



AKADEMIA GÓRNICZO-HUTNICZA  
IM. STANISŁAWA STASZICA W KRAKOWIE

ELEKTRONICZNE SYSTEMY DIAGNOSTYKI MEDYCZNEJ I  
TERAPII

---

## Wygładzanie sygnału metodą Savitzky-Golay

---

*Autorzy:*  
Piotr PAŁUCKI  
Filip KUBICZ

# Spis treści

<b>1</b>	<b>Wstęp</b>	<b>2</b>
<b>2</b>	<b>Algorytm</b>	<b>3</b>
2.1	Przyjęte parametry . . . . .	3
<b>3</b>	<b>Prototyp programu</b>	<b>4</b>
3.1	Wykorzystane moduły . . . . .	4
3.2	Oprogramowanie algorytmu . . . . .	4
3.3	Wartości brzegowe . . . . .	4
<b>4</b>	<b>Implementacja</b>	<b>6</b>
4.1	Wykorzystane biblioteki . . . . .	6
4.2	Wyjście programu . . . . .	6
4.3	Porównanie z modelem programowym . . . . .	7
<b>Dodatek A</b>	<b>Instrukcje</b>	<b>9</b>
A.1	Uruchomienie prototypu . . . . .	9
A.2	Kompilacja aplikacji . . . . .	9
A.3	Uruchomienie aplikacji . . . . .	9

# 1 Wstęp

Niniejsza praca jest częścią projektu mającego na celu identyfikację zespołu QRS w sygnale z elektrokardiografu. Sygnał EKG zaraz po zebraniu danych jest zaszumiony i wymaga wygładzenia oraz usunięcia zakłóceń. Zakłócenia takie jak pojedyncze ruchy pacjenta trudno wyeliminować bez posiadania dodatkowej wiedzy na temat jego zachowania podczas pomiaru. Możliwe jest jednak odfiltrowanie szumu niskiej częstotliwości pochodzącego od ruchów klatki piersiowej podczas oddychania.

Jednym ze sposobów wstępnego przetwarzania sygnału EKG zanim przystąpi się do jego analizy jest filtr Savitzky-Golay.

W naszej części projektu przygotowujemy prototyp algorytmu z użyciem języka Python, a następnie implementację w języku C++. Użyte w projekcie narzędzia przedstawia tabela 1.

Część projektu	Język programowania	Biblioteki
Prototyp	Python 3.x	NumPy, Matplotlib
Implementacja	C++	Eigen, gnuplot

Tablica 1: Zestawienie języków programowania i modułów

## 2 Algorytm

Filtr Savitzky-Golay pozwala wygładzić cyfrowy sygnał. Jego działanie opiera się na lokalnym przybliżeniu próbek sygnału wielomianami niskiego rzędu [?].

Mając dany sygnał  $x[n]$ , poszukujemy wielomianu

$$p(n) = \sum_{k=0}^N a_k n^k \quad (1)$$

który w otoczeniu  $2M + 1$  punktów minimalizuje kwadrat błędu aproksymacji

$$\epsilon_N = \sum_{n=-M}^M (p(n) - x[n])^2 \quad (2)$$

Wtedy dla próbki centralnej pośród  $2M + 1$  punktów (ma ona indeks  $n = 0$ ) wartość odfiltrowanego sygnału przyjmuje

$$y[0] = p(0) = a_0 \quad (3)$$

W praktyce filtr Savitzky-Golay realizuje swoje zadanie obliczając splot  $2M+1$  aktualnie branych pod uwagę punktów z wielomianową aproksymacją ciągu z jednostkowym impulsem w środku sekwencji.

$$y[n] = \sum_{m=n-M}^{n+M} h[n-m]x[m] \quad (4)$$

gdzie  $h$  jest wynikiem interpolacji wielomianem stopnia  $N$  sekwencji z jednostkowym impulsem, np.  $[0, 0, 0, 1, 0, 0, 0]$  dla  $M = 3$ .

### 2.1 Przyjęte parametry

Opracowanego w pracy filtru można użyć w filtracji sygnału EKG na dwa sposoby: do odfiltrowania zakłóceń wysokiej częstotliwości oraz do wyznaczenia izolinii reprezentującej na przykład ruchy związane z oddychaniem pacjenta. Mając wyznaczoną izolinię można potem odjąć ją od sygnału wejściowego otrzymując poprawny sygnał EKG. Tak rozbieżne cele wymagają użycia różnych rozmiarów okna filtracji. Parametry przyjęte w pracy przedstawia poniższa Tablica 2

Cel / Parametr	N	M
Wyznaczenie izolinii	2	900
Odfiltrowanie zakłóceń wysokiej częstotliwości	2	5

Tablica 2: Parametry filtru

## 3 Prototyp programu

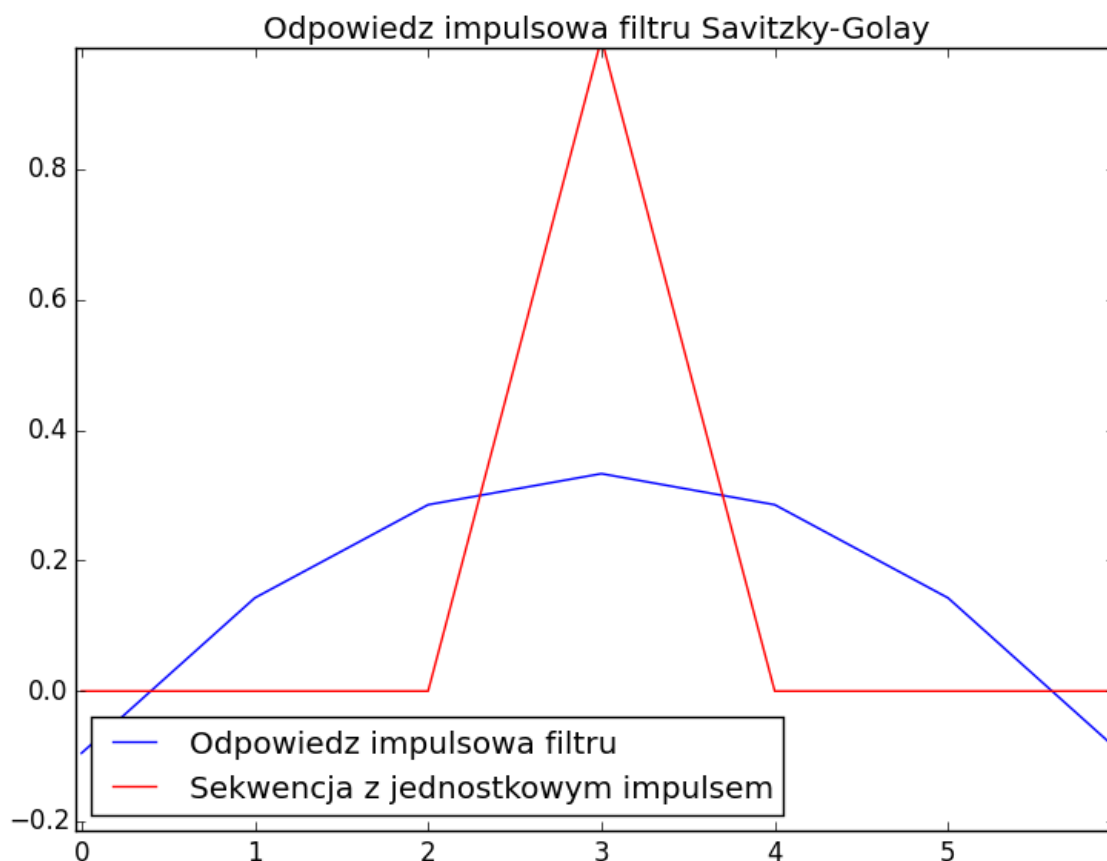
### 3.1 Wykorzystane moduły

Przed przystąpieniem do pracy nad algorytmem, użyliśmy biblioteki **WFDB** dla środowiska MATLAB, która umożliwiła pobranie danych EKG z bazy MIT-BIH[?].

W prototypowej wersji programu użyliśmy modułu **NumPy** do operacji na macierzach i operacji numerycznych, w tym: interpolacji wielomianowej, sklejana i obracania wektorów. Moduł matplotlib.pyplot posłużył do obrazowania wyników.

### 3.2 Oprogramowanie algorytmu

Ponieważ odpowiedź impulsowa filtru  $h$  jest niezależna od wartości sygnału, wartości wektora  $h$  o długości  $2m+1$  wystarczy obliczyć raz, a następnie obliczyć wartość konwolucji sygnału z taką maską. Wartości maski zostały pokazane na rysunku 1.



Rysunek 1: Odpowiedź impulsowa filtru, uzyskana jako interpolacja wielomianem  $N = 2$  stopnia serii  $2M + 1 = 7$  próbek z jednostkowym impulsem pośrodku

### 3.3 Wartości brzegowe

Algorytm w każdym punkcie potrzebuje do obliczeń  $M$  punktów z lewej i  $M$  punktów z prawej strony, zatem należy zdecydować o zachowaniu programu na początku i na końcu

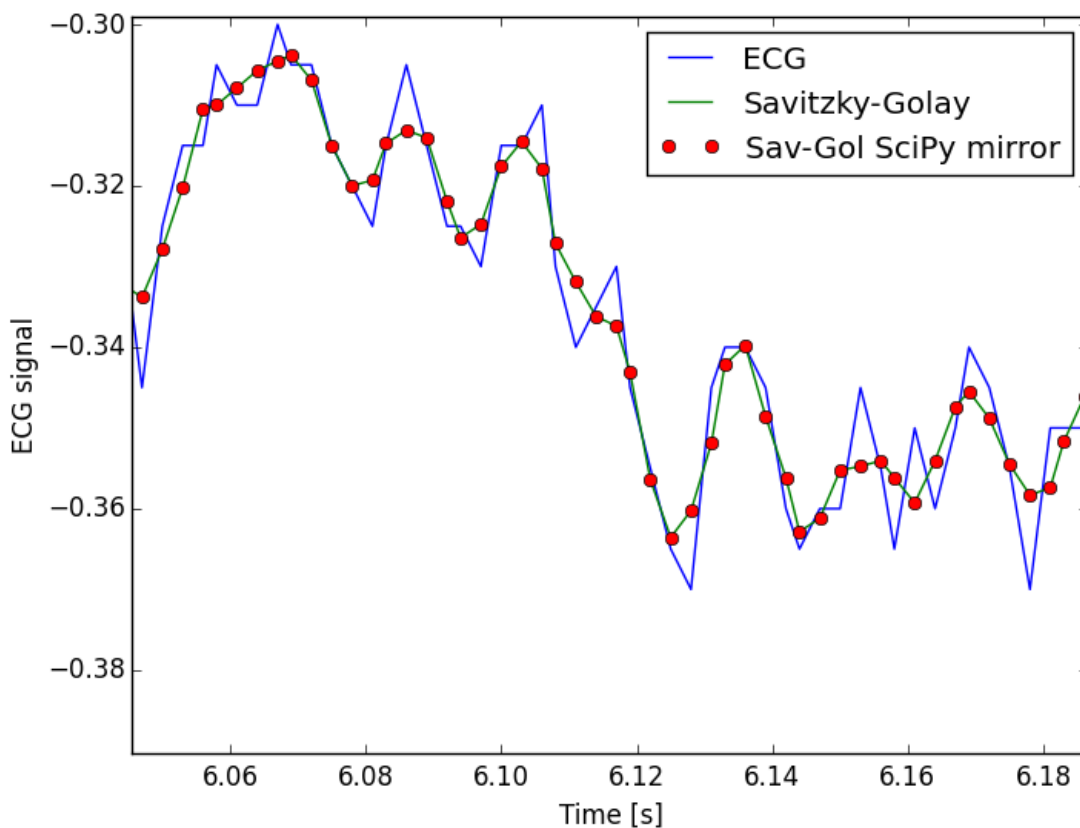
spróbkowanego sygnału.

Istnieje kilka możliwych rozwiązań:

1. Zawijanie - algorytm wykorzysta pierwsze  $M$  punktów na końcu oraz ostatnie  $M$  punktów na początku sygnału
2. Odbicie - na brzegach zostaną wykorzystane najbliższe punkty, jednak w odwróconej kolejności
3. Użycie stałej, ustalonej wartości
4. Najbliższy sąsiad - poza granicami sygnału zostaną dopisane punkty równe pierwszej próbkę (na początku) i ostatniej próbkę (na końcu)

W prototypie wybraliśmy rozwiązanie 2. Jeśli ostatnie punkty mają wartości  $[6, 7, 8, 9]$ , to dopisane poza prawą granicą punkty otrzymają wartości  $[8, 7, 6]$ .

Przebieg filtracji uzyskanej za pomocą prototypu Python przedstawia wykres 2



Rysunek 2: Wygładzanie metodą Savitzky-Golay z oknem 7 próbek ( $M=3$ ) i aproksymacją wielomianem  $N=2$  stopnia. Porównanie prototypu z funkcją z modułu SciPy.signals

## 4 Implementacja

Implementacja właściwa filtrowania metodą Savitzky-Golay została przygotowana z wykorzystaniem języka C++ w standardzie 11. Filtracja z uwagi na niewielkie wykorzystanie zasobów wykonywana jest w głównym wątku programu. Aplikacja pobiera z linii poleceń wszystkie niezbędne parametry takie jak nazwy plików wejściowego i wyjściowego, szerokość okna filtracji, rząd wykorzystanego wielomianu oraz parametr decydujący czy po zakończeniu obliczeń pokazać wykres porównujący sygnał surowy i przefiltrowany. Procedura zbudowania i uruchomienia aplikacji opisana jest w Dodatku A.

### 4.1 Wykorzystane biblioteki

W aplikacji wykorzystano bibliotekę Eigen (<http://eigen.tuxfamily.org>), która dostarcza funkcjonalności umożliwiające operować na wektorach i macierzach w sposób znany nam chociażby ze środowiska Matlab. Biblioteka oparta jest o szablony. Nie wymaga ona instalacji, gdyż dystrybuowana jest jako zbiór plików nagłówkowych, które wystarczy (opcjonalnie wybiórczo) umieścić w swoim projekcie.

Biblioteka Eigen oferuje użytkownikom typy o dynamicznie ustalonym rozmiarze reprezentujące wektor i macierz (np. liczb typu float):

- Eigen::VectorXf
- Eigen::MatrixXf

Dodatkowo dostarcza przydatne funkcje jak:

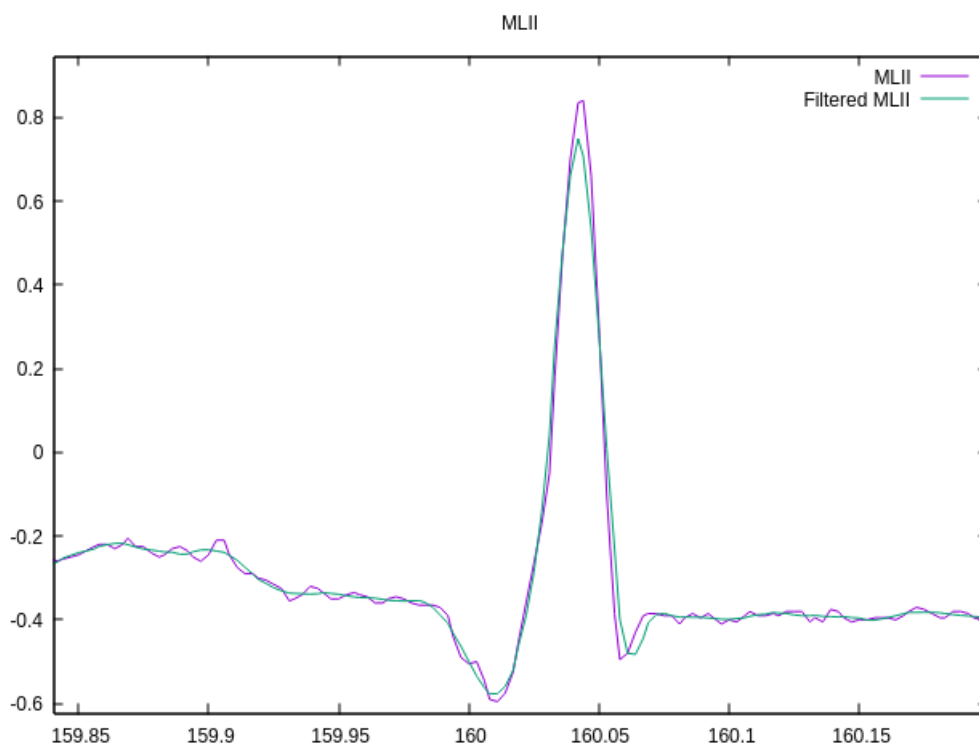
- Eigen::MatrixXf transpose() - zwracającą transpozycję macierzy,
- Eigen::MatrixXf inverse() - zwracającą macierz odwrotną.

### 4.2 Wyjście programu

Wyjściem programu jest plik z danymi. Plik posiada następujący format:

Time [s]	MLII [mV]	MLII_Filtered [mV]	V5 [mV]	V5_Filtered [mV]
0	-0.145	-0.314107	-0.065	-0.216375
0.003	-0.145	-0.314043	-0.065	-0.216346
0.006	-0.145	-0.313977	-0.065	-0.216311
0.008	-0.145	-0.313882	-0.065	-0.216267
0.011	-0.145	-0.313774	-0.065	-0.21621
0.014	-0.145	-0.313648	-0.065	-0.216148
0.017	-0.145	-0.313519	-0.065	-0.216095
0.019	-0.145	-0.313392	-0.065	-0.216052
0.022	-0.12	-0.313274	-0.08	-0.216001

Opcjonalnie na podstawie takiego pliku można wygenerować wykres z porównaniem sygnału przed i po filtracją. Wykres generowany jest przez środowisko **gnuplot**. Przykładowy wykres przedstawiono poniżej.

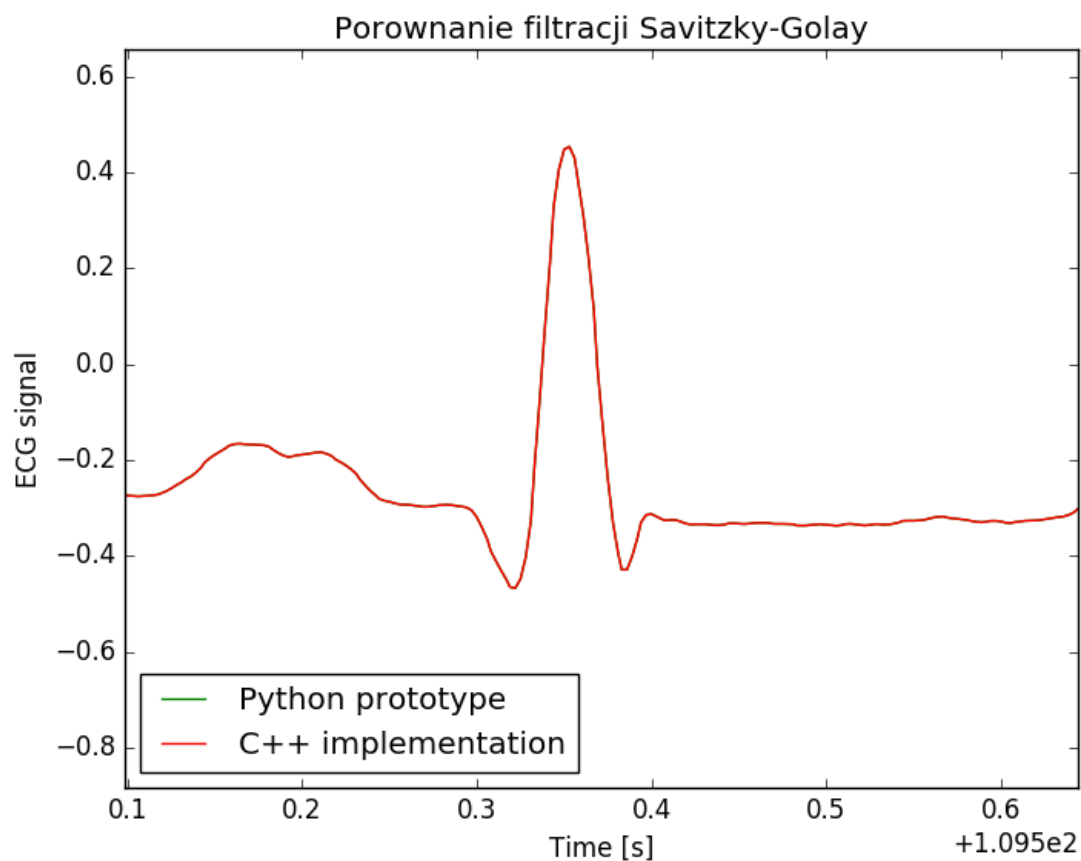


Rysunek 3: Porównanie sygnałów przy filtracji wielomianem  $N = 2$  stopnia serii  $2M + 1 = 15$  próbek

### 4.3 Porównanie z modelem programowym

Poniżej przedstawiono porównanie sygnałów przefiltrowanych w prototypie oraz we właściwej aplikacji. Nie widać większych różnic między sygnałami.





Rysunek 4: Rozbieżności przy filtracji wielomianem  $N = 2$  stopnia serii  $2M + 1 = 21$  próbek

Aby upewnić się jaka rozbieżność istnieje między sygnałami policzono MSE.....  
(ZROBIĆ)

## Dodatek A Instrukcje

Dodatek ten opisuje sposoby kompilacji i uruchomienia przygotowanej aplikacji.

### A.1 Uruchomienie prototypu

Uruchomienie prototypu wymaga zainstalowania środowiska Python. Popularną dystrybucją zawierającą niezbędne moduły jest pakiet Anaconda.

Uruchomienie prototypu z linii komend:

```
prototype$ ./savitzky_design.py
```

### A.2 Kompilacja aplikacji

Do budowania aplikacji wykorzystano kompilator **gcc** w wersji 5.4.0.

Flagi kompilacji: *-Ofast -Wall -std=c++11*. W celu ułatwienia budowy aplikacji w projekcie wykorzystano framework **cmake**. Instrukcję pobrania instalacji można znaleźć na stronie twórców narzędzia: <https://cmake.org/>.

Procedura kompilacji z użyciem cmake:

```
implem/build$ cmake ../  
implem/build$ make
```

Alternatywą do systemu budowania cmake może być zwykły system oparty o pliki Makefile lub nawet bezpośrednie użycie kompilatora wraz ze wspomnianymi flagami kompilacji.

### A.3 Uruchomienie aplikacji

Uruchomienie aplikacji odbywa się za pośrednictwem linii poleceń:

```
implem/build$ ./SavGol.o
```

Po uruchomieniu programu bez żadnych argumentów otrzymamy wiadomość zwrotną wraz z poprawnym użyciem programu i opisem poszczególnych parametrów:

```
implem/build$ ./SavGol.o  
Usage:  
./SavGol.o -i file -o file [-n N] [-m M] [plot]  
Options:  
--input, -i file : name of the input file  
--output, -o file : name of the output file  
--plot, -p      : plot filtered signal at the end; default: disabled  
-n N           : half of the filtering range; default: 2  
-m M           : half of the filtering range; default: 5
```

W ramach projektu przygotowano skrypt, który zbuduje i uruchomi aplikację:

```
implem$ ./runSavitzkyGolay
```

W celu zmiany parametrów uruchomienia najłatwiej poddać skrypt modyfikacji i uruchomić go ponownie.

Do poprawnego wyświetlania wykresów niezbędne jest środowisko **gnuplot**. Środowisko to jest z założenia międzyplatformowe, więc powinno działać na większości systemów operacyjnych. Instrukcja jego instalacji znajduje się pod adresem <http://www.gnuplot.info/> .

Aktualny kod projektu dostępny w repozytorium <https://github.com/Qbicz/Savitzky-Golay>