Amirkabir University of Technology
(Tehran Polytechnic)

پروژه ی درس سیگنال ها و سیستم ها

ترم دوم سال تحصیلی 1399-1400

استاد درس: خانم دکتر عبداللهی

عنوان پروژه:

پروژه ی مخابرات

اعضای گروه:

حمیدرضا دشت آبادی

(9823032)

فروغ افخمی

(9823006)

عرفان راستی

(9823034)

ابوالفضل میکانیکی

(9823088)

بخش مقدماتی:

قسمت 1:

الف) کد مربوط به این بخش فایل **Code 1 - ElementaryEx1Part1.m** می باشد:

```
% Defining Variables
f0 = 0;
f1 = 500;
T = 10;
fs = 8000;
c = (f1 - f0) / T;

% defining time domain
t = 0:(1 / fs):10;

% defining function
x = 0.5 * sin(2 * pi * ((c / 2) * (t.^2) + (f0 * t)));

% playing the sound
sound(x, fs)
```

Code 1 – ElementaryEx1Part1.m

در ابتدای کد متغیر های مورد نظر تعریف شده اند. در بخش دوم، محور زمان تابع خود را تعریف می کنیم که تا زمان 10 ثانیه می باشد. سپس تابع مورد نظر را تعریف می کنیم که بر حسب متغیر زمان می باشد. همچنین محور زمان به صورت یک آرایه تعریف شده‌است. پس اعمال انجام شده بر روی محور زمان، که در تابع استفاده شده اند، باید به صورت ماتریسی باشند. در نهایت با فراخوانی تابع **sound** با فرکانس نمونه برداری **8000hz** این تابع را پخش می کنیم.

ب) با ارجاع به فایل **Code 2 - ElementaryEx1Part2.m** برنامه ی این بخش قابل مشاهده می باشد:

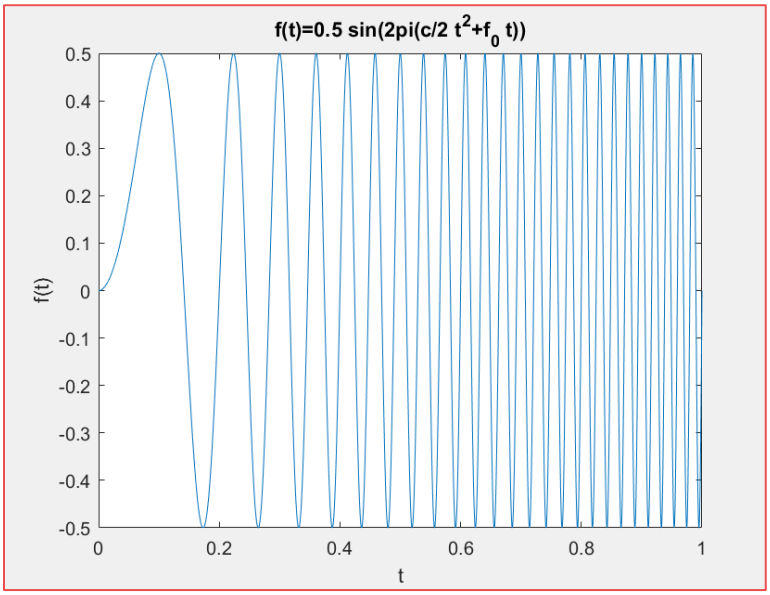
```
% Defining Variables
f0 = 0;
f1 = 500;
T = 10;
fs = 8000;
c = (f1 - f0) / T;

% defining time domain
t = 0:(1 / fs):1;

% defining function
x = 0.5 * sin(2 * pi * ((c / 2) * (t.^2) + (f0 * t)));

% plotting the function
plot(t, x)
xlabel("t")
ylabel("f(t)")
title("f(t)=0.5 sin(2\pi(c/2 t^2+f_0 t))");
grid on
```

Code 2 – ElementaryEx1Part2.m

خروجی این برنامه با استفاده از تابع **plot** می باشد که تصویر نمودار تابع قسمت الف را نشان می دهد. معادله ی تابع مورد نظر به شکل زیر می باشد:

$$f(t) = \frac{1}{2}\sin\left(2\pi\left(\frac{c}{2}t^2 + f_0 t\right)\right)$$

شکل خروجی نمودار این تابع با مقادیر مشخص شده در قسمت الف شکل 1 است:



شکل 1

با توجه به این نمودار این نکته قابل تشخیص است که با حرکت بر روی محور زمان از صفر به سمت مقادیر مثبت، نوسانات نمودار شکل 1بیشتر می شود و نمودار فشرده می شود. در نتیجه فرکانس لحظه ای افزایش می یابد. در نتیجه با گذشت زمان، فرکانس صدای شنیده شده بیشتر می شود. به عبارتی صدا با گذشت زمان از بم به سمت زیر حرکت می کند که با تحلیل نمودار مطابقت دارد.

پ) با مراجع به بخش **help** در نرم افزار **MATLAB** متوجه می شویم که بازه ی فرکانسی دستور**sound** نمی تواند از عدد **1000hz** کمتر باشد(شکل 2):

Fs — Sample rate
8192 (default) | positive number

Sample rate, in hertz, of audio data y, is specified as a positive number from 1000 through 384000. Valid values depend on both the sample rates permitted by MATLAB® and the specific audio hardware on your system. MATLAB has a hard restriction of 1000 Hz <= Fs <= 384000 Hz, although further hardware-dependent restrictions apply.

Data Types: `single` | `double`

شکل 2

به همین دلیل است که با تنظیم فرکانس نمونه برداری بر روی 800 با خطای زیر مواجه می شویم(شکل 3):

```
>> ElementaryEx1Part3
Error using sound (line 79)
Device Error: Invalid sample rate


Error in ElementaryEx1Part3 (line 15)
sound(x, fs)
```

شکل 3

برنامه ی این بخش به صورت زیر است(Code 3 - ElementaryEx1Part3.m):

```
% Defining Variables
f0 = 0;
f1 = 500;
T = 10;
fs = 800;
c = (f1 - f0) / T;

% defining time domain
t = 0:(1 / fs):10;

% defining function
x = 0.5 * sin(2 * pi * ((c / 2) * (t.^2) + (f0 * t)));

% playing the sound
sound(x, fs)
```

Code 3 - ElementaryEx1Part3.m

قسمت 2:

الف) برنامه فایل Code 4 - ElementaryEx2Part1.m می باشد:

```
% loading the signal
load("Audio_Signals\file1.asc")

% playing the original signal
soundsc(file1, 16000)
disp("This is the original signal.");
disp("Press enter to continue.");
disp("-----------------------------------------------");
pause;

% wrong interpolated signal
newInterpSig = interp(file1, 4);
soundsc(newInterpSig, 32000)
disp("This is the wrong interpolated signal(just for testing).");
disp("Press enter to continue.");
disp("-----------------------------------------------");
pause;

% correct interpolated signal
newInterpSig = interp(file1, 4);
soundsc(newInterpSig, 64000)
disp("This is the correct interpolated signal.");
```

Code 4 - ElementaryEx2Part1.m

ابتدا فایل صوتی را **load** کردیم. سپس این سیگنال را با فرکانس نمونه برداری اصلی خود یعنی **16000hz** به وسیله ی دستور **soundsc** پخش کردیم.

سپس با افزایش نرخ نمونه برداری سیگنال با استفاده از دستور **interp** نرخ نمونه برداری را **4** برابر کردیم. سپس برای پخش دوباره ی سیگنال با تغییر مقادیر فرکانس در تابع **soundsc** به این نتیجه می رسیم که باید آرگومان دوم تابع **soundsc** را نیز **4** برابر بکنیم تا صدا همانند قبل شنیده شود.

ب) در این بخش می خواهیم با استفاده از دستور **decimate** سیگنال مورد نظر را با نرخ نمونه برداری 0.2 بررسی کنیم. در ابتدا اگر تنها از این دستور استفاده کنیم، باید فرکانس تابع **soundsc** را بر روی 3200 تنظیم کنیم. در این حالت صدا با کیفیت اولیه پخش نمی شود. برای رفع این مشکل ابتدا با استفاده از دستور **interp** سیگنال را **expand** می کنیم. سپس از دستور **decimate** استفاده می کنیم و سیگنال را فشرده می کنیم.

در فایل **Code 5 - ElementaryEx2Part2.m** ابتدا سیگنال مورد نظر را با نرخ نمونه برداری 4 بررسی کردیم. سپس نرخ نمونه برداری 0.2 را برروی سیگنال اعمال کردیم. در نهایت فرکانس نهایی کمی از مقدار اولیه پایین تر خواهد آمد.(0.8 فرکانس اولیه خواهد شد.) پس فرکانس استفاده شده در دستور **soundsc** 12800 خواهد بود. در این حالت صدای شنیده شده با تقریب خوبی مانند صدای اصلی می باشد.

```
% loading the signal
load("Audio_Signals\file1.asc")

% playing the original signal
soundsc(file1, 16000)
disp("This is the original signal.");
disp("Press enter to continue.");
disp("-----------------------------------------------");
pause;

% wrong decimated signal
newDecimSig = decimate(file1, 5);
soundsc(newDecimSig, 3200)
disp("This is the wrong decimated signal(just for testing).");
disp("Press enter to continue.");
disp("-----------------------------------------------");
pause;

% correct signal
newSignal = interp(file1, 4);
newSignal = decimate(newSignal, 5);
soundsc(newSignal, 12800)
disp("This is the correct decimated signal.");
```

Code 5 - ElementaryEx2Part2.m

پ) با توجه به توضیحات بند الف و ب، نرخ نمونه برداری در بخش الف 4 برابر می شود و فرکانس نمونه برداری برابر 64000hz می شود. در بخش ب نیز نرخ نمونه برداری 0.8 برابر شد و فرکانس نمونه برداری برابر 12800 شد.

قسمت 3:

الف) در این بخش با توجه به فایل Code 6 - ElementaryEx3Part1.m ابتدا متغیر ها را تعریف کرده و سیگنال اصلی را load و پخش کردیم. سپس سیگنال را با نرخ نمونه برداری 0.125 بررسی کردیم و با استفاده از فیلتر Chebyshev از درجه ی N سیگنال را فیلتر کردیم. اکنون پس از پخش این صدا و مقایسه ی آن با سیگنال اصلی متوجه می شویم که سیگنال اصلی دست خوش تغییر شده است و کیفیت قبل را ندارد.

```
% defining variables
R = 8;
N = 60;

% loading the signal
load("Audio_Signals\file1.asc")

% playing the original signal
soundsc(file1, 16000)
disp("This is the original signal.");
disp("Press enter to continue.");
disp("-----------------------------------------------");
pause;

% decimated signal
decimSig = decimate(file1, R, N, 'fir');
soundsc(decimSig, 2000)
disp("This is the decimated and filtered signal(using chebyshev filter).");
disp("Press enter to continue.");
disp("-----------------------------------------------");
```

Code 6 - ElementaryEx3Part1.m

ب) در این بخش سیگنال قسمت قبلی را دوباره با نرخ 8 برابر به فرکانس قبلی بازگرداندیم(**Code 7 -** ElementaryEx3Part2.m):

```
% defining variables
R = 8;
N = 60;

% loading the signal
load("Audio_Signals\file1.asc")

% playing the original signal
soundsc(file1, 16000)
disp("This is the original signal.");
disp("Press enter to continue.");
disp("-----------------------------------------------");
pause;

% decimated signal
newSig = decimate(file1, R, N, 'fir');
newSig = interp(newSig, 8);
soundsc(newSig, 16000)
disp("This is the decimated and filtered signal(using chebyshev filter).");
disp("Press enter to continue.");
disp("-----------------------------------------------");
```

Code 7 - ElementaryEx3Part2.m

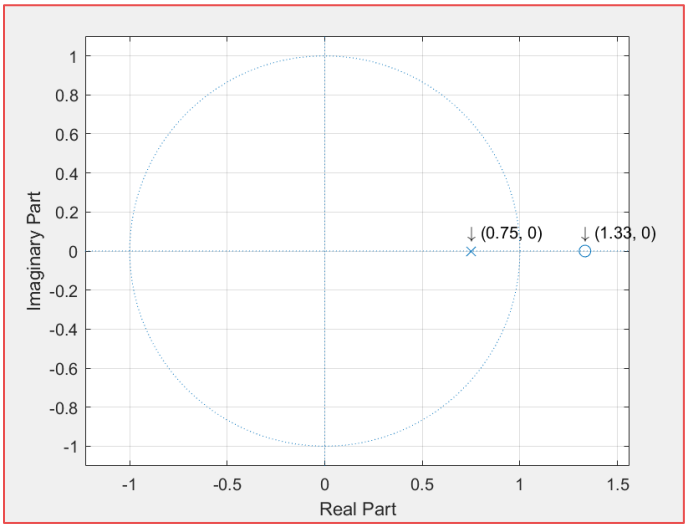
پ) بعد از اعمال تابع **decimate**، بخشی از فرکانس های موجود در سیگنال اصلی حذف می شوند. اکنون اگر سعی کنیم این سیگنال جدید را با استفاده از دستور **interp** به سیگنال اصلی بازگردانیم، به دلیل حذف بعضی از فرکانس ها در تابع **decimate** سیگنال کامل بازیابی نمی شود و کیفیت اصلی را ندارد.

ت) استفاده از بلوک **decimation** قبل از بلوک **interpolation** درست نمی باشد و باعث حذف بخشی از فرکانس های موجود در تابع می شود. در نتیجه صدای شنیده شده محتوای فرکانسی سیگنال اصلی را ندارد و کیفیت خود را از دست داده است. زیاد بعضی از فرکانس ها حذف شده و با صفر جایگزین شده اند.

بخش متوسط:

قسمت 1:

الف) در این بخش ابتدا ضرایب توابع **y** و **x** را در متغیرهای **a** و **b** ذخیره کردیم. سپس با استفاده از تابع **zplane**، قطب ها و صفرها را در صفحه ی **z** نمایش دادیم. همچنین مقدار صفر ها و قطب ها بر روی نمودار قابل مشاهده هستند(شکل **4**).



شکل 4

```
% diffrence equation:y[n] - 3/4y[n - 1] = -3/4x[n] + x[n - 1]
% defining variables
% a: coefficients of y
% b: coefficients of x
a = [1 -3/4];
b = [-3/4 1];

zplane(b, a);
text(0.73, 0.1, '\downarrow (0.75, 0)');
text(1.315, 0.1, '\downarrow (1.33, 0)');
grid on
```

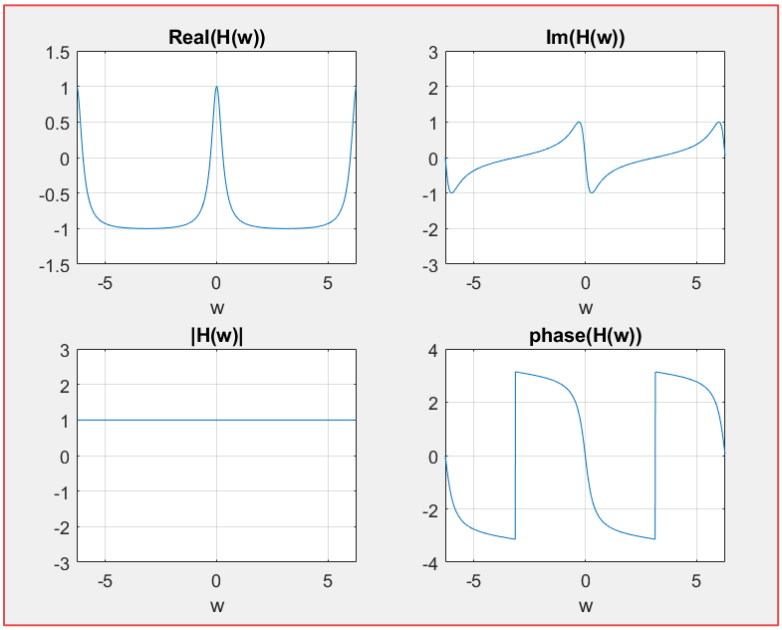
Code 8 - IntermediateEx1Part1.m

ب) در این بخش ابتدا پاسخ فرکانسی سیستم مطرح شده را بدست آوردیم. برای این کار از تابع **freqz** استفاد کردیم. سپس دامنه، فاز، بخش حقیقی و بخش موهومی این تابع را بدست آوردیم(**Code 9 - IntermediateEx1Part2.m**):

```
% diffrence equation:y[n] - 3/4y[n - 1] = -3/4x[n] + x[n - 1]
% defining variables
% a: coefficients of y
% b: coefficients of x
a = [1 -3/4];
b = [-3/4 1];

% defining w-axis
w = -2 * pi:0.01:2 * pi;

% H: frequency response
H = freqz(b, a, w);

% plotting the frequency response
subplot(2, 2, 1);
plot(w, real(H))
axis([-2 * pi 2 * pi -1.5 1.5])
title('Real(H(w))')
xlabel('w')
grid on

subplot(2, 2, 2);
plot(w, imag(H))
axis([-2 * pi 2 * pi -3 3])
title('Im(H(w))')
xlabel('w')
grid on

subplot(2, 2, 3);
plot(w, abs(H))
axis([-2 * pi 2 * pi -3 3])
title('|H(w)|')
xlabel('w')
grid on
```

```
subplot(2, 2, 4);
plot(w, angle(H))
axis([-2 * pi 2 * pi -4 4])
title('phase(H(w))')
xlabel('w')
grid on
```

Code 9 - IntermediateEx1Part2.m



شکل 5

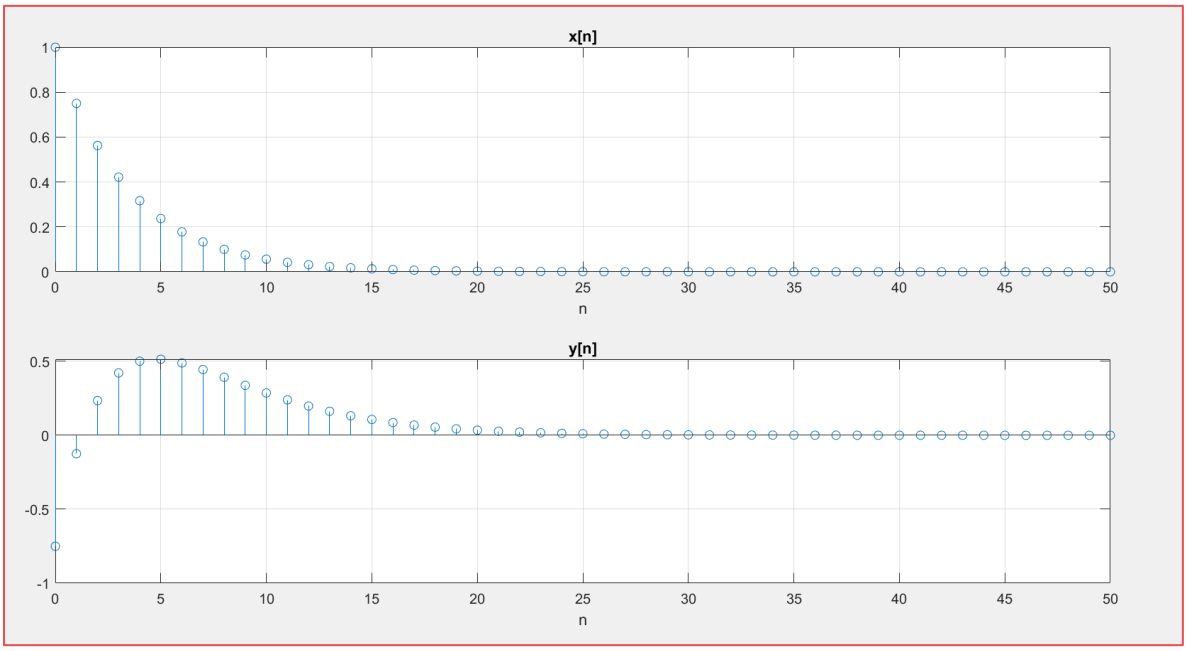
با مشاهده ی نمودار های رسم شده(شکل 5)و بررسی نمودار اندازه ی پاسخ فرکانسی، باتوجه به اینکه اندازه ی پاسخ فرکانسی عدد ثابت 1 می باشد، متوجه می شویم که این فیلتر تمام گذر می باشد.

قسمت 2:

نام فایل نوشته شده برای این بخش **Code 10 - IntermediateEx2.m** می‌باشد. با استفاده از دستور **filter**، سیستم مورد نظر را بر روی سیگنال ورودی اعمال کردیم. نمودار سیگنال های ورودی و خروجی به صورت زیر می باشند(شکل 6):

```
% defining the input signal
n = 0:50;
x = (3/4).^n;

% diffrence equation:y[n] - 3/4y[n - 1] = -3/4x[n] + x[n - 1]
% defining variables
% a: coefficients of y
% b: coefficients of x
a = [1 -3/4];
b = [-3/4 1];

% defining output function
y = filter(b, a, x);

% plotting input signal
subplot(2, 1, 1);
stem(n, x)
title('x[n]');
xlabel('n');
grid on

% plotting output signal
subplot(2, 1, 2);
stem(n, y)
title('y[n]');
xlabel('n');
grid on
```

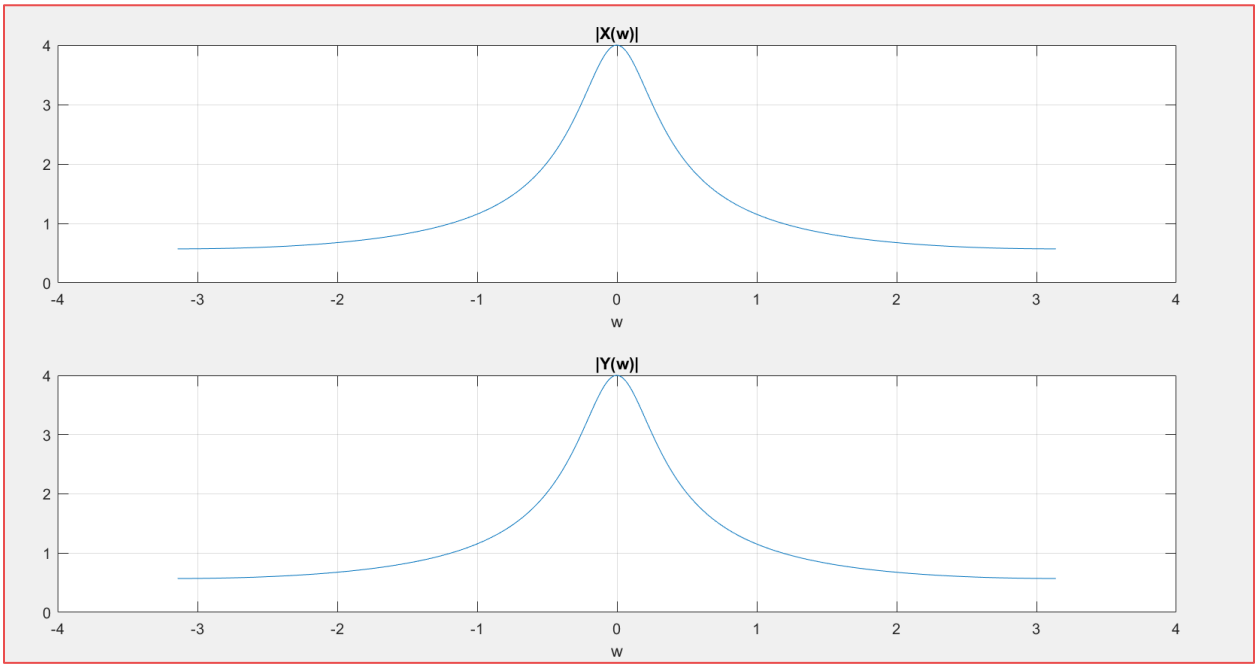
Code 10 - IntermediateEx2.m



شکل 6

قسمت 3:

الف) فایل مربوط به این بخش **Code 11 - IntermediateEx3.m** می‌باشد. نمودار تبدیل فوریه ی سیگنال های ورودی و خروجی به شکل زیر می باشند(شکل **7**):



شکل 7

```
% defining the input signal
n = 0:50;
x = (3/4).^n;

% defining w-axis
w = -pi:0.01:pi;

% diffrence equation:y[n] - 3/4y[n - 1] = -3/4x[n] + x[n - 1]
% defining variables
% a: coefficients of y
% b: coefficients of x
a = [1 -3/4];
b = [-3/4 1];
```

```
% defining output function
y = filter(b, a, x);

% defining fourier transform of input and output signals
Xw = fft(x, length(w));
Xw = fftshift(Xw);
Yw = fft(y, length(w));
Yw = fftshift(Yw);

% plotting fourier transform of input signal
subplot(2, 1, 1);
plot(w, abs(Xw))
title('|X(w)|');
xlabel('w');
grid on

% plotting fourier transform output signal
subplot(2, 1, 2);
plot(w, abs(Yw))
title('|Y(w)|');
xlabel('w');
grid on
```

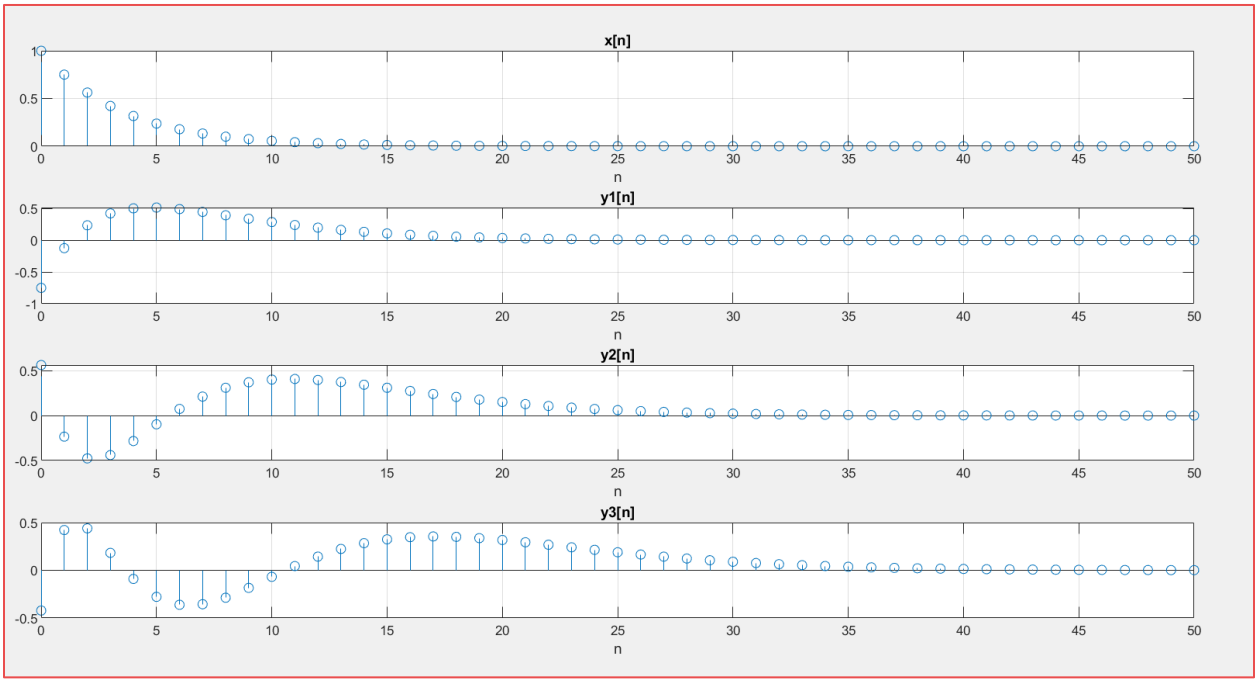
Code 11 - IntermediateEx3.m

برای محاسبه ی تبدیل فوریه ی این دو سیگنال، ابتدا از دستور **fft** استفاده کردیم. سپس با استفاده از دستور **fftshift** مکان تبدیل فوریه ی این دو سیگنال را به صورت متقارن حول محور قائم قرار دادیم.

ب) با توجه به قسمت های قبلی دیدیم که فیلتر استفاده شده تمام گذر می باشد. به همین دلیل اندازه ی تبدیل فوریه ی سیگنال های ورودی و خروجی کاملا مشابه بودند. هر چند که با توجه به بخش الف، می توان مشاهده کرد خود سیگنال خروجی متفاوت با سیگنال ورودی است.

قسمت 4:

الف) فایل مربوط به این قسمت **Code 12 - IntermediateEx4Part1.m** نام دارد. فیلتر را سه بار به صورت سری بر روی ورودی ذکر شده اعمال کردیم و خروجی هر مرحله را در متغیر جدیدی ذخیره کردیم. سپس نمودار مربوط به سیگنال ورودی و سیگنال های خروجی را رسم کردیم.(شکل 8)



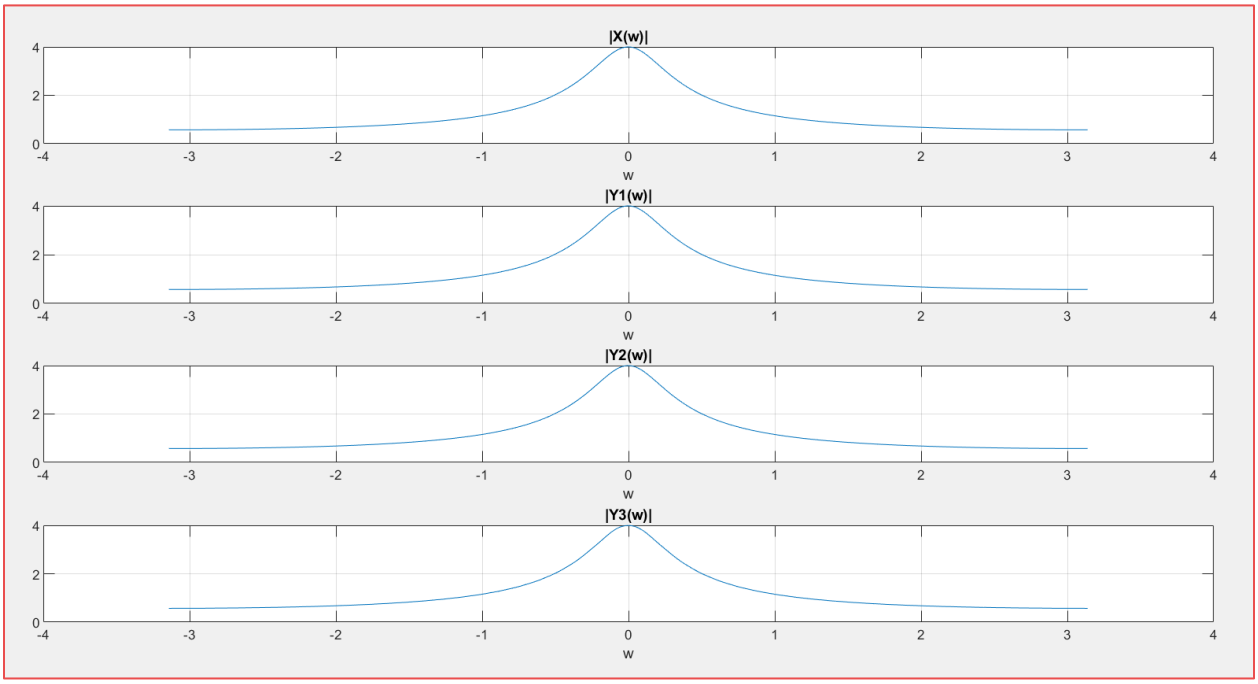
شکل 8

```
% diffrence equation:y[n] - 3/4y[n - 1] = -3/4x[n] + x[n - 1]
% defining variables
% a: coefficients of y
% b: coefficients of x
a = [1 -3/4];
b = [-3/4 1];

% defining the input signal
n = 0:50;
x = (3/4).^n;

% defining output function 1
```

```
y1 = filter(b, a, x);

% defining output function 2
y2 = filter(b, a, y1);

% defining output function 2
y3 = filter(b, a, y2);

% plotting input function
subplot(4, 1, 1);
stem(n, x)
title('x[n]');
xlabel('n');
grid on

% plotting output function 1
subplot(4, 1, 2);
stem(n, y1)
title('y1[n]');
xlabel('n');
grid on

% plotting output function 2
subplot(4, 1, 3);
stem(n, y2)
title('y2[n]');
xlabel('n');
grid on

% plotting output function 3
subplot(4, 1, 4);
stem(n, y3)
title('y3[n]');
xlabel('n');
grid on
```

Code 12 - IntermediateEx4Part1.m

ب) در این مرحله از سیگنال های قسمت الف، تبدیل فوریه گرفتیم. برای این کار همانند قسمت 3، از دستور های **fft** و **ffitshift** استفاده کردیم. مشاهده می شود اندازه ی تبدیل فوریه ی هر سه نمودار یکسان می باشند(شکل 9).



شکل 9

```
% diffrence equation:y[n] - 3/4y[n - 1] = -3/4x[n] + x[n - 1]
% defining variables
% a: coefficients of y
% b: coefficients of x
a = [1 -3/4];
b = [-3/4 1];

% defining the input signal
n = 0:50;
x = (3/4).^n;

% defining w-axis
w = -pi:0.01:pi;

% defining output function 1
```

```
y1 = filter(b, a, x);

% defining output function 2
y2 = filter(b, a, y1);

% defining output function 2
y3 = filter(b, a, y2);



% defining fourier transform of input signal
Xw = fft(x, length(w));
Xw = fftshift(Xw);

% defining fourier transform of 3 output siganls
Y1w = fft(y1, length(w));
Y1w = fftshift(Y1w);

Y2w = fft(y2, length(w));
Y2w = fftshift(Y2w);

Y3w = fft(y3, length(w));
Y3w = fftshift(Y3w);

% plotting fourier transform of input signal
subplot(4, 1, 1);
plot(w, abs(Xw))
title('|X(w)|');
xlabel('w');
grid on

% plotting fourier transform of output signal 1
subplot(4, 1, 2);
plot(w, abs(Y1w))
title('|Y1(w)|');
xlabel('w');
grid on
```

```
% plotting fourier transform of output signal 2
subplot(4, 1, 3);
plot(w, abs(Y2w))
title('|Y2(w)|');
xlabel('w');
grid on

% plotting fourier transform of output signal 3
subplot(4, 1, 4);
plot(w, abs(Y3w))
title('|Y3(w)|');
xlabel('w');
grid on
```

Code 13 - IntermediateEx4Part2.m

ج) برنامه ی این بخش **Code 14 - IntermediateEx4Part3.m** می باشد. در این بخش انرژی سیگنال ورودی و سیگنال های خروجی را به صورت سیگما محاسبه کردیم. برای این کار از تابع **sum** استفاده کردیم. در نهایت مقادیر بدست آمده را به وسیله ی تابع **fprintf** چاپ کردیم. خروجی به صورت زیر بدست آمد(شکل 10):

```
>> IntermediateEx4Part3
En{x[n]} = 2.2857

En{y1[n]} = 2.2857

En{y2[n]} = 2.2857

En{y3[n]} = 2.2857
```

شکل 10

مشاهده می شود که انرژی سیگنال ورودی، با سیگنال های خروجی برابر است. این موضوع از رابطه ی پارسوال نیز قابل استنباط است:

$$\sum_{n=-\infty}^{+\infty} |x[n]|^2 = \frac{1}{2\pi} \int_{2\pi} |X(j\omega)|^2 d\omega$$

از آنجایی که اندازه ی تبدیل فوریه ی سیگنال ها یکی بود، پس بخش راست تساوی برای هر **4** سیگنال یکی می باشد. در نتیجه انرژی سیگنال ها که با عبارت سمت چپ تساوی بدست می آید نیز یکسان خواهد بود.

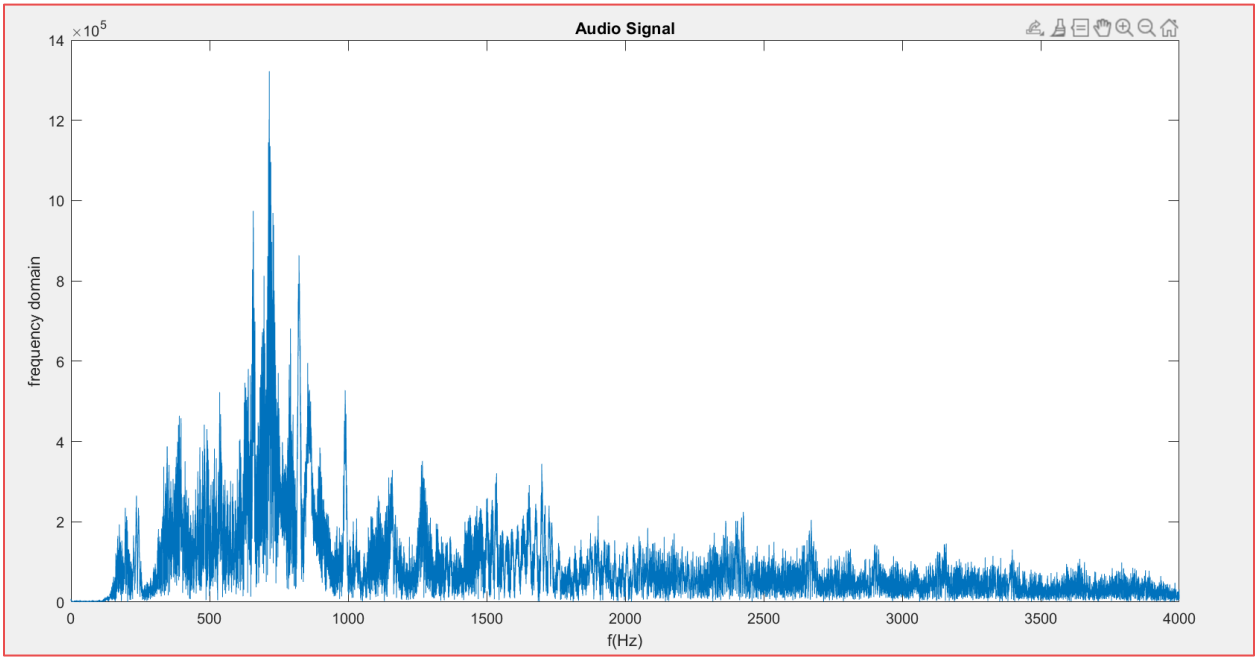
```
% diffrence equation:y[n] - 3/4y[n - 1] = -3/4x[n] + x[n - 1]
% defining variables
% a: coefficients of y
% b: coefficients of x
a = [1 -3/4];
b = [-3/4 1];

% defining the input signal
n = 0:50;
x = (3/4).^n;

% defining output function 1
y1 = filter(b, a, x);

% defining output function 2
y2 = filter(b, a, y1);

% defining output function 2
y3 = filter(b, a, y2);

Enx = sum(x.^2);
Eny1 = sum(y1.^2);
Eny2 = sum(y2.^2);
Eny3 = sum(y3.^2);
```

```
fprintf(['En{x[n]} = %.4f\n\n', ...
         'En{y1[n]} = %.4f\n\n', ...
         'En{y2[n]} = %.4f\n\n', ...
         'En{y3[n]} = %.4f\n\n'], ...
    Enx, Eny1, Eny2, Eny3);
```

Code 14 - IntermediateEx4Part3.m

بخش پیشرفته:

الف) فایل مربوط به این بخش **Code 15 - AdvancedEx1Part1.m** نام دارد. در این بخش ابتدا فایل صوتی **file2.asc** را پخش کردیم. سپس به کمک تابع **FT** که خودمان آن را تشکیل دادیم، از فایل صوتی تبدیل فوریه گرفتیم. طیف فرکانسی این سیگنال به صورت زیر می باشد(شکل 11):



شکل 11

تبدیل فوریه ی سیگنال تخمینی از طیف فرکانسی می باشد که درشکل 11 مشاهده می کنیم.

```
% loading the signal
load("Audio_Signals\file2.asc");

% defining variables
fs = 8000;

% playing the sound
disp("The advanced part of the project starts in 1 seconds...");
pause(1);
soundsc(file2, fs, 16);

% FF2 -> frequency-axis of file2
% FTF2 -> fourier transform of file 2
[FF2, FTF2] = FT(fs, file2);

% plotting the signal
plot(FF2, FTF2);
xlabel("f(Hz)");
ylabel("frequency domain");
title("Audio Signal");
```

Code 15 - AdvancedEx1Part1.m

تابع FT:

```
function [f, Xw] = FT(fs, xn)
    %{
    Inputs:
    fs - > sampling frequency
    xn - > discrete signal

    Outputs:
    f - > frequency range
    Xw - > fourier transform
    %}
    n = 2^nextpow2(length(xn));
    y = fft(xn, n);
    y = abs(y);
    N = length(y);
    Xw = y(1:N / 2);
    f = (0:N / 2 - 1) .* fs / N;
end
```

Code 16 - FT.m

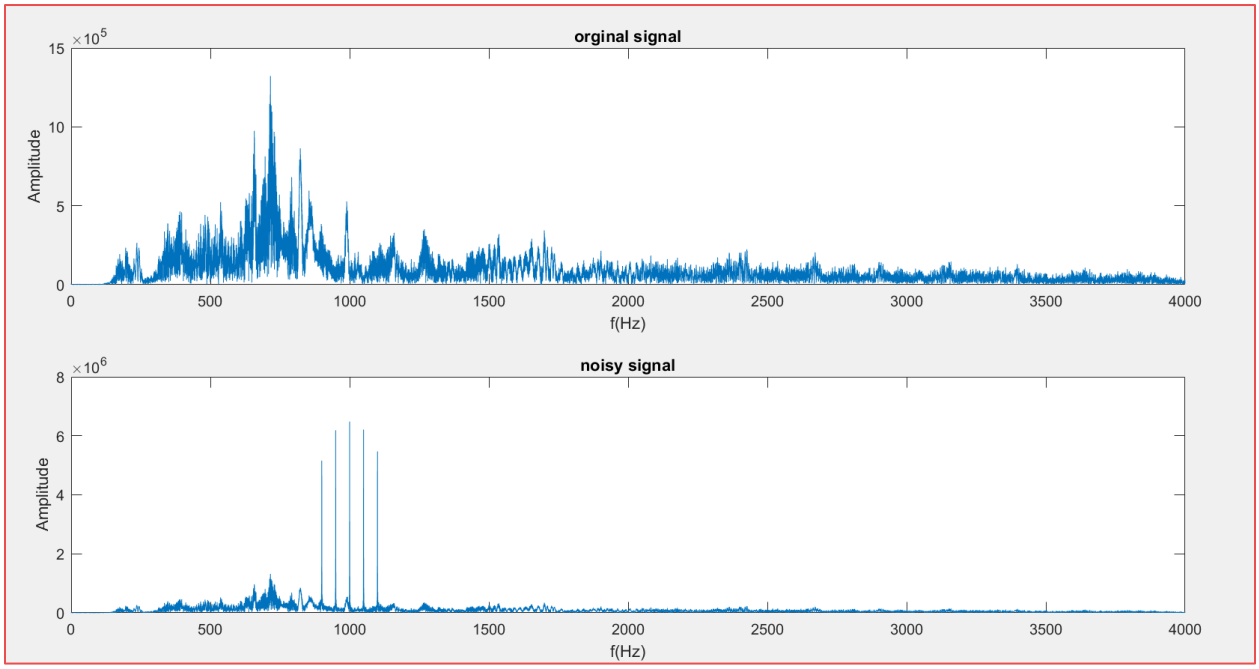
برای محاسبه ی تبدیل فوریه سیگنال صوتی و محور **f** تابعی به نام **FT** را تشکیل دادیم. در بطن این تابع از دستور **fft** استفاده شده است. دستور **fft** الگوریتم سریع محاسبه ی **DFT (Discrete Fourier Transform)** می باشد.

در علم کامپیوتر برای بررسی رفتار سیگنال ها در حوزه ی زمان و فرکانس، پیاده سازی محور پیوسته بسیار سخت می باشد. به همین دلیل باید از تبدیلی استفاده شود که هم ورودی این تبدیل سیگنال گسسته باشد و هم خروجی این تبدیل سیگنالی گسسته باشد. در روابط بررسی شده در درس سیگنال ها و سیستم ها تنها ضرایب سری فوریه ی سیگنال متناوب گسسته همچین ویژگی ای را دارا بود. به این صورت که هم خود سیگنال مورد آنالیز گسسته بود و هم ضرایب سری فوریه ی این سیگنال گسسته بودند. تبدیل **DFT** از روابط مربوط به ضرایب سری فوریه ی سیگنال گسسته حاصل می شود.

برای استفاده از دستور **fft** در کد **FT.m** دو آرگمان را به عنوان ورودی به این تابع می دهیم. آرگمان اول مقادیر **xn** و آرگمان دوم تعداد عناصر خروجی تابع **fft** می باشد که در غالب یک آرایه قرار می گیرند. برای مشخص کردن تعداد عناصر مقدار **n** را به

اینصورت تعریف کردیم که نخستین توان عدد 2 باشد که بزرگتر از طول آرایه ی **xn** است. (این که چرا تعداد عناصر خروجی را توانی از 2 فرض می کنیم به الگوریت محاسبه ی **fft** مربوط است که در هر مرحله از محاسبات ماتریسی، عملگر های مورد استفاده نصف می شوند.) در نهایت از تابع مورد نظر **abs** گرفتیم؛ زیرا دامنه ی تبدیل فوریه ی سیگنال برای این بخش اهمیت دارد. با توجه به روابط درس می دانیم دامنه ی تبدیل فوریه ی سیگنال حقیقی، تابعی زوج بر حسب محور فرکانس می باشد. از آنجایی که سیگنال صوتی یک سیگنال حقیقی است، پس دامنه ی تبدیل فوریه ی این سیگنال، تابعی زوج خواهد بود. پس با داشتن بخش اول ورودی و نصف دامنه ی تبدیل فوریه، باقی مقادیر قابل بازسازی هستند و اطلاعات جدیدی به ما نمی دهند. از این رو، تنها نصف تعداد مقادیر دامنه ی تبدیل فوریه را به عنوان خروجی برگرداندیم.

ب) قطعه کد نوشته شده برای این بخش **Code17 - AdvancedEx1Part2.m** نام دارد. ابتدا فایل های مربوطه را **load** کردیم. سپس سیگنال نویزی را پخش نمودیم. سپس از سیگنال اصلی و سیگنال نویزی تبدیل فوریه گرفتیم. دامنه ی حوزه ی فرکانس سیگنال ها به صورت زیر می باشند(شکل 12):



شکل 12

با مقایسه ی شکل دو سیگنال رسم شده، متوجه می شویم که فرق تبدیل فوریه ی این دو سیگنال در بازه ای حول فرکانس 1000hz می باشد. این نویز شامل 5 خط می باشد. از آنجایی که دامنه ی این نویز تفاوت فاحشی با نویز اصلی دارد، می توان متوجه شد تبدیل فوریه ی این نویز شامل 5 پالس ضربه می باشد. پس ماهیت اصلی این نویز سیگنال سینوسی می باشد.

```
% loading the signals
load("Audio_Signals\noisysignal.txt");
load("Audio_Signals\file2.asc");

% defining variables
fs = 8000;

% playing the signal
disp(" Press any key to play noisy signal.");
pause;
soundsc(noisysignal, fs, 16); %  The noisy one

% FNS -> frequency of noisysignal
% FTNS -> fourier transform of noisysignal
[FNS, FTNS] = FT(fs, noisysignal);

% FF2 -> frequency-axis of file2
% FTF2 -> fourier transform of file 2
[FF2, FTF2] = FT(fs, file2);

% displaying the signals
disp(" Press any key to plot the signals. ");
pause;

subplot(2, 1, 1);
plot(FF2, FTF2);
title("orginal signal");
xlabel("f(Hz)");
ylabel("Amplitude");

subplot(2, 1, 2);
plot(FNS, FTNS);
title("noisy signal");
```

```
xlabel("f(Hz)");
ylabel("Amplitude");
```

Code17 - AdvancedEx1Part2.m

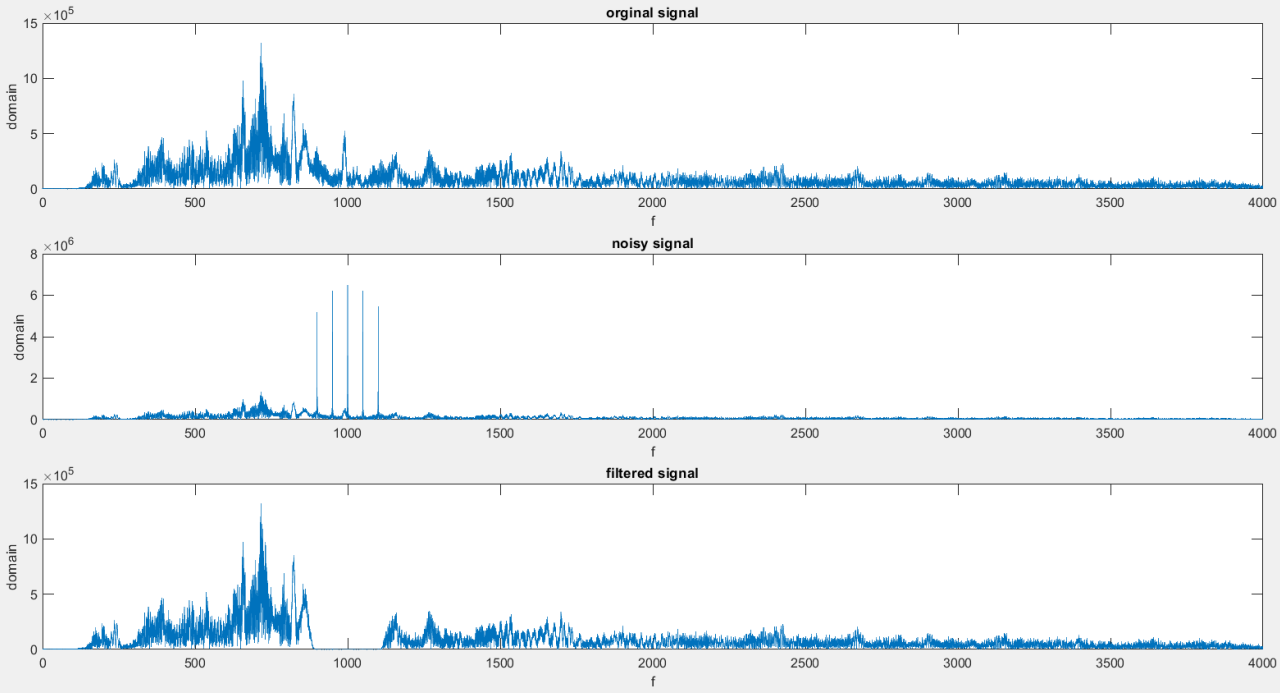
قسمت 2:

برنامه ی مربوط به این بخش **AdvancedEx2.m** نام دارد. در ابتدا سیگنال های مورد استفاده را **load** کردیم. سپس از سیگنال ها تبدیل فوریه گرفتیم تا در نمودار نهایی نمایش دهیم. در نهایت با استفاده از تابع **fir1** یک فیلتر میان نگذر در بازه ی حول **1000hz** تشکیل دادیم تا نویز سیگنال را پاک کند. در نهایت سیگنال فیلتر شده را پخش کردیم و سیگنال اصلی، سیگنال فیلتر شده و سیگنال نویزی را با یکدیگر به نمایش کشیدیم(شکل **13**).

```
% loading the signals
load("Audio_Signals\noisysignal.txt");
load("Audio_Signals\file2.asc");

% defining variables
fs = 8000;

% FNS -> frequency of noisysignal
% FTNS -> fourier transform of noisysignal
[FNS, FTNS] = FT(fs, noisysignal);

% FF2 -> frequency-axis of file2
% FTF2 -> fourier transform of file 2
[FF2, FTF2] = FT(fs, file2);

% definnig Wn_Frequency
% bound (Nyquist frequency is half the samplerate= fs/2 = 4000)
% Wn=[850 1150] 50hz offset is for possible noise.
% FIR -> Finite Impulse Response
Wn = [880 1120] / (fs / 2);
```

```
filt = fir1(2048, Wn, 'stop'); % 'stop' specifies a bandstop filter.

% Filter the output noisy signal.
filtered_signal = fftfilt(filt, noisysignal);

disp(" filtering...");
pause(1);

soundsc(filtered_signal, fs, 16);

% FFS -> frequency of filtered_signal
% FTFS -> fourier transform of filtered_signal
[FFS, FTFS] = FT(fs, filtered_signal);

% displaying signals
subplot(3, 1, 1);
plot(FF2, FTF2);
title("orginal signal");
xlabel("f");
ylabel("domain");

subplot(3, 1, 2);
plot(FNS, FTNS);
title("noisy signal");
xlabel("f");
ylabel("domain");

subplot(3, 1, 3);
plot(FFS, FTFS);
title("filtered signal");
xlabel("f");
ylabel("domain");
```

Code 18 - AdvancedEx2.m


orginal signal
noisy signal
filtered signal
domain
f
×10^5
×10^6

شکل13