" بسمه تعالی "

گزارش دوم آزمایشگاه DSP – زهرا لطیفی – ۹۹۲۳۰۶۹

بخش ۲-۱-الف)

```
function conv = myconv(h, x)
    lenh = length(h);
    lenx = length(x);
    lenRes = lenh + lenx - 1;
    fliph = fliplr(h);
    zeroPadx = [zeros(1, lenh - 1), x, zeros(1, lenh - 1)];
    conv = zeros(1, lenRes);
    for i=1:1:lenRes
        conv(i) = fliph*zeroPadx(i:i + lenh- 1)';
    end
end
```

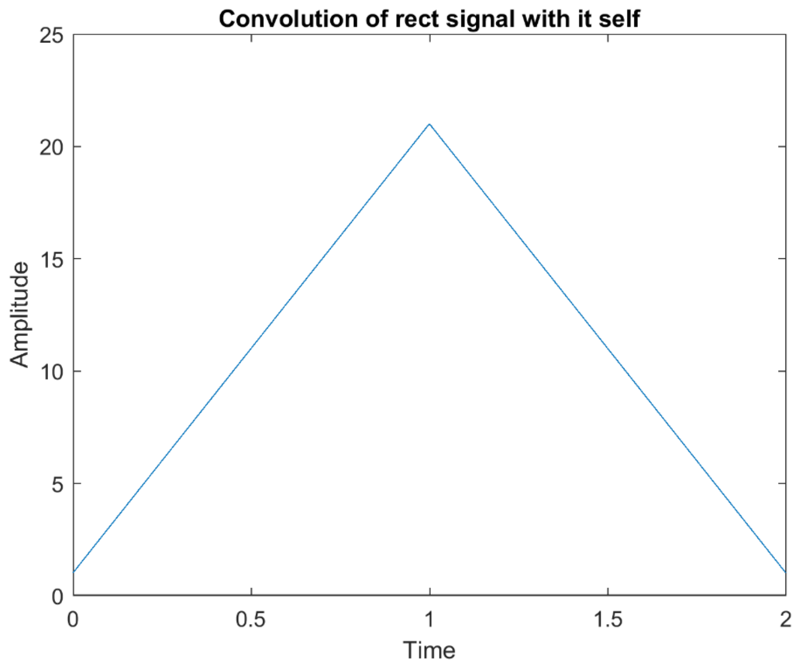
در این بخش، تابعی به نام myconv پیاده‌سازی کردیم که دو سیگنال x و h را به عنوان ورودی گرفته، h را flip می‌کند، x را به اندازه طول h-1 از دو طرف گسترش می‌دهد و حاصل این دو عملیات را نمونه به نمونه در هم ضرب کرده و در درایه متناظر آرایه conv که طولی برابر با مجموع طول h و x منهای ۱ دارد، ذخیره می‌کند. نهایتا آرایه تکمیل شده conv را به عنوان خروجی برمی‌گرداند. مبنای تعریف این تابع رابطه زیر است:

$$y_n = \sum_{m=0}^{\min(-l+n,M)} h_m x_{n-m} \qquad n = 0, 1, 2, \ldots, M + L - 1$$

برای آزمودن عملکرد این تابع در این بخش یک سیگنال rect ساخته و با خودش کانولوشن گرفتیم.

```
t = 0:0.05:2;
xa = ones(1, 21);
resa = myconv(xa, xa);


% Testing my function
figure(1);
plot(t, resa);
title("Convolution of rect signal with it self");
ylabel("Amplitude");
xlabel("Time");
```

نتیجه را رسم کردیم و دیدیم که مطابق انتظار Triangle بود.

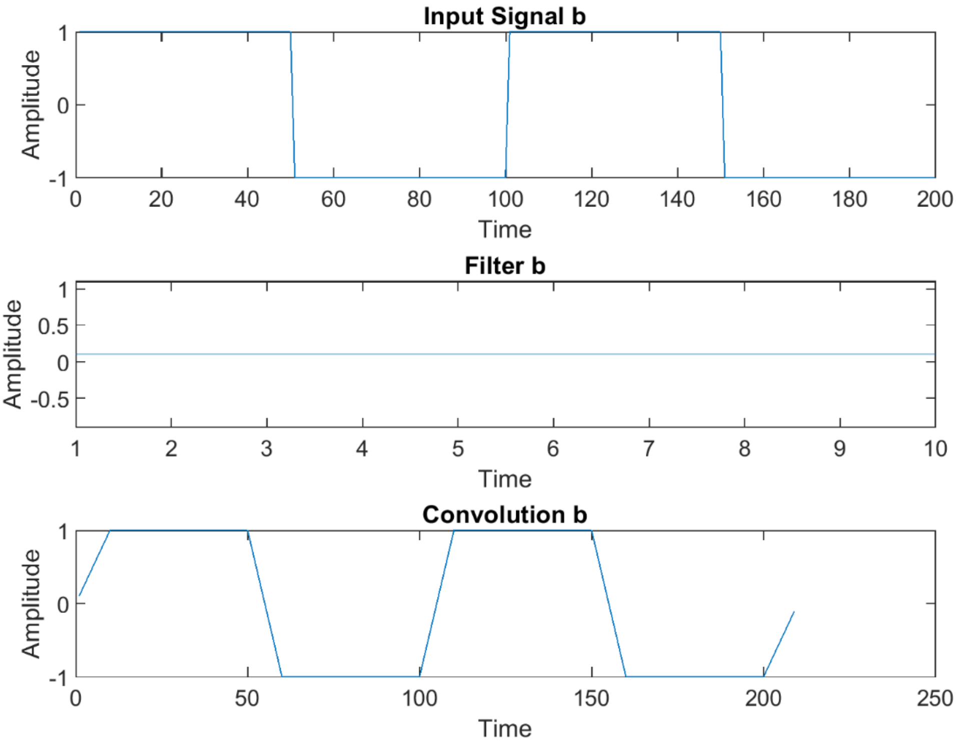


شکل ۱

بخش ۲-۱-ب)

```
t = 0:0.01:1.99;
xb = square(2*pi*t, 50);
hb = ones(1,10)/10;
resb = myconv(xb, hb);


figure(2);
subplot(3,1,1);
plot(xb)
title("Input Signal b");
ylabel("Amplitude");
xlabel("Time");


subplot(3,1,2);
plot(hb);
title("Filter b");
ylabel("Amplitude");
xlabel("Time");


subplot(3,1,3);
plot(resb);
title("Convolution b");
ylabel("Amplitude");
xlabel("Time");
```

در این بخش پاسخ ضربه فیلتری تعریف شده بود که با دستور `ones(1,10)/10` پیاده‌سازی شد و سیگنال مربعی خواسته شده هم با دستور `square(2*pi*t, 50)` ساخته شد. کانولوشن این دو سیگنال را با تابع تعریف شده در بخش قبل محاسبه و رسم کردیم:



شکل ۲

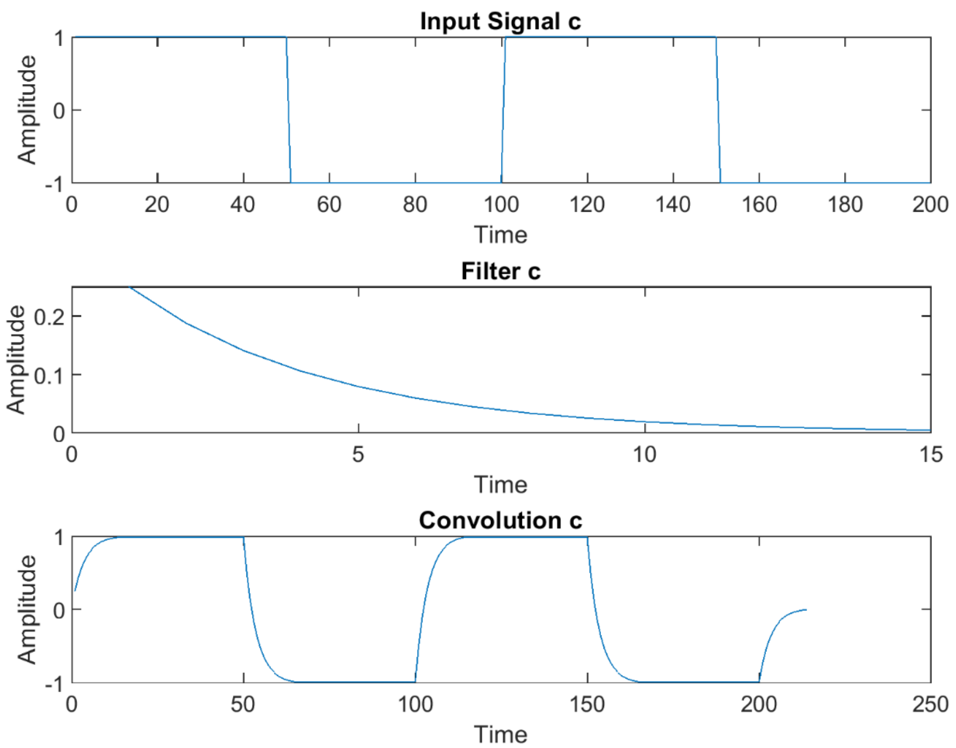
نکته قابل توجه درباره این بخش این است که اولا مطابق انتظار سیگنال خروجی طولی برابر با ۲۰۹ دارد و ثانیا چون فیلتر تعریف شده در این بخش پایین گذر هست، نقاطی که در سیگنال ورودی تغییرات sharp داشتند، نرم تر شده و کندتر تغییر می‌کنند.

بخش ۲-۱-ج)

```
t = 0:0.01:1.99;
xc = square(2*pi*t, 50);
hc =  zeros(1,15);


for i=1:1:15
    hc(i)=0.25*(0.75^(i-1));
end


resc = myconv(xc, hc);
```

```
figure(3);
subplot(3,1,1);
plot(xc)
title("Input Signal c");
ylabel("Amplitude");
xlabel("Time");


subplot(3,1,2);
plot(hc);
title("Filter c");
ylabel("Amplitude");
xlabel("Time");


subplot(3,1,3);
plot(resc);
title("Convolution c");
ylabel("Amplitude");
xlabel("Time");
```

در این بخش هم همانند بخش (ب) عمل کردیم با این تفاوت که فیلتر h ضابطه نمایی و طول ۱۵ داشت. نتیجتا طول سیگنال خروجی برابر ۲۱۴ شده و تغییرات هم شکل نمایی گرفتند:

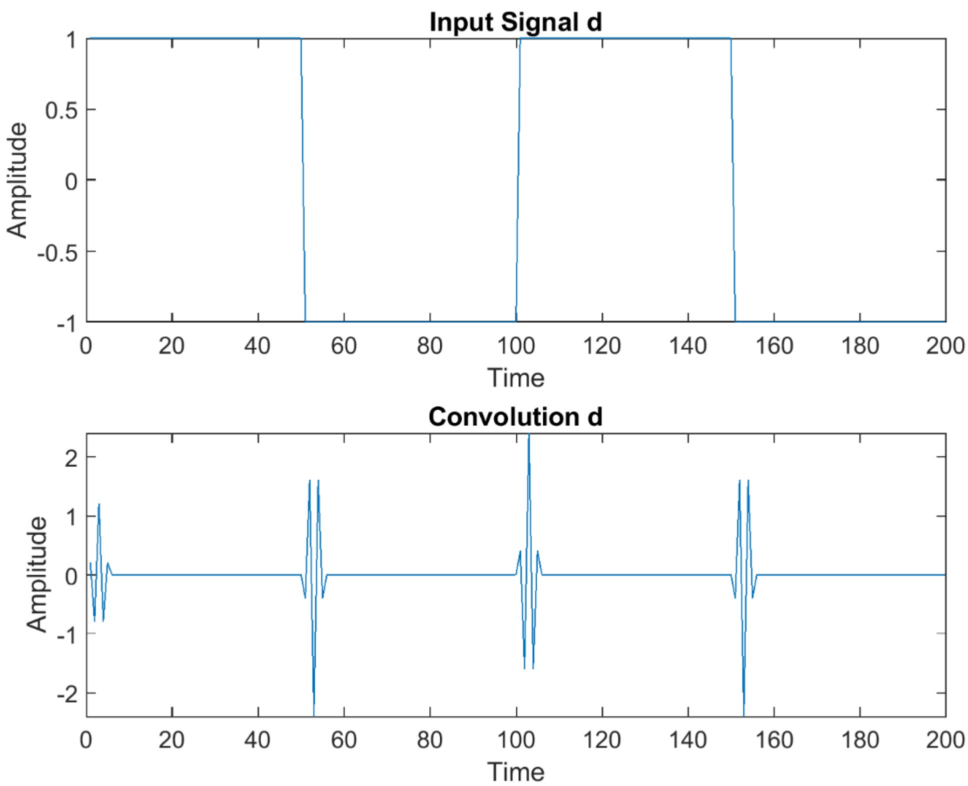


شکل ۳

بخش ۲-۱-د)

```
t = 0:0.01:1.99;
xd = square(2*pi*t, 50);
resd = filter([1, -5, 10, -10, 5, -1], 5, xd);


figure(4);
subplot(2,1,1);
plot(xd)
title("Input Signal d");
ylabel("Amplitude");
xlabel("Time");


subplot(2,1,2);
plot(resd);
title("Convolution d");
ylabel("Amplitude");
xlabel("Time");
```

روش پیشنهادی برای انجام این بخش این بود که تابع H(z) را باز کرده و با یافتن ضرایب فیلتر، از دستور filter برای اعمال بر سیگنال مربعی ورودی استفاده کنیم.



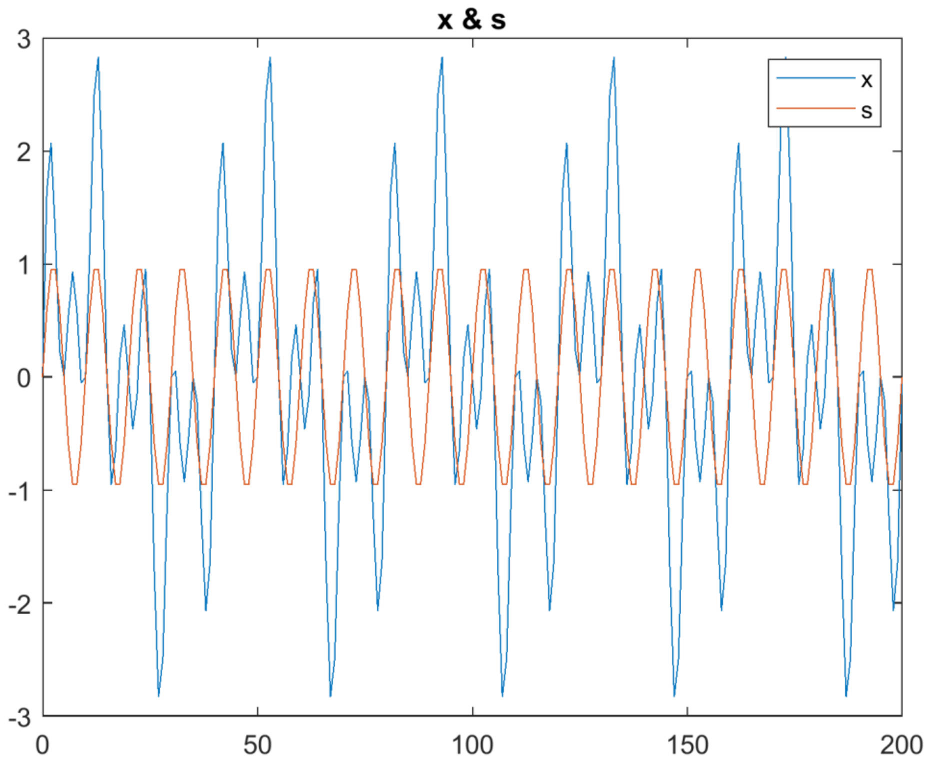
شکل ۴

همانطور که انتظار داشتیم، ۵ بار مشتق گرفته شده پس در نقاط تغییر ورودی، در خروجی ۵ ضربه ظاهر می‌شود.

بخش ۲-۲-الف)

```
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(5)
plot(t, x);
hold on;
plot(t, s);
title("x & s");
```

در این بخش از ما خواسته شده تا سیگنال سینوسی s[n] و x[n] که حاصل جمع دو سیگنال سینوسی s[n] و v[n] است را ساخته و رسم کنیم.

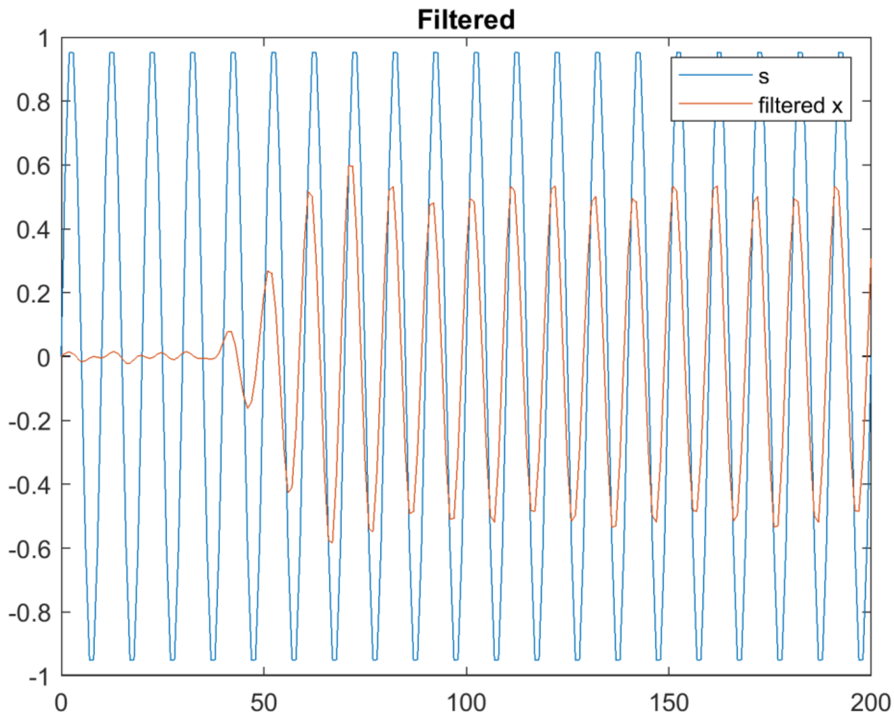


شکل ۵

بخش ۲-۲-ب)

```
M = 100;
w = zeros(1, M);
h = zeros(1, M);


for i=1:1:M
    w(i) = 0.54 - 0.46*sin(2*pi*i/M);
end


for i=1:1:M
    h(i) = w(i)*((wb/pi)*sinc((wb/pi)*(i-M/2)) - (wa/pi)*sinc((wa/pi)*(i-
M/2)));
end


y = filter(h, 1, x);


figure(6);
plot(t, s);
hold on;
plot(t, y);
title("Filtered");
```

در این قسمت، w[n] تعریف شده را ساخته، با تبدیل ضوابط داده شده به شکل تعریف تابع sinc، پنجره hamming با پاسخ ضربه h[n] را هم ساخته و با دستور filter به x[n] اعمال کردیم. سپس خروجی را به همراه سیگنال s[n] رسم کردیم:



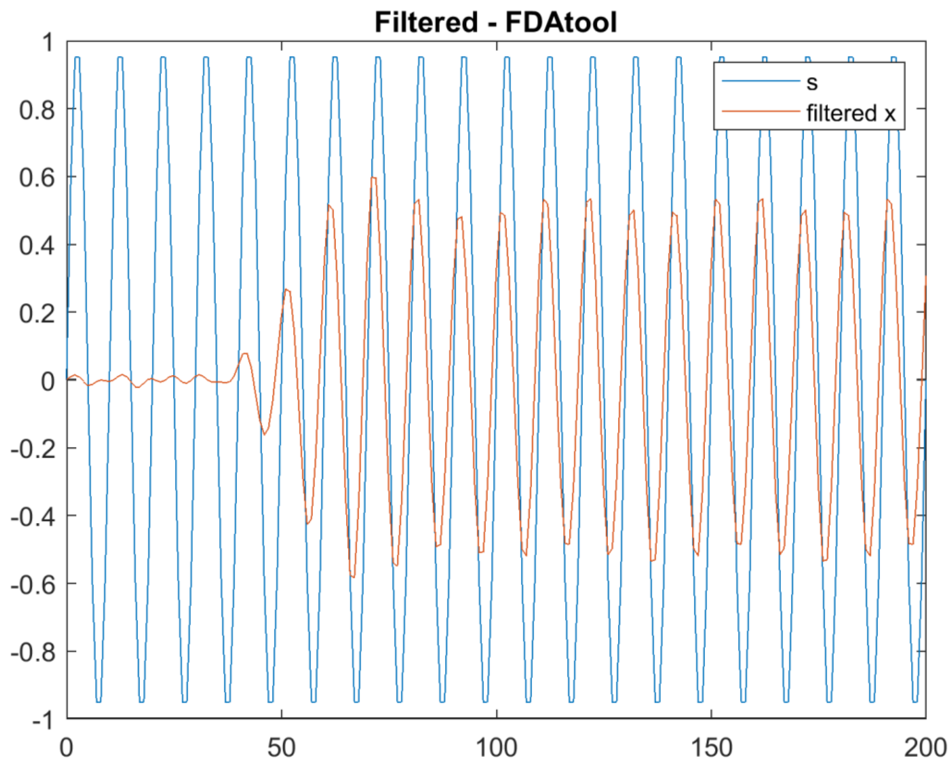
شکل ۶

می‌بینیم که تفاوت اصلی این دو، تنها در تاخیر ناشی از استفاده از فیلتر است.

بخش ۲-۲-ج)

```
coef = load("coef.mat").Num;
y2 = filter(coef , 1, x);


figure();
plot(t, s);
hold on;
plot(t, y);
legend("s","filtered x");
title("Filtered - FDAtool");
```

این بار فیلتری با مشخصات داده شده در FDATool طراحی کردیم و ضرایب آن را به عنوان ورودی به دستور filter داده و بر سیگنال x اعمال کردیم و خروجی و سیگنال s را رسم کردیم.



شکل ۷

بازهم شاهدیم که تفاوت اصلی در تاخیر ناشی از فیلتر است.

بخش ۲-۳-الف)

```
[audio, fs] = audioread("Audio01.wav");
audio = audio';
```

در این قسمت فایل صوتی داده شده را لود کرده و فرکانس نمونه برداری آن را هم دریافت کردیم. سپس آن را Transpose کردیم تا در سایر بخش‌ها قابل استفاده باشد.

بخش ۲-۳-ب)

```
audio_filter = load("coef2.mat").filter;
```

فیلتری با مشخصات داده شده در FDATool طراحی کردیم و ضرایب آن را ذخیره کردیم.

بخش ۲-۳-ج)

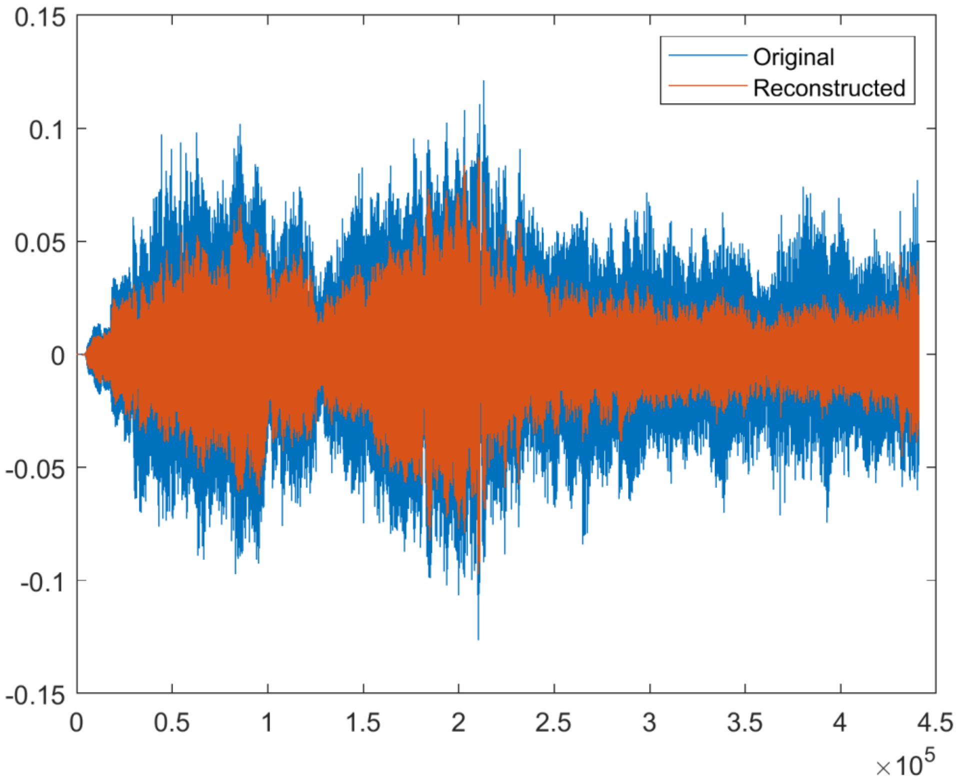
```
carrier = zeros(1, size(audio, 2));


for i=1:1:size(audio, 2)
    carrier(i) = 2*cos(pi*i/2);
end


filtered1 = filter(audio_filter, 1, audio);
carrMult1 = filtered1.*carrier;
filtered2 = filter(audio_filter, 1, carrMult1);
```

در این قسمت فیلتر ساخته شده را بر سیگنال صوت اعمال کردیم. سپس حامل کسینوسی را طبق خواسته سوال ساخته و در خروجی فیلتر شده ضرب کردیم. بار دیگر نتیجه را فیلتر کردیم.

بخش ۲-۳-د)

```
filtered3 = filter(audio_filter, 1, filtered2);
carrMult2 = filtered3.*carrier;
filtered4 = filter(audio_filter, 1, carrMult2);
%% sound(audio', fs);
sound(filtered4, fs);
```

تمام مراحل بخش قبل را یک بار دیگر عینا تکرار کردیم تا سیگنال صوت اصلی کاملا بازسازی شود. صوت را پخش کردیم، بازسازی تا میزان خوبی به درستی انجام شده بود.

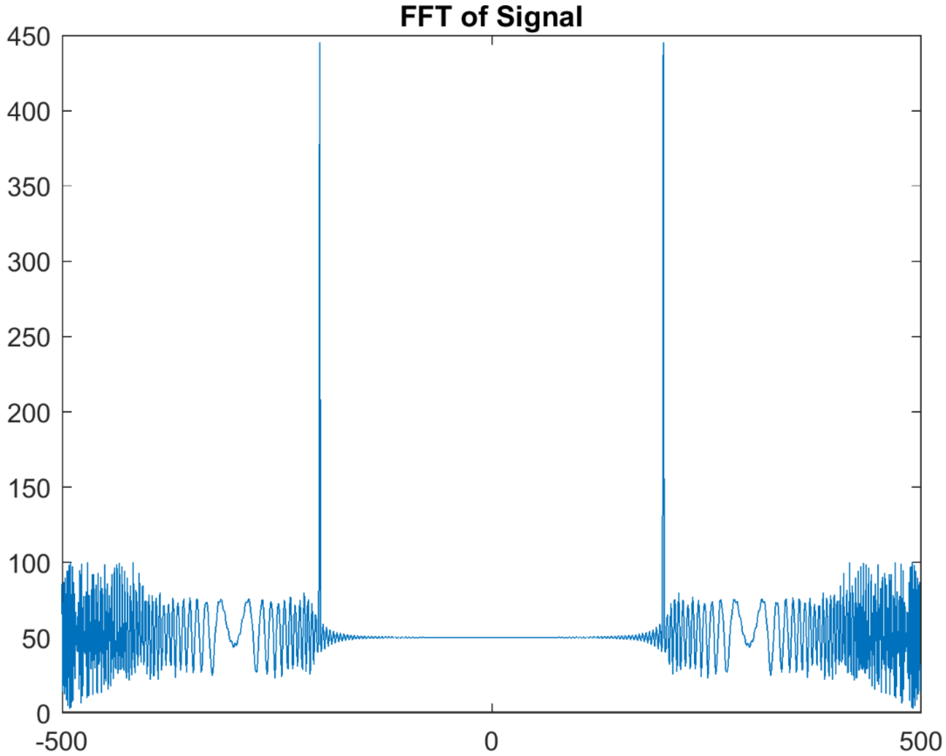


شکل ۸

بخش ۲-۴-الف)

```
t = 0:0.002:1.998;
f0 = 400;
f1 = 200;
t1 = 2;


x1 = chirp(t, f0 , t1, f1, 'linear');
x2 = sin(2*pi*100*t);
x3 = zeros(1, 1000);
x3(250) = 50;


s = x1 + x2 + x3;


figure(8);
n = -500:1:499;
plot(n, fftshift(abs(fft(s))));
title("FFT of Signal");
```

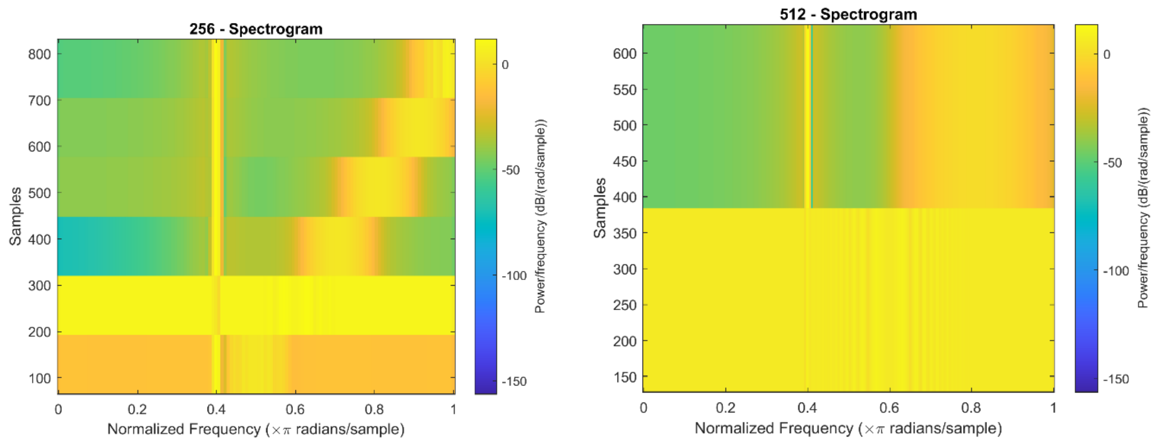
سه سیگنال chirp، سینوسی و ضربه‌ای که در نمونه ۲۵۰ مقدار ۵۰ دارد، در این بخش ساخته شده و با هم جمع شدند. سپس تبدیل فوریه این سیگنال، رسم شده است.



شکل ۹

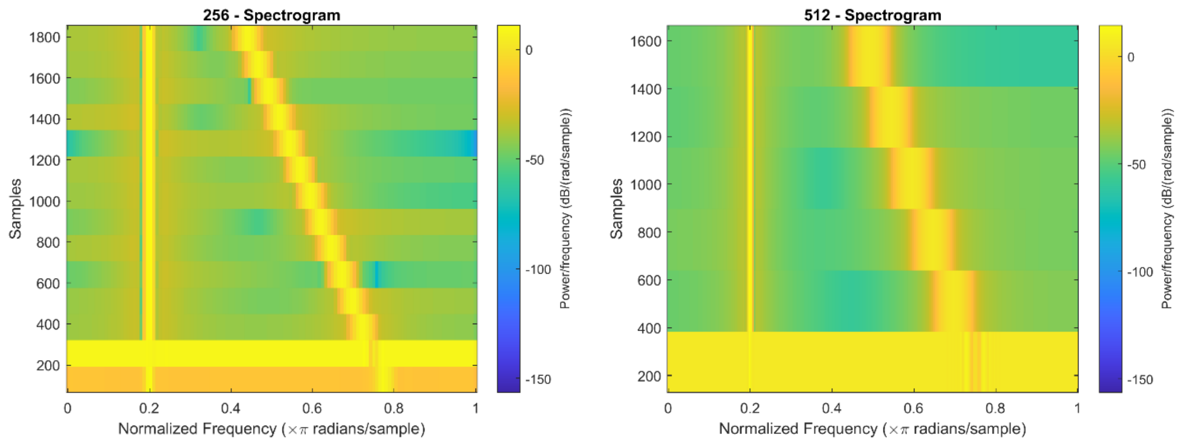
بخش ۲-۴-ب)

```
figure(8);
[spc1, w1] = spectrogram(s, hamming(256));
title("256 - Spectrogram");

figure(9);
[spc, w] = spectrogram(s, hamming(512));
spectrogram(s, hamming(512));
title("512 - Spectrogram");
```



شکل ۱۰

بخش زیگزاگی در این نمودارها، حاصل حضور سیگنال chirp است که فرکانس آن به صورت خطی افزایش می‌یابد، پس شاهدیم که در هر بازه زمانی کوتاه، سیگنال chirp به صورت یک سیگنال سینوسی با فرکانس تقریبا ثابت ظاهر می‌شود و در هر بازه نسبت به بازه قبلی، مقدار این فرکانس افزایش می‌یابد. البته طبق صورت سوال نباید چنین اتفاقی بیفتد و باید با گذر زمان فرکانس chirp کاهش یابد. علت این است که فرکانس نمونه برداری داده شده در صورت سوال برای سیگنال sin کمتر از ۸۰۰ (۴۰۰*۲) است و این مسئله باعث شده سیگنال chirp دچار مشکل شود. اگر فرکانس را به جای ۵۰۰ برابر ۱۰۰۰ هرتز قرار دهیم می‌بینیم که نتیجه درست خواهد شد. (شکل ۱۱) خط عمودی در فرکانس نرمالیزه ۰.۴ شکل ۱۰ و ۰.۲ شکل ۱۱ اثر سیگنال سینوسی ۱۰۰ هرتز است زیرا مولفه فرکانسی این سیگنال فقط یک مقدار دارد که در تمام بازه زمانی موجود و ثابت است. خط افقی که حدودا در سمپل ۲۵۰ ظاهر شده هم متعلق به سیگنال ضربه می‌باشد که در بازه زمانی کوتاهی، تمامی فرکانس‌ها را اشغال کرده اما در بازه‌های بعدی تاثیر چندانی نداشته است.



شکل ۱۱

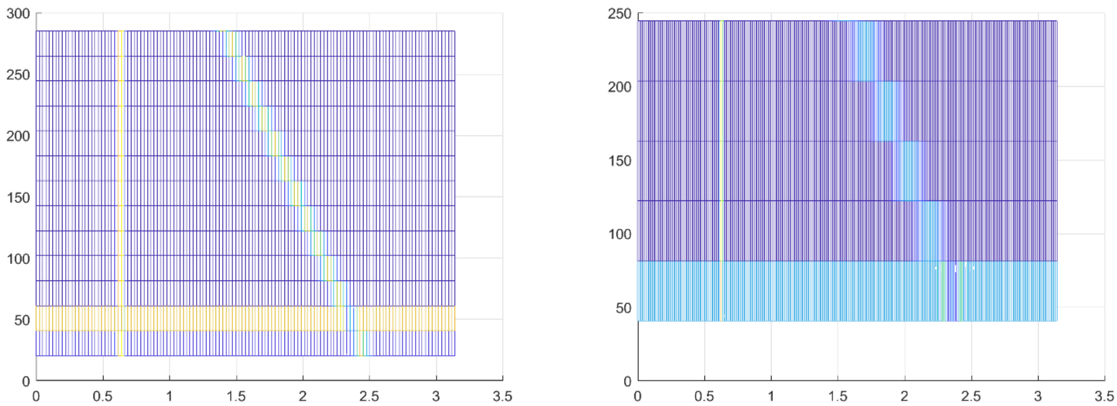
DTFT و STFT هر دو ابزارهایی هستند که در پردازش سیگنال برای تجزیه و تحلیل محتوای فرکانسی سیگنال‌ها استفاده می‌شوند، اما کاربردها و ویژگی‌های متفاوتی دارند. DTFT برای تجزیه و تحلیل کل سیگنال به طور همزمان استفاده می‌شود و یک سیگنال زمان گسسته را به تابعی پیوسته در حوزه فرکانس تبدیل می‌کند. DTFT معمولاً زمانی استفاده می‌شود که سیگنال به صورت متناوب یا با طول بی‌نهایت در نظر گرفته شود. از سوی دیگر، STFT برای تجزیه و تحلیل سیگنال‌هایی استفاده می‌شود که دارای ویژگی‌های غیر ثابت هستند، به این معنی که محتوای فرکانسی آنها در طول زمان تغییر می‌کند. STFT این کار را با تقسیم سیگنال به قسمت‌های کوتاه‌تر و اعمال تبدیل فوریه به هر بخش به طور جداگانه انجام می‌دهد. این کار اجازه می‌دهد تا محتوای فرکانسی سیگنال را در طول زمان بررسی کنیم.

اندازه پنجره Hamming مورد استفاده در STFT، بر رزولوشن فرکانسی و رزولوشن زمانی تجزیه و تحلیل تأثیر می‌گذارد. رزولوشن فرکانسی به توانایی تمایز بین دو فرکانس نزدیک به هم اشاره دارد. اندازه پنجره بزرگتر به وضوح فرکانسی بهتری منجر می‌شود زیرا نمایش دقیق‌تری از محتوای فرکانسی سیگنال ارائه می‌دهد. رزولوشن زمانی اما به توانایی نمایش دقیق تغییرات سیگنال در طول زمان اشاره دارد. اندازه پنجره کوچکتر رزولوشن زمانی بهتری را ارائه می‌دهد و امکان تجزیه و تحلیل دقیق‌تری از چگونگی تغییر سریع محتوای فرکانسی سیگنال در طول زمان را فراهم می‌کند. در واقع، هنگام تغییر اندازه پنجره Hamming در STFT، بین رزولوشن فرکانسی و رزولوشن زمانی یک Trade-off وجود خواهد داشت. یک پنجره بزرگتر رزولوشن فرکانسی را بهبود می بخشد اما رزولوشن زمانی را بدتر می‌کند و بالعکس. وقتی اندازه پنجره Hamming افزایش می‌یابد، لوب اصلی تبدیل فوریه پنجره باریک‌تر می‌شود، که این مسئله رزولوشن فرکانسی را بهبود می‌بخشد اما لوب‌های جانبی را گسترده می‌کند باعث افزایش نشت به فرکانس‌های مجاور می‌شود. به عکس اندازه پنجره کوچکتر دارای لوب اصلی گسترده‌تری

خواهد بود که رزولوشن فرکانسی را کاهش می دهد اما لوب‌های جانبی را نیز کاهش می‌دهد که می‌تواند باعث بهتر شدن رزولوشن زمانی شود. در شکل ۹ هم به وضوح تغییرات زمانی (افقی) در طول ۲۵۶ نقطه واضح‌تر است و تغییرات فرکانسی (عمودی) در طول ۵۱۲ واضح‌تر شده.

نهایتا اینکه می‌توان هر سه خروجی spec را گرفت و با دستور mesh هم رسم کرد:

```
figure(13);
mesh(w, T, abs(spc)');

figure(14);
mesh(w1, T1, abs(spc1)');
```

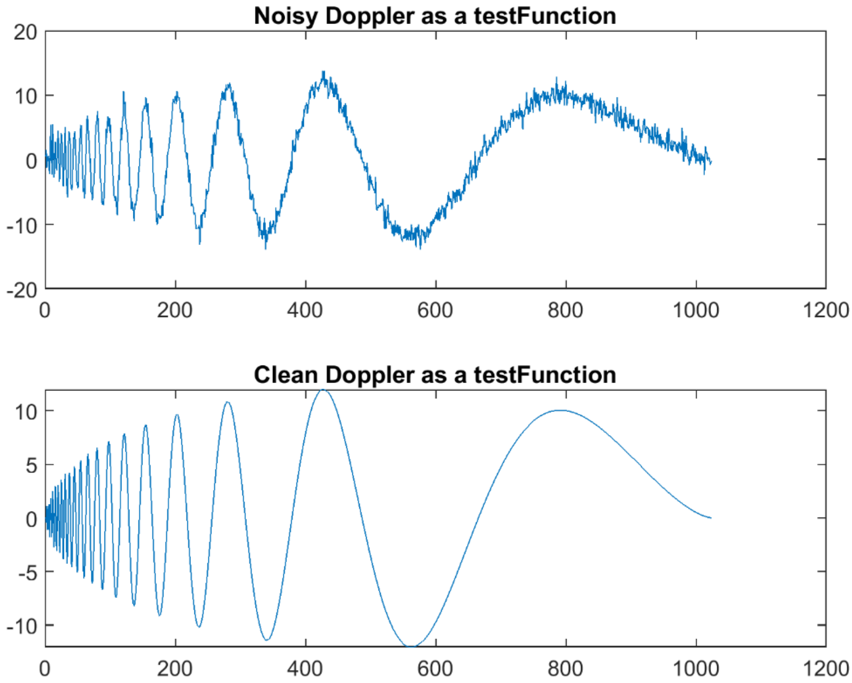


شکل ۱۲

بخش ۲-۵)

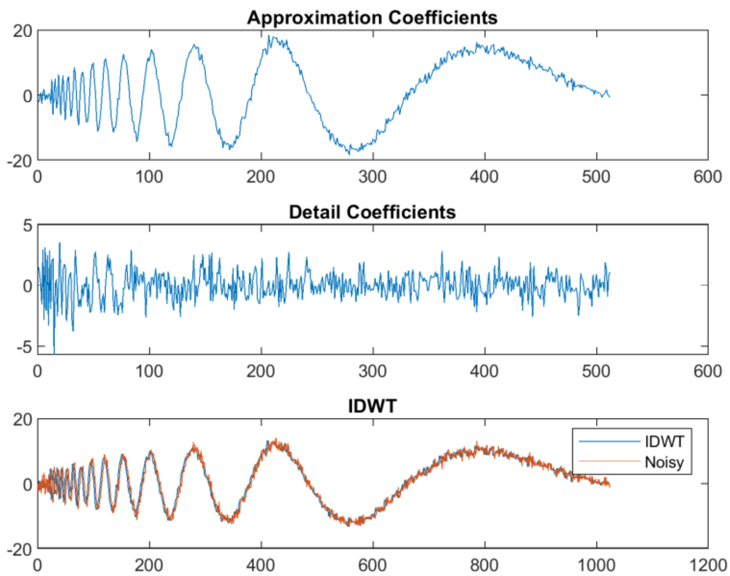
```
[clean, noisy] = wnoise('doppler', 10, 7);


figure(10);
subplot(2,1,1)
plot(noisy);
title("Noisy Doppler as a testFunction");
subplot(2,1,2)
plot(clean);
title("Clean Doppler as a testFunction");


figure(11);
subplot(3,1,1);
plot(cA);
title("Approximation Coefficients");
subplot(3,1,2);
plot(cD);
title("Detail Coefficients");
xrec = idwt(cA, zeros(size(cA)),'sym4');
subplot(3,1,3);
plot(xrec);
title("IDWT");
```

در این بخش با استفاده از تابع wnoise سیگنال نویزی Doppler ایجاد کردیم و هر دو سیگنال تمیز و نویزی را رسم کردیم.



شکل ۱۳

سپس از سیگنال نویزی dwt گرفته و خروجی‌های cA و cD را رسم کردیم. Dwt با شروع از سیگنال ورودی، دو مجموعه از ضرایب را محاسبه می‌کند؛ ضرایب تقریبی cA، و ضرایب دیتیلد cD. با گذشتن سیگنال از فیلتر موجک پایین گذری و به دنبال آن downsample با نسبت ۲ ضرایب تقریبی به دست می‌آیند. با گذشتن سیگنال ورودی از فیلتر موجک بالاگذری و به دنبال آن downsample با نسبت ۲ ضرایب دیتیلد به دست می‌آیند. یک بار هم از cA حاصله، idwt گرفته و نتیجه را رسم کردیم.



شکل ۱۴