به نام خدا

گزارش کار آزمایشگاه پردازش سیگنال دیجیتال

آزمایش ۲

آیدین روزبه - ۹۹۲۳۰۳۷

پاییز ۱۴۰۲

کد مربوط به تمام بخش های این آزمایش در فایل lab2.mat آورده شده است و برای اجرای درست آن لازم است سایر فایل ها که در بند بعدی شرح داده شده اند، در کنار یکدگیر باشند.

فایل audio_filter.mat: ضرایب فیلتر طراحی شده با ابزار fdatool برای فیلتر کردن صدا

فایل audio_filter.fda : فایل session مرتبط با فیلتر صدا

فایل fda.mat : ضرایب فیلتر طراحی شده با fdatool برای سیگنال بخش ۲-۲

فایل fda_filter.fda : فایل session مرتبط با فیلتر بخش ۲-۲

فایل sample-3 : ویس ۱۰ ثانیه ای استفاده شده برای پردازش صوت (فرکانس نمونه برداری اصلی این صدا برابر با 22996 هرتز و مدت زمان آن دقیقا برابر با ۱۰ ثانیه می باشد. به همین

بخش ۲-۱-الف) پیاده سازی تابع myconv

برای پیاده سازی تابع کانولوشن، طبق تعریف آن عمل کرده و با قرینه کردن، شیفت دادن و ضرب کردن (سمپل به سمپل) یکی از سیگنال ها در دیگری، حاصل را محاسبه می‌کنیم.

$$z[n] = x[n] * y[n] = \sum_{m=-\infty}^{m=+\infty} y[m]x[n-m]$$

سیگنال temp، سیگنال y را از طرفین به اندازه ((یک واحد کمتر از طول x)) با مقادیر صفر گسترش می‌دهد تا امکان شبیه سازی عملیات کانولوشن را فراهم سازد. (گسترش بیش از این لازم نیست زیرا طول سیگنال ها محدود است و بعد از شیفت یافتن بیشتر از طول یکی از سیگنال ها، عملا سیگنال شیفت یافته تماما در صفر ضرب می شود).

سیگنال نهایی اندازه ای برابر با len(x)+len(y)-1 دارد، به همین ترتیب سیگنال res با این ابعاد تعریف می-شود و مقادیر آن در یک حلقه، به روشی که در پارگراف قبلی توضیح داده شد، محاسبه می‌شوند.

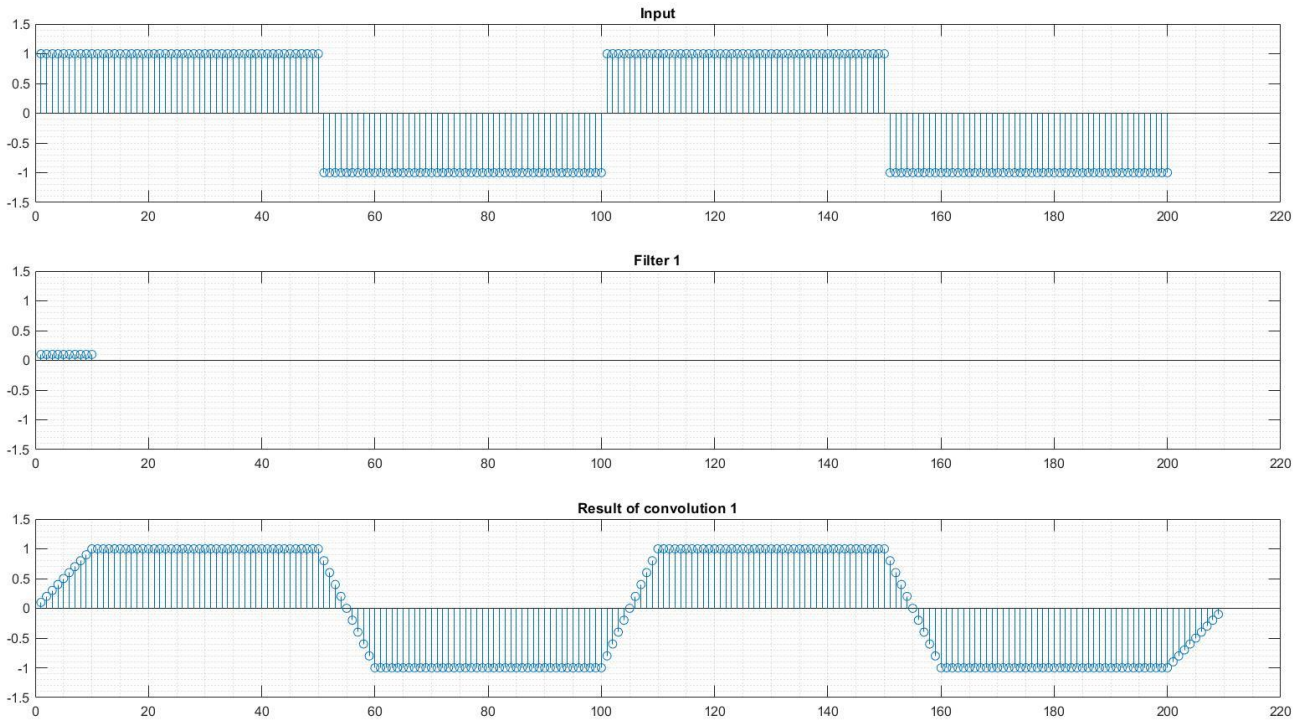
```
function res = myconv(x,y)
    lenX=size(x,2);
    lenY=size(y,2);
    fx=fliplr(x);
    disp(fx);

    res=zeros(1,lenX+lenY-1);

    temp = [zeros(1,lenX-1) , y , zeros(1,lenX-1)];

    for i=1:1:lenX+lenY-1
        res(i) = fx*temp(i:i+lenX-1)';
    end
end
```

بخش ۲-۱-ب) فیلتر شبه انتگرال گیر

با دستور ones، پاسخ ضربه فیلتر مورد نظر ساخته شده است.

```
%% 2-1-b)
sig = [ones(1,50) , -1*ones(1,50) , ones(1,50), -1*ones(1,50)];
fil1 = 1/10*ones(1,10);
res1 = myconv(sig,fil1);

figure(1);
subplot(3,1,1);
stem(sig)
title("Input");
axis([0 220 -1.5 1.5]);
grid minor;

subplot(3,1,2);
stem(fil1);
title("Filter 1");
axis([0 220 -1.5 1.5]);
grid minor;

subplot(3,1,3);
stem(res1);
title("Result of convolution 1");
axis([0 220 -1.5 1.5]);
grid minor;
```

نتایج شبیه سازی به صورت زیر می‌باشد



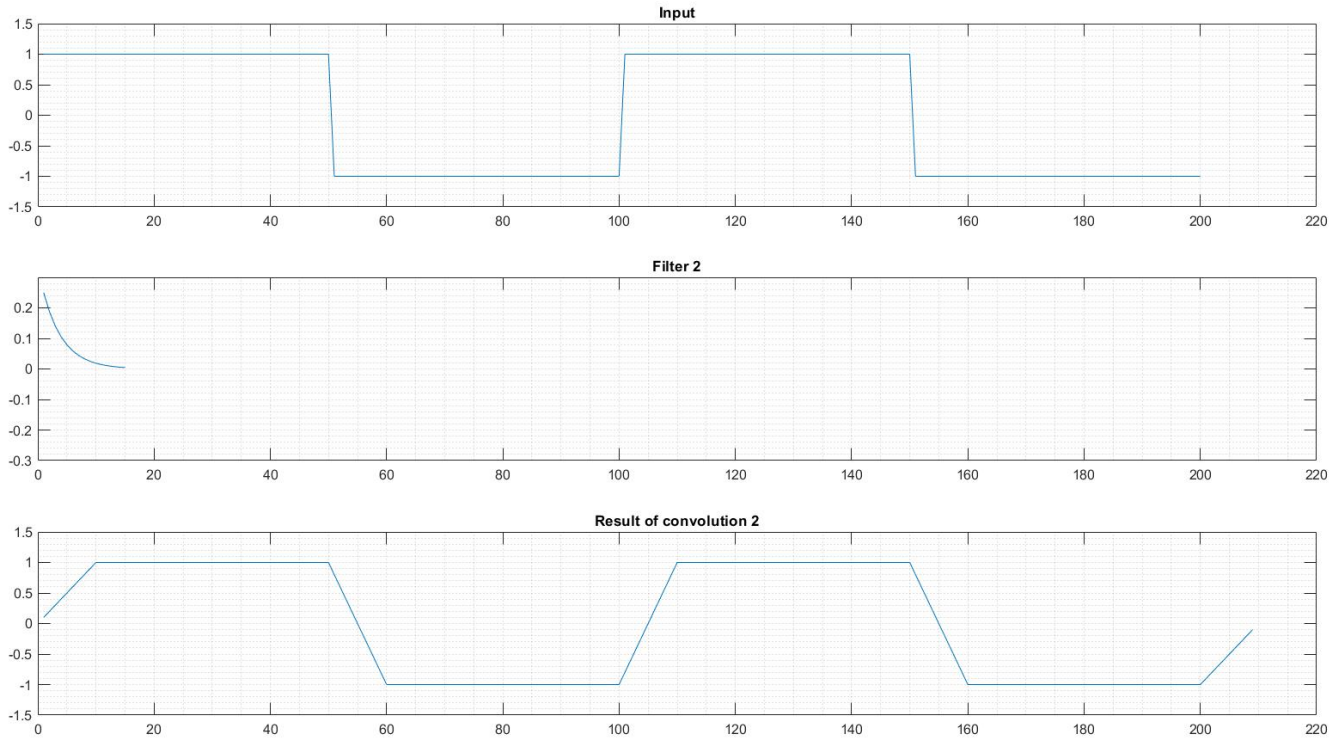
از آنجایی که فیلتر اعمال شده به این سیگنال، پایین گذر بوده (پاسخ فرکانسی = تابع sinc)، انتظار می‌رود خروجی، شامل محتوای فرکانس بالای کمتری باشد. همانطور که از شکل مشخص است، لبه هایی که پرش داشتند، پس از عبور از فیلتر دیگر مثل قبل تغییرات ناگهانی ندارند و با سرعت کمتر تغییر می‌کنند.

بخش ۲-۱-ج)

فرایند حل سوال کاملا مشابه حالت قبل است

```
%% 2-1-c)
fil2 =  zeros(1,15);
for i=1:1:15
    fil2(i)=0.25*(0.75^(i-1));
end

res2 = myconv(sig , fil2);

figure(2);
subplot(3,1,1);
plot(sig)
title("Input");
axis([0 220 -1.5 1.5]);
grid minor;

subplot(3,1,2);
plot(fil2);
title("Filter 2");
axis([0 220 -0.3 0.3]);
grid minor;

subplot(3,1,3);
plot(res1);
title("Result of convolution 2");
axis([0 220 -1.5 1.5]);
grid minor;
```

نتیجه شبیه سازی:



بخش ۲-۱-د)

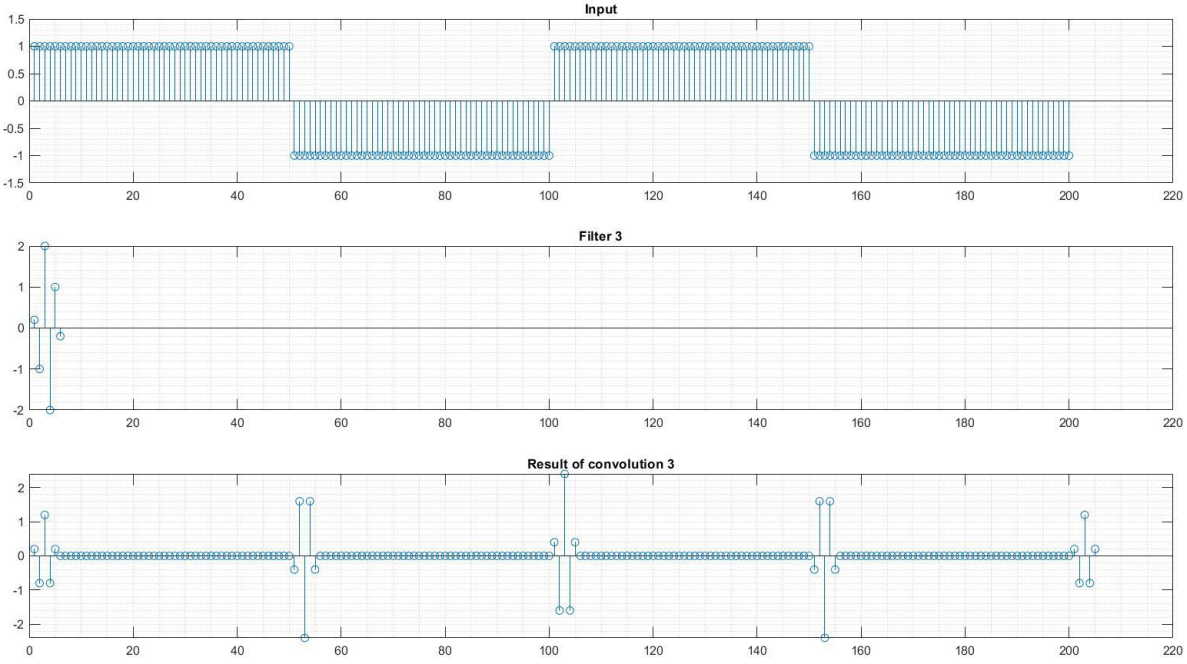
برای پیاده سازی فیلتر مورد نظر سوال، به جای محاسبه چند جمله ای از درجه ۵، چند جمله ای درجه ۱ که به صورت [1 -1] تعریف می‌شود را، پنج بار به سیگنال اصلی اعمال می‌کنیم.

$$Y(z) = X(Z)\frac{1}{5}(1 - z^{-1})^5$$

$$\Rightarrow y[n] = \left(\frac{1}{5}\right) y[n] * (\delta[n] - \delta[n-1]) * (\delta[n] - \delta[n-1]) * (\delta[n] - \delta[n-1]) * (\delta[n] - \delta[n-1]) * (\delta[n] - \delta[n-1])$$

بدین ترتیب نیازی به محاسبه چند جمله ای از درجه ۵ یا تعریف تابع دقیق H(z) نیست.

```
%% 2-1-d)
fil3 = [1 -1];

% applying the filter 5 times in a row
fil3 = myconv(fil3 , fil3);
fil3 = myconv(fil3 , fil3);
fil3 = 0.2*myconv(fil3 , [1 -1]);

res3 = myconv(fil3 , sig);

figure(3);
subplot(3,1,1);
stem(sig)
title("Input");
axis([0 220 -1.5 1.5]);
grid minor;

subplot(3,1,2);
stem(fil3);
title("Filter 3");
axis([0 220 -1*max(fil3) 1*max(fil3)]);
grid minor;

subplot(3,1,3);
stem(res3);
title("Result of convolution 3");
axis([0 220 -1*max(res3) max(res3)]);
grid minor;
```

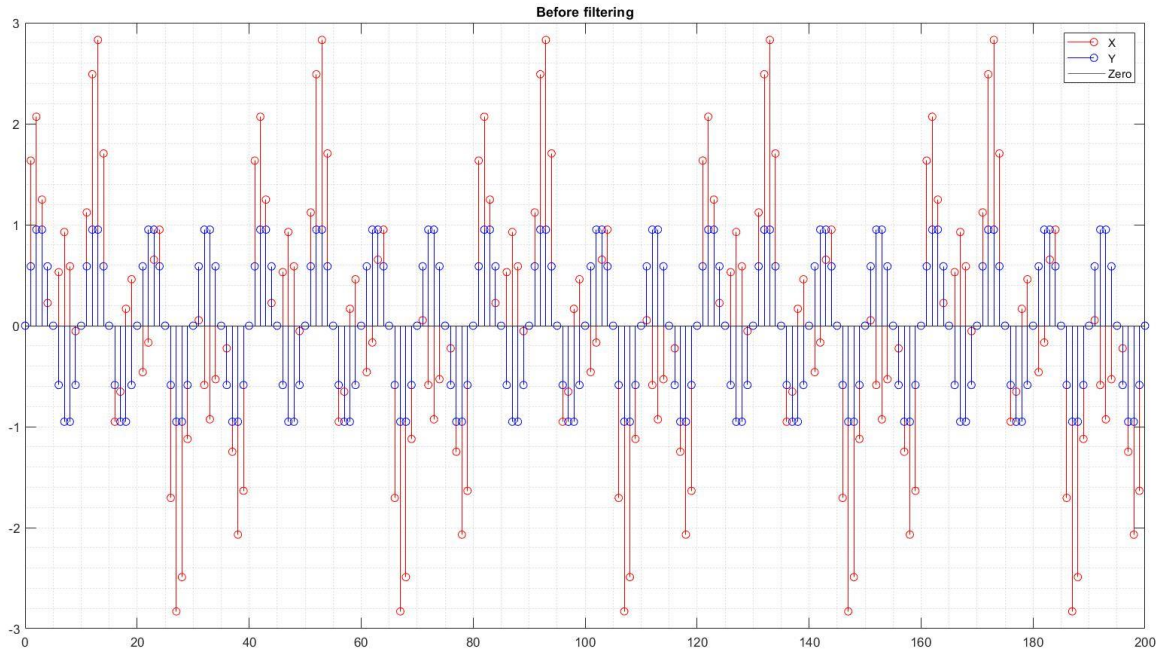


سیگنال اصلی مشابه پله بوده و انتظار می رود با ۵ بار مشتق گرفتن از آن، مواقعی که سیگنال اصلی تغییر می‌کند، ۵ ضربه متوالی ظاهر بشود، که از شکل نهایی نیز همین اتفاق مشهود است.

بخش ۲-۲-الف) فیلتر سیگنال نویزی

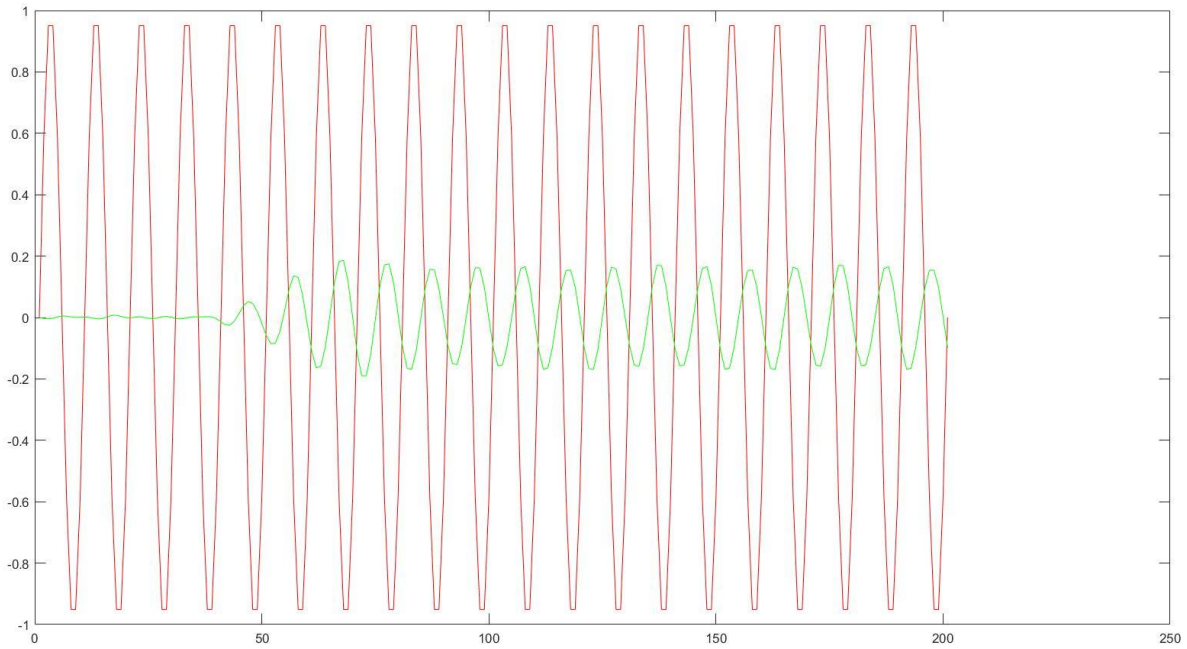
ابتدا سیگنال ها و توابع مطابق خواسته سوال تعریف شده اند:

```
%% 2-2-a)
w1 = 0.05*pi;
w2 = 0.20*pi;
w3 = 0.35*pi;

Wa = 0.15*pi;
Wb = 0.25*pi;

t=0:1:200;

s = sin(w2*t);
v = sin(w1*t) + sin(w3*t);
x = s+v;

figure(4)
plot(t,x , color='red');
hold on;
plot(t,s , color='blue');
grid minor;
yline(0);
legend("X","Y",'Zero');
title("Before filtering");
```



بخش ۲-۲-ب)

```
%% 2-2-b)
M=100;
W = zeros(1,100);
H = zeros(1,100);
for i=1:1:M
    W(i)=0.54 - 0.46*sin(2*pi*i/M);
end

for i=1:1:100
    H(i)=W(i)*( (Wa/(pi^2))*sinc((Wa/pi)*(i-M/2)) - (Wb/(pi^2))*sinc((Wb/pi)*(i-M/2))
);
end

y = filter(H , 1 , x);
figure(5);
title("After Filtering");
plot(s,color='red');
hold on;
plot(y,color='green');
```



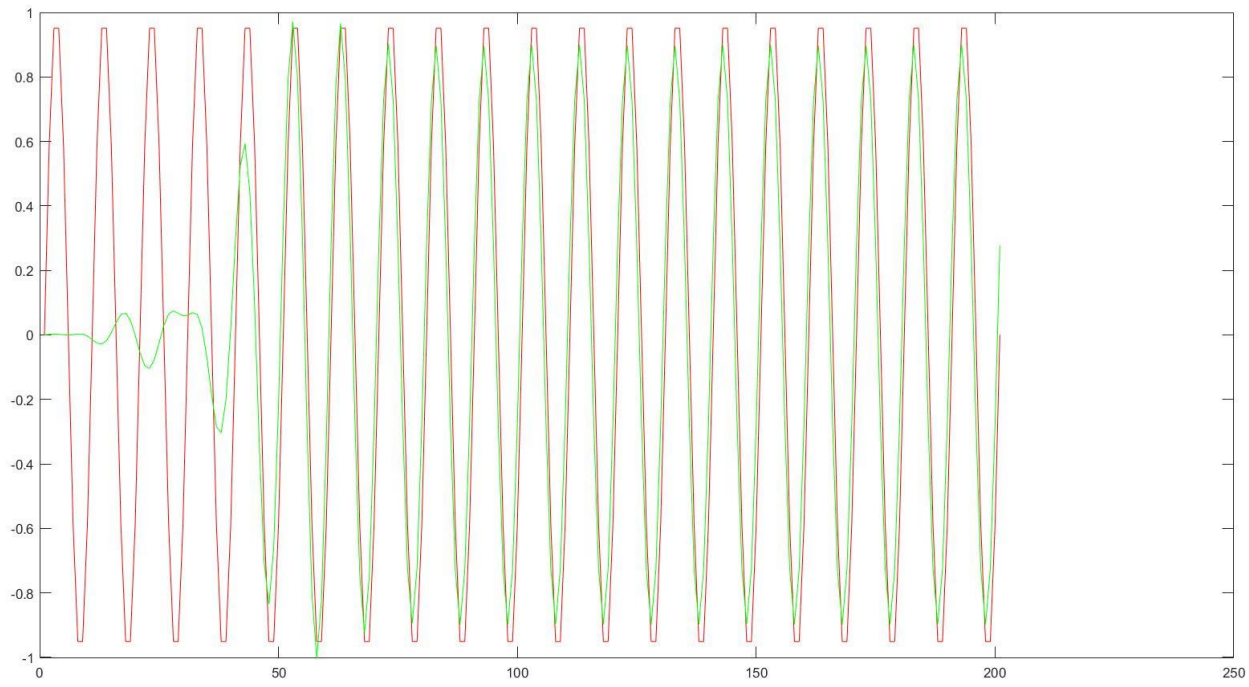
در شکل بالا، سیگنال S با رنگ قرمز و X بعد از عبور از فیلتر با رنگ سبز نمایش داده شده است. مشاهده می‌شود که تفاوت اصلی آن ها صرفا در یک تاخیر مرتبط با فیلتر است.

بخش ۲-۲-ج)

فیلتر با ابزار fdatool طراحی شده و در اینجا صرفا از ضرایب آن استفاده می‌کنیم.

```
%% 2-2-c)
% The filter is already designed using the FDATOOL and here I just load the
% coefficients

coeff = load("fil.mat").coeff;
fda_fil = filter(coeff , 1 , x);

figure(6);
title("After Filtering using FDATOOL");
plot(s,color='red');
hold on;
plot(fda_fil,color='green');
```

نتیجه شبیه سازی به صورت زیر است:



در شکل بالا، سیگنال s با رنگ قرمز و x بعد از عبور از فیلتر با رنگ سبز نمایش داده شده است. مشاهده می‌شود که تفاوت اصلی آن ها صرفا در یک تاخیر مرتبط با فیلتر است.

بخش ۲-۳-الف) بارگذاری صوت:

```
%% 2-3-a)
fs=22296;
f0=fs/4;
audio = audioread(“sample-3.mp3”,[1,10*fs]);
audio = audio’;
```

فایل مورد استفاده در این بخش، ۱۰ ثانیه بوده و شامل 222960 سمپل است. به همین دلیل فرض می شود فرکانس نمونه برداری به جای ۴۸۰۰۰ برابر با ۲۲۲۹۶ است و فرکانس عبور و قطع فیلتر نیز به همین نسبت، در مرحله طراحی، تغییر می‌کنند.

بخش ۲-۳-ب) طراحی و لود کردن فیلتر

```
%% 2-3-b)
% The coefficients are stored as 'Audio_filter'
aud_fil = load("audio_filter.mat").coeff;
```

بخش ۲-۳-ج) در هم ریختن سیگنال صوت

```
%% 2-3-c)
S=zeros(1,size(audio,2));
for i=1:1:size(audio,2)
    S(i)=2*cos(pi*i/2);
end
% Scrambling
y0 = filter(aud_fil , 1 , audio);
y1 = y0.*S;
y2 = filter(aud_fil , 1 , y1);
sound(y2,fs);
% Applying this system all over again
```

بخش ۲-۳-د) بازگردانی صوت

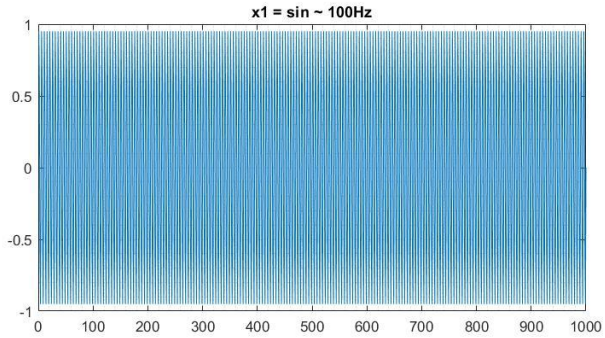
```
%% 2-3-d)
y3 = y2.*S;
y4 = filter(aud_fil, 1, y3);
```

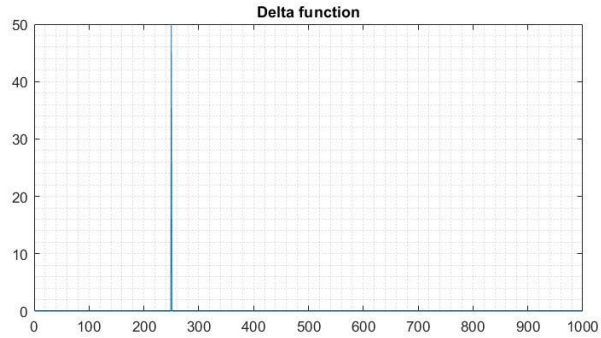
مشاهده می شود سیگنال صوت پس از این مراحل دچار کاهش کیفیت شده است که می‌توان آن را مرتبط با ایده آل نبودن فیلتر دانست.

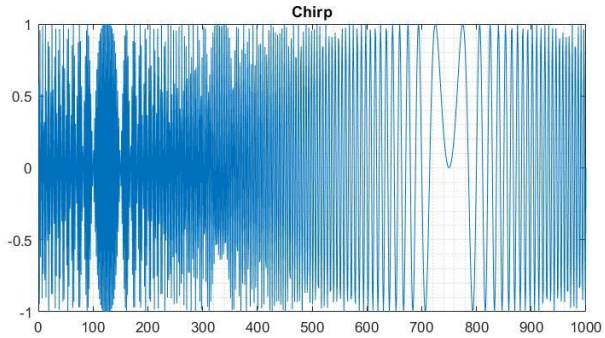
بخش ۲-۴-الف) تبدیل فوریه زمان کوتاه

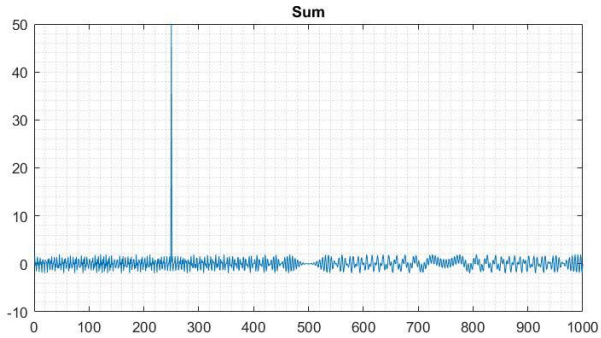
سیگنال سینوسی با x1، سیگنال ضربه با x2 و سیگنال chirp با x3 تعریف شده است.
هر سه سیگنال و مجموع آن ها در شکل ۷ نشان داده شده است.
همچنین در شکل ۸، تبدیل فوریه سیگنال x4 نشان داده شده است (۱۰۰۰ نقطه)

```
%% 2-4-a)

n=1:1:1000;
x1 = sin(2*pi*100*n/500);
x2 = zeros(1,1000);
x2(250)=50;
x3 = sin(2*pi*(200 + 0.2.*n).*n/500);

x4 = x1 + x2 + x3;

figure(7);
subplot(2,2,1);
plot(x1);
title("x1 = sin ~ 100Hz");
grid minor;


subplot(2,2,2);
plot(x2);
title("Delta function");
grid minor;

subplot(2,2,3);
plot(x3);
title("Chirp");
grid minor;

subplot(2,2,4);
plot(x4);
title("Sum");
grid minor;

x4_fft = fftshift(abs(fft(x4)));

figure(8);
m=-500:1:499;
plot(m,x4_fft);
```

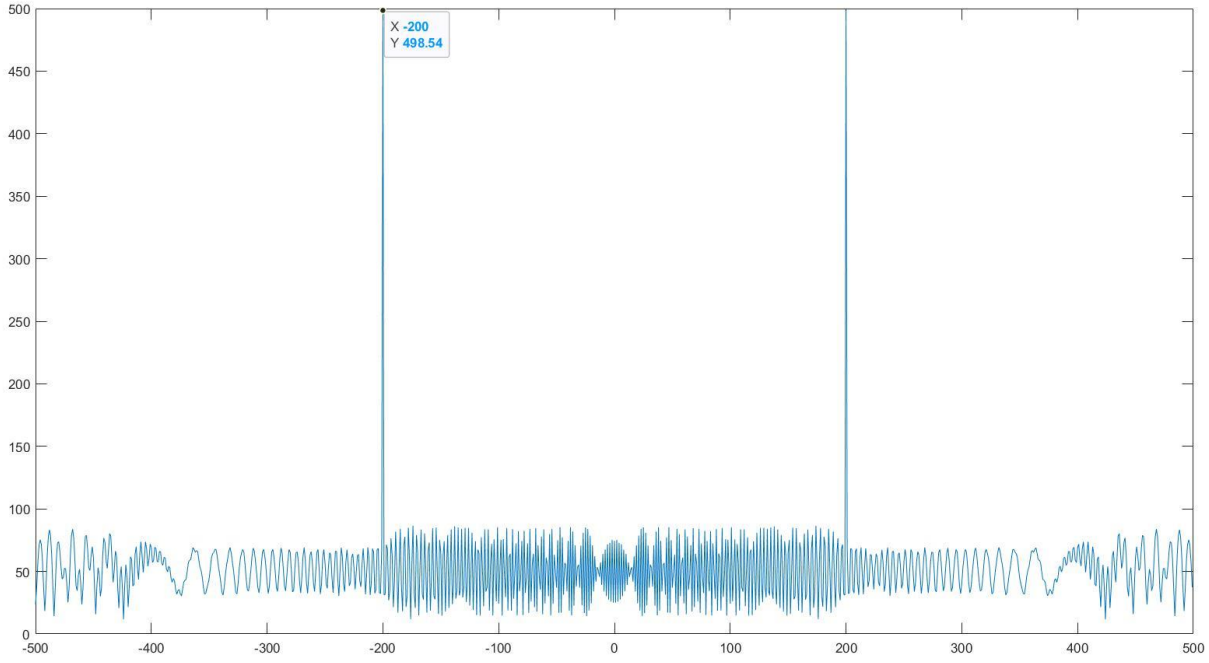






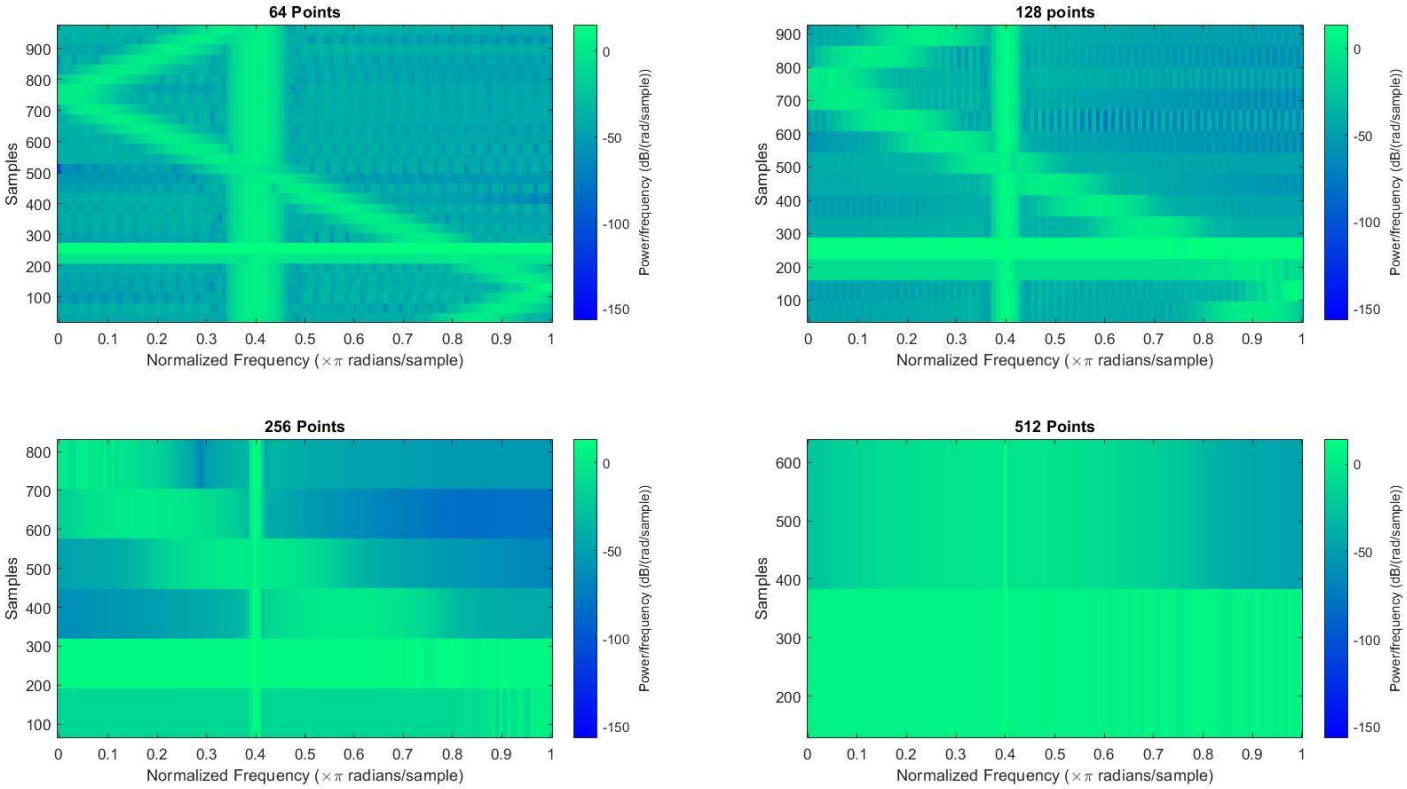


تبدیل فوریه سیگنال x4:

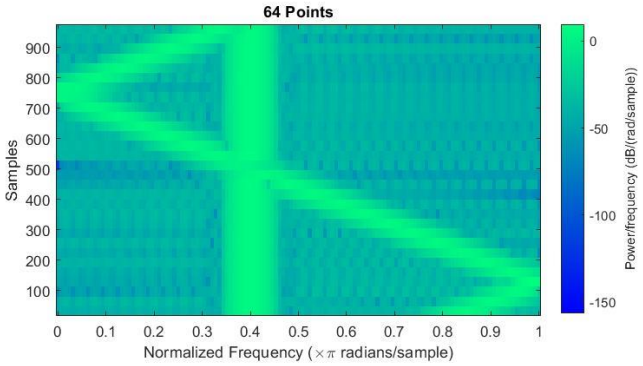


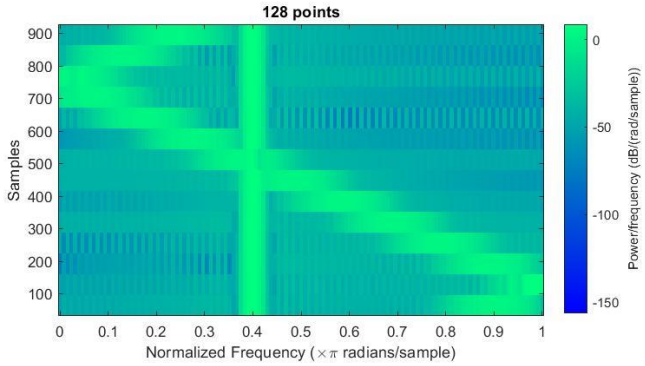
بخش ۴-۲-ب) اسپکتوگرام

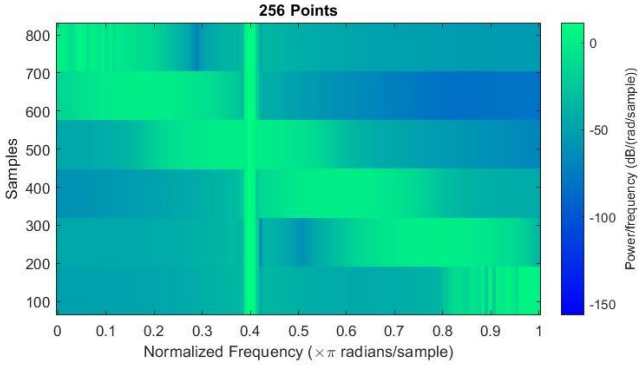
تبدیل فوریه زمان کوتاه این سیگنال برای ۴ حالت (پنجره هایی به طول ۶۴، ۱۲۸، ۲۵۶ و ۵۱۲ سمپل) ترسیم شده است.

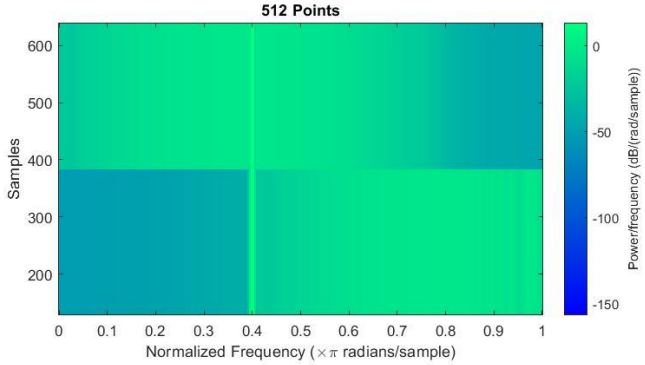


وجود خط در فرکانس نرمالیزه 0.4، ناشی از سیگنال سینوسی ۱۰۰ هرتز است، زیرا مولفه فرکانسی این سیگنال فقط یک مقدار دارد و در همه بازه های زمانی نیز موجود و ثابت است. همچنین بخش زیگزاگی در این نمودار ها، مرتبط با سیگنال chirp است که فرکانس لحظه ای آن به صورت خطی افزایش می‌یابد، بدین ترتیب انتظار می رود که در هر بازه زمانی کوتاه، سیگنال chirp به صورت یک سیگنال سینوسی با فرکانس تقریبا ثابت ظاهر بشود، در هر بازه نسبت به بازه قبلی، مقدار این فرکانس اندکی افزایش می‌یابد.
خط افقی که در حوالی سمپل ۲۵۰ ظاهر شده است (در شکل ۶۴ نقطه ای بسیار واضح تر است) مرتبط با سیگنال ضربه می باشد، که در بازه زمانی کوتاه اطراف خود، تمامی فرکانس ها را اشغال کرده است و در بازه های بعدی تاثیری چندانی نذاشته است. برای نمایش اثر ضربه، یک بار دیگر شبیه سازی را بدون حضور x2 در x4 انجام می دهیم:









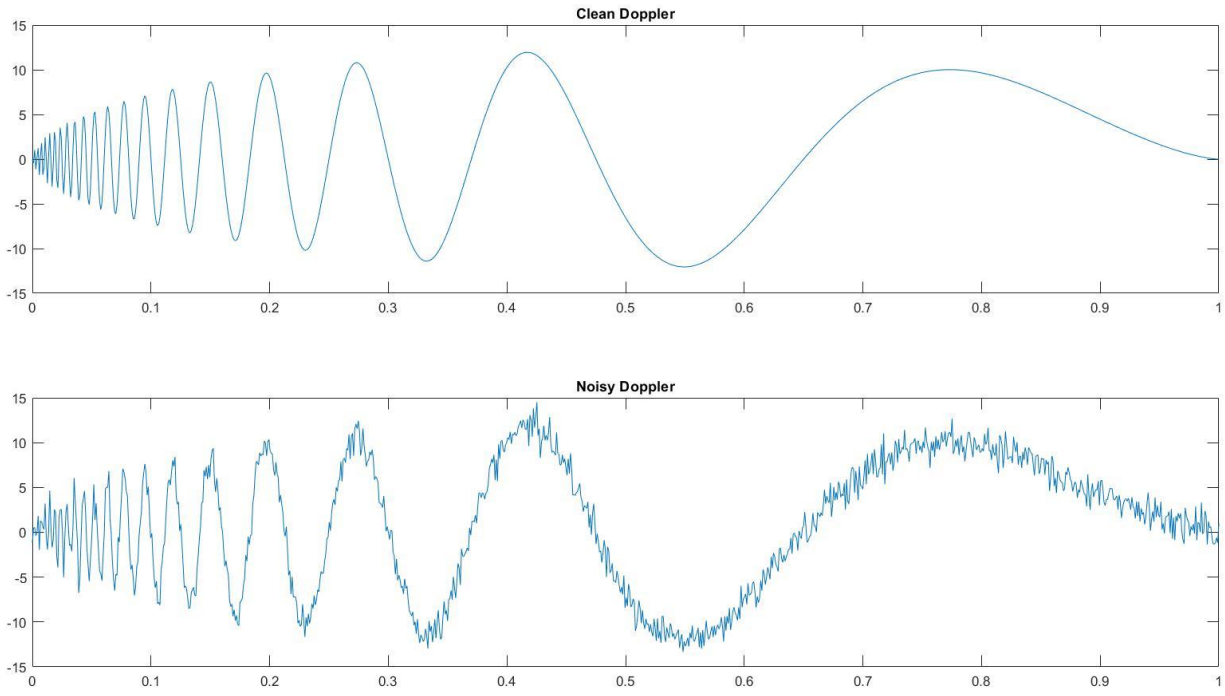
همانطور که انتظار می‌رفت، خط افتی حذف شده و سایر بازه های زمانی تغییر قابل توجهی نداشته اند.

بخش ۲-۵) تبدیل موجک

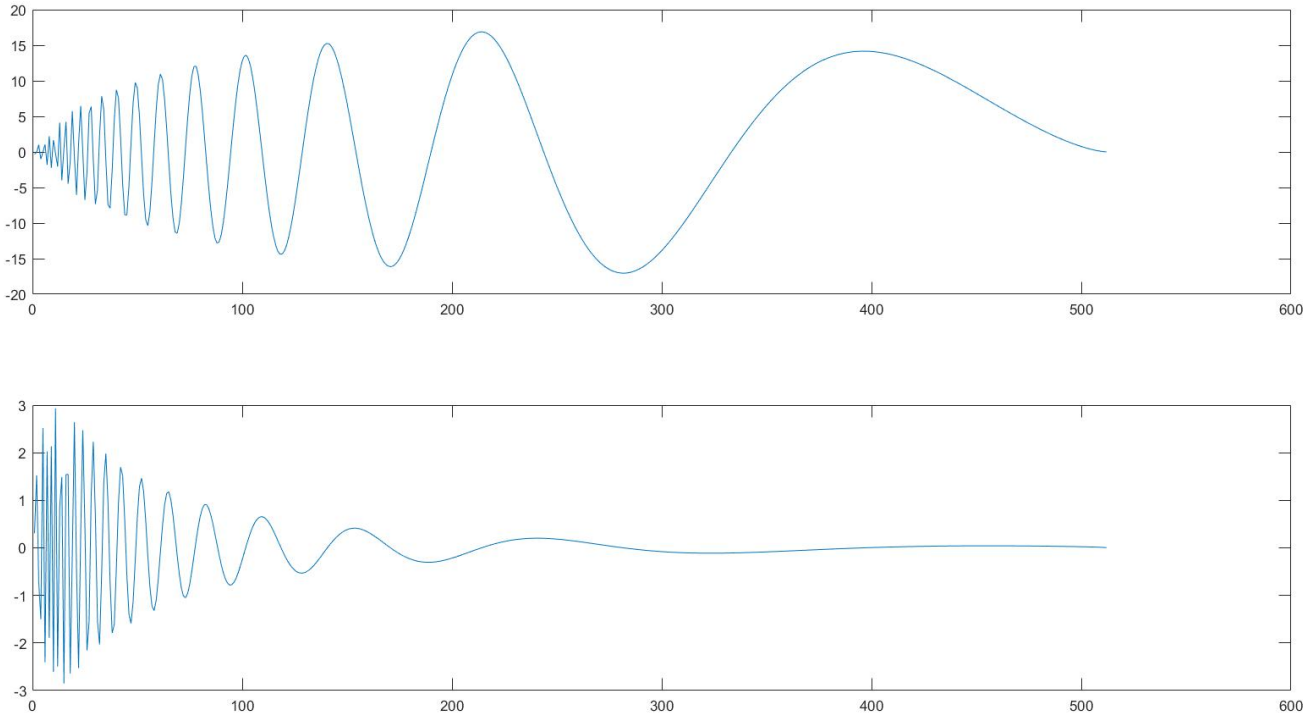
برای نمایش تبدیل موجک و ساب-بند های مرتبط با آن، از تابع داپلر استفاده کرده ایم و سپس آن را با ویولت db1 (Daubechies) تجزیه کرده ایم.

```
%% 2-5)
[wav,noisyx] = wnoise('doppler',10,7);
loc = linspace(0,1,2^10);

figure(10);
subplot(2,1,1);
plot(loc,wav);
title('Clean Doppler');
ylim([-15 15]);

subplot(2,1,2);
plot(loc,noisyx);
title('Noisy Doppler');
ylim([-15 15]);

% Analysis:
[cA, cD]=dwt(wav,'db1');

figure(11);

subplot(1,2,1);
plot(cA);

subplot(1,2,2);
plot(cD);
```

سیگنال داپلر قبل از تجزیه:



بعد از تجزیه:


20
15
10
5
0
-5
-10
-15
-20
0
100
200
300
400
500
600

3
2
1
0
-1
-2
-3
0
100
200
300
400
500
600