wnioski

Jak wynika z wykresu dla podpunktu 1 rozwiązania dla obu metod w większości się pokrywają. Jest to spowodowane dobrym uwarunkowaniem macierzy dla obu metod. Natomiast w podpunkcie 2 metoda QR dokładniej wyznacza rozwiązania jednak tylko dla pewniej liczby równań. Jest to spowodowane złym uwarunkowaniem dla obu algorytmów. Wyniki nie spełniają żadnej znanej zależności, szczególnie w przypadku metody QR ponieważ większa ilość operacji uwydatnia błędy.