Mariusz Paluch

Sprawozdanie projekt nr 1 MNUM

1. Układy równań
   - Układ równań A:

$$a_{i,j}=\begin{cases} -10, dla\ j=i \\ 3.5+\frac{j}{n}, dla\ j=i-3\ lub\ j=i+3, \\ 0, w\ p.\ p. \end{cases} \quad for\ i=1:n;\ j=1:n$$

$$b_i=0.5+2.5i$$

   - Układ równań B:

$$a_{i,j}=4(i-j)+2, j\neq i; a_{i,i}=\frac{1}{3};$$

$$b_i=3.5-0.4i$$

2. Rozwiązanie układów równań liniowych z użyciem eliminacji Gaussa z częściowym wyborem elementu głównego.

Eliminacja Gaussa wykorzystuje rozkład LU macierzy. Wykorzystanie częściowego wyboru elementu głównego zapobiega sytuacji, gdy na przekątnej macierzy znajduje się element równym zeru, co skutkuje dzieleniem przez 0.

Wybór elementu głównego dla macierzy $A^{(k)}$, gdzie k to k-ty krok algorytmu, spośród elementów $a_{jp}^{(k)}(j\geq k, p\geq k)$ wygląda następująco:

$$|a^{(k)}|=\max_j\left\{\left|a_{jk}^{(k)}\right|, j=k, k+1, \ldots, n\right\}$$

Metoda ta jako produkt swojego działania, zwraca macierze ***L***, ***U***, ***P***.

Macierz **P** jest macierzą przekształceń, która mówi nam, jak zmieniła się kolejność wierszy podczas obliczeń. Macierz **P** ma charakterystyczne własności:

$$det(P)=1$$
$$P^{-1}=P$$

Dla układu równań liniowych **Ax** = **b**, spełnione jest poniższe równanie:

$$\boldsymbol{LU} = \boldsymbol{PA}$$

Metoda ta sprawdza się dobrze do obu zadanych równań. Dla równania A algorytm nie musiał wykonywać częściowego wyboru elementu głównego, ponieważ na diagonali zawsze znajdowały się elementy o największym module. W takiej sytuacji macierz **P** jest macierzą jednostkową. W przypadku równania B, algorytm za każdym razem musiał wybierać element główny.

Pseudokod algorytmu gaussa z częściowym wyborem wygląda następująco:
Skoro A = LU, to:

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ l_{2,1} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \dots & 1 \end{pmatrix} . \begin{pmatrix} u_{1,1} & u_{1,2} & \dots & u_{1,n} \\ 0 & u_{2,2} & \dots & u_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{n,n} \end{pmatrix}$$

$$P^{(1)} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

$P = P^{(1)}$

dla każdego $k = 1 \dots n-1$:

$A^{(k)} = PA$

$m = \underset{j}{\operatorname{argmax}} \left\{ \left| a_{j,k}^{(k)} \right|, j = k+1, \dots, n \right\}$

Jeśli $m \neq k$

$$P^{(k)} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

$P_{m,k}^{(k)} = 1$ *oraz* $P_{k,m}^{(k)} = 1$

$P_{m,m}^{(k)} = 0$ *oraz* $P_{k,k}^{(k)} = 0$

$L := P^{(k)} L$

$U := P^{(k)} U$

$P := P^{(k)} P$

W przeciwnym wypadku

Dla $j = k+1 \dots n$

$$L_{j,k} = \frac{U_{j,k}}{U_{k,k}}$$

$U_j = U_j - L_{j,k} * U_k$

Aby znaleźć interesujące nas rozwiązanie, musimy znaleźć rozwiązanie układu równań:

$$LUx = Pb$$

Aby znaleźć rozwiązanie rozbijamy te równanie na dwa kolejne i re rozwiązujemy:

$$Ly = Pb$$
$$Ux = y$$

Kod algorytmu:

```
function [L,U,P] = partialPivoting(A, n)
L = zeros(n,n);
U = A;
P = diag(ones(n,1));
for k=1:n-1
   [~,i] = max(abs(U(k:n,k)));
   if (i ~= 1)
      max_idx = i+k-1;
      P_temp = diag(ones(n,1));
      P_temp(sub2ind(size(P_temp),[max_idx k],[k max_idx])) = 1;
      P_temp(sub2ind(size(P_temp),[max_idx k],[max_idx k])) = 0;
      L = P_temp*L;
      U = P_temp*U;
      P = P_temp*P;
   end
   L(k+1 : n, k) = U(k+1 : n, k) / U(k,k);
   U(k+1:n, : ) = U(k+1:n, : ) - L(k+1 : n, k) * U(k, :);
end
L = L + diag(ones(n,1));
end
```

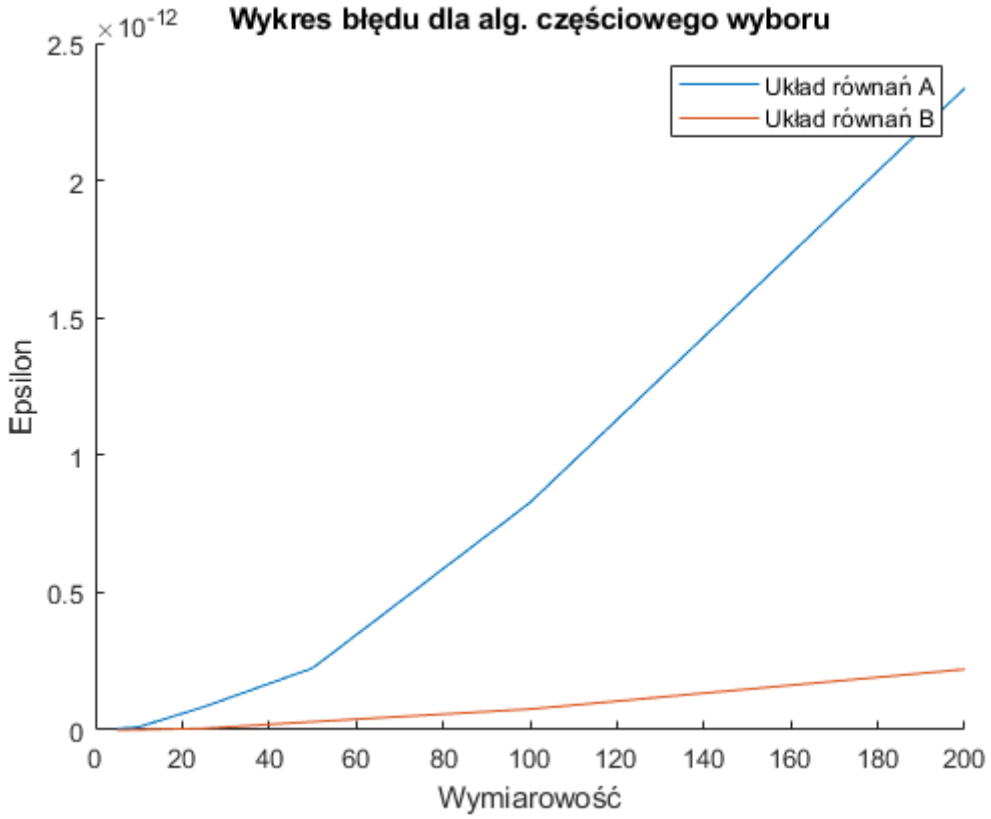
Kod solvera:

```
function [x] = equationSolver(A, b, n)
[L, U, P] = partialPivoting(A,n);
y = zeros(n, 1);
b2 = P*b;
for k=1:n
   y(k) = (b2(k) - L(k, 1:k-1)*y(1:k-1)) / L(k,k);
end

x = zeros(n,1);
```

```
for k=n:-1:1
    x(k) = (y(k) - U(k, k+1:n) * x(k+1:n)) / U(k,k);
end
```

Wykres dla błędu tej metody:



3. Rozwiązywanie układów równań liniowych z użyciem metody iteracyjnej Jacobiego.

Jako kryterium stopu, zgodnie z treścią zadania, przyjmuję błąd rozwiązania mniejszy od 10^{-8}.

Metoda Jacobiego wykorzystuje rozkład macierzy wejściowej na macierze ***L, D, U***. Macierz L składa się z elementów macierzy pierwotnej znajdującej się pod diagonalą, D to elementy znajdujące się na diagonali, a macierz U zawiera elementy znajdujące się nad pierwotną diagonalą.

Macierze ***L,D,U*** otrzymuję poprzez odpowiednie indeksowanie macierzy wejściowej.
Układ równań Ax = b można zapisać jako:

$$Ax = (L+D+U)x = b$$
$$Dx = -(L+U)x + b$$

Stąd łatwo już zaproponować metodę iteracyjną:

$$Dx^{(i+1)} = -(L+U)x^{(i)} + b$$
$$x^{(i+1)} = -D^{-1}(L+U)x^{(i)} + D^{-1}b$$

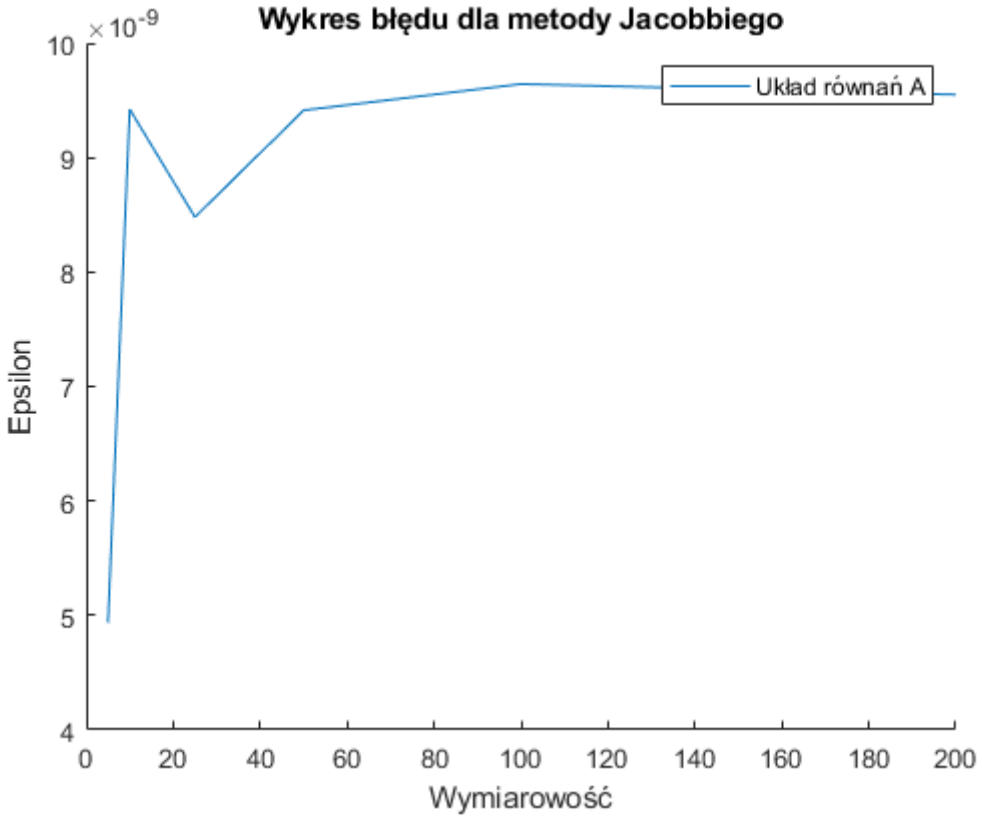
Warunkiem dostatecznym zbieżności dla metody Jacobiego jest silna diagonalna dominacja macierzy A, wierszowa bądź kolumnowa.

Ponieważ macierz A dla układu równań w wersji B, nie posiada silnej dominacji diagonalnej to metoda ta nie sprawdzi się dla niej. Dlatego też nie będę badał jej umieszczał w programie.

Kod algorytmu wygląda następująco:

```
function x = jacobi(A, b, n, prec)
x = zeros(n,1);
x_old = zeros(n,1);
stop = false;
while ~stop
    for i=1:n
        x(i) = ( b(i) - A(i,1:i-1) * x_old(1:i-1) - A(i,i+1:n) * x_old(i+1:n)) /
A(i,i);
    end
    x_old = x;
    stop = norm(A*x_old - b) < prec;
end
```

Wykres błędu dla równania A:



Tak jak można było się spodziewać, algorytm zatrzymał się w okolicach kryterium stopu.

Porównanie średnich czasów działania algorytmów Jacobiego i Gaussa z wyborem czasowym dla równania A:

Badany wymiar	Gauss z cz. wyborem	Jacobi
5	0,00097	0.00156
10	0.00079	0.00525
25	0.00167	0.02352
50	0.00545	0.03606
100	0.00669	0.12464
200	0.03839	0.28341

Jak widać metoda Gaussa z częściowym wyborem ma znacząco lepsze wyniki niż metoda Jacobiego. Wynik może być niemiarodajny, ponieważ dla układów równań A metoda Gaussa nie musi odwracać wierszy, przez co nie musi wykonywać jednej z najbardziej kosztownych operacji.

Możliwym też jest, że kopiowanie macie w metodzie Jacobiego znacząco wpływa na wydajność.

4. Dane potrzebne do aproksymacji funkcji:

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

5. Aproksymacja z użyciem równań normalnych w zadaniu najmniejszych kwadratów

Jako parametry wejściowe przyjmuję wektory kolumnowe $X, Y \in \mathbb{R}^N$.
W dalszych przypuszczeniach przyjmuję następujące oznaczenia:

- n – wymiar wielomianu
- $A = [A^0 A^1 \dots A^n]$
- a – wektor współczynników wielomianów o wymiarze n+1

Funkcja celu zadania aproksymacji możemy zapisać w postaci:

$$H(a) = (\|Y - Aa\|_2)^2$$

Gdzie H definiujemy także jako:

$$H(a) = \sum_{j=1}^{N} [Y_j - \sum_{i=0}^{n} a_i X_j^i]^2$$

Celem zadania aproksymacji jest wyznaczenie wektora a tak, aby zminimalizować błąd średniokwadratowy.

Aby znaleźć konieczne minimum liczymy pochodną dla każdego współczynnika i przyrównujemy ją do zera.

$$Dla\,każdego\,k=0,\dots n$$
$$\frac{\partial H}{\partial a_k}=-2\sum_{j=0}^{N}[Y_j-\sum_{i=0}^{n}a_i X_j^i]*X_j^i=0$$
$$a_0\sum_{j=0}^{N}X_j^{0+k}+a_1\sum_{j=0}^{N}X_j^{1+k}+\dots+a_n\sum_{j=0}^{N}X_j^{n+k}=\sum_{j=0}^{N}Y_j X_j^i$$

Zgodnie z notacją z wykładów, wprowadzam oznaczenia pomocnicze:

$$g_{ik}=\sum_{j=0}^{N}X_j^{i+k}$$
$$\rho_k=\sum_{j=0}^{N}Y_j X_j^k$$

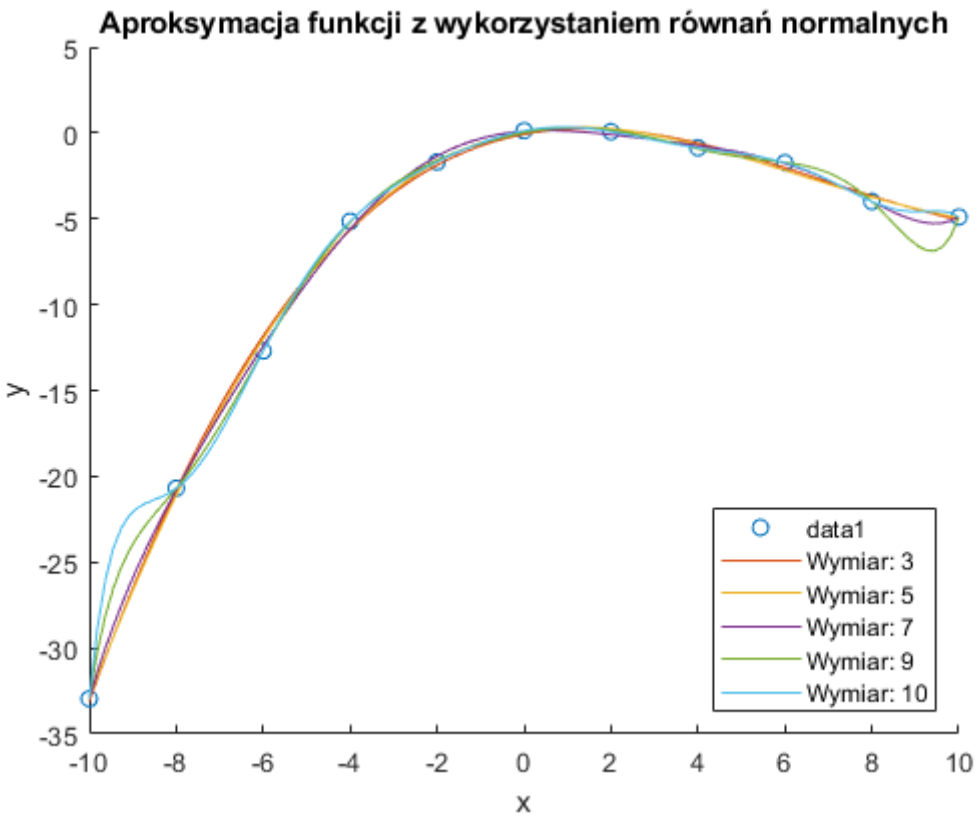
Uzyskujemy dzięki temu układ równań normalnych:

$$\begin{aligned}&a_0 g_{00}+a_1 g_{10}+\dots a_n g_{n0}=\rho_0\\&a_0 g_{01}+a_1 g_{11}+\dots a_n g_{n1}=\rho_1\\&\dots\dots\dots\dots\dots\dots\dots\dots\quad\equiv\quad Ga=\rho\\&a_0 g_{0n}+a_1 g_{1n}+\dots a_n g_{nn}=\rho_n\end{aligned}$$

Kod algorytmu wygląda następująco:

```
function [a] = polinomialAprox(x,y,n)
G = zeros(n+1,n+1);
r = zeros(n+1,1);
for k = 0:n
   for i = 0:n
      G(k+1,i+1) = sum(x.^(k+i));
   end
   r(k+1) = sum(y'*(x.^(k)));
end
   a = G^(-1)* r;
end
```

Wykres przedstawiający aproksymację funkcji wielomianowej przy użyciu równań normalnych:



6. Aproksymacja z użyciem rozkładu QR w zadaniu najmniejszych kwadratów.

Aby rozwiązać zadanie najmniejszych kwadratów, musimy rozwiązać następujący problem:

$$\min_a \|Aa - y\|_2$$

Rozkład QR ma kilka przydatnych właściwości:

1. $A = QR$
2. $\|Qx\| = \|x\| = \|Q^T x\|$
3. $Q^T Q = I$

Korzystamy z ww. wzorów, aby przekształcić problem do użytecznej formy:

$$\min_a \|Aa-y\|_2 = \min_a \|QRa-y\|_2 = \min_a \|Ra-Q^T y\|_2 = \min_a \|Ra-\tilde{y}\|_2$$

Stąd wynika, że:

$$Ra-\tilde{y}=0$$
$$Ra=\tilde{y}$$

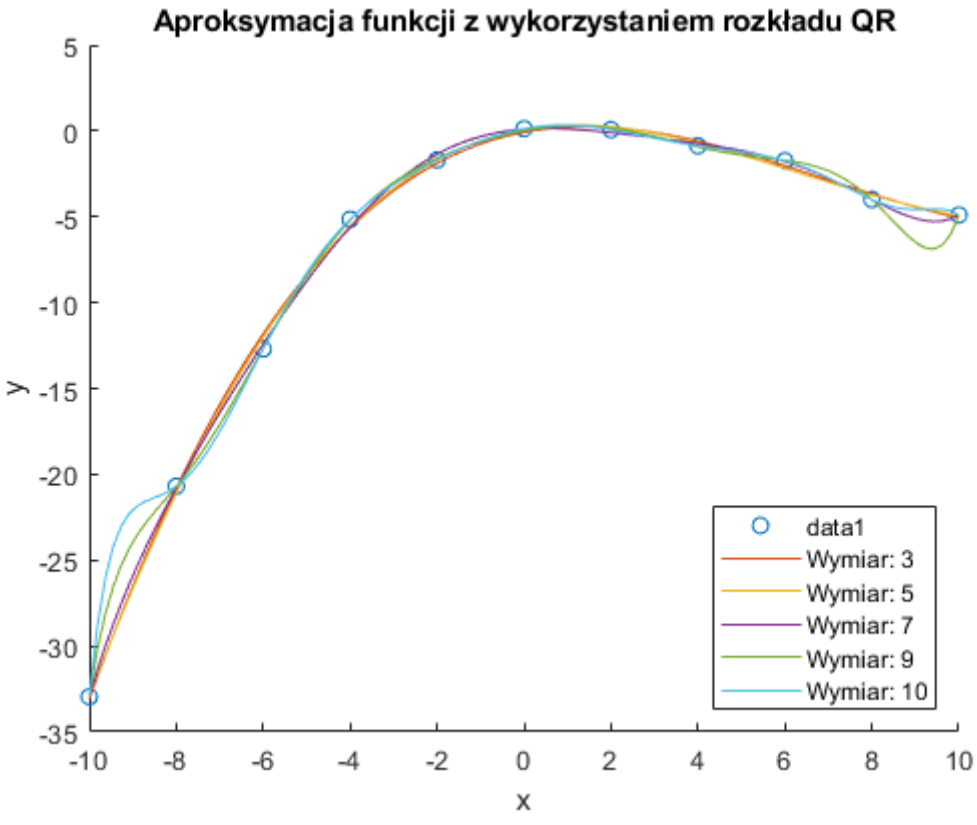
Stąd już łatwo wyznaczyć wektor współczynników wielomianów:

$$a=R^{-1}\tilde{y}$$

Kod algorytmu wygląda następująco:

```
function [a] = aproxQR(x,y,n)
    A = zeros(size(x, 1),n+1);
    for i=1:n+1
            A(:,i) = x.^(i-1);
    end
    [Q,R] = qr(A);
    y_tilde = Q'*y;
    a = linsolve(R,y_tilde);
end
```

Wykres przedstawiający aproksymację funkcji wielomianowej przy użyciu rozkładu QR:



7.Błędy aproksymacji obliczane z wykorzystaniem norm.

Norma 2:

Wymiarowość	Równania normalne	Rozkład QR
3	1.20879260687413	1.20879260687413
5	1.08988994867973	1.08988994867973
7	0.67075925557438	0.670759255574389
9	0.20699689929498	0.206996899294985
10	1.464741781287e-08	2.58288556773014e-13

Norma nieskończoność:

Wymiarowość	Równania normalne	Rozkład QR
3	0.884303030303037	0.884303030303032
5	0.69909207459223	0.699092074592077
7	0.440645660222072	0.440645660222131
9	0.121357076361372	0.121357076360176
10	1.03090513903226e-08	1.49213974509621e-13

8. Komentarz do wyników.

Rezultat działania obu algorytmów jest bardzo podobny. Dla obu algorytmów błąd okazał się prawie identyczny, przy czym rozkład QR okazał się nieznacznie lepszy.

Wraz ze wzrostem wymiarowości wielomianu, maleje błąd aproksymacji. Szczególnie duży spadek można zauważyć przy przechodzeniu z 9 na 10 wymiar wielomianu.

Gdy spojrzymy na wykres przybliżonego wielomianu o rozmiarze 10, zauważymy, że aproksymowana funkcja stara się jak najbardziej wpasować w dane testowe na podstawie których obliczany jest błąd, jednocześnie radzi sobie gorzej w pozostałych częściach wykresu.