Szczecin, 18 marca 2022

dr Sławomir Marczyński slawek@zut.edu.pl

Aproksymacja szeregiem Fouriera

Opiszę tu metodę nieco przestarzałą, bo nie wykorzystującą algorytmu FFT, aproksymacji danych numerycznych za pomocą szeregu Fouriera. Zakładamy że dane to dwa wektory, jeden $\mathbf{x}$, drugi $\mathbf{y}$, każdy mający n elementów. Chcemy znaleźć współczynniki a_k oraz b_k rozwinięcia w skończony szereg Fouriera (przyjmujemy że $a_0 = 0$):

$$\begin{aligned} y = f(x) &= \sum_{k=0}^{m} [a_k \sin(kx) + b_k \cos(kx)] \\ &= b_1 + a_1 \sin(x) + b_1 \cos(x) + a_2 \sin(2x) + b_2 \cos(2x) + a_3 \sin(3x) + b_3 \cos(3x) \\ &\quad + \cdots + a_m \sin(mx) + b_m \cos(mx) \end{aligned}$$

Jeżeli podstawimy konkretne wartości x i y to otrzymamy układ n równań liniowych dla $2m-1$ niewiadomych, które w postaci macierzowej możemy zapisać jako $\mathbf{W}\mathbf{q} = \mathbf{y}$, gdzie macierz $\mathbf{W}$ jest taka jak poniżej:

$$\mathbf{W} = \begin{pmatrix} 1 & \sin(x_1) & \cos(x_1) & \sin(2x_1) & \cos(2x_1) & \cdots & \cos(mx_1) \\ 1 & \sin(x_2) & \cos(x_2) & \sin(2x_2) & \cos(2x_2) & \cdots & \cos(mx_2) \\ 1 & \sin(x_3) & \cos(x_3) & \sin(2x_3) & \cos(2x_3) & \cdots & \cos(mx_3) \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \sin(x_n) & \cos(x_n) & \sin(2x_n) & \cos(2x_n) & \cdots & \cos(mx_n) \end{pmatrix}$$

natomiast wektor $\mathbf{q}$ zawiera szukane współczynniki:

$$\mathbf{q} = \begin{pmatrix} b_1 \\ a_2 \\ b_2 \\ \cdots \\ a_m \\ b_m \end{pmatrix}$$

Jak widzimy wystarczy skonstruować macierz $\mathbf{W}$ i rozwiązać układ równań liniowych a będziemy mieli współczynniki rozwinięcia w szereg Fouriera. Nie jest specjalnym wyzwaniem jeżeli mamy np. Matlaba (lub NumPy), wątpliwość może budzić jedynie co robić gdy n i $(2m-1)$ są różne (czyli nie mamy dokładnie tyle równań ile niewiadomych). Jeżeli $n > (2m-1)$ to Matlab, gdy używamy dzielenia macierzowego **W \ y** (dzielimy wektor **y** przez macierz **W** od-prawej-do-lewej), jako wynik dostarczy optymalne wartości **q**, tj. takie by zminimalizować sumę kwadratów odchyleń. Tak samo będzie gdy użyjemy metody Banachiewicza do rozwiązania układu nadokreślonego - co samo w sobie jest ciekawe - ale wymaga trochę wysiłku umysłowego nie przekładającego się na jakiś konkretny zysk. Jak się uprzeć to można policzyć **q = pinv(W) * y**, choć pinv w Matlabie jako funkcja liczy się zdecydowanie dłużej niż dzielenie macierzowe. Gdy $n < (2m-1)$ to dobrym pomysłem mogłoby być jednak użycie pseudoodwrotności macierzy C, czyli obliczanie **q = pinv(W) * y**, zamiast

po prostu **q = W \ y**, bo pseudoodwrotność da jako (jedno z możliwych) rozwiązanie minimalizujące sumę kwadratów samych współczynników b_1, a_2 , b_2, …, a_m, b_m co wydaje się korzystne.

Co możemy mając już współczynniki b_1, a_2 , b_2, …, a_m, b_m? Po pierwsze możemy wyliczyć wartości szeregu Fouriera w zakresie w jakim były zapodane oryginalne dane (aproksymacja). Po drugie możemy wyliczyć wartości spoza tego przedziału - przewidywać przyszłe lub otworzyć przeszłe (nie zmierzone).

Problemem jest że wybierając jako bazę $\sin(kx)$, $\cos(kx)$ itd. nijak nie zastanawialiśmy się nad tym czy dane reprezentowały cokolwiek okresowego o okresie 2π. Zauważmy, że jeżeli, na przykład, dane wejściowe x obejmowały zakres od 0 do 4π, ale wartości y nie powtarzały się (nie były okresowe), to obliczone powyższą metodą współczynniki szeregu i tak dadzą funkcję $f(x)$ powtarzającą w przedziale $(2\pi, 4\pi)$ wartości z przedziału $(0, 2\pi)$. Czyli będziemy mieli wymuszoną okresowość i zupełnie złe wyniki. Aby tego uniknąć należy najpierw przeskalować $\mathbf{x}$, tak aby wartości leżały w przedziale od zero do 2π, czyli $0 \leq x_i \leq 2\pi$.

Kolejnym problemem może być różnica wartości $y_n - y_1$, czyli skok pomiędzy y_n a spodziewanym y_{n+1}. Można temu zapobiec usuwając liniowy trend przez dodanie kolumny do macierzy $\mathbf{W}$ tak jak poniżej:

$$\mathbf{W} = \begin{pmatrix} 1 & x_1 & \sin(x_1) & \cos(x_1) & \sin(2x_1) & \cos(2x_1) & \cdots & \cos(mx_1) \\ 1 & x_2 & \sin(x_2) & \cos(x_2) & \sin(2x_2) & \cos(2x_2) & \cdots & \cos(mx_2) \\ 1 & x_3 & \sin(x_3) & \cos(x_3) & \sin(2x_3) & \cos(2x_3) & \cdots & \cos(mx_3) \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_n & \sin(x_n) & \cos(x_n) & \sin(2x_n) & \cos(2x_n) & \cdots & \cos(mx_n) \end{pmatrix}$$

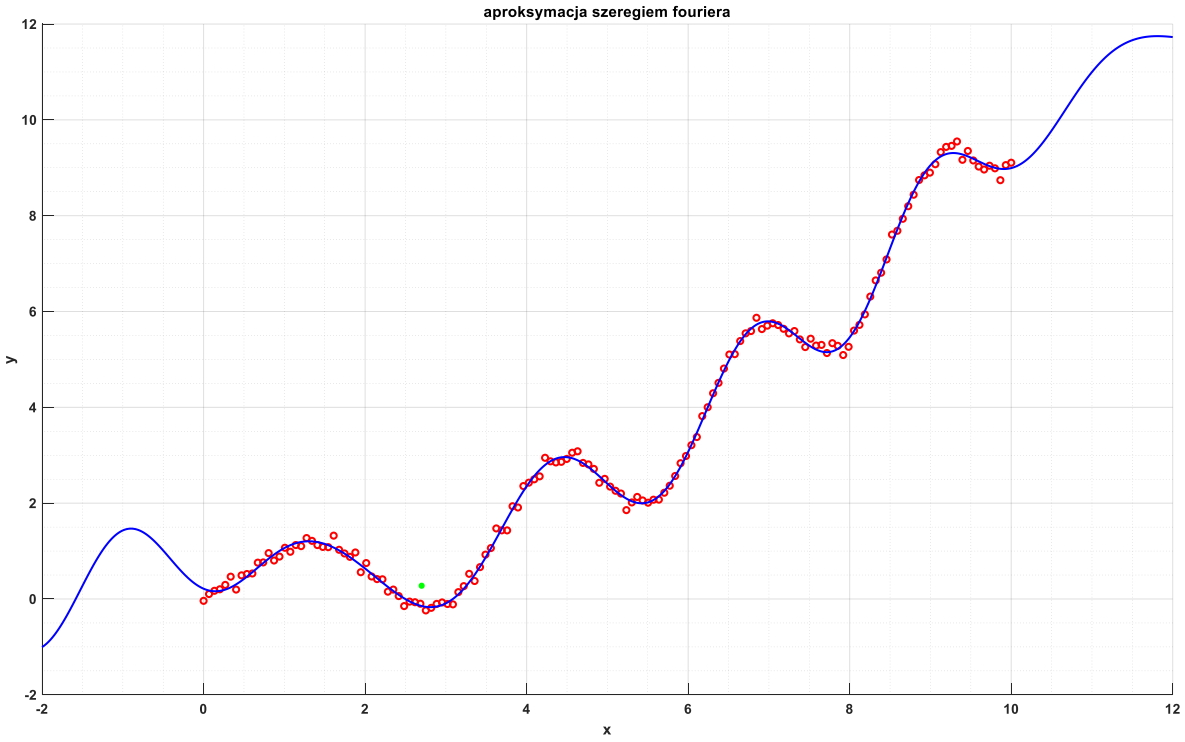
Zauważmy że kolejność kolumn ma niewielkie znaczenie, można macierz W zapisać z inną kolejnością kolumn i z wielomianem wyższego stopnia.

$$\mathbf{W} = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & \sin(x_1) & \sin(2x_1) & \cdots & \cos(x_1) & \cos(2x_1) & \cdots & \cos(mx_1) \\ 1 & x_2 & x_2^2 & \cdots & \sin(x_2) & \sin(2x_2) & \cdots & \cos(x_2) & \cos(2x_2) & \cdots & \cos(mx_2) \\ 1 & x_3 & x_3^2 & \cdots & \sin(x_3) & \sin(2x_3) & \cdots & \cos(x_3) & \cos(2x_3) & \cdots & \cos(mx_3) \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_n & x_n^2 & \cdots & \sin(x_n) & \sin(2x_n) & \cdots & \cos(x_n) & \cos(2x_n) & \cdots & \cos(mx_n) \end{pmatrix}$$

Wektor q będzie wtedy złożony z współczynników wielomianu oraz współczynników przy sinusach i cosinusach: $\mathbf{q}' = (p_1, p_2, p_3, \ldots, p_k, a_1, a_2, a_3, \ldots, a_m, b_1, b_2, b_3, \ldots, b_m)$. Dla procedury obliczeniowej to bez znaczenia, Matlab i tak w czasie rozwiązywania równań przestawi wiersze i kolumny.

Można też zmodyfikować obliczenia tak, aby szukać nie tylko rozkładu na coraz to wyższe składowe harmoniczne, lecz także dodawać składowe sub-harmoniczne. Przy takiej organizacji obliczeń większe m to nie tylko coraz finezyjniej dopasowywana krzywa, ale również coraz szerszy zakres w jakim szukana jest powtarzalność sygnału.

Zaletą bezpośredniego rozwiązywania układu równań nad algorytmem FFT jest możliwość stosowania do nierównomiernie rozmieszczonych wartości x. Wadą, i to niestety poważną, jest że jest znacząco mniejsza wydajność, po prostu obliczenia trwają dłużej niż FFT. To nie jest aż tak straszne gdy ma się w miarę szybki komputer, niezbyt duże n oraz $(2m - 1)$ a obliczenia nie muszą być przeprowadzane w czasie rzeczywistym. Jednak po prostu FFT jest znacznie lepsze.



Rysunek 1. Przykład aproksymacji szeregiem Fouriera.

Listing 1.

```
function [yi, p, a, b] = fsapp(x, y, xi, mh, mp)
% fsapp
%
% Aproksymacja danych przy pomocy szeregu Fouriera.
%
%       yi = fsapp(x, y, xi, mh, mp)
%
% Dane:
%
%       x  -- wektor odciętych
%       y  -- wektor rzędnych
%       xi -- wektor odciętych dla których mają być obliczone wartości yi
%       mh -- ilość harmonicznych (5 oznacza że liczone będą sin(5*x) itd.)
%       mp -- składniki wielomianowe (0 nie, 1 tylko stała, 2 liniowy itd.)
%
% Wyniki:
%
%       yi -- wektor z wartościami obliczonymi z szeregu Fouriera
%       p  -- współczynniki wielomianu p(0) + p(1) * x**2 + ...
%       a  -- współczyniki przy sinusach
%       b  -- współczynniki przy cosinusach
%
% 2022, dr Sławomir Marczyński

    x = x(:);
    y = y(:);
```

```
    s = size(xi);
    xi = xi(:);

    if length(x) ~= length(y)
        error 'wektory x i y mają mają różne długości'
    end

    n = length(x);
    ni = length(xi);

      if nargin < 5
        mp = 1;
    end

    if nargin < 4
        mh = fix(n / 4);
    end

    minx = min(x);
    maxx = max(x);

    % Przeskalowanie odciętych do standardowego zakresu

    x  = 2*pi * (x  - minx) / (maxx - minx);
    xi = 2*pi * (xi - minx) / (maxx - minx);

    % Utworzenie macierzy W

    W = create_W(x, mh, mp);

    % Rozwiązanie równania

    q = W \ y;

    % Utworzenie macierzy Wi

    Wi = create_W(xi, mh, mp);

    % Obliczenie wartości aproksymowanych

    yi = Wi * q;
    if s(1) == 1
        yi = yi.';
    end

    if nargout > 1
        p = q(1:mp);
        a = q((mp+1):(mp+mh));
        b = q((mp+mh+1):end);
    end

end


function W = create_W(x, mh, mp)
    n = length(x);
    W = zeros(n, mp + 2 * mh);
    for k = 1 : mp
        W(:, k) = x(:).^k;
    end
    for k = 1:mh
        W(:, k + mp) = sin(k * x(:));
        W(:, k + mp + mh) = cos(k * x(:));
    end
end
```