Filtracja sygnałów metodą Savitzky'ego-Golaya
Raport cząstkowy

Klaudia Luks, Wojciech Karkoszka

22 grudnia 2016

1 Wstęp

W niniejszym raporcie przedstawiona została prototypowa implementacja filtru Savitzky'ego-Golaya na potrzeby analizy sygnału elektrokardiograficznego. Algorytm został zbudowany i przetestowany w oparciu o sygnały pochodzące z bazy MIT-BIH, a otrzymane wyniki przedstawione zostały w dalszej części raportu.

2 Filtracja Savitzky'ego Golaya

Filtracja Savitzky'ego-Golaya polega na wygładzaniu danych wejściowych za pomocą aproksymacji wielomianu metodą najmniejszych kwadratów. Lokalne dopasowywanie wielomianów i obliczanie wartości wielomianu dla środkowego punktu analizowanego interwału jest równoważne działaniu na sygnał odpowiednim filtrem [1]. Przesuwając okno, w którym następuje dopasowanie odpowiedniej krzywej, uzyskuje się – dla środkowej próbki przedziału – wartość wygładzoną. Filtry Savitzky'ego-Golaya mona traktować jak filtry o skończonej odpowiedzi impulsowej (FIR). Szczególnym przypadkiem tego rodzaju filtracji jest filtr ruchomej średniej (w przypadku, gdy rząd wielomianu aproksymacyjnego wynosi 0).

Co istotne, wygładzanie zaszumionego sygnału tą metodą zwiększa stosunek sygnału do szumu, ale jednocześnie dość dobrze zachowuje kształt i amplitudę poszczególnych pików [2]. Ta własność znajduje szczególne zastosowanie w sygnałach elekardiograficznych [2], gdzie na podstawie morfologii zespołów QRS można rozpoznać wiele zaburzeń, tak więc sygnał przefiltrowany musi być jak najmniej zniekształcony. Algorytm może jednak powodować niekorzystne efekty brzegowe, które mogą zostać uniknięte poprzez nadbudowanie sygnału wejściowego na brzegach o określoną liczbę próbek.

Zalety filtracji Savitzky'ego-Golaya to tendencja do zachowywania wysokości i szerokości szczytów oryginalnego sygnału, płaska charakterystyka w zakresie pasma przepustowego i liniowe przesunięcie fazowe. Do wad algorytmu można zaliczyć pojawienie się efektów brzegowych na końcach sygnału (czemu można jednak zaradzić rozszerzając odpowiednio sygnał) oraz konieczność obliczania współczynników wielomianu dla każdorazowo przesuniętego okna, co wiąże się z kosztowną obliczeniowo operacją odwracania macierzy [3].

3 Opis algorytmu

3.1 Opis matematyczny

Zadaniem algorytmu jest dopasowanie w kolejnych fragmentach sygnału wielomianu, który najlepiej przybliża otrzymane dane. Wynaczanie współczynników takiego wielomianu może zostać oparte o metodę najmniejszych kwadratów [4] – należy znaleźć takie, które minimalizują wartość funkcji:

$$S(a_0, ..., a_K) = \sum_{i=1}^{N} (y_i - y(x_i))^2 \tag{1}$$

gdzie K – rząd dopasowywanego wielomianu, (x_i, y_i) – punkty poddawane filtracji, y – funkcja opisująca dopasowany wielomian.

Aby znaleźć minimum takiej funkcji możemy znaleźć miejsca, w których pochodne cząstkowe funkcji są równe zeru. Oznacza to rozwiązanie układu równań składającego się z wyrażeń dla każdej k-tej wartości współczynnika:

$$\frac{\delta S(a_0, ..., a_K)}{\delta a_k} = 0 \tag{2}$$

Zatem:

$$\frac{\delta}{\delta a_k} \sum_{i=1}^{N} \left(y_i - y(x_i)\right)^2 = 0 \tag{3}$$

$$\frac{\delta}{\delta a_k} \sum_{i=1}^{N} \left(y_i - \left(a_0 + ... + a_K x_i^K\right)\right)^2 = 0 \tag{4}$$

$$2\sum_{i=1}^{N} \left(y_i - a_0 - ... - a_K x_i^K\right)\left(-x_i^k\right) = 0 \tag{5}$$

Stąd otrzymać możemy układ równań o wyrazach w postaci:

$$\sum_{j=0}^{K} a_j \sum_{i=1}^{N} x_i^{j+k} = \sum_{i=1}^{N} y_i x_i^k \tag{6}$$

Co można przedstawić w postaci macierzowej:

$$\begin{bmatrix} N & \sum_{i=1}^{N} x_i & \dots & \sum_{i=1}^{N} x_i^K \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \dots & \sum_{i=1}^{N} x_i^{K+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{N} x_i^K & \sum_{i=1}^{N} x_i^{1+K} & \dots & \sum_{i=1}^{N} x_i^{2K} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} y_i x_i \\ \vdots \\ \sum_{i=1}^{N} y_i x_i^K \end{bmatrix} \tag{7}$$

Mnożąc lewostronnie przez odwrotność pierwszej z macierzy otrzymujemy wartości współczynników:

$$(X^{-1})XA = X^{-1}Y \tag{8}$$

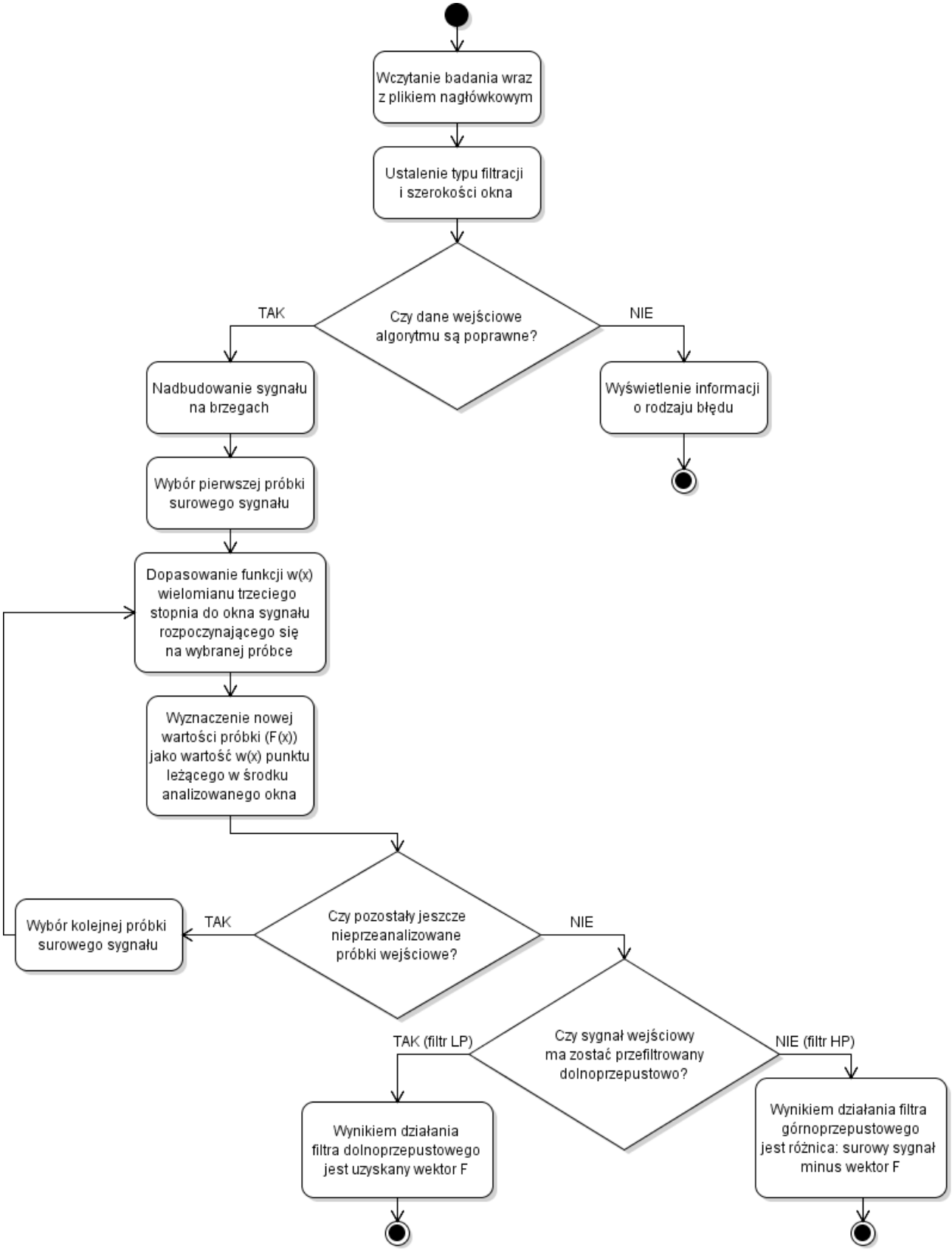
$$A = X^{-1}Y \tag{9}$$

Kolejnym krokiem jest obliczenie wartości dopasowanego wielomianu dla jednej z próbek znajdującej się w oknie – w naszym przypadku jest to środek okna.

$$y_{filtered} = \sum_{i=1}^{K} a_k x_i^k \tag{10}$$

3.2 Opis implementacyjny

Schemat blokowy algorytmu realizującego filtrację Savitzky'ego-Golaya został przedstawiony na rys. 1.



Rysunek 1: Diagram UML przedstawiający działanie zaimplementowanego algorytmu filtracji Savitzky'ego - Golaya.

Na początku sprawdzana jest poprawność danych wejściowych, między innymi:

- Czy sygnał wejściowy jest niepustą tablicą?
- Czy sygnał wejściowy składa się z dwóch kanałów? (specyfika bazy MIT-BIH)
- Czy zmienna określająca typ filtra ma wyłącznie wartość 1 (filtr dolnoprzepustowy) lub 2 (filtr górnoprzepustowy)?
- Czy szerokość okna filtracji jest liczbą całkowitą dodatnią?
- Czy szerokość okna jest liczbą nieparzystą? Jest to konieczne, ponieważ w każdej iteracji algorytmu wybierana jest środkowa próbka okna. Jeżeli szerokość okna jest liczbą parzystą, następuje jej inkrementacja o 1.

W następnej kolejności surowy sygnał jest poszerzany zarówno na początku, jak i na końcu o liczbę próbek równą całkowitoliczbowej połowie szerokości okna.

Następnie iterując po surowym sygnale tworzone są okna, do których dopasowywany jest wielomian trzeciego stopnia. Wartość przybliżonego wielomianu w samym środku okna jest uznawana za próbkę wyjściową danej iteracji, co jest widoczne we fragmencie kodu poniżej (listing 1).

Listing 1: Fragment algorytmu realizujący dopasowanie wielomianu trzeciego stopnia i obliczenie wartości wyjściowej danej próbki jako wartość funkcji wielomianowej w środku okna rozpoczynającego się w tej próbce.

```
1       bnd = floor(window_width / 2);
2
3       % ...
4
5       for i=1:length(raw_signal)
6
7           % interpolation coefficients
8           coeffs = polyfit((i:i+window_width-1)', ...
9               expanded_sig(i:i+window_width-1), 3);
10
11          % output signal for every sample
12          filtered_signal(i) = coeffs(1) * (i+bnd)^3 + ...
13              coeffs(2) * (i+bnd) ^2 + ...
14              coeffs(3) * (i+bnd) + coeffs(4);
15      end
```

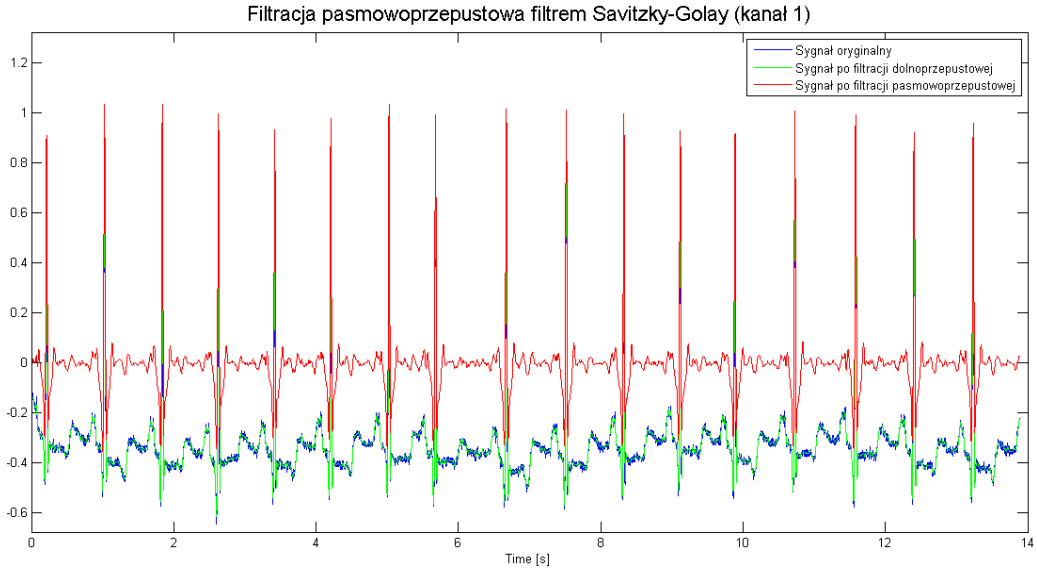
Algorytm jest dostosowany zarówno do filtracji dolno-, jak i górnoprzepustowej, a różnica polega jedynie na wykonywaniu bądź nie operacji odjęcia uzyskanego wektora od wektora surowego sygnału.

4 Rezultaty

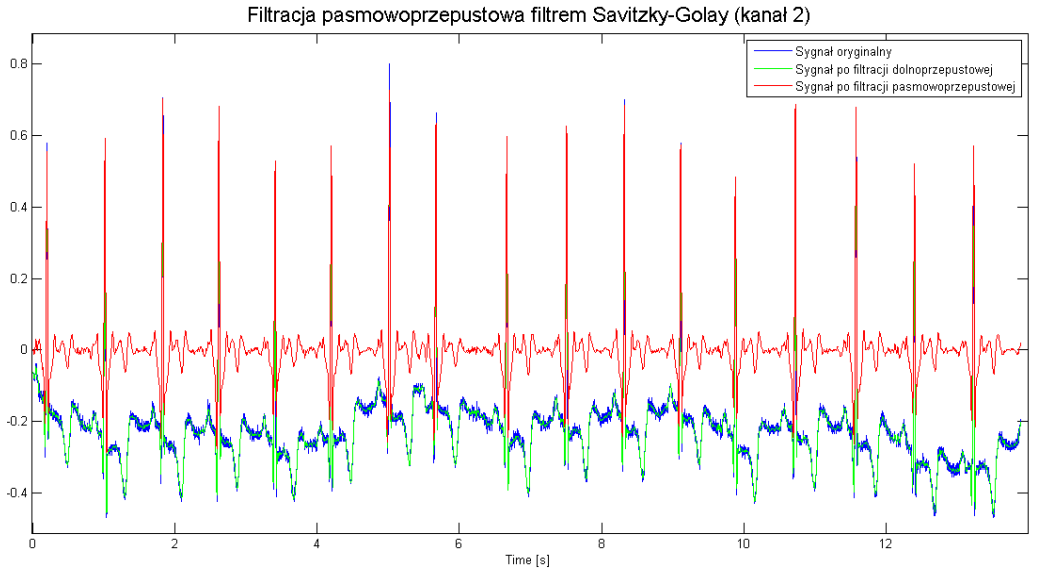
Na rysunkach 2, 3, 4, 5 przedstawiono zastosowanie filtru w praktyce na fragmentach badania z bazy nagrań MIT-BIH. Sygnały najpierw zostały poddane filtracji dolnoprzepustowej, a następnie wynikowy sygnał poddano filtracji górnoprzepustowej, co zostało oznaczone odpowiednimi kolorami.

Filtracja dolnoprzepustowa zredukowała zakłócenia sygnału. W rezultacie widoczne jest, że oscylacje sygnału surowego "wystają" poza sygnał wygładzony po filtracji dolnoprzepustowej (rys. 2, 3, 4, 5). W filtracji dolnoprzepustowej przyjęto szerokość okna równą 11 próbek.

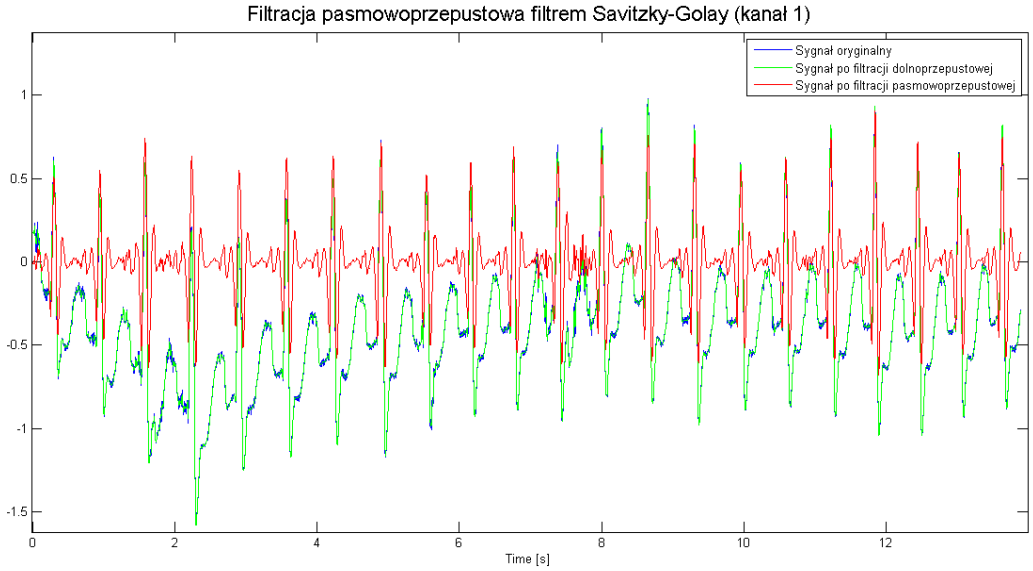
Filtracja górnoprzepustowa wyrównała izolinię sygnału do poziomu zera i zlikwidowała pływającą izolinię (szczególnie dobrze widoczne jest to na rysunkach 3, 4 i 5). W filtracji górnoprzepustowej przyjęto szerokość okna równą 85 próbek.



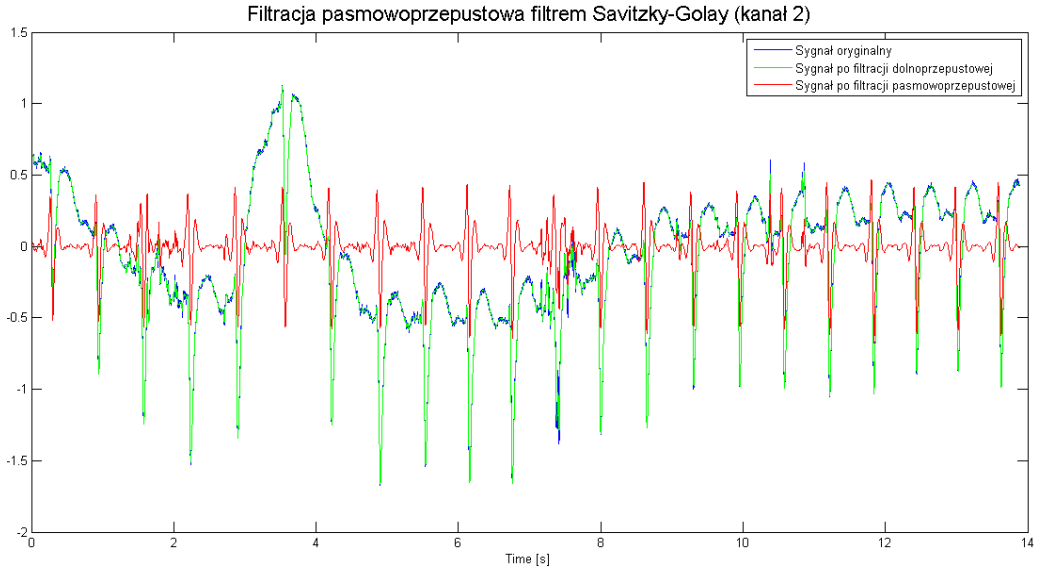
Rysunek 2: Fragment sygnału *'100.dat'*, odprowadzenie MLII, poddany filtracji pasmowoprzepustowej filtrami Savitzkyégo - Golaya na poszczególnych etapach przetwarzania.



Rysunek 3: Sygnał *'100.dat'*, odprowadzenie V5, poddany filtracji pasmowoprzepustowej filtrami Savitzkyégo - Golaya na poszczególnych etapach przetwarzania.



Rysunek 4: Sygnał *'109.dat'*, odprowadzenie MLII, poddany filtracji pasmowoprzepustowej filtrami Savitzkyégo - Golaya na poszczególnych etapach przetwarzania.



Rysunek 5: Sygnał *'109.dat'*, odprowadzenie V1, poddany filtracji pasmowoprzepustowej filtrami Savitzkyégo - Golaya na poszczególnych etapach przetwarzania.

5 Podsumowanie

W chwili obecnej działanie algorytmu filtracji Savitzky'ego - Golaya daje satysfakcjonujące efekty, jednak zajmuje dużo czasu, co utrudnia jego uruchomienie na większej ilości danych, w tym na całej bazie MIT-BIH.

W związku z tym dalszej części projektu uwaga zostanie skupiona na dwóch aspektach:

- implementacja tegoż algorytmu w języku C++ w celu zwiększenia jego wydajności;

- przygotowanie systemu walidacji algorytmu w celu porównania wyników z innymi zespołami w ramach raportu zbiorczego.

Literatura

[1] Krishnan S. R., Seelamantula C. S.*On the Selection of Optimum Savitzky-Golay Filters*. IEEE transactions on signal processing 61.2 (2013): 380-391.

[2] Schafer R. W. *What is a Savitzky-Golay filter?[lecture notes]*. IEEE Signal Processing Magazine 28.4 (2011): 111-117.

[3] Persson, P., Strang, G. *Smoothing by Savitzky-Golay and legendre filters*. Mathematical systems theory in biology, communications, computation, and finance (2003): 301-315.

[4] Miller, S. J. *The method of least squares*. Mathematics Department Brown University (2006): 1-7.

Dodatek A: Kod źródłowy prototypów programu

Listing 2: Wywołanie funkcji realizującej filtrację Savitzky'ego-Golaya.

```
clc, clear;
close all
[TIME, M] = read_dat_file(109);

sig = M(1:5000,:);
time = TIME(1:5000);

lp = sav_gol_multichannel(sig, 1, 11);
hp = sav_gol_multichannel(lp, 2, 85);

figure;
plot(time, sig(:,1))
hold on
plot(time, lp(:,1), 'g')
hold on
plot(time, hp(:,1), 'r')
title('\fontsize{16}Filtracja pasmowoprzepustowa filtrem Savitzky-Golay (kanal 1)')
legend('Sygnal oryginalny', 'Sygnal po filtracji dolnoprzepustowej', 'Sygnal po filtracji ...
    pasmowoprzepustowej')
xlabel('Time [s]')

figure;
plot(time, sig(:,2))
hold on
plot(time, lp(:,2), 'g')
hold on
plot(time, hp(:,2), 'r')
title('\fontsize{16}Filtracja pasmowoprzepustowa filtrem Savitzky-Golay (kanal 2)')
legend('Sygnal oryginalny', 'Sygnal po filtracji dolnoprzepustowej', 'Sygnal po filtracji ...
    pasmowoprzepustowej')
xlabel('Time [s]')
```

Listing 3: Funkcja filtrująca dla wielu kanałów.

```
function filtered_signal = sav_gol_multichannel (raw_signal, filter_type, ...
    window_width)
    %% Filter input signal with Savitzky-Golay filter

    % Parameters:
    % ----------------
    % raw_signal:       2D (or 1D) array, double
    %                   raw ECG signal from a few (at least one) channels
    % filter_type:      strictly specified uint: 1 or 2
    %                   1 - low-pass filter
    %                   2 - high-pass filter
    % window_width:     uint
    %                   width of filtration window
    %
    % Returns:
    % ----------------
    % filtered_signal:  2D (od 1D) array, double
    %                   filtered ECG signals returned in the same number of
    %                   channels as input signal

    %% Input data verification

    if (size(raw_signal,1)<1 || size(raw_signal,2)<1)
        error('Input signal must be non-empty array.')
    end

```

```
27      if (filter_type ≠ 1 && filter_type ≠ 2)
28          error('Incorrect filter type. Choose 1 for low-pass and 2 for high-pass filter.')
29      end
30
31      if (floor(window_width) ≠ window_width || window_width ≤ 0)
32          error('Window width must be positive integer')
33      end
34
35      % window width must be odd value
36      if (rem(window_width, 2) == 0)
37          window_width = window_width + 1;
38      end
39
40      %% calculate width of left/right artificial band
41      bnd = floor(window_width / 2);
42
43      %% iterate every channel
44      for chan = 1 : size(raw_signal, 2)
45
46          % expand signal
47          left_side = repmat(raw_signal(1, chan), bnd, 1);
48          right_side = repmat(raw_signal(end, chan), bnd, 1);
49          expanded_sig = [left_side; raw_signal(:, chan); right_side];
50
51
52          for i=1:length(raw_signal(:,chan))
53
54              % interpolation coefficients
55              coeffs = polyfit((i:i+window_width-1)', ...
56                  expanded_sig(i:i+window_width-1), 3);
57
58              % output signal for every sample
59              filtered_signal(i, chan) = coeffs(1) * (i+bnd) ^3 + ...
60                  coeffs(2) * (i+bnd) ^2 + ...
61                  coeffs(3) * (i+bnd) + coeffs(4);
62          end
63
64          filtered_signal(:,chan) = filtered_signal(:,chan)';
65      end
66
67      %% in case of high-pass filter
68
69      if filter_type == 2
70          filtered_signal = raw_signal - filtered_signal;
71      end
72
73  end
```