Jan Jędrzejewski 310 713
14.04.2023r. - grupa 103

MNUM - PROJEKT 1

zadanie 1.2

Zadanie 1

Solwer rozwiązujący układ równań liniowych $Ax = b$. Metoda faktoryzacji: LDL^T

Dane

Funkcja do generowania danych

```
function [A, b] = data1(n)
   A = zeros(n,n);
   b = zeros(n,1);
   for i=1:n
       for j=1:n
           A(i,j) = i + j + 1;
       end
   end
   for i=1:n
       A(i,i) = 3*n^2 + (1.5*i+2)*n;
       b(i,1) = 2.5+0.6*i;
   end
end
```

Wygenerowane przykładowe dane dla n=5

A=

```
   92.5000    4.0000    5.0000    6.0000    7.0000
    4.0000  100.0000    6.0000    7.0000    8.0000
    5.0000    6.0000  107.5000    8.0000    9.0000
    6.0000    7.0000    8.0000  115.0000   10.0000
    7.0000    8.0000    9.0000   10.0000  122.5000
```

b=

```
    3.1000
    3.7000
    4.3000
    4.9000
    5.5000
```

Metoda LDL^T

Zadanie: rozwiązać układ równań liniowych $Ax = b$

$$Ax = LDL^Tx = L(DL^Tx) = b$$
$$DL^Tx = y$$
$$Ly = b$$

1. Faktoryzacja $A = L * DL^T$
2. Rozwiązanie równania $Ly = b$ dla niewiadomej y
3. Rozwiązanie równania $DL^Tx = y$ dla niewiadomej x

Slower

```
function x = solverLDLt(A,b)
   % A = LDL' decomposition
   [L, D] = LDLt(A);
   % solve Ly = b for y (lower triangular matrix)
   y = solveY(L, b);
   % solve DL'x = y for x (upper triangular matrix)
   x = solveX(D*L', y);
end
```

Faktoryzacja macierzy A na macierze L i D

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdot & \cdot & \cdot & \cdot \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & & & 0 \\ \bar{l}_{21} & 1 & & \\ \vdots & & \ddots & \\ \bar{l}_{n1} & \bar{l}_{n2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} d_{11} & d_{11}\bar{l}_{21} & \cdots & d_{11}\bar{l}_{n1} \\ 0 & d_{22} & & d_{22}\bar{l}_{n2} \\ & & \ddots & \\ 0 & 0 & \cdots & d_{nn} \end{bmatrix}$$

Algorytm wyznaczania elementów macierzy

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

$$l_{ji} = (a_{ji} - \sum_{k=1}^{i-1} l_{jk} * d_{kk} * l_{ik}) / d_{ii}$$

```
function [L, D] = LDLt(A)
   [n, ~] = size(A);
   L = zeros(n);
   D = zeros(n);
   for i = 1:n
       % calculate values of the matrix D
       D(i,i) = A(i,i);
       for k = 1:i-1
           D(i,i) = D(i,i) - L(i,k) ^ 2 * D(k,k);
       end
       % calculate values of the matrix L
       for j = i+1:n
           L(j,i) = A(j, i);
           for k = 1:i-1
               L(j,i) = L(j,i) - (L(j,k) * D(k,k) * L(i,k));
           end
           L(j,i) = L(j,i)/D(i,i);
       end
       L(i,i) = 1;
   end
end
```

Rozwiązanie układu równań z macierzą trójkątną dolną

Kolejno rozwiązujemy równania od góry z jedną niewiadomą. Wyznaczamy wartość i przy rozwiązywaniu równania w poniższym wierszu podstawiamy wartość i znowu rozwiązujemy równanie z jedną niewiadomą. Algorytm:

$$x_1 = \frac{b_1}{a_{11}} \qquad x_k = \frac{b_k - \sum_{j=1}^{k-1} a_{kj} x_k}{a_{kk}}$$

```
function x = solveY(A,b)
   % linear equation solver with lower triangular matrix
   [n, ~] = size(A);
   x = zeros(n,1);
   for k = 1:n
       x(k) = b(k);
       for j = 1:k-1
           x(k) = x(k) - A(k,j) * x(j);
       end
       x(k) = x(k)/A(k,k);
   end
end
```

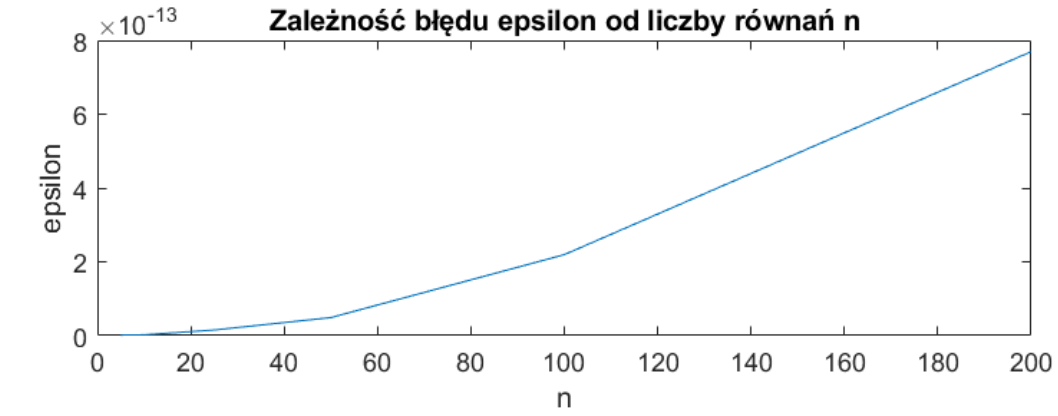
Rozwiązanie układu równań z macierzą trójkątną górną

Analogiczna metoda jak przy macierzy trójkątnej dolnej, tylko rozwiązywanie równań należy zacząć od dołu. Wyznaczamy niewiadome po koeli od tyłu. Algorytm:

$$x_n = \frac{b_n}{a_{nn}} \qquad x_k = \frac{b_k - \sum_{j=k+1}^{n} a_{kj} x_k}{a_{kk}}$$

```
function x = solveX(A,b)
   % linear equation solver with upper triangular matrix
   [n, ~] = size(A);
   x = zeros(n,1);
   for k = n:-1:1
       x(k) = b(k);
       for j = k+1:n
           x(k) = x(k) - A(k,j) * x(j);
       end
       x(k) = x(k)/A(k,k);
   end
end
```

Wyniki



Wnioski

Zarówno błąd $\epsilon = \left|\left|A\hat{x} - b\right|\right|_2$, jak i czas obliczeń wydaje się rosnąć liniowo od liczby równań n. Na wykresie czasu obserwuję nietypowy wzrost dla niewielkich rozmiarów układów równań $(n = 5, 10)$. Może być on spowodowany na przykład potrzebą załadowania funkcji przy pierwszym użyciu. Na wykresie $\epsilon(n)$ poziom błędu dla równań z większą liczbą niewiadomych jest niski, solwer poprawnie znajduje rozwiązania równania.

Zadanie 2

Porównanie solwera z zadania 1 z danym solwerem GS dla macierzy A i wektora b danych wzorami:

$a_{ii} = -20 \quad a_{ij} = 7.5, j = i \pm 3 \quad a_{ij} = 0$ - dla pozostałych

$b_i = 2.5 + 0.6i$

Dane

Funkcja do generowania danych

```
function [A, b] = data2(n)
   A = zeros(n, n);
   b = zeros(n, 1);
   for i=1:n
       for j=1:n
           if abs(i-j) == 3
               A(i,j) = 7.5;
           end
       end
   end
   for i=1:n
       A(i,i) = -20;
       b(i,1) = 4.0-0.5*i;
   end
end
```

Wygenerowane przykładowe dane dla n=5

A= b=

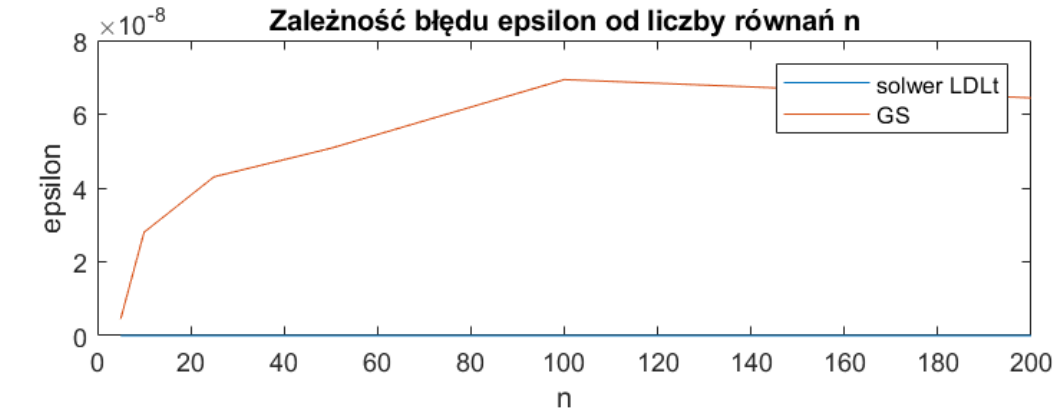
```
  -20.0000         0         0    7.5000         0         3.5000
         0  -20.0000         0         0    7.5000         3.0000
         0         0  -20.0000         0         0         2.5000
    7.5000         0         0  -20.0000         0         2.0000
         0    7.5000         0         0  -20.0000         1.5000
```

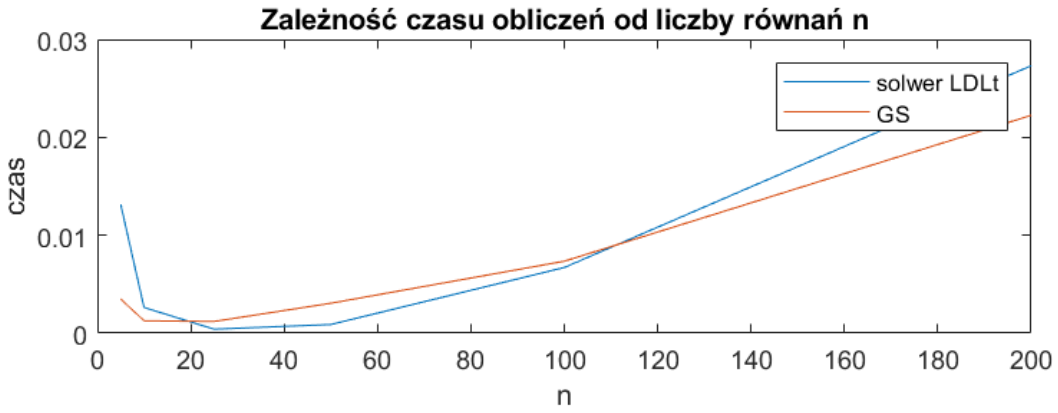
Parametry metody Gaussa-Seidela

$itmax = 1000 * n$

$delta = 1e - 8$

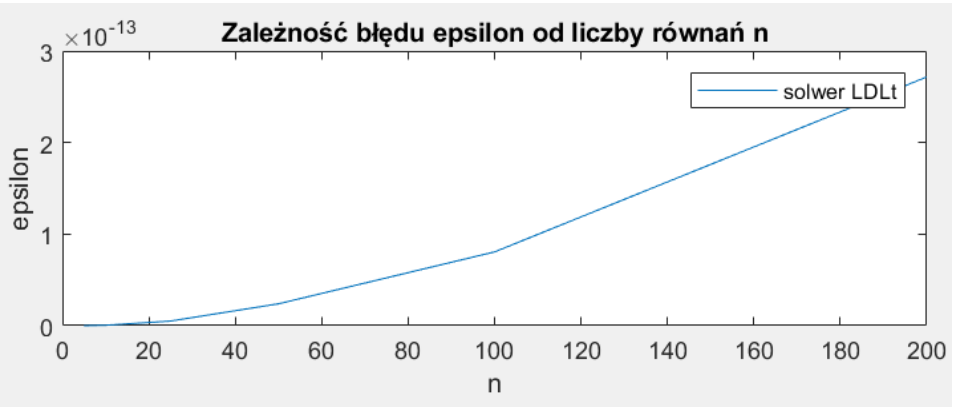
Wyniki





Wnioski

Błąd ε dla porównywanych solwerów na danych z zadania 2 jest na zupełnie różnym poziomie. Dla solwera GS od 10^{-8} do 10^{-7}. Natomiast dla solwera z poprzedniego zadania błąd jest na poziomie 10^{-13} (co słabo widać na powyższym wykresie). Jeśli chodzi o czas rozwiązania równania, to są porównywalne. Liniowo zależą od liczby równań. W obydwu przypadkach zaobserwowałem anomalie dla małej liczby równań (tak jak w poprzednim zadaniu). Dla większej liczby równań solwer GS szybciej oblicza rozwiązania.



Zadanie 3

Wyznaczanie funkcji aproksymującej dane metodą najmniejszych kwadratów przy użyciu:

A. solwera z zadania 1 (rozkładem LDL')
B. solwera GS
C. rozkładu SVD

Dane

Bezpośrednio z tabeli w zadaniu

```
X = [-10:2:10];
Y = [-5.460 -3.880 -1.969 -1.666 -0.076 -0.397 -1.030 -4.548 -11.528 -21.641 -34.445];
N = [3 5 7 9 10]; (stopień wielomianu)
```

Aproksymacja

Zadanie aproksymacji: wyznaczyć wartości współczynników a0, a1, a2, …., an określających funkcję aproksymującą aby zminimalizować błąd średniokwadratowy.

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

Możemy zdefiniować macierz A postaci

$$\mathbf{A} = \begin{bmatrix} \phi_0(x_0) & \phi_1(x_0) & \cdots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_n(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \cdots & \phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \cdots & \phi_n(x_N) \end{bmatrix}$$

która w przypadku aproksymacji funkcją wielomianową będzie wyglądała w następujący sposób:

$$A = \begin{bmatrix} x_0^0 & x_0^1 & \dots & x_0^n \\ x_1^0 & x_1^1 & \dots & x_1^n \\ x_2^0 & x_2^1 & \dots & x_2^n \\ \dots & \dots & \dots & \dots \\ x_N^0 & x_N^1 & \dots & x_N^n \end{bmatrix}$$

```
function A = calculateMatrixA(x,n)
   A = zeros(length(x), n+1);
   for i = 1:length(x)
       for j = 0:n
           A(i,j+1) = x(i)^j;
       end
   end
end
```

Wówczas zadanie aproksymacji możemy zapisać w prostszej postaci

$$H(a) = (||y - A * a||_2)^2$$

Za pomocą metody liniowego zadania najmniejszych kwadratów (LZNK) wystarczy rozwiązać powyższy układ równań normalnych dla wektora współczynników a w następujący sposób

$$A^T * A * a = A^T * y$$

Aproksymacja solwerem LDLt

```
function a = approxLDLt(x,y,n)
   % calculate matrix A
   A = calculateMatrixA(x,n);
   % solve linear equation A'Aa = A'y
   a = solverLDLt2(A' * A, A' * y');
end
```

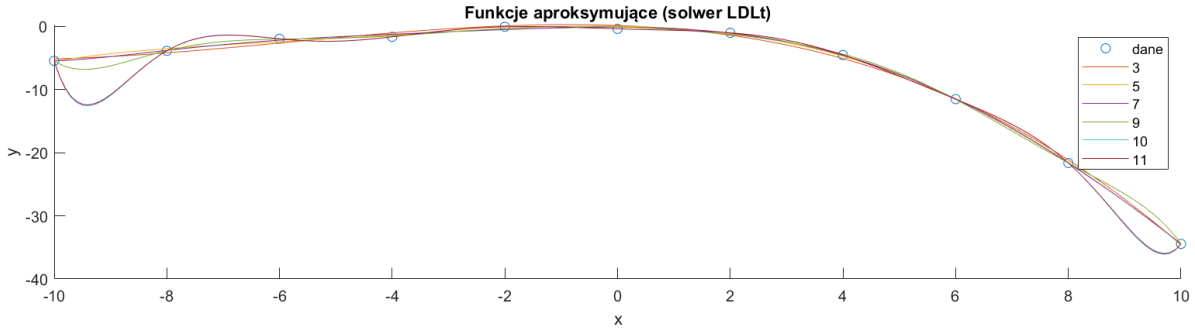
Aproksymacja solwerem GS

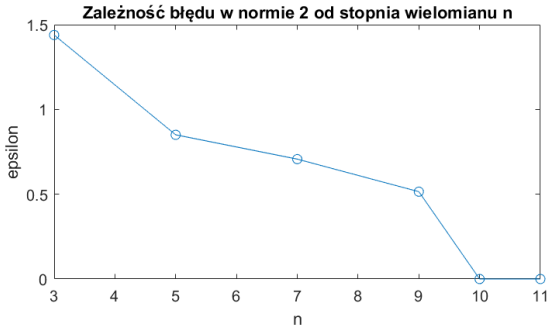
```
function a = approxGS(x,y,n,delta,itmax)
   % calculate matrix A
   A = calculateMatrixA(x,n);
   % solve linear equation A'Aa = A'y
   [a, ~, ~] = GS(A' * A, A' * y', delta, itmax*n);
end
```

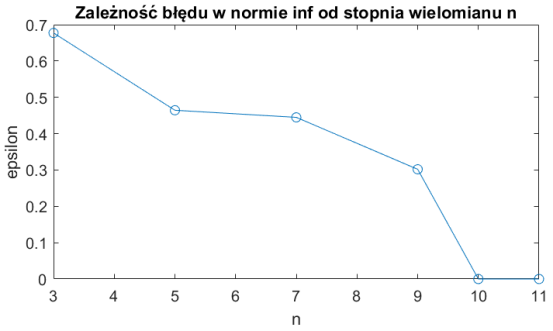
Aproksymacja rozkładem SVD

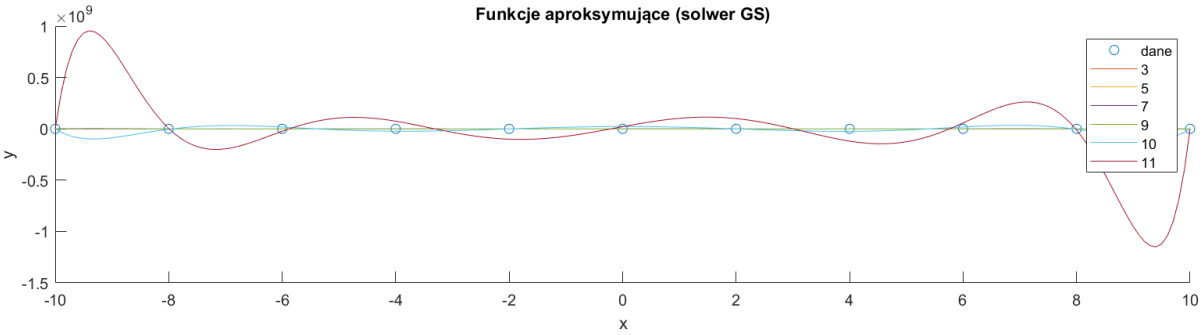
```
function a = approxSVD(x,y,n)
   % calculate matrix A
   A = calculateMatrixA(x,n);
   % decomposition of matrix A by SVD
   [U, S, V] = svd(A);
   % solve for a using SVD
   s = diag(S);
   k = rank(A);
   y_ = U' * y';
   a_ = [y_(1:k, 1) ./ s(1:k, 1); zeros(n-k,1)];
   a = V * a_;
end
```

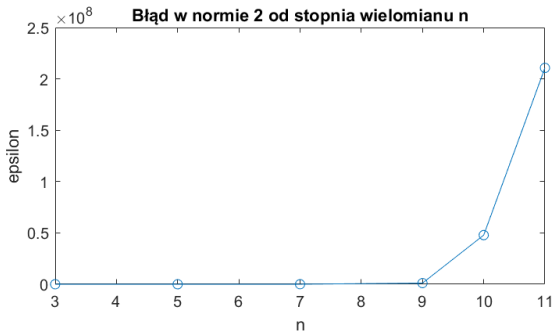
Wyniki

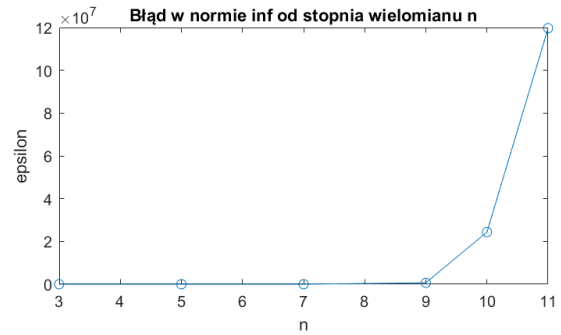

Funkcje aproksymujące (solwer LDLt)
dane
3
5
7
9
10
11
y
x

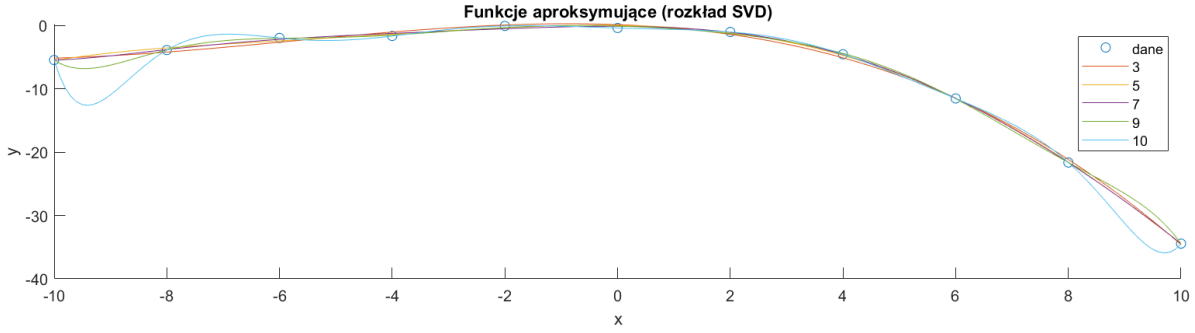

Zależność błędu w normie 2 od stopnia wielomianu n
epsilon
n

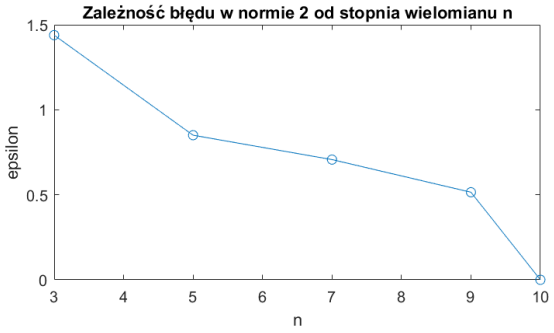

Zależność błędu w normie inf od stopnia wielomianu n
epsilon
n

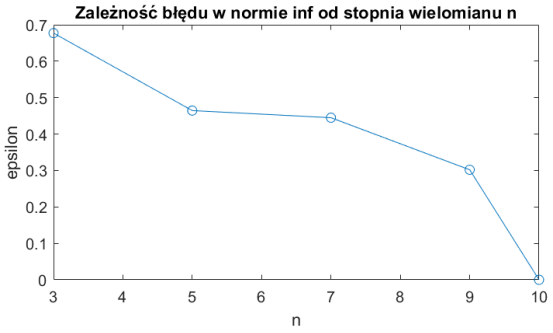

Funkcje aproksymujące (solwer GS)
×10^9
dane
3
5
7
9
10
11
y
x


Błąd w normie 2 od stopnia wielomianu n
×10^8
epsilon
n


Błąd w normie inf od stopnia wielomianu n
×10^7
epsilon
n







Wnioski

Aproksymacja solwerem LDLt oraz metodą SVD daje dobre przybliżenie i niskie błędy. Przy wyższych stopniach wielomianów funkcja aproksymująca się zbyt dopasowuje do punktów. Aproksymacja przy użyciu solwera GS daje zadowalające rezultaty dla niewielkich stopni wielomianów. Przy stopniu większym od 5 błędy i niedopasowania wykładniczo wzrastają.