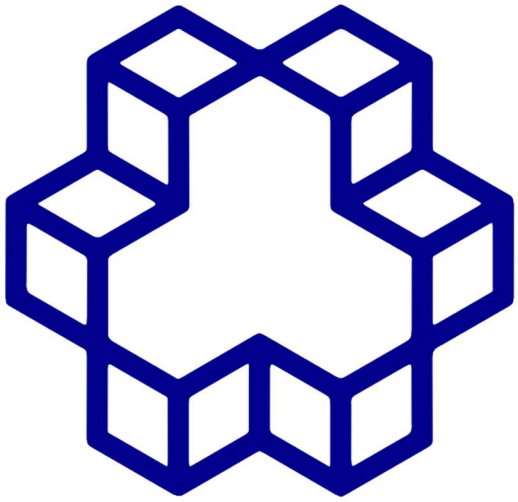


دانشگاه صنعتی خواجه نصیرالدین طوسی

گزارشکار تمرین کامپیوتری سری اول

درس پردازش سیگنال‌های دیجیتال

آنائیس گل‌بوداغیانس ۴۰۱۲۲۱۱۳

بخش اول

سوال اول

```
clc; clear; close all
```

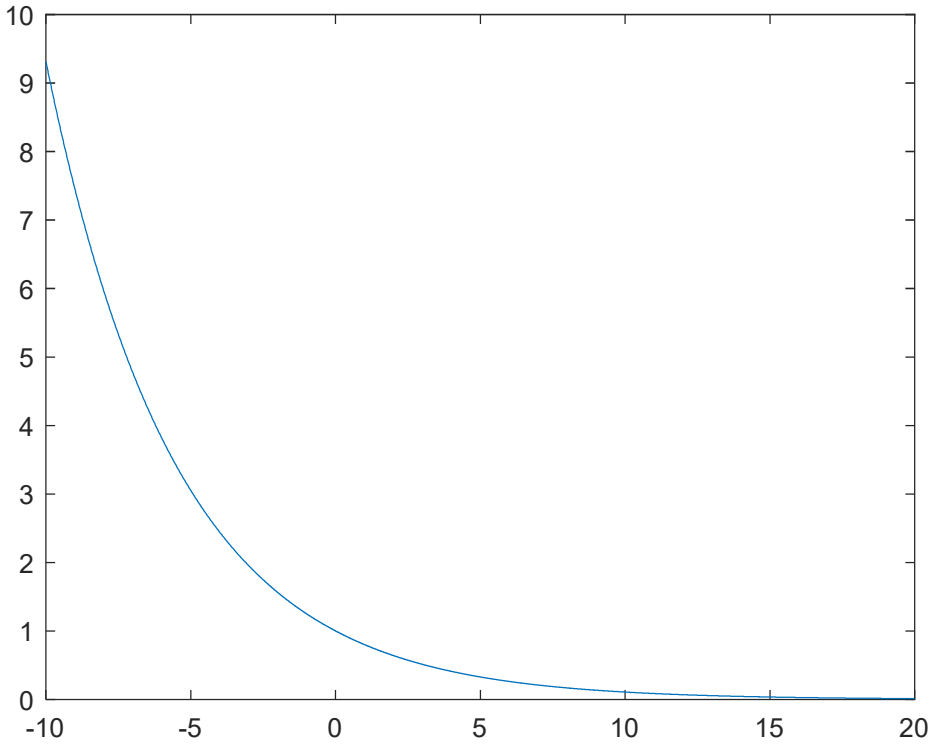
ابتدای هر کدی، با این سه دستور کارهای زیر را انجام می‌دهیم:

1. Command window را پاک می‌کنیم؛
2. متغیرها را پاک می‌کنیم؛
3. تمام پنجره‌ها را می‌بندیم.

ابتدا سیگنال را می‌خواهیم تعریف کنیم. درواقع نمی‌توان سیگنال پیوسته در متلب تعریف کرد و تمام سیگنال‌ها گسسته در زمان تعریف می‌شوند. درواقع همان نرخ نمونه برداری یا T، گام بردار n می‌باشد.

```
%CA1, Anaies Golboudaghians 40122113 DSP
%Part 1
%Q1
%x1
n1_1 = -10;
n2_1 = 20;
T_1 = 0.01;
n=n1_1:T_1:n2_1;
x1 = (0.8).^n;
```

اینجا نرخ را 0.01 انتخاب کرده‌ایم. می‌توانستیم عدد کوچک‌تری نیز انتخاب کنیم اما سرعت اجرا و پردازش کد کندتر می‌شد. در عین حال گام انتخاب شده سیگنال را به‌اندازه کافی پیوسته نشان می‌دهد.



دستور رسم تصویر فوق در کد اصلی موجود نیست اما چون می‌خواستیم پیوستگی را به‌طور موقت بررسی کنیم، کد رسم نمودار در بخش command window زده شد.

```
>> n=n1_1:T_1:n2_1;
>> plot(n,x1)
```

در ادامه خواسته شده تا تبدیل فوریه سیگنال محاسبه شود.

```
figure
DTFT(x2, n1_2, n2_2, T_1);
```

تابع DTFT در پایان کد نوشته‌شده چون سینتکس متلب طوری است که تابع‌ها را یا باید در اسکریپت جداگانه یا آخر کد نوشت.

```
function DTFT(x, n1,n2,T)
    syms w
    n=n1:T:n2;
    j = sqrt(-1);
    X = x.*exp(-j.*n*w);
    X_sum = sum(X);
```

```
    X_m = matlabFunction(X_sum);
    i=1;
    for w_sample=-pi:0.001:pi
        X_data(i) = X_m(w_sample);
        i=i+1;
    end
    w_sample=-pi:0.001:pi;
    subplot(1,2,1)
    plot(w_sample,abs(X_data));
    xlabel('\omega');
    ylabel('|H(e^{j\omega})|')
    hold on
    subplot(1,2,2)
    plot(w_sample,angle(X_data));
    xlabel('\omega');
    ylabel('arg(H(e^{j\omega}))')
end
```

در این تابع، ورودی‌ها سیگنال مورد نظر، ابتدا و انتهای بازه سیگنال و نرخ نمونه‌برداری می‌باشد.

ابتدا امگا یا w به‌صورت سمبولیک تعریف کردیم. لازم بود که به‌صورت syms تعریف شود چون ابتدا می‌خواهیم تابع تبدیل فوریه را به‌دست می‌آوردیم. به نرخ نمونه‌برداری نیز لازم داشتیم چون برای محاسبات آرایه‌ای مقادیر سیگنال در حوزه زمان، باید ابعاد آرایه n و x باهم برابر باشد. اما اگر w را نیز آرایه تعریف می‌کردیم در ابتدا، به مشکل برمی‌خوردیم و خطایی دریافت می‌کردیم که ابعاد آرایه‌ها باهم برابر نیستند. بازه خواسته‌شده برای امگا با بازه قابل پردازش برابر نبود.

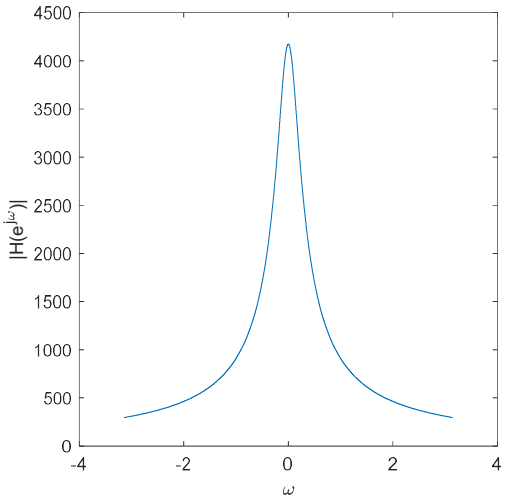
با تشکیل آرایه X، به‌ازای nهای مختلف نمونه X را به‌دست می‌آوریم و در انتها با استفاده از دستور sum، سیگمای موجود در فرمول تبدیل فوریه را اعمال می‌کنیم. در حال حاضر هنوز X سمبولیک است یعنی مقدار ندارد و به شکل تابع ریاضی است.

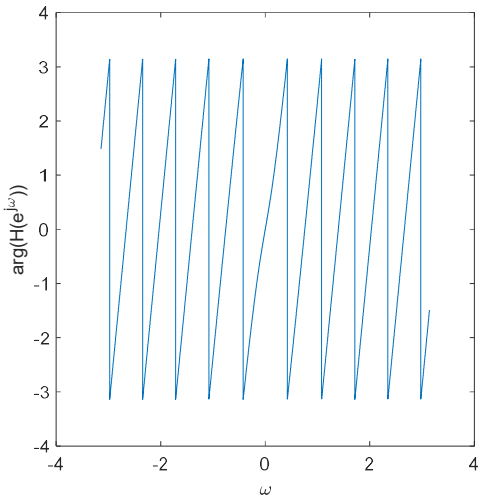
در خط بعدی با دستور matlabFunction، X را به تابع handle تبدیل می‌کنیم تا بتوانیم با جایگذاری مقادیر مختلف در w، مقدار عددی به‌دست بیاوریم.

```
    for w_sample=-pi:0.001:pi
        X_data(i) = X_m(w_sample);
        i=i+1;
    end
```

در این حلقه که در کد بالا نیز آمده بود، مقادیر عددی تبدیل فوریه را به‌دست می‌آوریم.

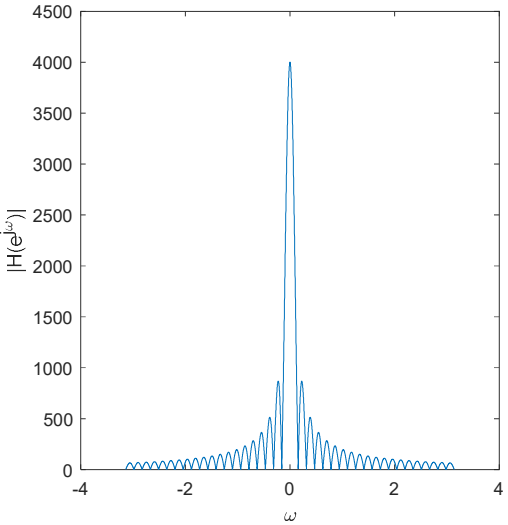
در خطوط بعدی به رسم شکل می‌پردازیم. از abs برای به‌دست آوردن اندازه و از angle برای به‌دست آوردن فاز استفاده کرده‌ایم. از دستور hold on استفاده شد چون در سوال دوم خواسته‌شده بود دوتا شکل را روی یک نمودار رسم کنیم. دستور subplot برای رسم دو نمودار در یک پنجره است و labelهای هر بردار x و y هم نام‌گذاری شده‌اند.

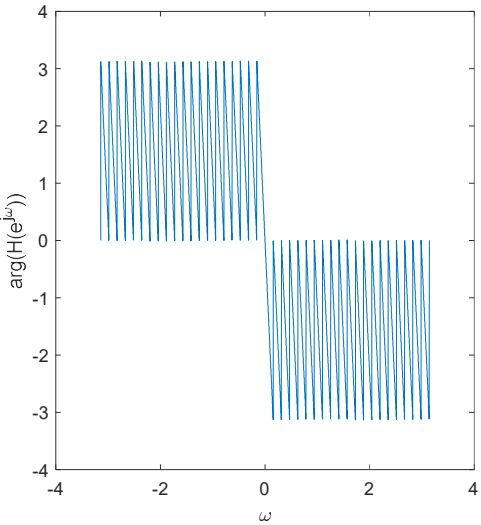




همین کارها را برای سیگنال بعدی انجام می‌دهیم.

```
%x2
n1_2 = 0;
n2_2 = 40;
T_1 = 0.01;
n=n1_2:T_1:n2_2;
x2 = 1;
figure
DTFT(x2, n1_2, n2_2, T_1);
```





سوال دوم

```
%Q2
%Compressor and sampling
```

```
clear n;
f = 10;
T = 1/f;
n1_3 = -4;
n2_3 = 4;
```

فرکانس نمونه‌برداری در این سوال کم‌تر است پس پیوستگی شکل یا سیگنال کم‌تر خواهد بود.

```
x_d = @(n) (sinc(n)).^2;
y1 = Compressor(x_d,2, n1_3, n2_3, T);
```

تابع به‌صورت handle تعریف‌شده تا محاسبه مقادیر سیگنال فشرده‌شده راحت‌تر باشد.

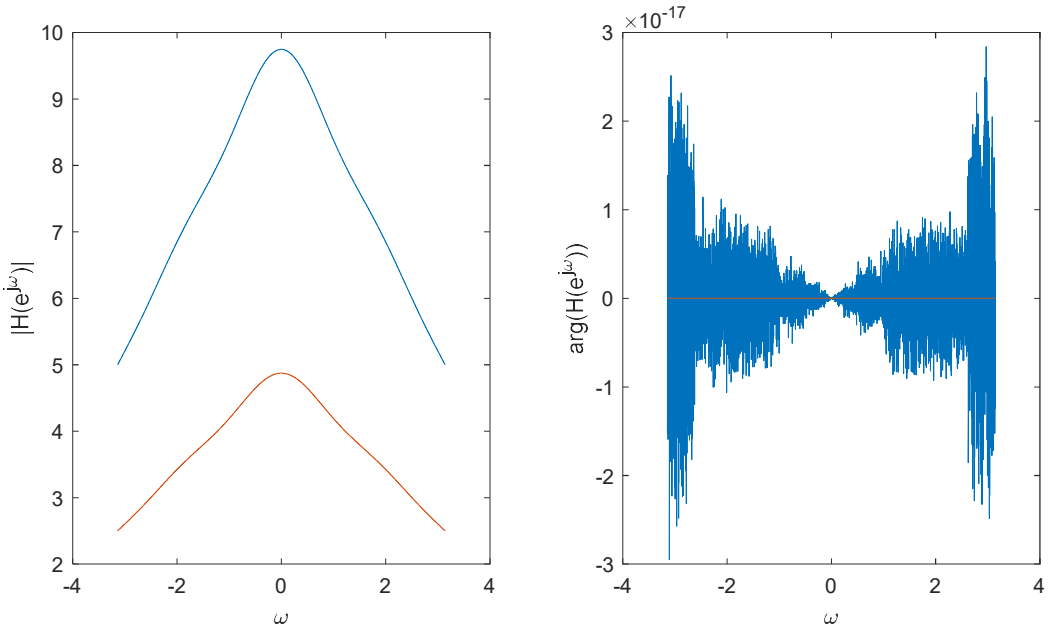
تابع فشرده‌ساز در آخر کد آمده که این‌جا می‌نویسیم:

```
function y = Compressor(x, M, n1, n2, T)
    n = n1:M*T:n2;
    y = x(n);
end
```

فشرده‌سازی درواقع تغییری در نرخ نمونه‌برداری است و کافی‌ست M مورد نظر را در T ضرب کنیم.

حال تبدیل فوریه را در دو حالت قبل و بعد از فشرده‌سازی محاسبه می‌کنیم.

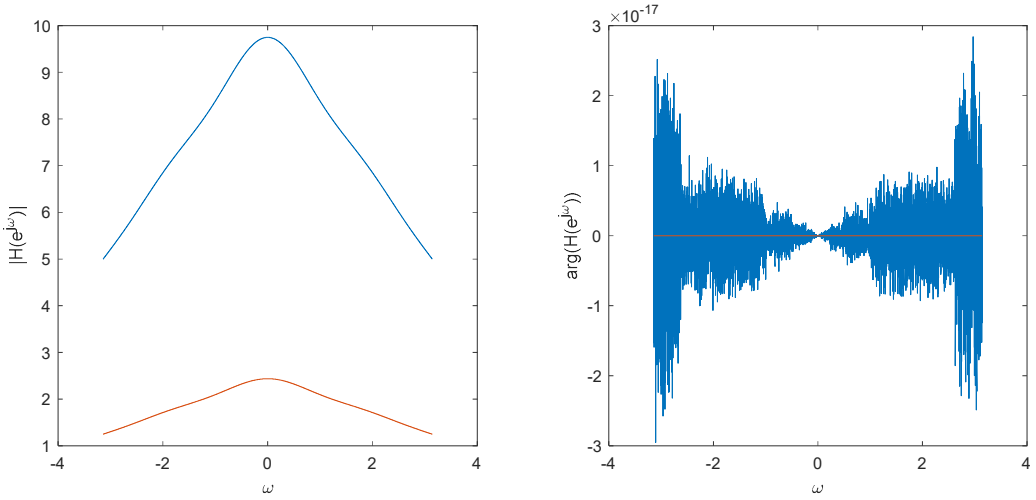
```
n = n1_3:T:n2_3;
x_d_f = x_d(n);
figure
DTFT(x_d_f,n1_3,n2_3,T);
hold on
DTFT(y1,n1_3,n2_3,T*2);
```



نمودار نارنجی سیگنال فشرده‌شده است. می‌بینیم که علاوه‌بر تاثیر آن در امگا، اندازه آن 1/M برابر (نصف) شده است و تغییرات فاز ثابت شده است.

همین کار را برای M=4 تکرار می‌کنیم.

```
y2 = Compressor(x_d,4,n1_3,n2_3,T);
figure
DTFT(x_d_f,n1_3,n2_3,T);
hold on
DTFT(y2,n1_3,n2_3,T*4);
```



بخش دوم

```
clc; clear; close all
%CA1, Anaies Golboudaghians 40122113 DSP
%Part 2
[y, Fs] = audioread("HesapirateDSP.mp3");
disp(Fs);
```

ابتدا فایل صوتی را می‌خوانیم. خروجی آن اطلاعات صوتی و فرکانس نمونه‌برداری می‌باشد. خروجی:

44100

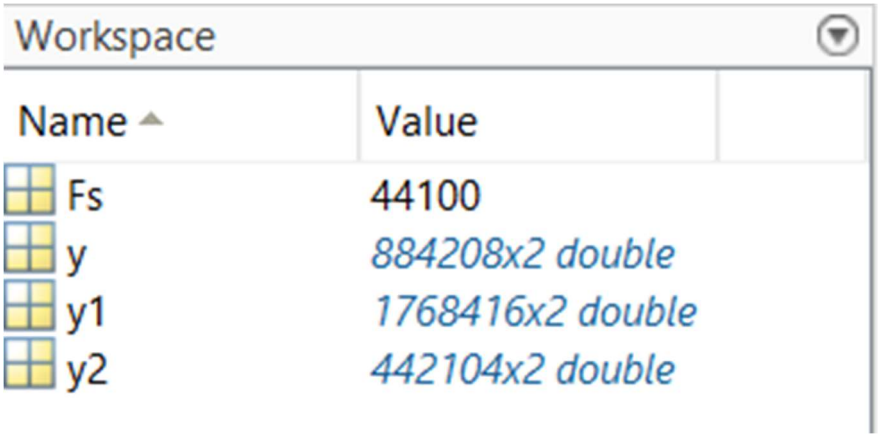
```
sound(y, Fs);
pause(22);
```

ابتدا با دستور sound فایل اولیه را گوش می‌دهیم. برای آن‌که پخش شدن آهنگ‌ها درهم نشود و همزمان صورت نگیرد، از دستور pause استفاده می‌کنیم.

```
y1 = resample(y,Fs*2,Fs);
sound(y1,Fs*2);
audiowrite('Part2_output1.wav',y1,Fs*2);
```

در گام بعدی با دستور resample فرکانس نمونه‌برداری را دوبرابر و ذخیره می‌کنیم. حتماً باید این دستور را استفاده کنیم. اگر فقط به تغییر دادن فرکانس در دستور sound بسنده کنیم، تنها فرکانس پخش تغییر می‌کند و در نمونه‌برداری تغییری به‌وجود نیامده است.

با دوبرابر شدن فرکانس نمونه‌برداری، تعداد داده نمونه‌برداری شده دوبرابر می‌شود و به اصطلاح در این حالت حجم اشغالی آهنگ بالاتر می‌رود. هرچه فرکانس نمونه‌برداری بیشتر باشد، کیفیت موسیقی نیز بیشتر است.



برای نصف شدن فرکانس همین کارها را تکرار می‌کنیم.

```
pause(22);
y2 = resample(y,Fs*.5,Fs);
sound(y2, Fs*.5);
audiowrite('Part2_output2.wav',y2,Fs*.5);
```

نتیجه می‌گیریم که با کاهش فرکانس نمونه‌برداری، کیفیت آهنگ کاهش پیدا می‌کند.

بخش سوم

سوال اول

با توابع گفته‌شده سیستم را تعریف می‌کنیم.

```
clc; clear; close all
%CA1, Anaies Golboudaghians 40122113 DSP
%Part 3

%Q1
num = [1 2 3 4];
den = [4 3 2 1];
w = -4*pi:8*pi/511:4*pi;

h = freqz(num, den, w);
```

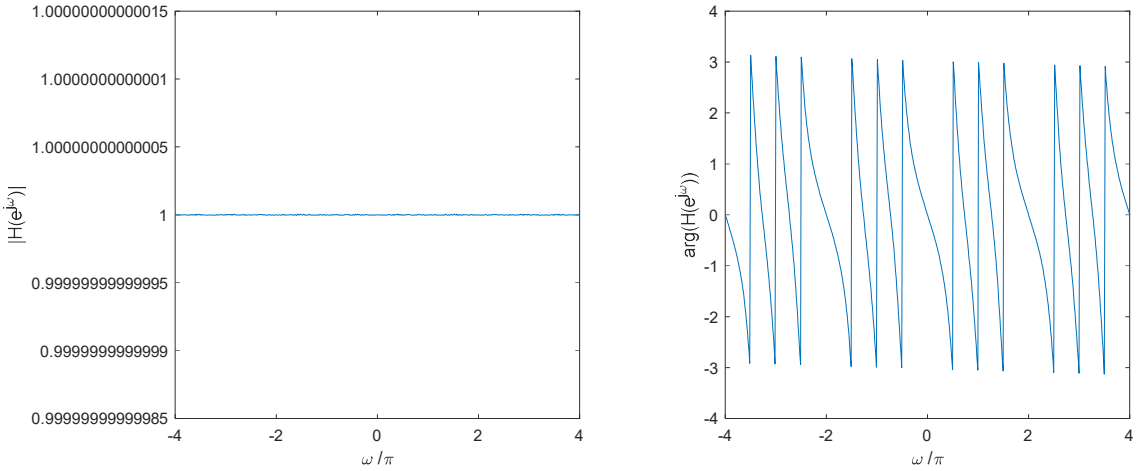
ضرایب چندجمله‌ای صورت و مخرج را در بردارهای num و den باید ذخیره کنیم.

مانند بخش اول نمودار اندازه و فاز را رسم می‌کنیم.

```
figure
```

```
subplot(1,2,1)
plot(w/pi,abs(h));
xlabel('\omega /\pi');
ylabel('|H(e^{j\omega})|')
subplot(1,2,2)
plot(w/pi,angle(h));
xlabel('\omega /\pi');
ylabel('arg(H(e^{j\omega}))');
```

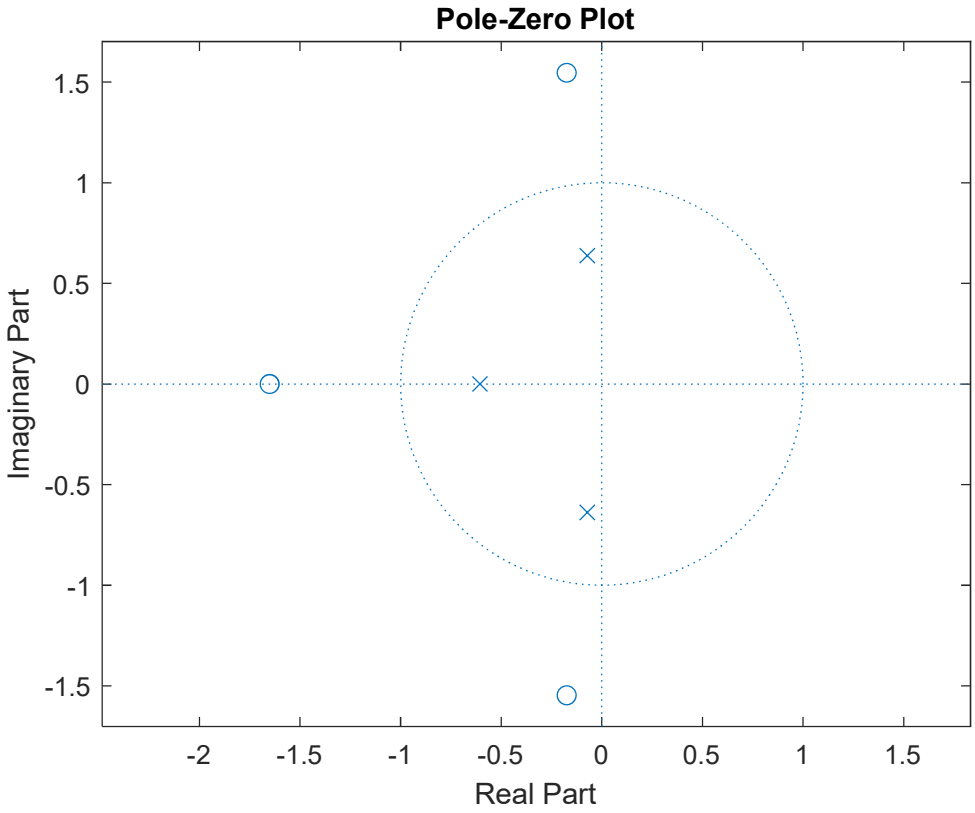
خروجی:



اندازه فیلتر تقریباً یک است.

```
figure
z = roots(num);
p = roots(den);
zplane(z,p);
```

در خطوط بالا، ریشه چندجمله‌ای‌های صورت و مخرج، یعنی صفر و قطب‌ها را پیدا و رسم می‌کنیم.



سوال دوم

```
%Q2
[x, Fs] = audioread("HesapirateDSP.mp3");
sound(x, Fs)
pause(22);
y = filter(num, den,x,[],2);
sound(y,Fs)
audiowrite('output1.wav',y,Fs);
pause(22);
```

با دستور filter، سیگنال صوتی را از فیلتر عبور می‌دهیم. تفاوت چندانی شنیده نمی‌شود.

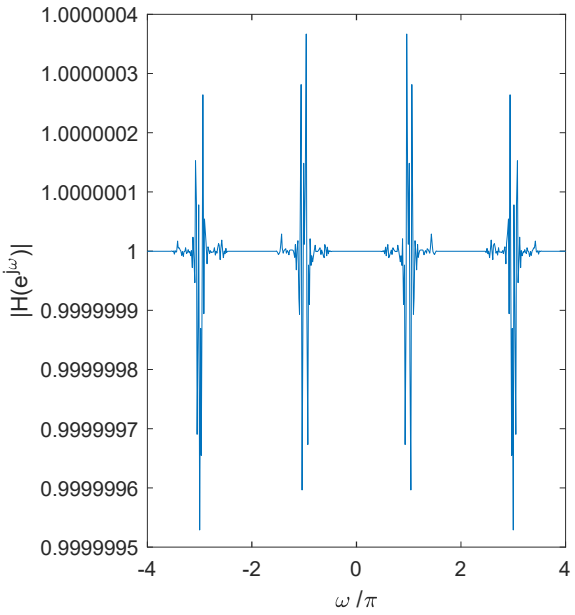
گوش ما به عنوان یک حسگر شنوایی، در درجه اول به تغییرات در دامنه (بلندی صدا) و فرکانس (زیر و بمی صدا) حساس است. این بدان معناست که گوش ما می‌تواند تشخیص دهد که یک صدا چقدر بلند است و چه میزان زیر و بم دارد. اما به طور معمول، گوش ما نمی‌تواند تفاوت بین دو موج صوتی با دامنه و فرکانس یکسان را تشخیص دهد که تنها در فاز با هم متفاوت هستند.

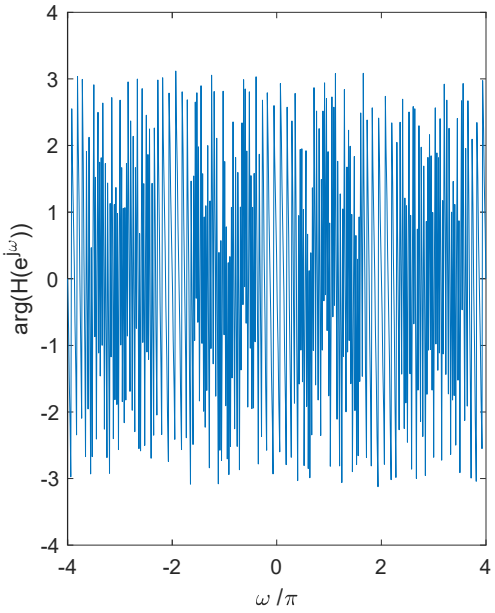
سوال سوم

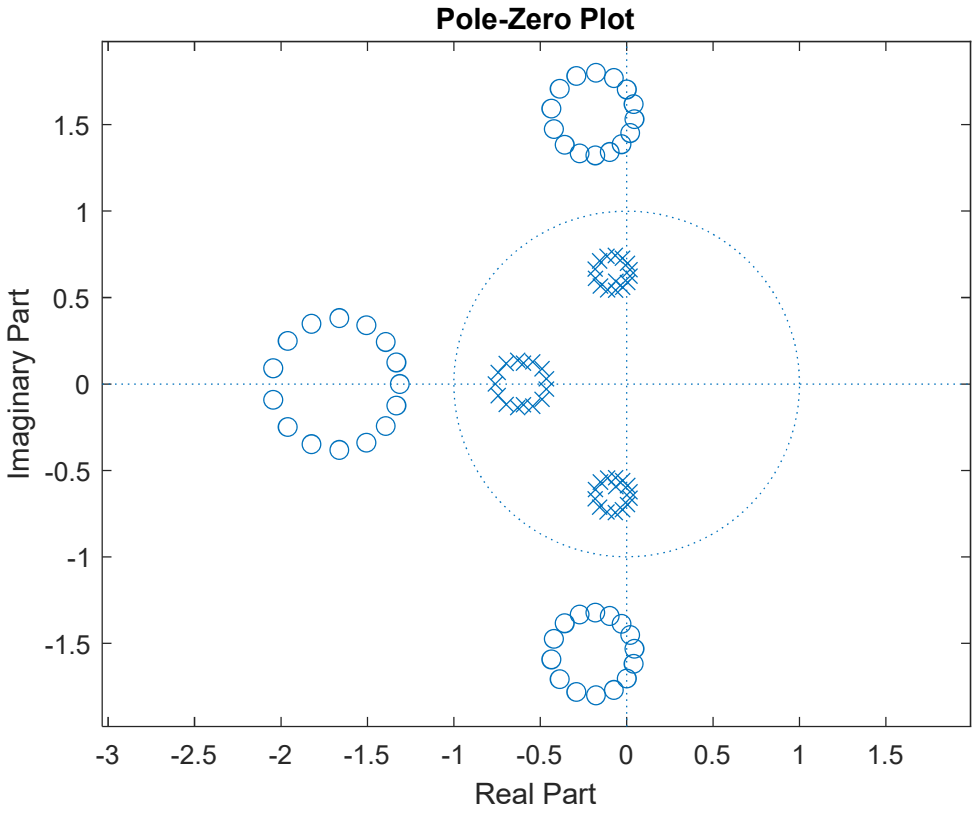
برای به توان رساندن، چندین بار از دستور conv استفاده کرده‌ایم.

```
%Q3
num2 = conv(num,num);
num4 = conv(num2, num2);
num8 = conv(num4, num4);
num12 = conv(num4,num8);
num14 = conv(num12,num2);
num15 = conv(num,num14);

den2 = conv(den,den);
den4 = conv(den2, den2);
den8 = conv(den4, den4);
den12 = conv(den4,den8);
den14 = conv(den12,den2);
den15 = conv(den,den14);

h2 = freqz(num15,den15, w);
figure
subplot(1,2,1)
plot(w/pi,abs(h2));
xlabel('\omega /\pi');
ylabel('|H(e^{j\omega})|')
subplot(1,2,2)
plot(w/pi,angle(h2));
xlabel('\omega /\pi');
ylabel('arg(H(e^{j\omega}))');
figure
z = roots(num15);
p = roots(den15);
zplane(z,p);
```

خروجی:







نمودار صفر و قطب یک فیلتر تمام‌گذر را نشان می‌دهد. زیرا هرجا که قطب باشد، زوج conjugate آن نیز صفر است. پس صحیح می‌باشد.

سوال چهارم

```
%Q4
y2 = filter(num15, den15,x,[],2);
sound(y2,Fs)
pause(22);
audiowrite('output2.wav',y2,Fs);
```

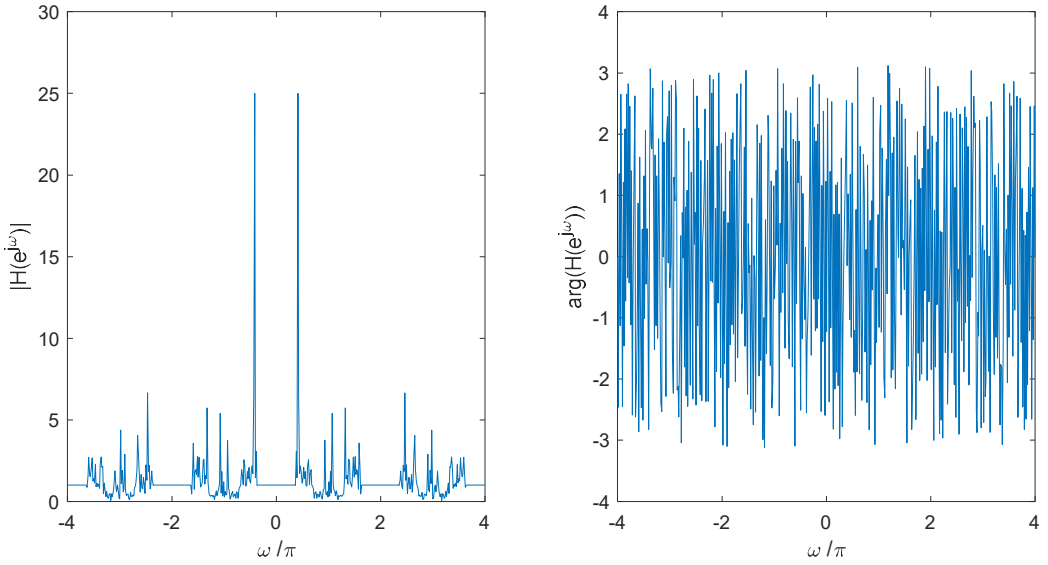
تاکنون تفاوت چندانی احساس نمی‌شود.

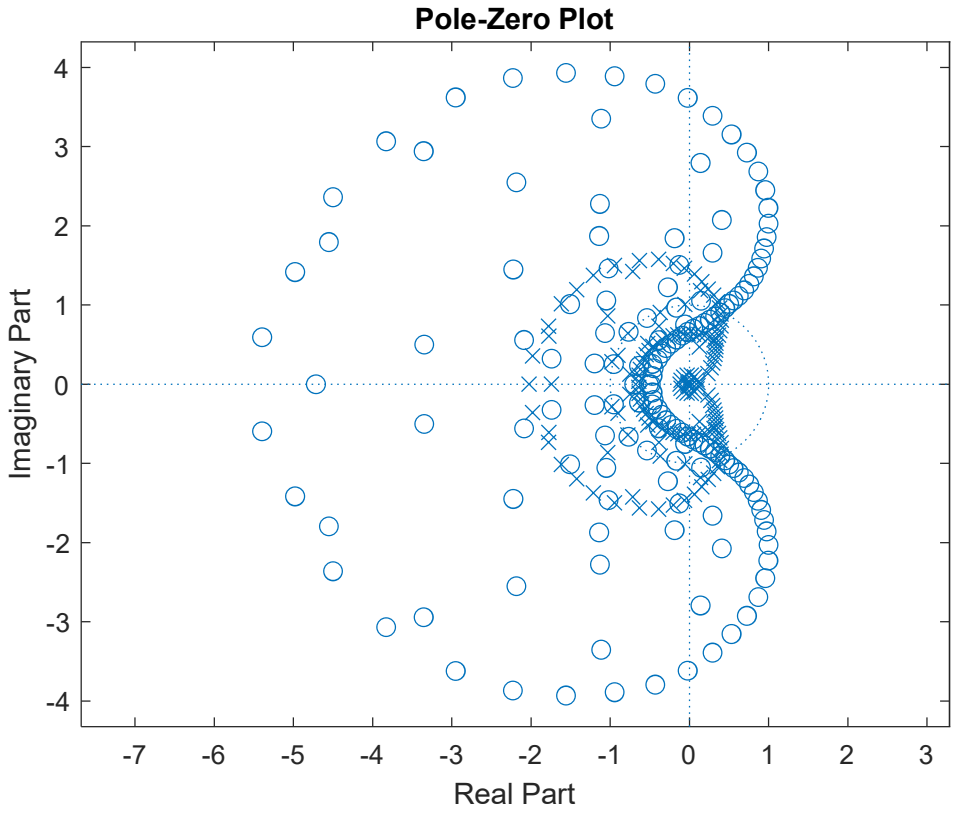
سوال پنجم

```
%Q5
num30 = conv(num15,num15);
num45 = conv(num15,num30);
num49 = conv(num45,num4);
num50 = conv(num49,num);

den30 = conv(den15,den15);
den45 = conv(den15,den30);
den49 = conv(den45,den4);
den50 = conv(den49,den);
```

```
h3 = freqz(num50,den50, w);
figure
subplot(1,2,1)
plot(w/pi,abs(h3));
xlabel('\omega /\pi');
ylabel('|H(e^{j\omega})|')
subplot(1,2,2)
plot(w/pi,angle(h3));
xlabel('\omega /\pi');
ylabel('arg(H(e^{j\omega}))');
figure
z = roots(num50);
p = roots(den50);
zplane(z,p);

y3 = filter(num50, den50,x,[],2);
sound(y3,Fs)
audiowrite('output3.wav',y,Fs);
```

خروجی:





باز هم تفاوت شنیدار خاصی محسوس نبود.

بخش چهارم

```
clc; clear; close all
%CA1, Anaies Golboudaghians 40122113 DSP
```

```
%Part 4
b = [0.45 0.4 -1];
a = [1 -0.4 -0.45];
y = [0 3];
x = [2 2];
```

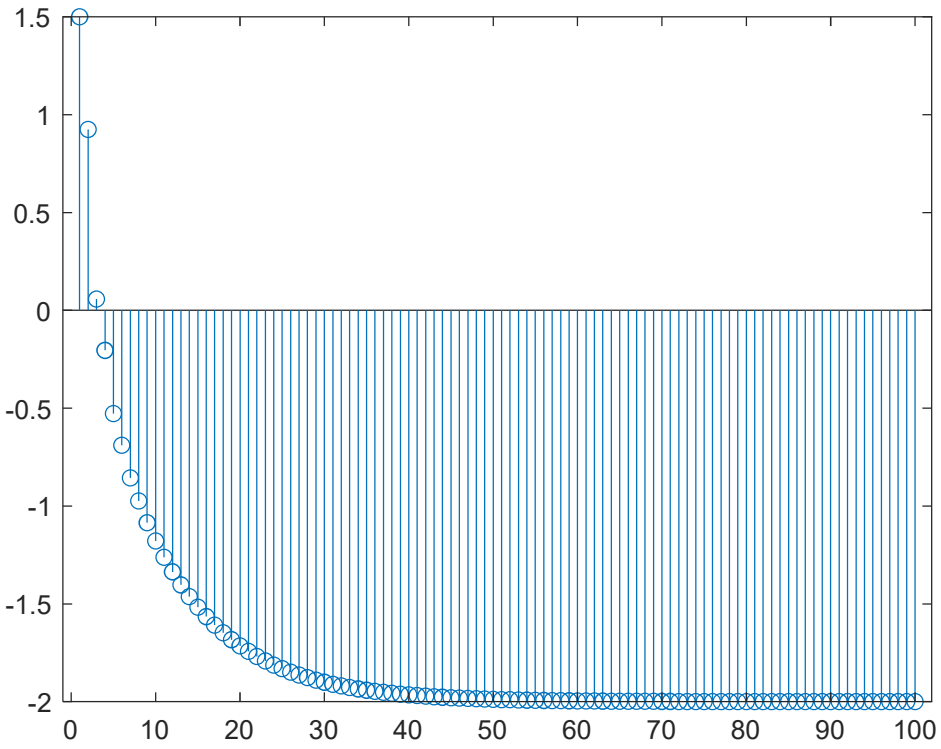
داده‌های سوال را در چهار آرایه بالا ذخیره می‌کنیم.

```
zi = filtic(b, a, y, x);
```

با استفاده از تابع بالا وضعیت اولیه سیستم یا فیلتر را ایجاد می‌کنیم.

```
n = 0:99;
x_input = 2 + ((0.5).^n);
y1 = filter(b, a, x_input, zi);
figure
stem(y1);
```

ورودی را به‌صورت آرایه‌ای تعریف می‌کنیم و از فیلتر عبور می‌دهیم و با دستور stem، پاسخ را به‌صورت گسسته رسم می‌کنیم.

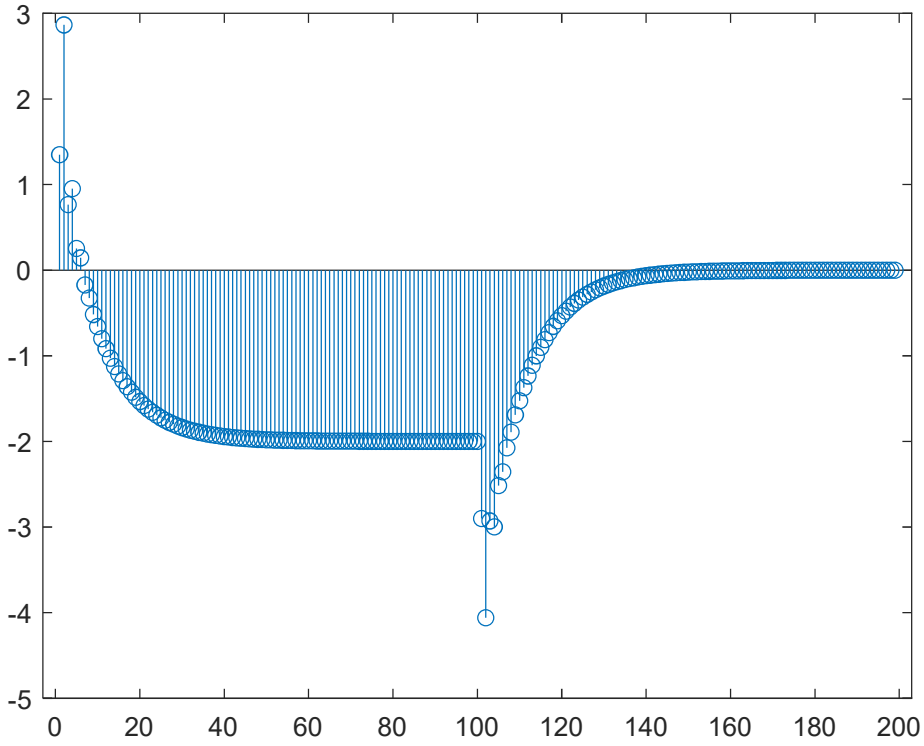


در ادامه با استفاده از residuez و conv پاسخ ورودی به سیستم را محاسبه می‌کنیم. دستور residuez تابع تبدیل را به کسرهای جزئی تجزیه می‌کند.

```
[r,p,k] = residuez(b,a);
syms z
```

```
H = r(1)/(1-(p(1)*(z^(-1)))) + r(2)/(1-(p(2)*(z^(-1)))) + k;
h = iztrans(H);
h_mf = matlabFunction(h);
h_n = h_mf(n);
y2 = conv(x_input,h_n);
figure
stem(y2);
```

خروجی:



علت تفاوت این دو شکل احتمالاً به‌خاطر تشخیص متفاوت ROC فیلتر است. اما در ۱۰۰ نمونه اول شباهت دارند.