Przetwarzanie Sygnałów

Jakub Dudarewicz

Semestr letni, 2016

Instrukcja 1. Podstawy generacji sygnałów

Instrukcja 1, zadanie 1 Operacje tablicowe i macierzowe

Wynik operacji mnożenia macierzowego i tablicowego:

```
A =
1   43   45

B =
6   12   45
2    4   56
12    3    1
```

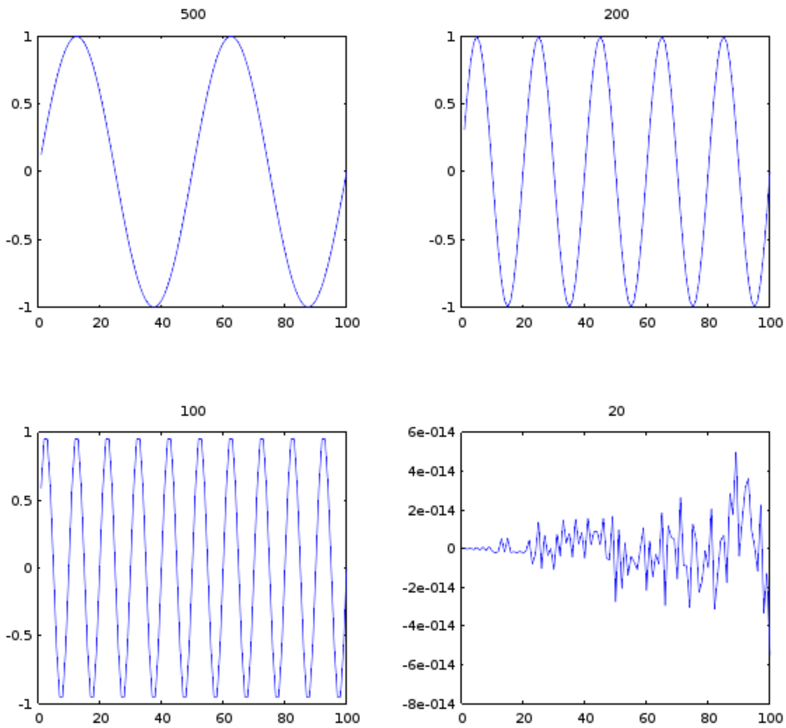
```
tablicowe
6     516    2025
2     172    2520
12     129      45

macierzowe
632     319    2498
```

Operacje te są fundamentalnie od siebie różne

Instrukcja 1, zadanie 2 Próbkowanie przebiegu sinusoidalnego

M-plik użyty do generacji wykresów:

```
% probkowanie sinusa

n=10; %długość obserwacji
k=100; %liczba próbek
t=[1:k];

H = figure;

m=500;
a=sin(2*pi*n*t/m);
subplot(221);
plot(a);
title("500");

m=200;
a=sin(2*pi*n*(t/m));
subplot(222);
plot(a);
title("200");

m=100;
a=sin(2*pi*n*t/m);
subplot(223);
plot(a);
title("100");

m=20;
a=sin(2*pi*n*t/m);
subplot(224);
plot(a);
title("20");

clear;
```



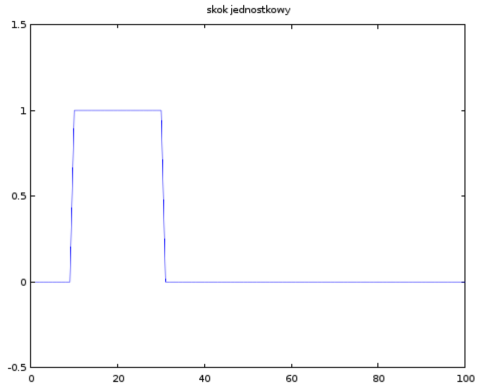
Rysunek 1: Przebieg sinusoidalny próbkowany z różnymi częstotliwościami

Liczba zarejestrowanych okresów i rozdzielczość wynika z częstotliwości próbkowania. W tym zadaniu nie przekształcam dziedziny sygnału do czasu.

Instrukcja 1, zadanie 3 Skok jednostkowy

M-plik użyty do genracji wykresów:

```
%skok jednostkowy

be = 10;
en = 30;
dur = 100;

a = zeros(dur,1);

for i = be:en
a(be, 1) = 1;
be = be + 1;
end

figure;
plot(a);
axis([0, 100, -0.5, 1.5]);
title("skok jednostkowy");
```

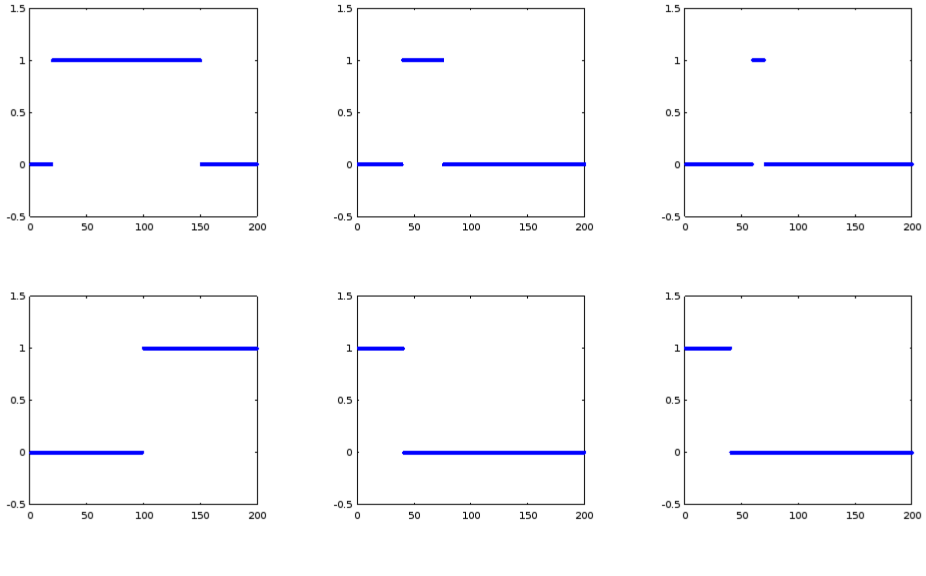


Rysunek 2: Przykładowy wydruk skoku jednostkowego

Instrukcja 1, zadanie 4 Skok jednostkowy, użycie varargin

M-plik użyty do generowania wykresów:

```
function skok(varargin) %(duration,
begin 1, end 1, begin 2, end 2...

k = 2;
l=1;
figure;
r = ceil((nargin-1)/4);
dur = varargin{1};

for i = 1:((nargin-1)/2)
be = varargin{k};
en = varargin{k+1};

a = zeros(dur,1);

for j = be:en
a(be, 1) = 1;
be += 1;
end

subplot(r, 2, l);
plot(a, '.');
axis([0, dur, -0.5, 1.5]);

printf("bound %d - %d\n", varargin{k},
varargin{k+1});

if(k+1<(nargin-1))
k=k+2;
endif
l=l+1;
end

endfunction
```



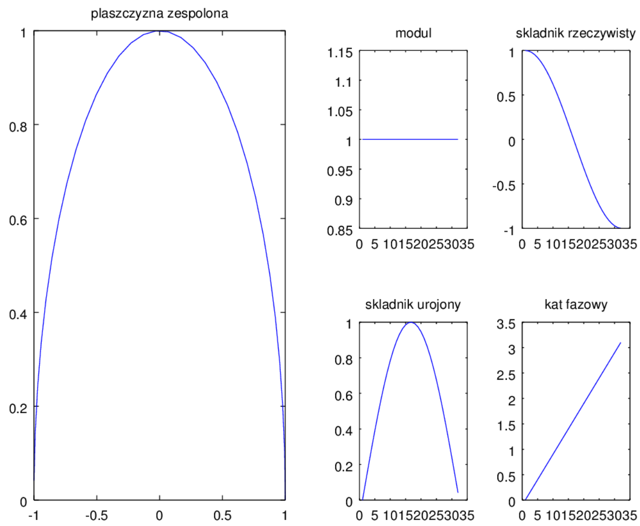
Rysunek 3: Przykładowy wydruk dla siedmiu argumentów

Funkcja varargin i wszystkie jej towarzyszące są bardzo użyteczne, bo pozwalają na napisanie bardziej uniwersalnej funkcji.

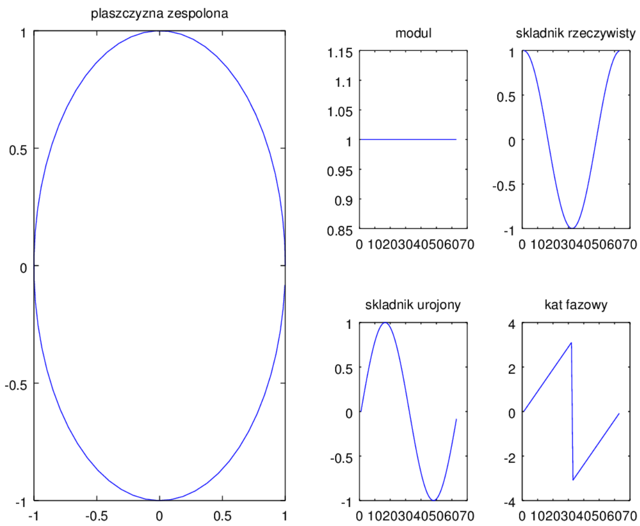
Instrukcja 1, zadanie 5 Przebieg wykładniczy zespolony

M-plik użyty do generacji wykresów funkcji $f(x) = e^{xi}$:

```
%przebieg wykładniczy zespolony

t=[0:0.1:2*pi];

zesp=e.^(t.*j);
rzecz=real(zesp);
uroj=imag(zesp);
modul=abs(zesp);
faza=angle(zesp);

figure

subplot(121)
plot(zesp)
title("plaszczyzna zespolona")
subplot(243)
plot(modul)
title("modul")

subplot(244)
plot(rzecz)
title("skladnik rzeczywisty")
subplot(247)
plot(uroj)
title("skladnik urojony")
subplot(248)
plot(faza)
title("kat fazowy")
```



(a) $0i - \pi i$



(b) $0i - 2\pi i$

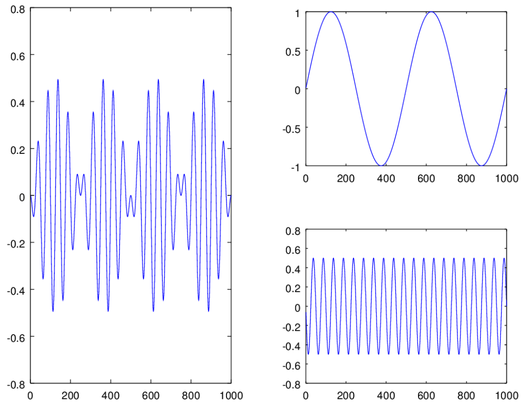
Rysunek 4: Zespolone przebiegi wykładnicze dla dwóch zakresów zmiennej zespolonej

Z powyższych wykresów połączenie funkcji wykładniczej zespolonej z funkcjami trygonometrycznymi jest oczywiste. Zrozumiała też staje się słynna Tożsamość Eulera $e^{\pi i} + 1 = 0$.

Instrukcja 1, zadanie 6 Modulowanie przebiegu sinusoidalnego

M-plik użyty do generacji wykresów:

```
%modulowanie sinusa

n=1; %długość obserwacji
k=1000; %liczba próbek
t=[1:k]; %probki
m=500; %czestotliwosc obserwacji

a=sin(2*pi*n*t/m);
b=(0.5).*sin((10*2*pi*n*t/m)+pi);
c=a.*b;

figure

subplot(121)
plot(c)
subplot(222)
plot(a)
subplot(224)
plot(b)
```



Rysunek 5: Z lewej - sygnał zmodulowany, z prawej - sygnały wyjściowe

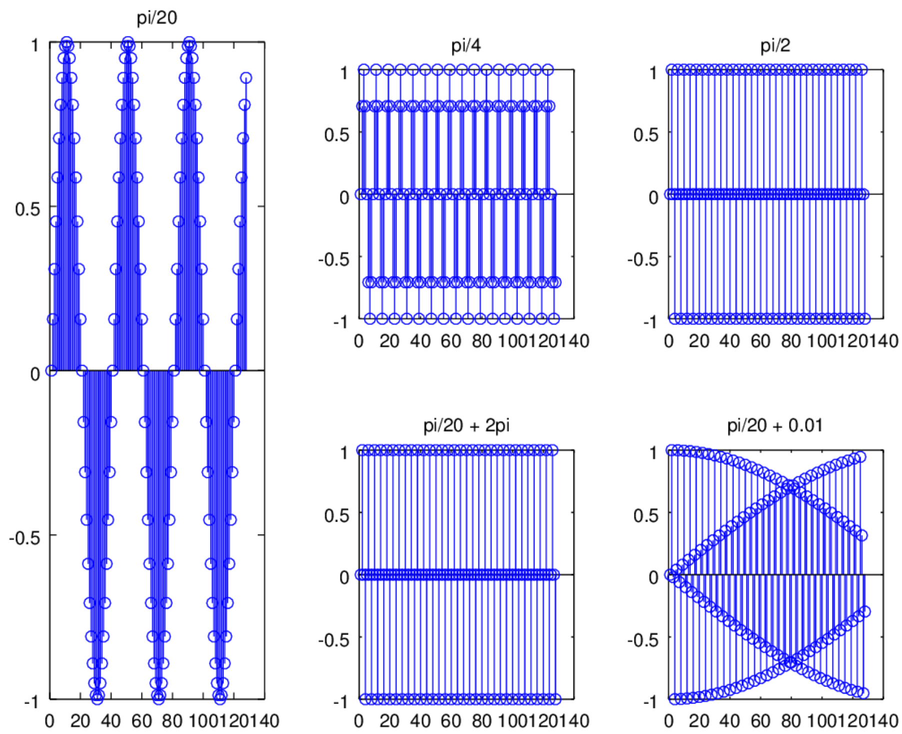
Modulacja osiągana jest poprzez pomnożenie tablicowe przez siebie dwóch sygnałów.

Instrukcja 2. Podstawy próbkowania sygnałów

Instrukcja 2, zadanie 1 Generowanie przebiegów sinusoidalnych o pulsacji znormalizowanej

M-plik użyty do generacji wykresów:

```
n=[0:127];

xa=sin((pi/20)*n);
xb=sin((pi/4)*n);
xc=sin((pi/2)*n);
xd=sin((pi/2+2*pi)*n);
xe=sin((pi/2+0.01)*n);

subplot(131)
stem(xa)
title('\pi/20')
subplot(232)
stem(xb)
title('\pi/4')
subplot(233)
stem(xc)
title('\pi/2')
subplot(235)
stem(xd)
title('\pi/20 + 2\pi')
subplot(236)
stem(xe)
title('\pi/20 + 0.01')
```



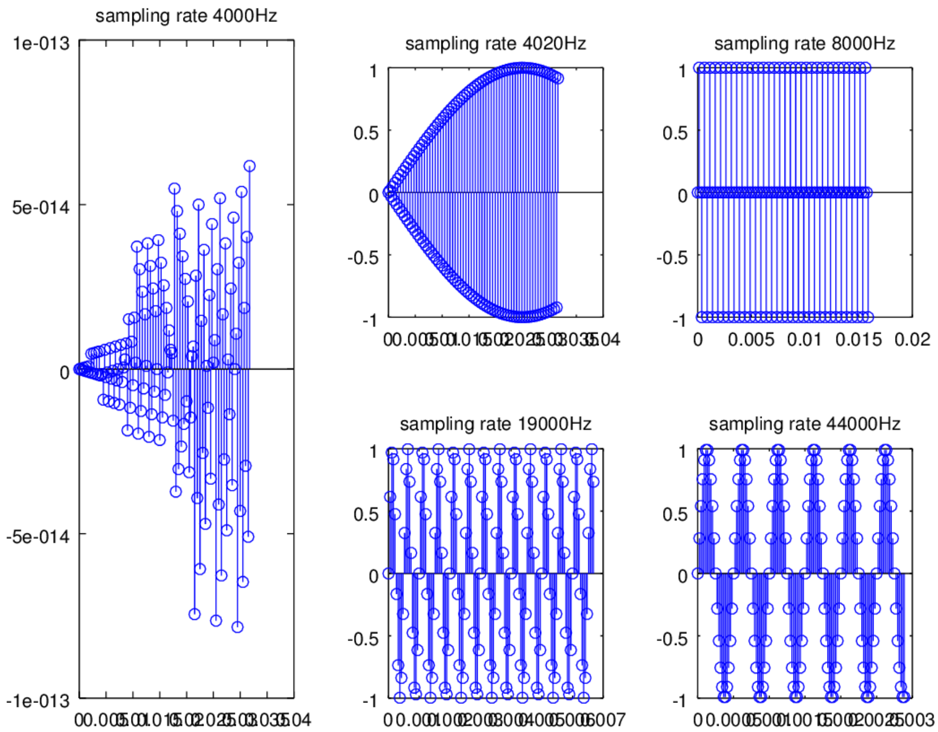
Rysunek 6: Przebiegi generowane dla różnych pulsacji.

Zbyt duża częstotliwość powoduje niedokładne rejestrowanie przebiegu.

Instrukcja 2, zadanie 2 Generowanie przebiegów sinusoidalnych próbkowanych z różną częstotliwością

M-plik użyty do generacji wykresów:

```
%sampling frequency
fsa=4000;
fsb=4020;
fsc=8000;
fsd=19000;
fse=44000;

%samples vector
n=[0:127];

%signal frequency
f=2000;

%time vectors
ta=n/fsa;
tb=n/fsb;
tc=n/fsc;
td=n/fsd;
te=n/fse;


%signal model
xa=sin(2*pi*f*ta);
xb=sin(2*pi*f*tb);
xc=sin(2*pi*f*tc);
xd=sin(2*pi*f*td);
xe=sin(2*pi*f*te);

subplot(131)
stem(ta, xa)
title('sampling rate 4000Hz')
subplot(232)
stem(tb, xb)
title('sampling rate 4020Hz')
subplot(233)
stem(tc, xc)
title('sampling rate 8000Hz')
subplot(235)
stem(td, xd)
title('sampling rate 19000Hz')
subplot(236)
stem(te, xe)
title('sampling rate 44000Hz')
```

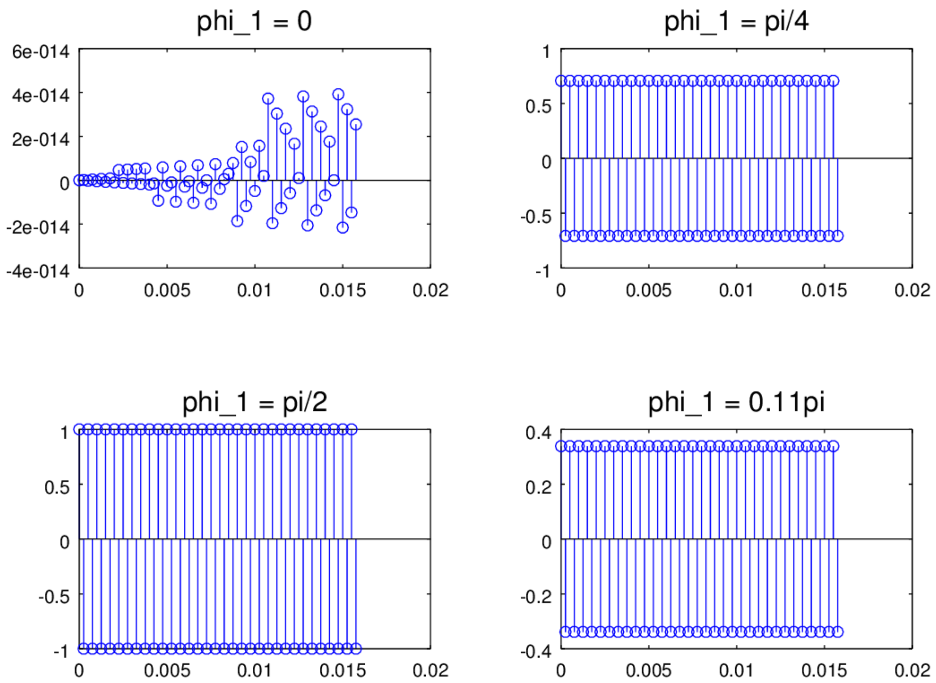


Rysunek 7: Przebiegi o równej częstotliwości próbkowane z różną częstotliwością wyświetlone w dziedzinie czasu

Instrukcja 2, zadanie 3 Generowanie przebiegów sinusoidalnych przesuniętych w fazie

M-plik użyty do generacji wykresów:

```
%sampling frequency
fs=4000;

%samples vector
n=[0:63];

%signal frequency
f=2000;

%time vectors
t=n/fs;

%signal model
xa=sin(2*pi*f*t);
xb=sin(2*pi*f*t + pi/4);
xc=sin(2*pi*f*t + pi/2);
xd=sin(2*pi*f*t + 0.11*pi);

subplot(221)
stem(t, xa)
title('\phi_1 = 0', 'fontsize', 15)
subplot(222)
stem(t, xb)
title('\phi_1 = \pi/4', 'fontsize', 15)
subplot(223)
stem(t, xc)
title('\phi_1 = \pi/2', 'fontsize', 15)
subplot(224)
stem(t, xd)
title('\phi_1 = 0.11\pi', 'fontsize', 15)
```



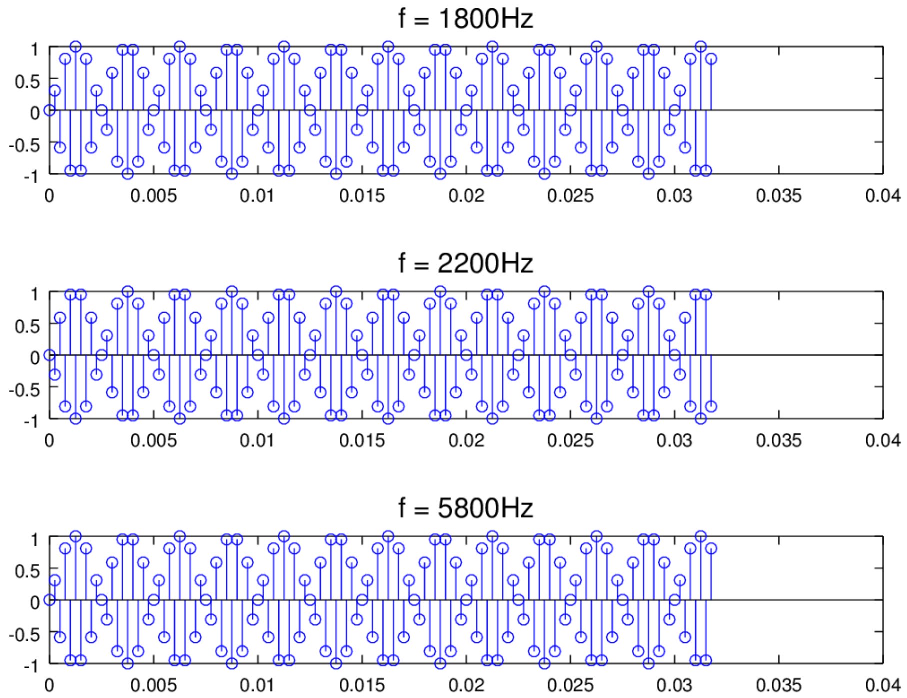
Rysunek 8: Przebiegi o równej częstotliwości z różnym przesunięciem fazowym i próbkowane z równą częstotliwością

Ponieważ częstotliwość próbkowania sygnału jest dwukrotnie większa od częstotliwości sygnału sygnał jest próbkowany zawsze w tych samych dwóch miejscach w okresie. Od przesunięcia fazowego sygnału zależy jakie wielkości osiągną próbki.

Instrukcja 2, zadanie 4 Generowanie przebiegów sinusoidalnych o różnych częstotliwościach

M-plik użyty do generacji wykresów:

```
%sampling frequency
fs=4000;

%samples vector
n=[0:127];

%signal frequency
fa=1800;
fb=2200;
fc=5800;

%time vectors
t=n/fs;

%signal model
xa=sin(2*pi*fa*t);
xb=sin(2*pi*fb*t);
xc=sin(2*pi*fc*t);

subplot(311)
stem(t, xa)
title('f = 1800Hz', 'fontsize', 15)
subplot(312)
stem(t, xb)
title('f = 2200Hz', 'fontsize', 15)
subplot(313)
stem(t, xc)
title('f = 5800Hz', 'fontsize', 15)
```

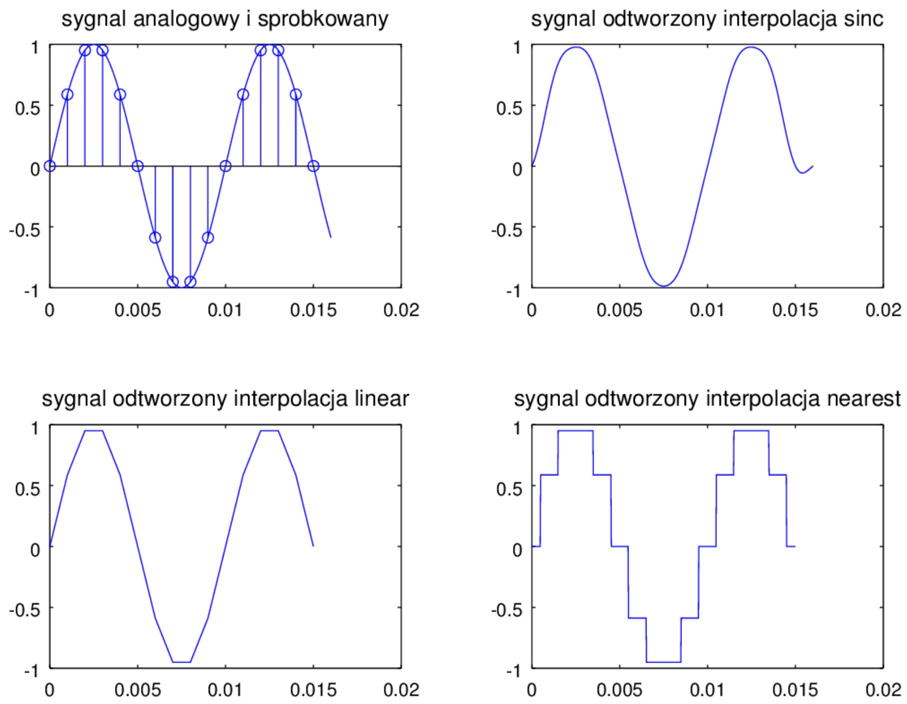


Rysunek 9: Przebiegi o różnej częstotliwości próbkowane z równą częstotliwością wyświetlone w dziedzinie czasu

Instrukcja 2, zadanie 5 Rekonstrukcja sygnałów cyfrowych

M-plik użyty do generacji wykresów:

```
%sampling frequency
fsa=44000;
fsd=1000;

%samples vector
nd=[0:15];
na=[0:44*length(nd)];

%signal frequency
f=100;

%time vectors
ta=na/fsa;
td=nd/fsd;

%signal model
xa=sin(2*pi*f*ta);
xd=sin(2*pi*f*td);

%reconstructions
xl=interp1(td, xd, ta, "linear");
xn=interp1(td, xd, ta, "nearest");

xs=zeros(size(ta));
for  k = 1:length(td);
st = xd(k)*sinc(fsd*(ta-td(k)));
xs = xs + st;
end

hold on

subplot(222)
plot(ta, xs, 'r')
hold on
plot(ta, xa)
title("sygnal odtworzony interpolacja sinc"
, "fontsize", 12)

subplot(223)
plot(ta, xl)
title("sygnal odtworzony interpolacja linear"
, "fontsize", 12)

subplot(224)
plot(ta, xn)
title("sygnal odtworzony interpolacja nearest"
, "fontsize", 12)

subplot(221)
stem(td, xd)
hold on
plot(ta, xa)
title("sygnal analogowy i sprobkowany"
, "fontsize", 12)
```



Rysunek 10: Sygnały zrekonstruowane za pomocą trzech algorytmów

Z wybranych algorytmów najbliżej oryginalnego jest przybliżenie za pomocą funkcji sinc. Interpolacja typu linear wprowadza dużo składowych o wysokiej częstotliwości. natomiast interpolacja nearest nie nadaje się w zasadzie do niczego.

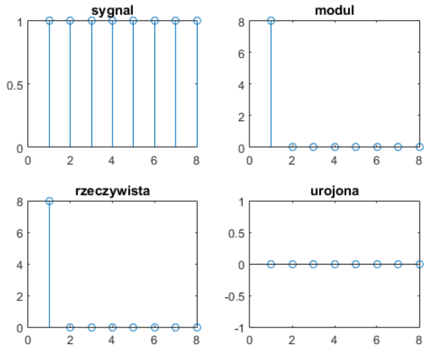
Instrukcja 3. Transformata Fourriera

Instrukcja 3, zadanie 1 Podstawy DFT

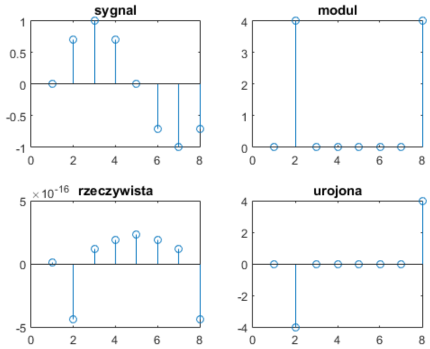
Równania sygnałów:

a) $x = 1$, długość sygnału 8 próbek
b) $x = \sin(2\pi*2000*t)$, długość sygnału 8 próbek
c) $x = \sin(2\pi*4000*t)$, długość sygnału 8 próbek
d) $x = \sin(2\pi*6000*t)$, długość sygnału 8 próbek
e) $x = \sin(2\pi*8000*t)$, długość sygnału 8 próbek
f) $x = \cos(2\pi*4000*t)$, długość sygnału 8 próbek
g) $x = \cos(2\pi*8000*t)$, długość sygnału 8 próbek
h) $x = \sin(2\pi*4000*t + \pi/8)$, długość sygnału 8 próbek
i) $x = \sin(2\pi*8000*t + \pi/8)$, długość sygnału 8 próbek
j) $x = -1$, długość sygnału 8 próbek
k) $x = 1$, długość sygnału 16 próbek
l) $x = \sin(2\pi*2000*t)$, długość sygnału 16 próbek
m) $x = \sin(2\pi*2000*t)$, długość sygnału 18 próbek
n) $x = \sin(2\pi*2000*t)$, długość sygnału 20 próbek
o) $x = j*\sin(2\pi*4000*t)$, długość sygnału 8 próbek
p) $x = j*\cos(2\pi*4000*t)$, długość sygnału 8 próbek
q) $x = \sin(2\pi*4000*t) + j*\sin(2\pi*4000*t)$, długość sygnału 8 próbek
r) $x = \sin(2\pi*4000*t) + j*\cos(2\pi*4000*t)$, długość sygnału 8 próbek

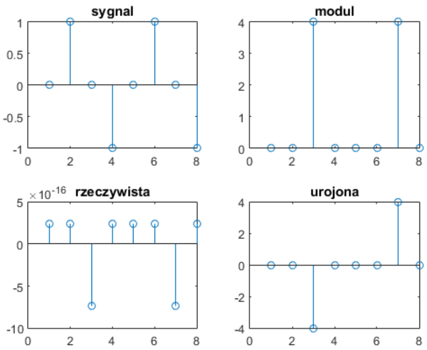
Różnica fazy nie wpływa na moduł transformaty Fourriera sygnału, zmienia natomiast jej część rzeczywistą co widać wyraźnie w przypadku sygnałów f i h z rysunku 11.Częstotliwość sygnału wyraźnie wpływa na położenie wysokiego prążka w module tranformaty.Transformata sygnalu urojonego (Rysunek 12, g) daje zamienione wartości dla składników rzeczywistego i urojonego w prównaniu dla transformaty analogicznego sygnału rzeczywistego (Rysunek 11, c).



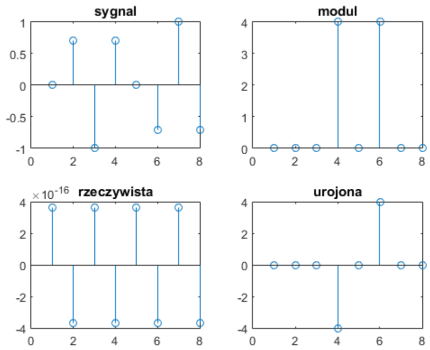
(a)



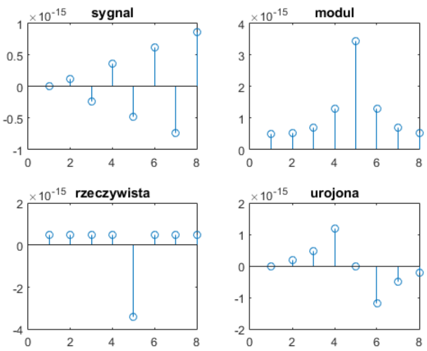
(b)



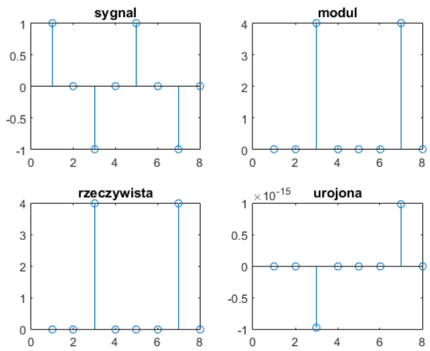
(c)



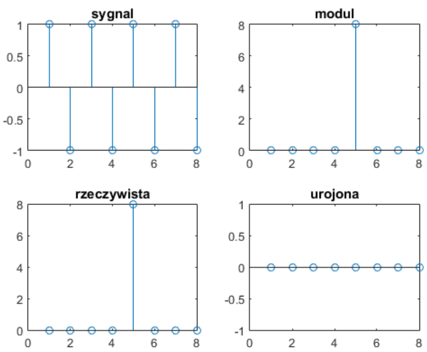
(d)



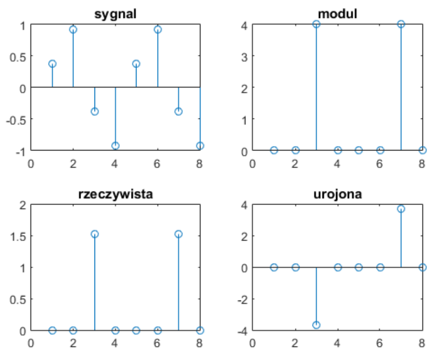
(e)



(f)

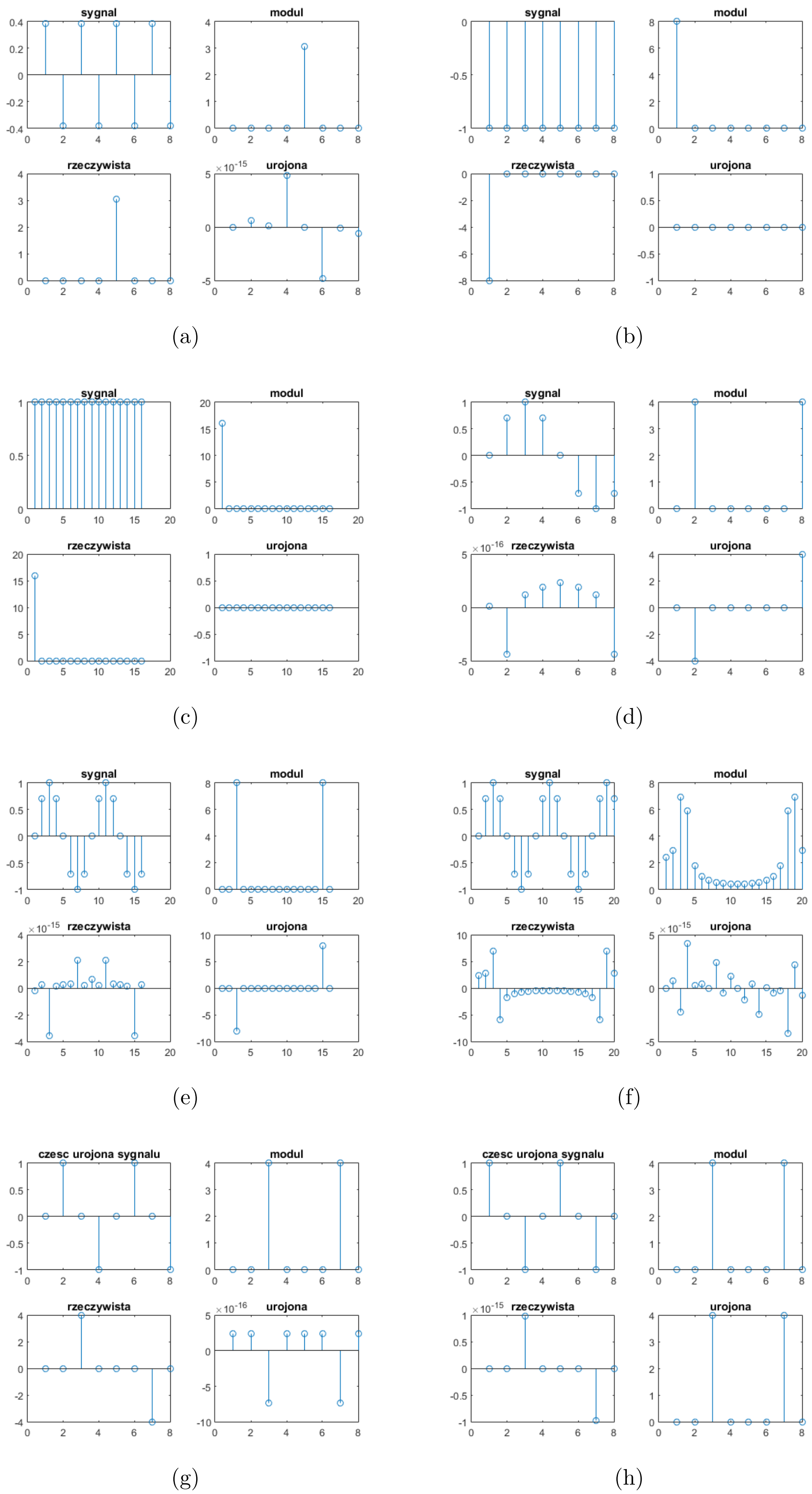


(g)

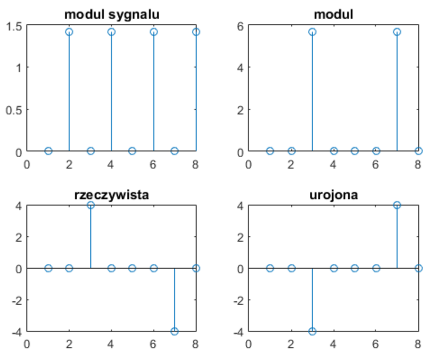


(h)

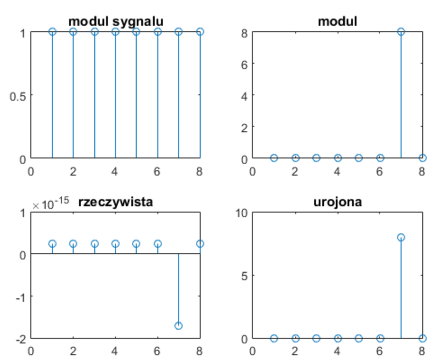
Rysunek 11: Wyniki transformacji fourriera a - h



Rysunek 12: Wyniki transfpormacji Fourriera i - p



(a)

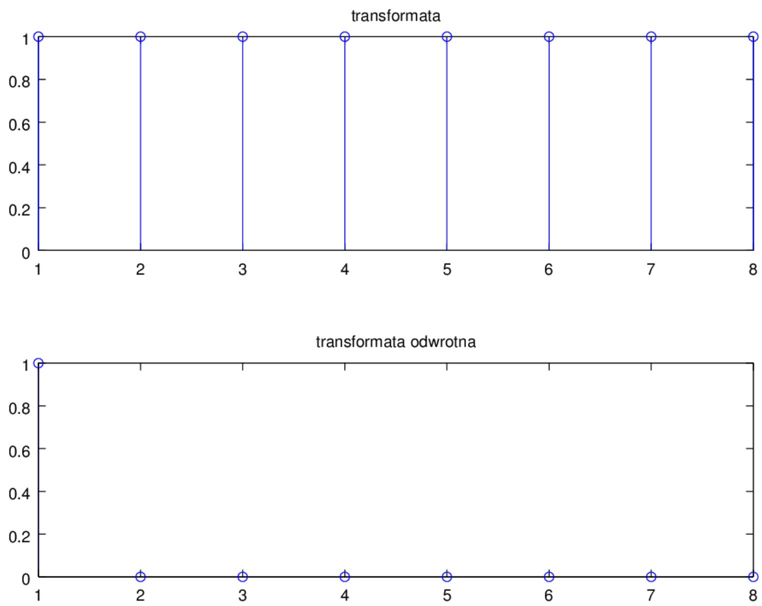


(b)

Rysunek 13: Wyniki transformaty Fourriera q - r

Instrukcja 3, zadanie 2 Odwrotna DFT

```
figure
subplot(211);
stem(ones(1,8));
title("transformata");
subplot(212);
stem(abs(ifft(ones(1,8))));
title("transformata odwrotna");
```



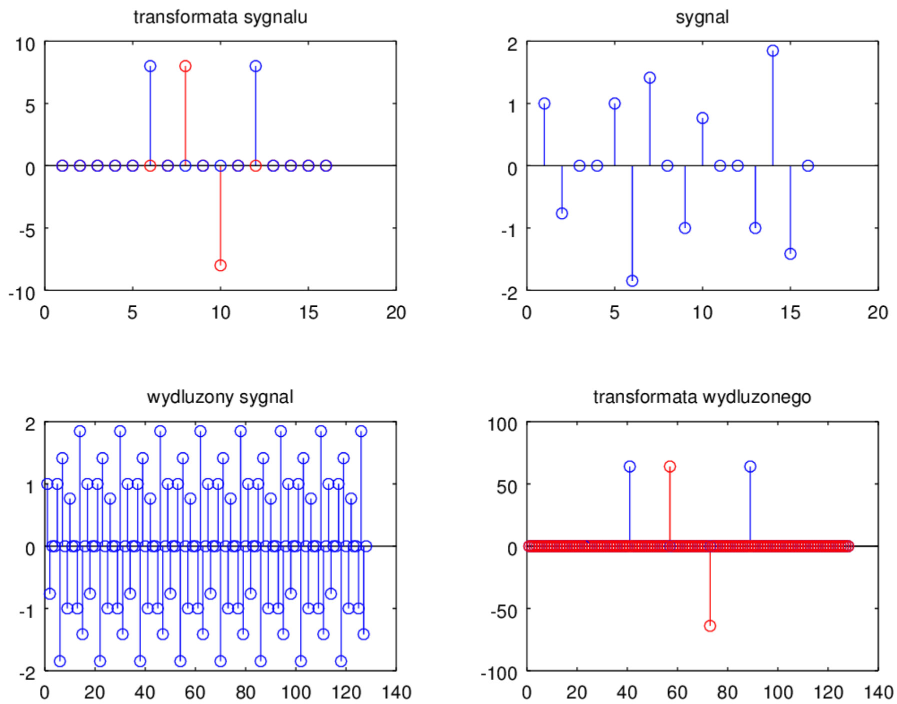
Rysunek 14

Aby uzyskać sygnał dla którego wszystkie prążki widma będą równe jeden, wystarczy obliczyć transformatę odwrotną takiego wektora.

Instrukcja 3, zadanie 3 Rekonstrukcja sygnału z jego transformaty

kod użyty do rekonstrukcji:

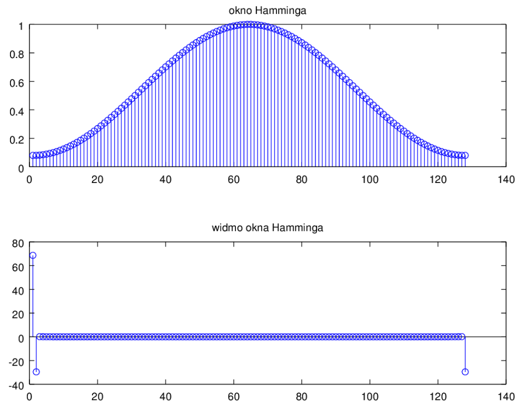
```
a = [0,0,0,0,0,8,0,j*8,0,
-j*8,0,8,0,0,0,0];
x = ifft(a);
x2 = [x,x,x,x,x,x,x,x];
a2 = fft(x2);
figure
subplot(221);
stem(real(a));
hold on
stem(imag(a), 'r');
title("transformata sygnalu");
subplot(222);
stem(x);
title("sygnal");
subplot(223);
stem(x2);
title("wydluzony sygnal");
subplot(224);
stem(real(a2));
hold on
stem(imag(a2), 'r');
title("transformata wydluzonego");
```



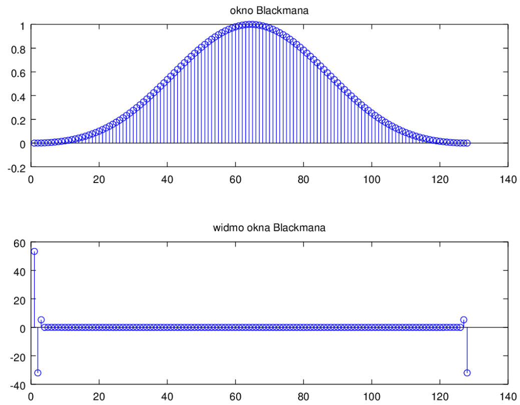
Rysunek 15

Po wprowadzeniu transformaty synału wystarczy obliczyć jej odwrotną transformatę. Po parokrotnym skopiowaniu sygnału wynikowego i obliczeniu kontrolnej transformaty dochodzimy do wniosku, że metoda ta działa.

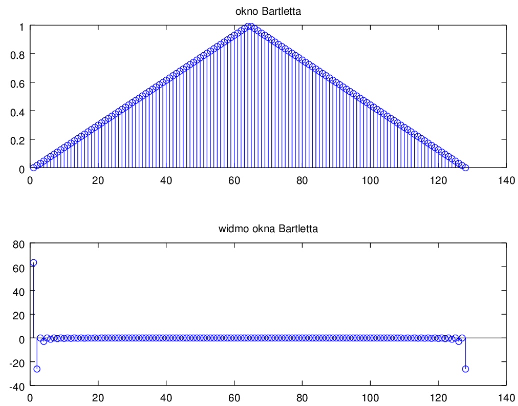
Instrukcja 3, zadanie 4 Okna Hamminga, Bartletta i Blackmana



(a)



(b)



(c)

Rysunek 16: Kształty okien i ich widma

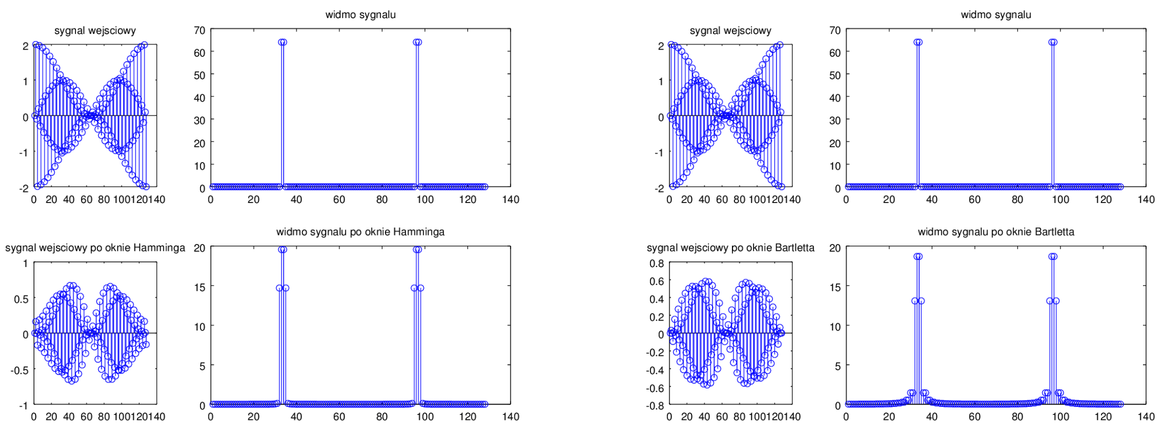
Okno Bartletta jest na pierwszy rzut oka najprostsze i wydaje się być najmniej przydatne z powyższych okien. Najbardziej równomierne widmo ma natomiast okno Hamminga i wydaje się być najbardziej użyteczne.

Instrukcja 3, zadanie 5 Uzycie okien do obserwacji widm przebiegów

Obserwowane sygnały: Skrypt użyty do generacji przebiegów:

a) $x = \sin(2\pi*4000*t) + \sin(2\pi*4125*t)$
b) $x = \sin(2\pi*4000*t) + \sin(2\pi*4250*t)$
c) $x = \sin(2\pi*4000*t) + \sin(2\pi*4500*t)$
d) $x = \sin(2\pi*4062*t)$

```
%sampling frequency
fs=16000;

%samples vector
n=[0:127];

%time vectors
t=n/fs;

%signal model
xa=sin(2*pi*4000*t)+sin(2*pi*4125*t);
xb=sin(2*pi*4000*t)+sin(2*pi*4250*t);
xc=sin(2*pi*4000*t)+sin(2*pi*4500*t);
xd=sin(2*pi*4062*t);

%windows
ham = rot90(hamming(128));
bla = rot90(blackman(128));
bar = rot90(bartlett(128));

figure
subplot(231);
stem(xd);
title("sygnal wejsciowy");
subplot(234);
stem(xd.*ham);
title("sygnal wejsciowy po oknie Hamminga");
subplot(2,3,[2,3]);
stem(abs(fft(xd)));
title("widmo sygnalu");
subplot(2,3,[5,6]);
stem(abs(fft(xd.*ham)));
title("widmo sygnalu po oknie Hamminga");

figure
subplot(231);
stem(xd);
title("sygnal wejsciowy");
subplot(234);
stem(xd.*bla);
title("sygnal wejsciowy po oknie Blackmana");
subplot(2,3,[2,3]);
stem(abs(fft(xd)));
title("widmo sygnalu");
subplot(2,3,[5,6]);
stem(abs(fft(xd.*bla)));
title("widmo sygnalu po oknie Blackmana");

figure
subplot(231);
stem(xd);
title("sygnal wejsciowy");
subplot(234);
stem(xd.*bar);
title("sygnal wejsciowy po oknie Bartletta");
subplot(2,3,[2,3]);
stem(abs(fft(xd)));
title("widmo sygnalu");
subplot(2,3,[5,6]);
stem(abs(fft(xd.*bar)));
title("widmo sygnalu po oknie Bartletta");
```

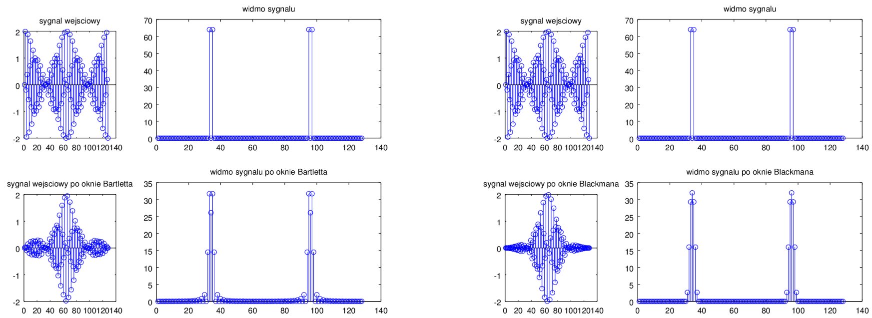


(a) (b)

Rysunek 17: Obserwacja sygnału A przez okna Hamminga i Bartletta

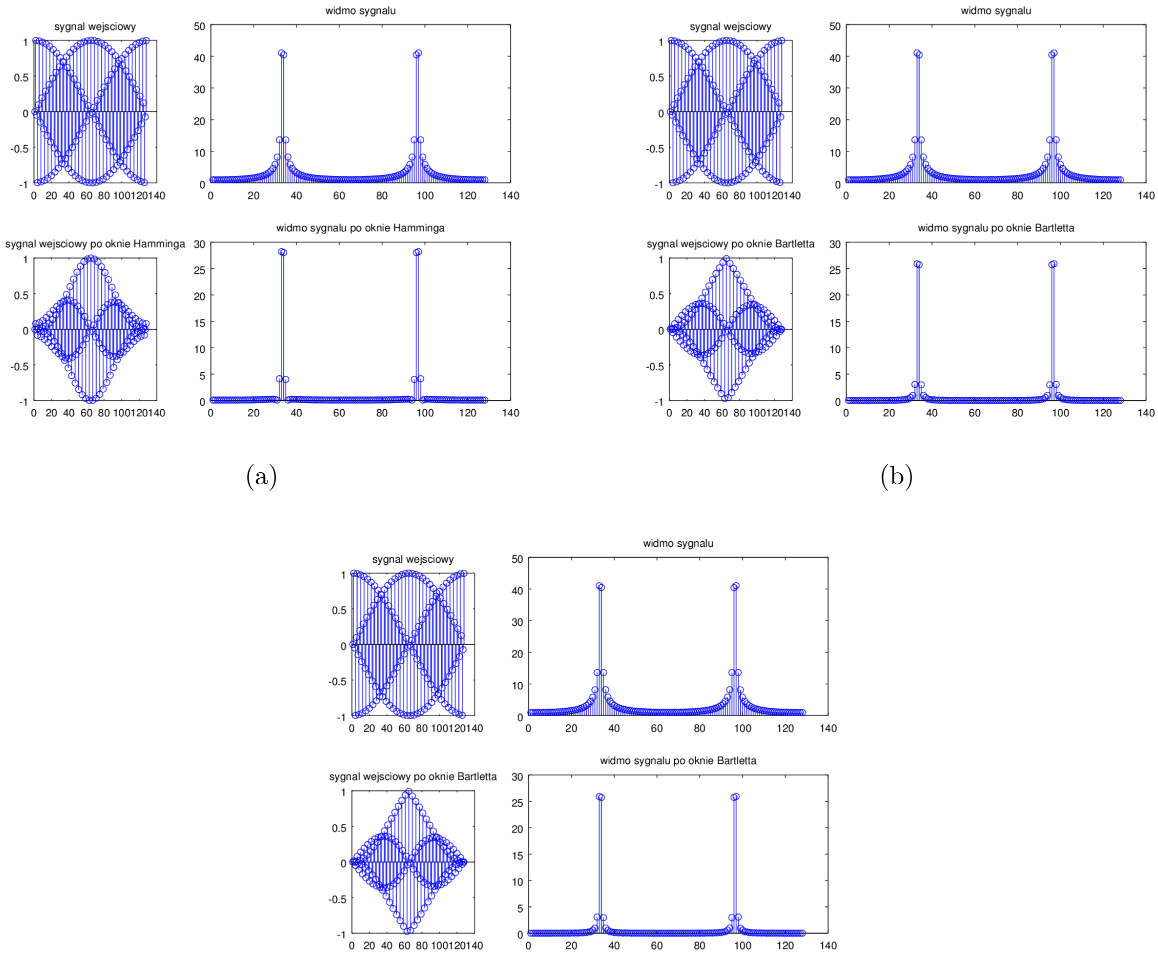
Widoczna jest większa przydatność okna Hamminga, gdyż okno Bartletta wprowadza zbyt duże zniekształcenia.

Zarówno okno Bartletta i Blackmana wprowadzają zauważalne zniekształcenia sygnału, lecz okno Blackmana jest odrobinę lepsze pod tym względem.



(a) (b)

Rysunek 18: Obserwacja sygnału B przez okna Bartletta i Blackmana



(a) (b)

(c)

Rysunek 19: Obserwacja sygnału D przez okna Hamminga, Bartletta i Blackmana

Dla poprzednich sygnałów nie występował przeciek widma ze względu na częstotliwości sygnałóW dopasowane do częstotliwości próbkowania. Okna skutkowały wtedy tylko wprowadzeniem zniekształceń. W przypadku sygnału D występuje duży przeciek i okna mogą mu zapobiec. Tak jak przewidywałem, okno Hamminga okazuje się być najbardziej przydatne

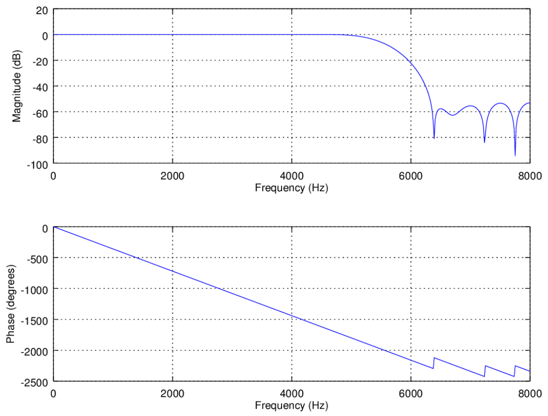
Jak widać, okna moga wprowadzać niepożądane czestotliwości i czynić pomiar mniej dokładnym. W większości przypadków jednak okażą się bardzo pomocne.

Instrukcja 4. Filtry SOI

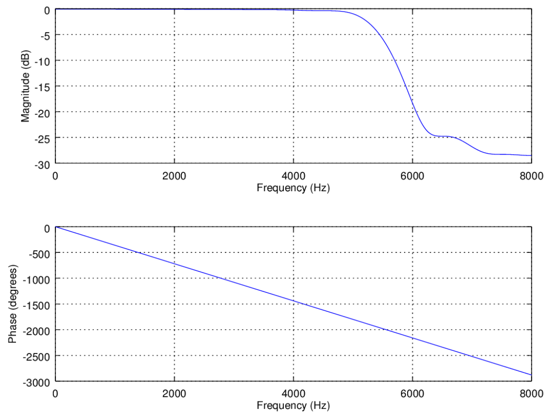
Instrukcja 4, zadanie 1 Filtr dolnoprzepustowy

Skrypt użyty do projektowania filtru i przefiltrowania sygnału kwadratowego:

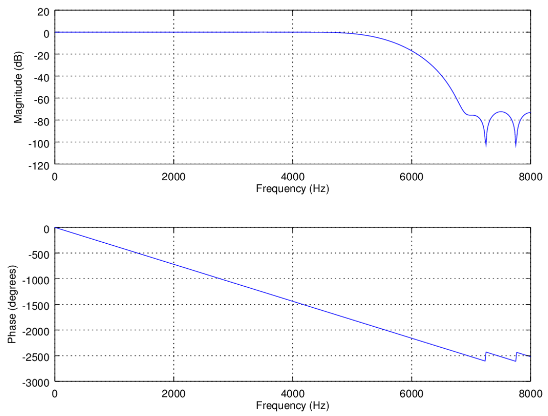
```
n = [-16:16]; %rzad filtru

ft = 5500; %czestotliwosc graniczna
fs = 16000;

wg = 2*pi*ft/fs;
h = (wg/pi)*sinc(wg*n/pi); %odpowiedz impulsowa
w = blackman(33)';
hw = h.*w;
freqz(hw,1,512,fs);

%samples vector
na=[0:127];

%signal frequency
f=500;

%time vectors
t=na/fs;

%signal model
a=sin(2*pi*f*t);

xa=sign(a);

xf = filter2(hw, xa);

figure
subplot(211)
plot(xa)
subplot(212)
plot(xf)
```



(a) Okno Hamminga



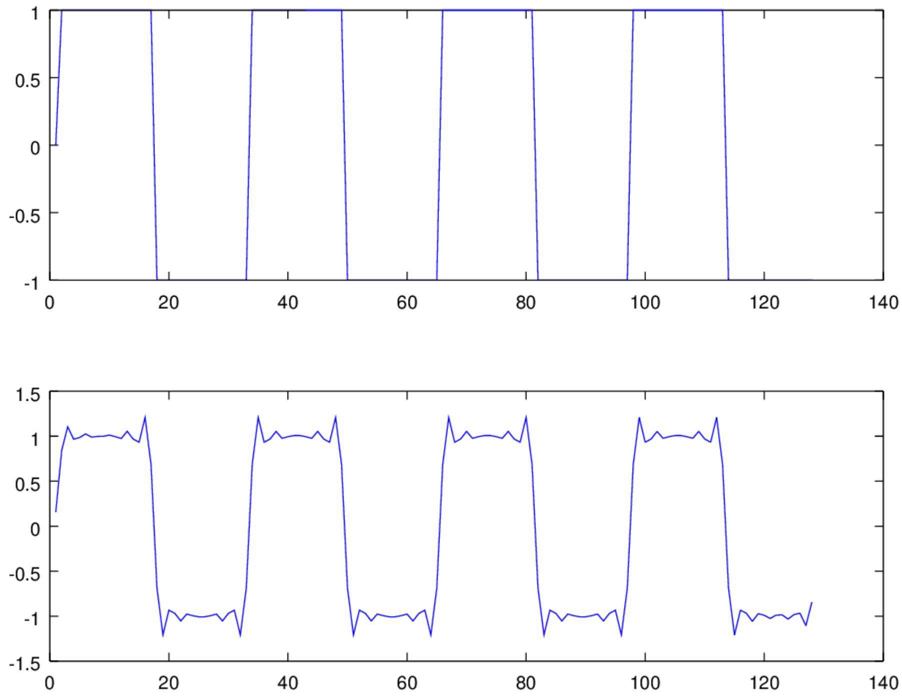
(b) Okno Bartletta



(c) okno Blackmana

Rysunek 20: Charakterystyki projektowanych filtrów rzędu 17

W przypadku filtru 17 rzędu okno Blackmana zapewnia największe tłumienie w paśmie zaporowym poniżej -80 dB ze stosunkowo niewielką amplitudą listków bocznych w porównaniu do okna Hamminga. Okno Bartleta ma najmniejszą amplitudę listków bocznych, alecz również niewielkie tłumienie.



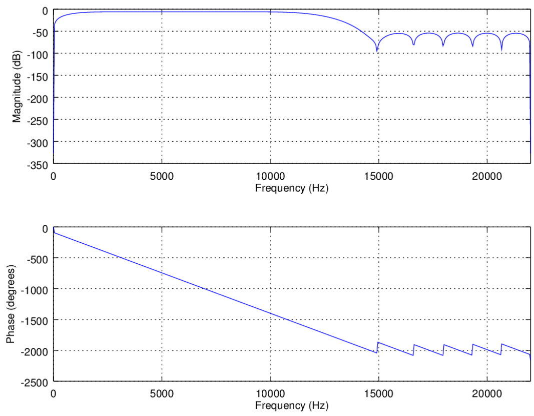
Rysunek 21: Przebieg kwadratowy i wynik filtrowania dolnoprzepustowego z oknem Blackmana

Wybrałem okno Blackmana ponieważ daje ono niską amplitudę listków bocznych. Aby uzyskać wymagane 70 dB tłumienia w paśmie zaporowym zwiększyłem rząd filtru z 17 do 33.

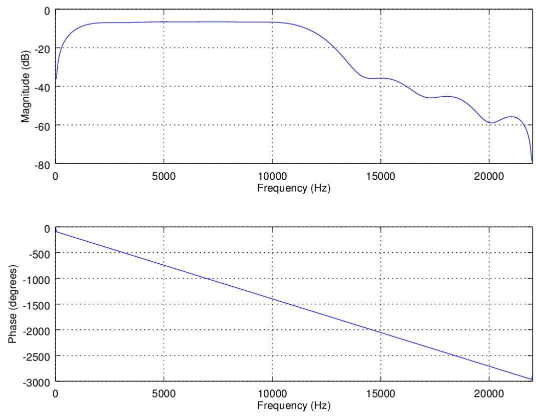
Instrukcja 4, zadanie 2 Filtr pasmowo-przepustowy

Skrypt użyty do projektowania filtru:

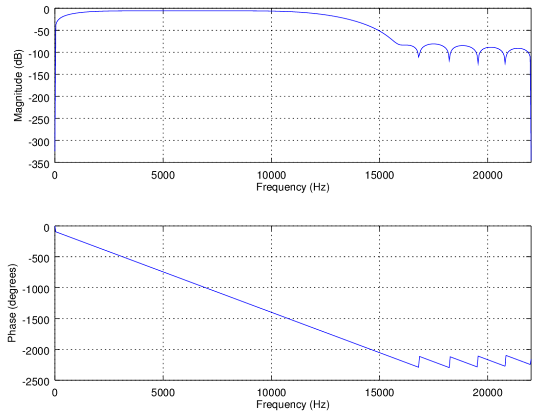
```
n = [-16:16]; %rzad filtru

ft = 6000; %czestotliwosc graniczna
fs = 44000;
t=n/fs;

wg = 2*pi*ft/fs;

h = (wg/pi)*sinc(wg*n/pi).*sin(2*pi*6000*t); %odpowiedz impulsowa
w = hamming(33)';
hw = h.*w;
freqz(hw,1,512,fs);

%samples vector
na=[0:511];

%signal frequency
f=500;

%time vectors
ta=na/fs;
```



(a) Okno Hamminga



(b) Okno Bartletta



(c) okno Blackmana

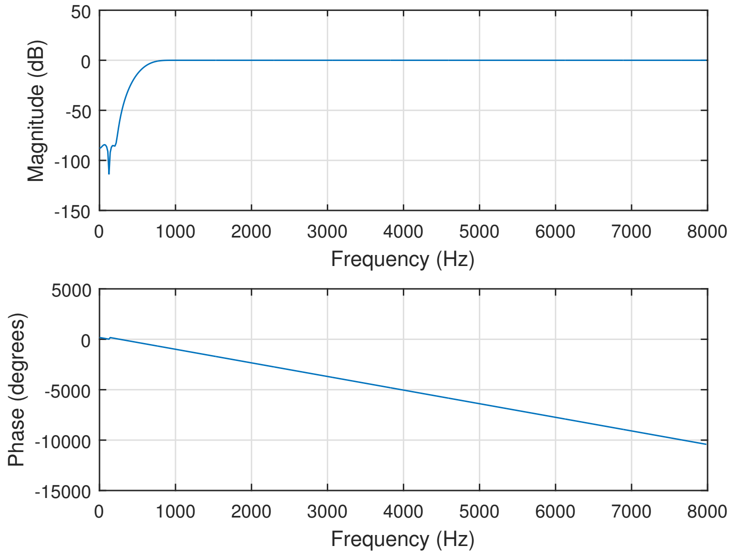
Rysunek 22: Charakterystyki projektowanych filtrów rzędu 33

Tak jak w poprzednim punkcie, Okno Bartletta charakteryzuje się najkorzystniejszą charakterystyką fazową i małą amplituda listków bocznych. Okna Hamminga i Blackmana dają za to większe tłumienie w paśmie zaporowym.

Instrukcja 4, zadanie 3 Filtr górnoprzepustowy

Skrypt użyty do projektowania filtru:

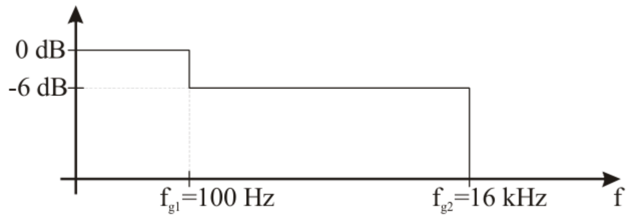
```
n=120;
ft = 600 /8000;
w=blackman(n+1); %funkcja okna
h=fir1(n, ft, 'high', w);
H = fft(h);
freqz(h,1,512,16000)
```



Rysunek 23: Filtr górnoprzepustowy

Powyższy filtr został wykonany w Matlabie. Poprzednie zadania wykonywane były w Octave. Zostałem zmuszony do przesiadki przez słabe wsparcie Octave dla DSP.

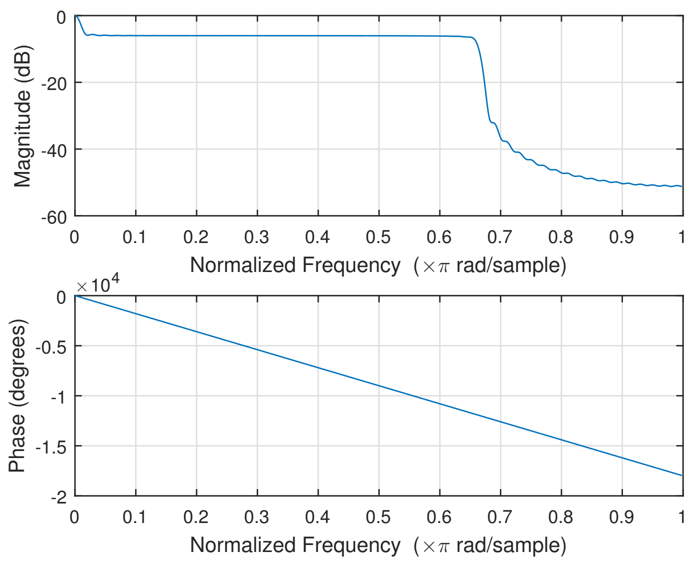
Instrukcja 4, zadanie 4   Filtr schodkowy dolnoprzepustowy



Rysunek 24: Zadany kształt charakterystyki filtru

Skrypt użyty do projektowania filtru:

```
n=200;
fs=48000;
ft1 = 100*(2/fs);
ft2 = 16000*(2/fs);
w=bartlett(n+1);
h1=impz(fir1(n, ft1, 'low', w));
h2=impz(fir1(n, ft2, 'low', w));
h=(h1+h2)/2;
freqz(h)
```



Rysunek 25: Charakterystyka zaprojektowanego filtru

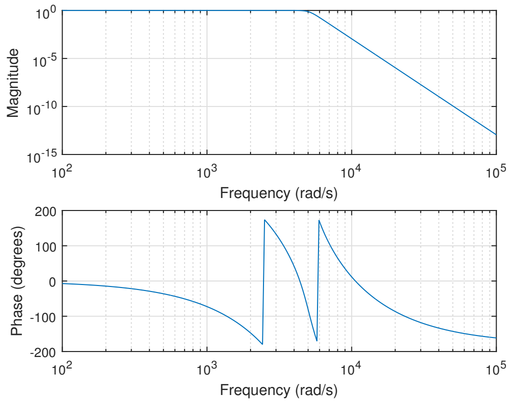
Aby sprostać wymaganiom zadania zwiększyłem rząd filtru do 200.

Instrukcja 5. Filtry NOI

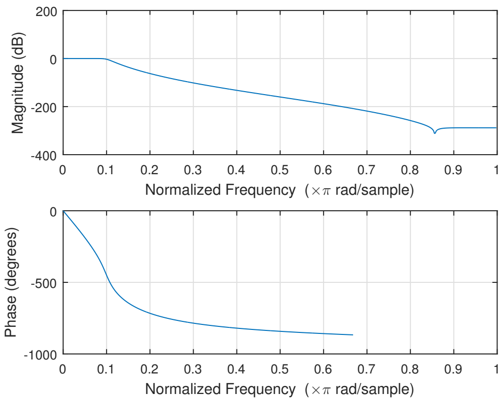
Instrukcja 5, zadanie 1 Filtr dolnoprzepustowy

Skrypt użyty do projektowania filtrów:

```
n = 10; %rzad
fs = 16000; %probkowanie
teta = 0.1; %graniczna
%czestotliwosci analogowe
omega = 2*fs*tan(pi*teta/2);
%znormalizowany prototyp filtru analogowego
[z, p, k] = buttap(n);
%figure(1);
%zplane(z,p);
%transmitancja filtru znormalizowanego
[num, den] = zp2tf(z,p,k);
%denormalizacja
[numt, dent] = lp2lp(num, den, omega);
figure(2);
freqs(numt,dent);
%transmitancja filtru cyfrowego
[BB, AA] = bilinear(numt, dent, fs);
figure(3);
freqz(BB,AA);
```

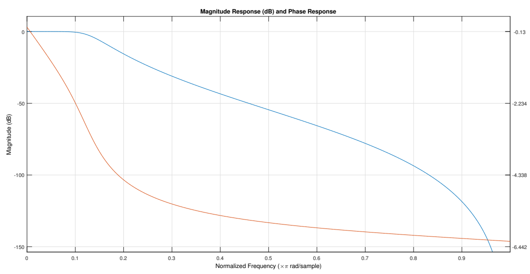


(a) Filtr analogowy

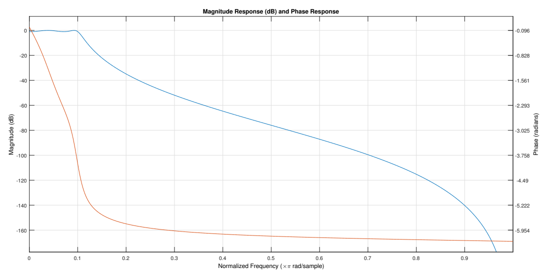


(b) Filtr cyfrowy

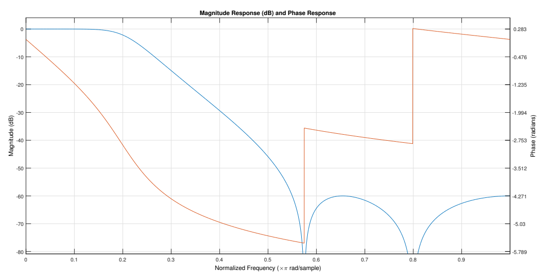
Rysunek 26: Charakterystyki amplitudowe i fazowe filtrów



(a) Butterworth



(b) Chebyshev 1



(c) Chebyshev 2

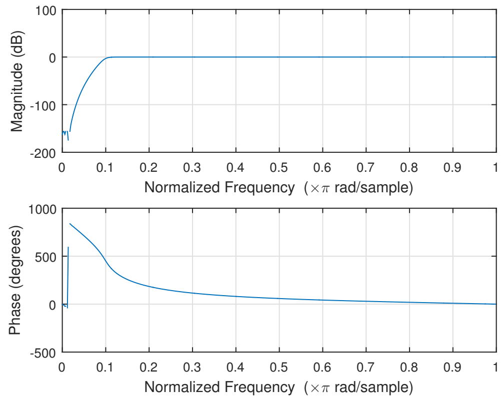
Rysunek 27: Filtry projektowane z użyciem różnych prototypów

Prototyp Butterwortha powoduje, że filtr ma łagodniejsze charakterystyki i Chebyshev 1 typu jest zbliżony. Prototyp Chebysheva 2 typu ma dużą amplitudę listków bocznych, ale też bardziej stromą charakterystykę w paśmie przejściowym.

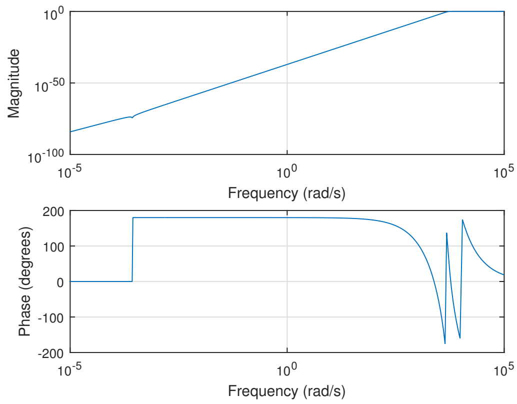
Instrukcja 5, zadanie 2 Filtr górnoprzepustowy

Skrypt użyty do projektowania filtrów:

```
n = 10; %rzad
fs = 16000; %probkowanie
teta = 0.1; %graniczna
%czestotliwosci analogowe
omega = 2*fs*tan(pi*teta/2);
%znormalizowany prototyp filtru analogowego
[z, p, k] = buttap(n);
%figure(1);
%zplane(z,p);
%transmitancja filtru znormalizowanego
[num, den] = zp2tf(z,p,k);
%denormalizacja
[numt, dent] = lp2hp(num, den, omega);
figure(2);
freqs(numt,dent);
%transmitancja filtru cyfrowego
[BB, AA] = bilinear(numt, dent, fs);
figure(3);
freqz(BB,AA);
```

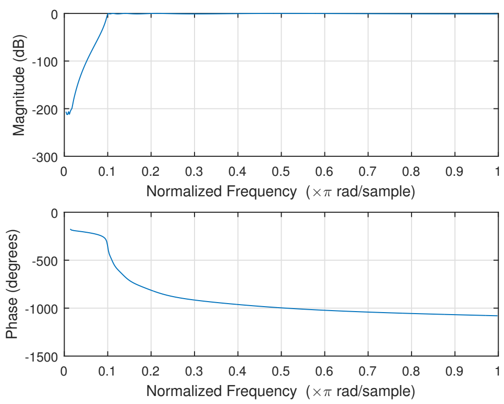


(a) Filtr cyfrowy

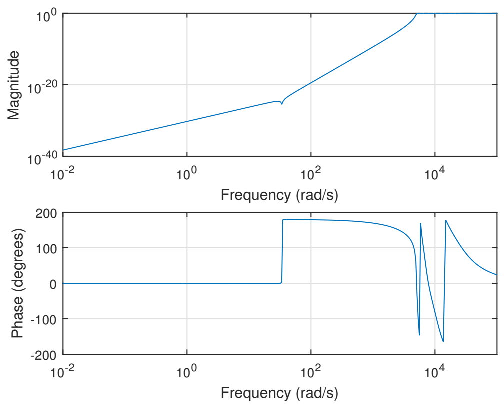


(b) Filtr analogowy

Rysunek 28: Filtr Butterwortha

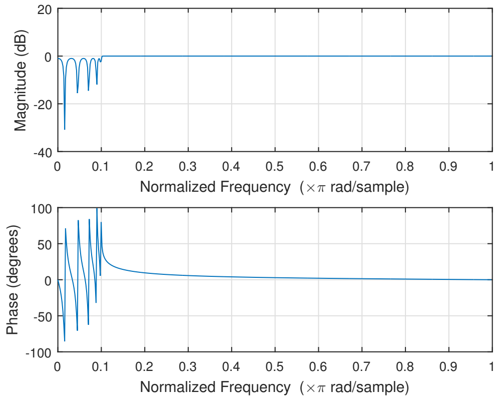


(a) Filtr cyfrowy

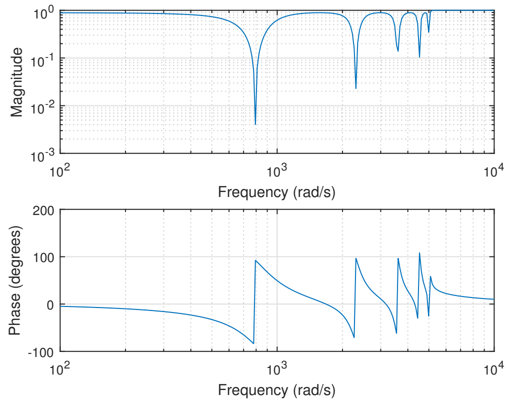


(b) Filtr analogowy

Rysunek 29: Filtr Chebysheva 1 typu



(a) Filtr cyfrowy



(b) Filtr analogowy

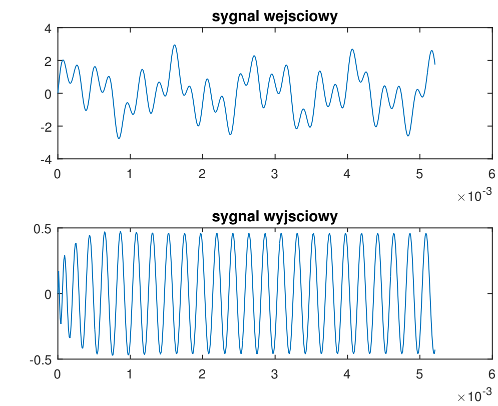
Rysunek 30: Filtr Chebysheva 2 typu

Aby filtr Chebysheva 2 typu był opłacalny, musiałby mieć wysoki rząd. Przy rzędzie 10 najbardziej użyteczny jest filtr Butterwortha ze względu na małą amplitudę listków w pasmie zaporowym oraz łagodne charakterystyki.

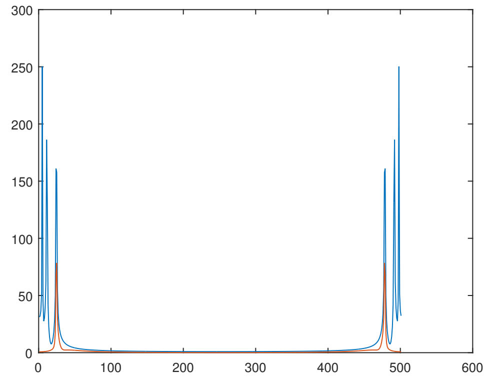
Instrukcja 5, zadanie 3 Filtracja NOI

Skrypt użyty do filtrowania sygnałów:

```
%sampling frequency
fs=96000;

%samples vector
n=[0:500];

%signal frequency
f1=800;
f2=2000;
f3=4500;

%time vectors
t=(n/fs);
t1=2*pi*t;

%signal model
x=sin(t1*f1)+sin(t1*f2)+sin(t1*f3);

rzad = 10; %rzad
fs = 16000; %probkowanie
teta = 0.1; %graniczna
%czestotliwosci analogowe
omega = 2*fs*tan(pi*teta/2);
%znormalizowany prototyp filtru analogowego
[z, p, k] = cheb2ap(rzad,1);
%figure(1);
%zplane(z,p);
%transmitancja filtru znormalizowanego
[num, den] = zp2tf(z,p,k);
%denormalizacja
[numt, dent] = lp2hp(num, den, omega);
%transmitancja filtru cyfrowego
[BB, AA] = bilinear(numt, dent, fs);

y=filter(BB, AA, x);

subplot(211);
plot(t, x);
title('sygnal wejsciowy');
subplot(212);
plot(t, y);
title('sygnal wyjsciowy');
```

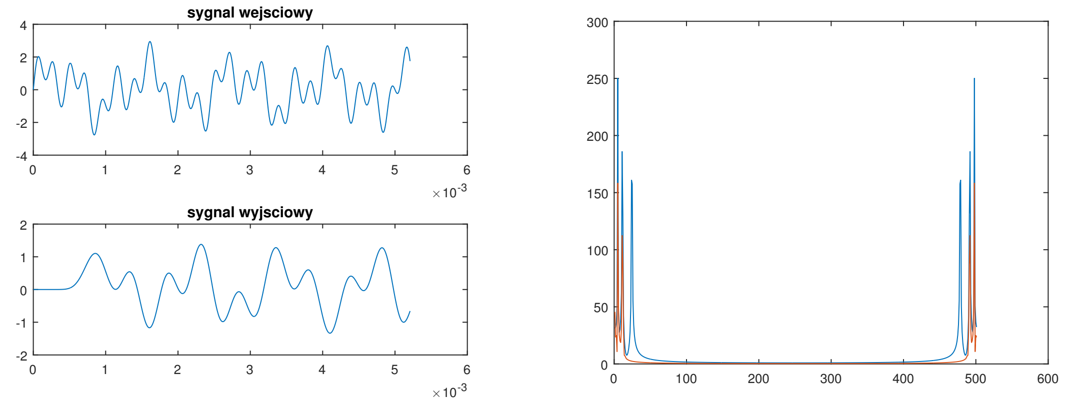


(a) Przebiegi


(b) Widma

Rysunek 31: Górnoprzepustowa filtracja Butterwortha

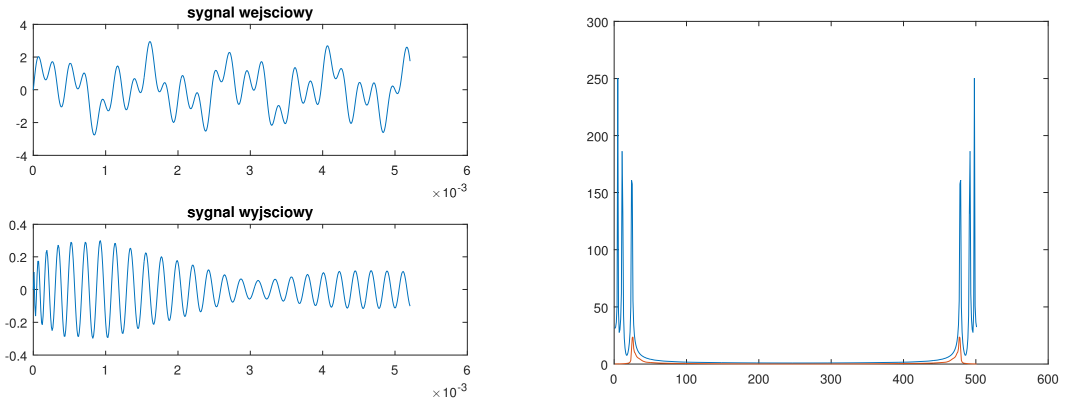
Bardzo czysta filtracja zostawiła jedną harmoniczną i bardzo znacznie stłumiła pozostałe. Bardzo wyraźnie widać to zarówno w przebiegu jak i w widmie sygnałów.



(a) Przebiegi (b) Widma

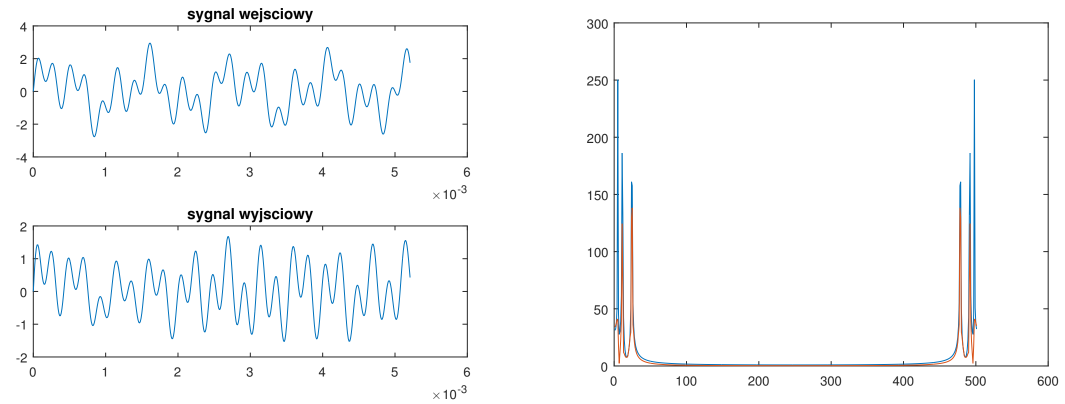
Rysunek 32: Dolnoprzepustowa filtracja Butterwortha

Tak jak w przypadku filtracji górnoprzepustowej, filtracja dolnoprzeputowa Butterwortha daje bardzo dobre rezultaty lekko tylko tłumiąc interesuące nas pasmo.



(a) Przebiegi (b) Widma

Rysunek 33: Górnoprzepustowa filtracja Chebysheva 1 typu



(a) Przebiegi (b) Widma

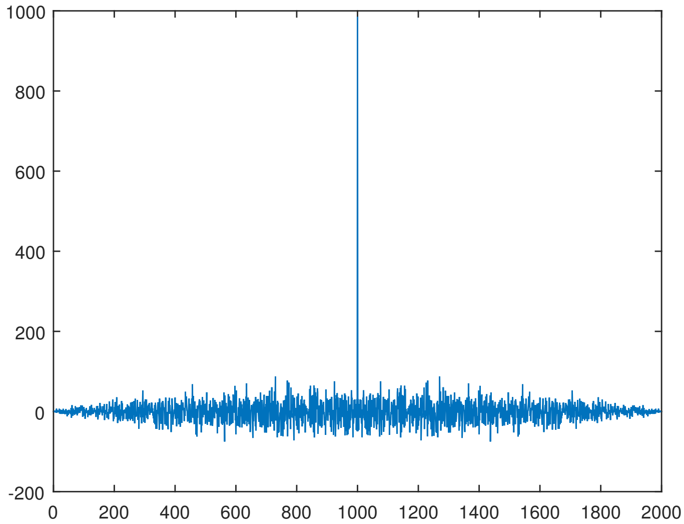
Rysunek 34: Górnoprzepustowa filtracja Chebysheva 2 typu

Filtr Chebysheva 1 typu daje w miarę przyzwoite wyniki. Filtrując niskie czestotliwości jednocześnie wprowadza znaczne zniekształcenia. Filtr Chebysheva 2 typu daje bardzo słabe wyniki.

Instrukcja 6. Analiza korelacyjna sygnałów

W tej sekcji nie zamieszczam zbednych skrawków koduograniczających się do wywoływania funkcji.

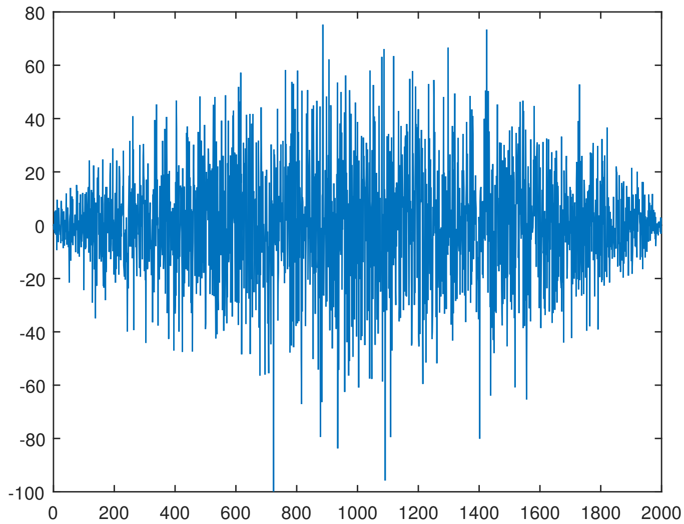
Instrukcja 6, zadanie 1 Autokorelacja szumu



Rysunek 35: Wyniki autokorelacji szumu

Ze wzgledu na swoj losowy charakter, autokorelacja sygnału szumowego skutkuje jedną duża wartością dla zerowego przesunięcia.

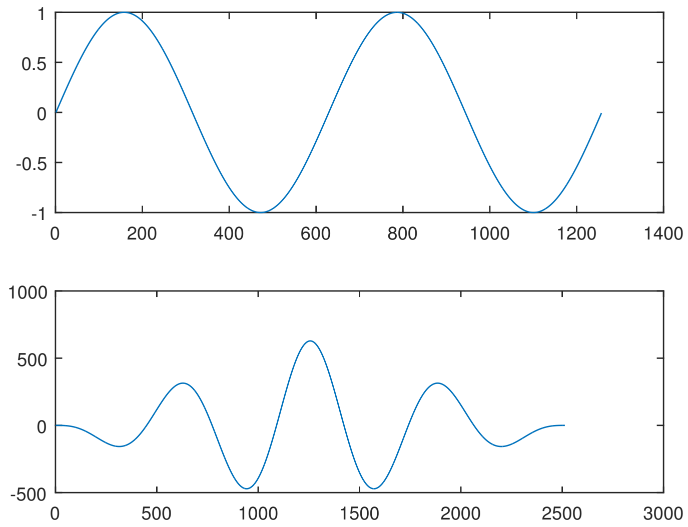
Instrukcja 6, zadanie 2 Korelacja dwóch sygnałów szumowych



Rysunek 36: Wyniki korelacji dwóch sygnałów szumowych

Z powodu szumowego charakteru obydwu sygnałów nie jesteśmy w stanie wyznaczyć żadnej korelacji.

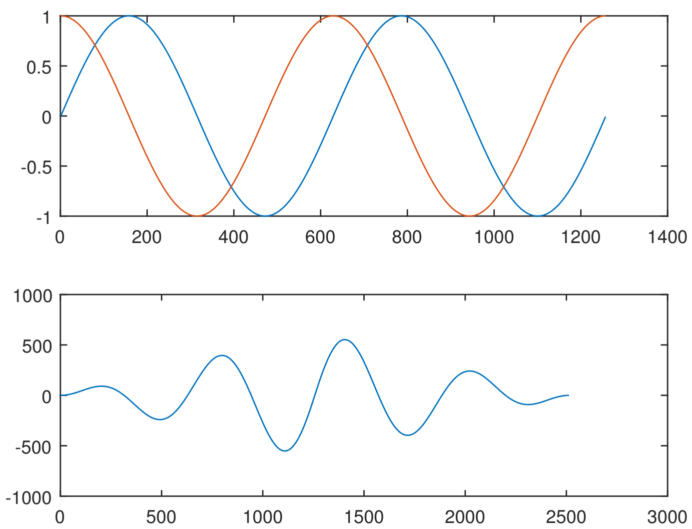
Instrukcja 6, zadanie 3 Autokorelacja sygnału sinusoidalnego



Rysunek 37: Wynik autokorelacji sygnału sinusoidalnego

Wynik autokorelacji świadczy o samopodobieństwie sygnału sinusoidalnego.

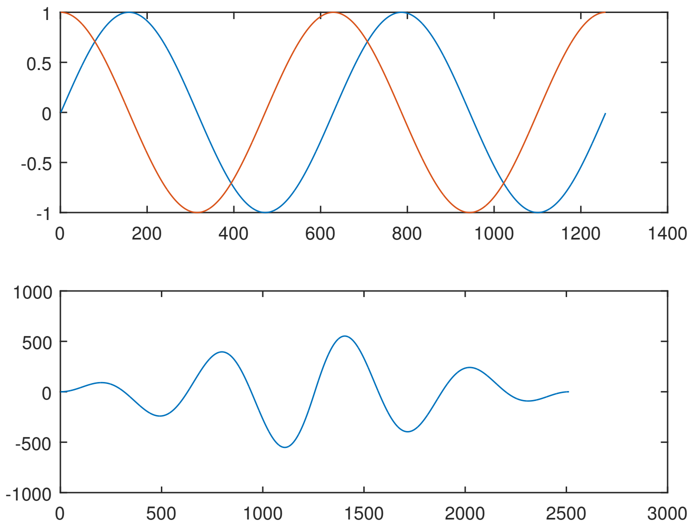
Instrukcja 6, zadanie 4 Korelacja sygnałów *sin*, oraz *cos*



Rysunek 38: Wynik korelacji sygnałów

Ponieważ sygnał kosinusoidalny jest tak podobny do sinusoidalnego, wynik korelacji nie dziwi i świadczy o podobieństwie tych sygnałów.

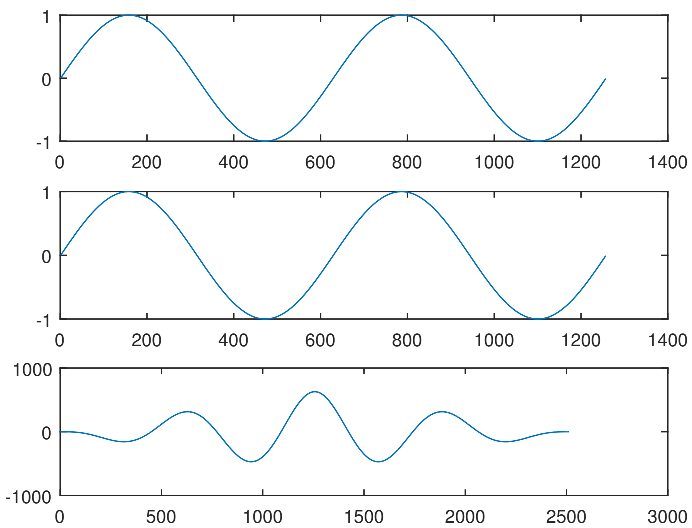
Instrukcja 6, zadanie 5 Korelacja sygnałów przesuniętych o pół fazy



Rysunek 39: Wynik korelacji sygnałów

Wynik jest identyczny jak przy korelacji z zadania 4, bo przesunięta sinusoida to to samo co kosinusoida.

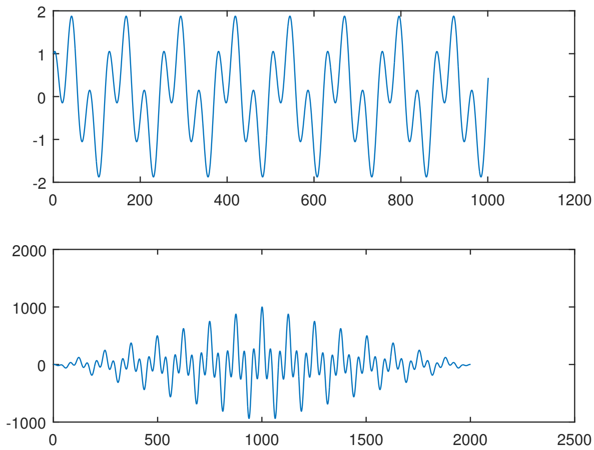
Instrukcja 6, zadanie 6 Korelacja sygnałów przesuniętych całą fazę



Rysunek 40: Wynik korelacji sygnałów

Wynik jest identyczny jak przy autokorelacji sinusoidy, ponieważ sygnały są identyczne.

Instrukcja 6, zadanie 7 Autokorelacja sygnału złożonego



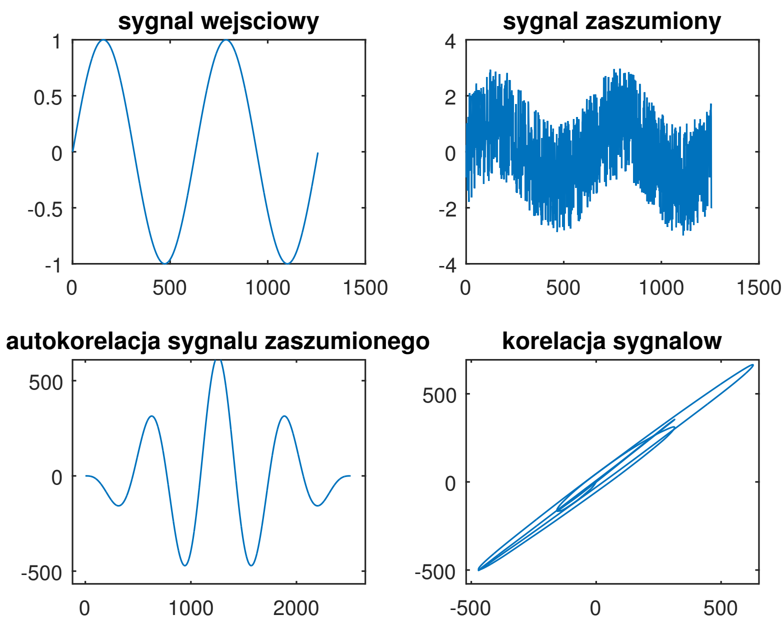
Rysunek 41: Wynik autokorelacji

Autkorelacja wykazuje samopodobieństwo, ponieważ sygnał jest okresowy.

Instrukcja 6, zadanie 8 Korelacja sygnałów zaszumionych

Skrypt użyty do generacji wykresów:

```
s=sin([0:0.01:4*pi]);
ss=s+rand(1,length(s))*0.2-0.1;

[c, lags] = xcorr(s);
[cc, lags1] = xcorr(s,ss);

subplot(221)
plot(s)
title('sygnal wejsciowy')
subplot(222)
plot(ss)
title('sygnal zaszumiony')

subplot(223)
plot(c)
title('autokorelacja sygnalu zaszumionego')
subplot(224)
plot(c,cc)
title('korelacja sygnalow')
```



Rysunek 42

Nawet sygnał zaszumiony jest samopodobny. Co więcej, korelacja sygnału przed i po dodaniem szumu wykazuje bardzo mocne podobieństwo. Pokazuje to jak przydatna jest opercja korelacji przy wyszukiwaniu wzorów.