سیستم های مخابراتی
پروژه درس
استاد : دکتر محمد هادی

محمدحسین شفیعی زادگان ۹۹۱۰۴۷۸۱
امیرحسین یاری ۹۹۱۰۲۵۰۷
ایلیا هاشمی راد ۹۹۱۰۲۴۵۶

۱۶ بهمن ۱۴۰۱

فهرست مطالب

۱ معرفی کد ها و بلوک دیاگرام سیستم ۲
۲ Question b ۸
۳ Question c ۱۱
۴ Question d ۱۳
۵ Question e ۱۵
۶ Question g ۱۸

۱ معرفی کد ها و بلوک دیاگرام سیستم

ابتدا کد های نوشته شده برای هریک از بلوک دیاگرام های سیستم مخابراتی موردنظر را شرح می دهیم و سپس آن ها را برای یک نمونه، آزمایش می کنیم.
کد های این بخش را در فایل مجزایی به نام $test.m$ قرار داده ایم.

ADC

بلوک دیاگرام مبدل آنالوگ به دیجیتال استفاده شده شامل یک بخش نمونه گیر (sampler) ، کوانتایزر یونیفرم (midrise uniform quantizer) و در نهایت یک واحد encoder می باشد.
کد متلب نوشته شده برای این بلوک به صورت زیر است :

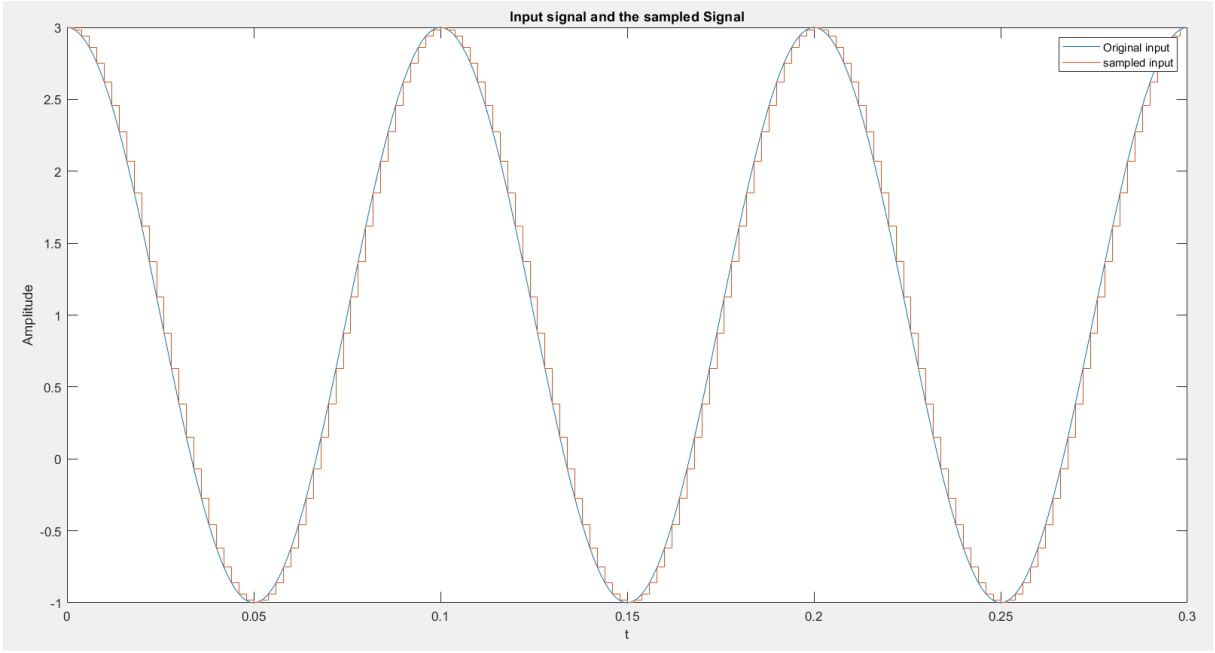
```
function [Q_encoded, delta] = ADC(x, nu, fs, F, plot)
   x = x(1:end)';
   sam_x = x(1:fs/F:end);
   [Q, delta] = uniform_quan(nu, sam_x);
   Q_encoded = encoder(Q, delta, min(x));
   if(plot)
      ADC_plot(x,F,fs,sam_x,Q)
   end
end

function [Q , delta] = uniform_quan(v, x)
   N = 2^v;
   delta = ceil((max(x) - min(x)))/N;
   Q = zeros(1,length(x));
   for n=0:N-1
      Q(x>=min(x)+n*delta & x<=min(x)+(n+1)*delta) = min(x)+n*delta + delta/2;
   end
end

function x_encoded = encoder(x_quan, delta, min_x)
   level = (x_quan-delta/2-min_x) / delta;
   x_encoded = de2bi(level, 'left-msb');
end
```

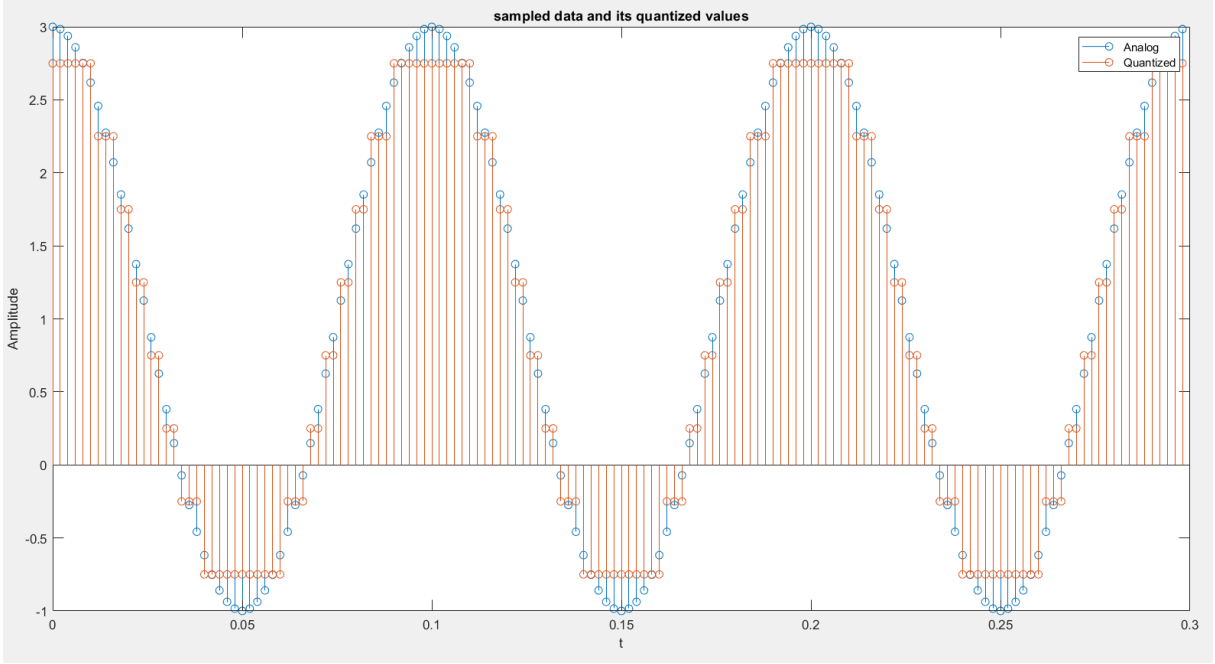
تابعی به نام ADC_plot نوشته ایم که در صورت نیاز با وارد کردن مقدار plot در آرگومان ورودی تابع ADC برابر با ۱ ، نمودار سیگنال های مربوط را به صورت مناسب رسم کند.
در تابع uniform_quan هدف آن است که با گرفتن سیگنال زمان گسسته و مقدار پیوسته، سیگنالی زمان گسسته و مقدار گسسته خروجی دهیم. کوانتیزاسیون نشان دهنده این است که مقادیر نمونه برداری شده از سیگنال عضو یک مجموعه محدود از مقادیر هستند. نحوه ی عملکرد این تابع بدین گونه است که با در دست داشتن تعداد بیت ها، تعداد سطح ها را از رابطه ی $N = 2^{\nu}$ بدست می آوریم. سپس به دلیل اینکه می دانیم کوانتایزر ما یکنواخت است، فاصله ی هر دو سطح از حاصل تقسیم طول سیگنال بر تعداد سطوح محاسبه خواهد شد. حال بصورت زیر به هر بازه یک سطح نسبت می دهیم.

$$[x_{min} + n\Delta\,,\, x_{min} + (n+1)\Delta] \xrightarrow{mapped\ to} x_{min} + k\Delta + \frac{\Delta}{2}$$

با استفاده از تابع encoder ، با یافتن سطح های متناظر با مقادیر نسبت داده شده به ورودی ها که اعدادی در بازه صفر تا $N-1$ می باشند که در آن N برابر با تعداد سطوح کوانتیزاسیون است، با کمک تابع$de2bi$ اعداد باینری متناظر را به دست می آوریم.
حال کد های این بخش را به ازای ورودی زیر آزمایش می کنیم و خروجی ها را نمایش می دهیم.

```
F = 1000;
t = 0:10e-6:0.003-10e-6;
x = 2*cos(2*pi*F.*t)+1;

nu = 3;
x = x';

fs = 2*F;
[Q_encoded, delta] = ADC(x, nu, fs, F, 1);
```



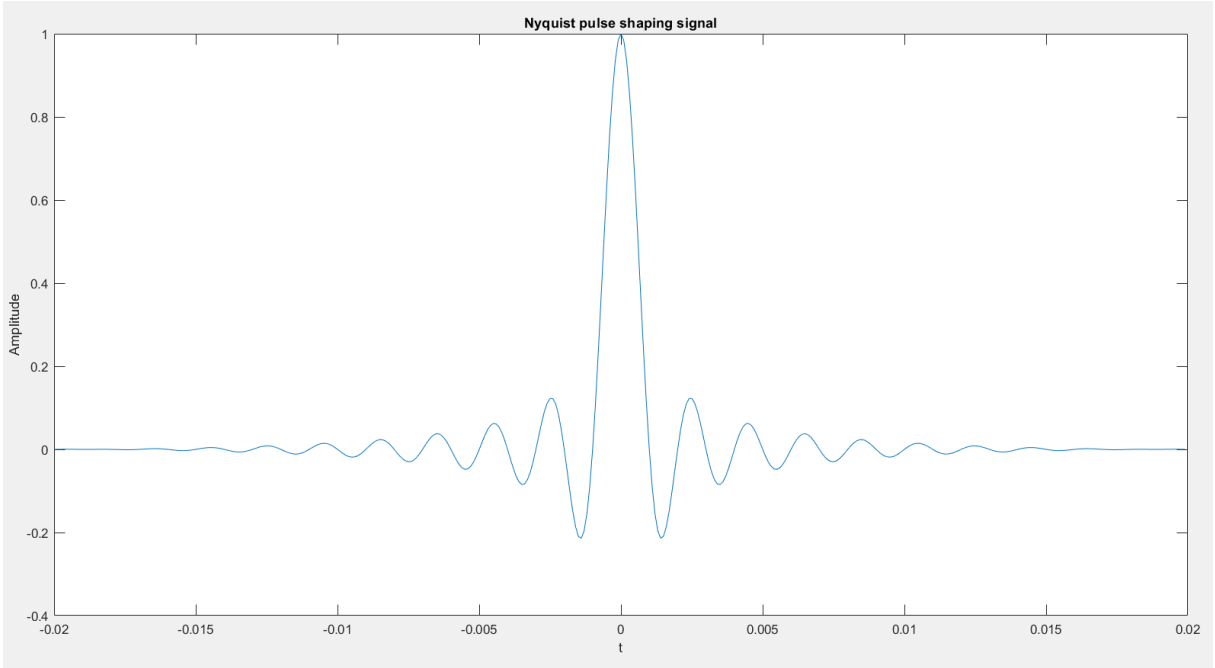


LineCoder

کد متلب استفاده شده در این بخش به صورت زیر است :

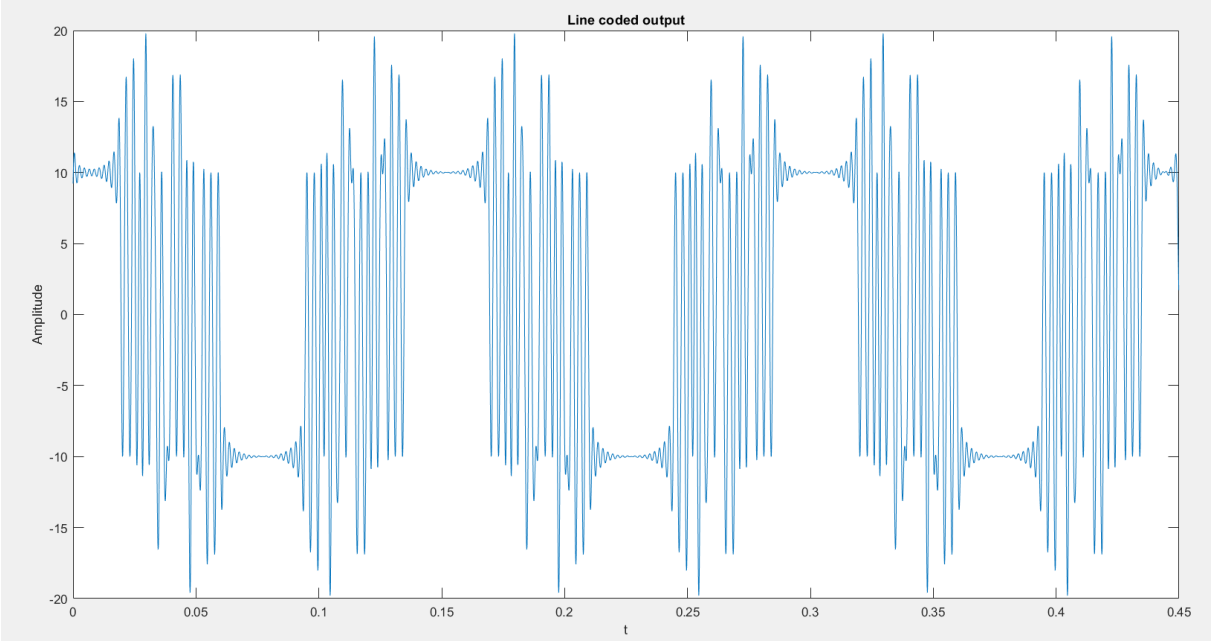
```
function [result, t2] = lineCoder(bit_stream, beta, baudRate, plot)
   A = 10;
   [Rows,Cols] = size(bit_stream);
   res = 10; %resolution
   t2 = 0:1/baudRate/res:(Rows*Cols-1/res)/baudRate;
   result = zeros(1,length(t2)+400);
   t3 = -200*1/baudRate/res:1/baudRate/res:199*1/baudRate/res;
   p = cos(2*pi*beta.*t3).*sinc(baudRate*t3) ./ (1-(4*beta.*t3).^2) ;
   for i=0:Rows*Cols-1
      if(bit_stream(i+1) == 1)
         result(1+i*res:400+i*res) = result(1+i*res:400+i*res) + A*p;
      end
      if(bit_stream(i+1) == 0)
         result(1+i*res:400+i*res) = result(1+i*res:400+i*res) - A*p;
      end
   end
   result = result(200:end-201);

   if(plot)
      lineCoder_plot(t2, result, baudRate, beta);
   end
end
```

با توجه به مقادیر β و baudRate ابتدا شکل موج سیگنال نایکوئیست $(p(t))$ را تشکیل می دهیم و سپس با توجه به مقدار رشته بیت ورودی، در صورت صفر بودن بیت موردنظر سیگنال حاصل را از $Ap(t-kD)$ کم کرده و در صورت یک بودن بیت موردنظر، سیگنال حاصل را با $Ap(t-kD)$ جمع می کنیم. مقدار D برابر با عکس مقدار baudrate می باشد.



به ازای همان سیگنال ورودی سینوسی، کد های این بخش را نیز آزمایش کرده و خروجی ها را مشاهده می کنیم.

```
beta = 40;
baudRate = 1000;
[x_lineCoded, t2] = lineCoder(Q_encoded', beta, baudRate, 1);
```



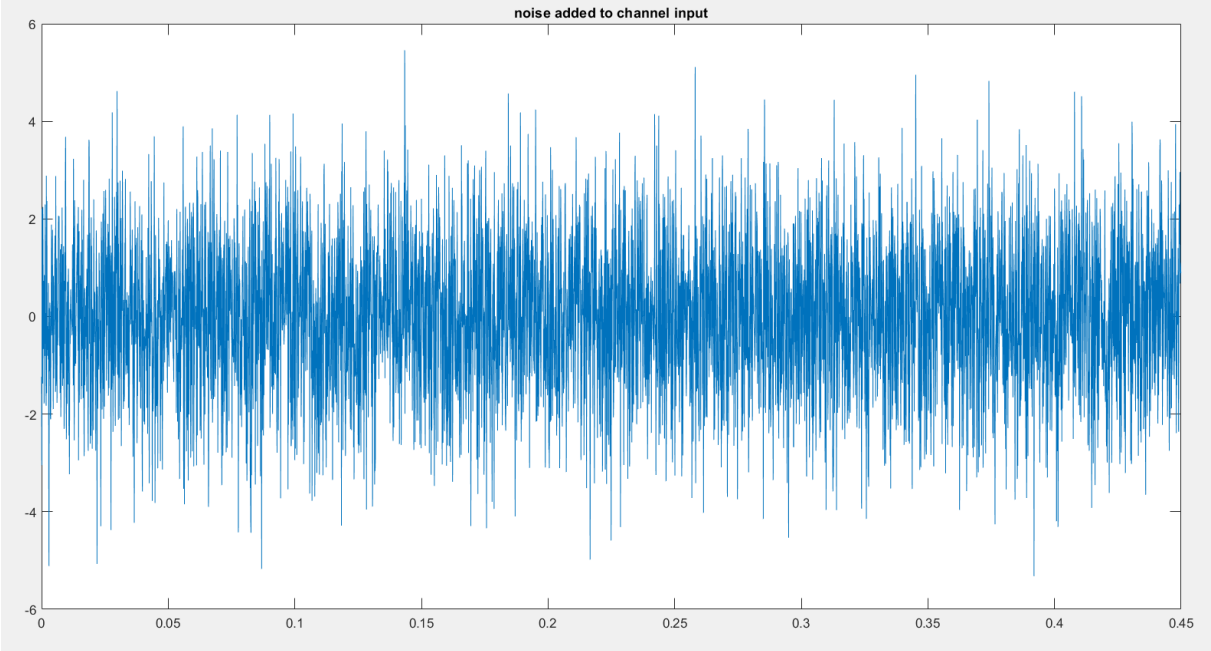
Channel

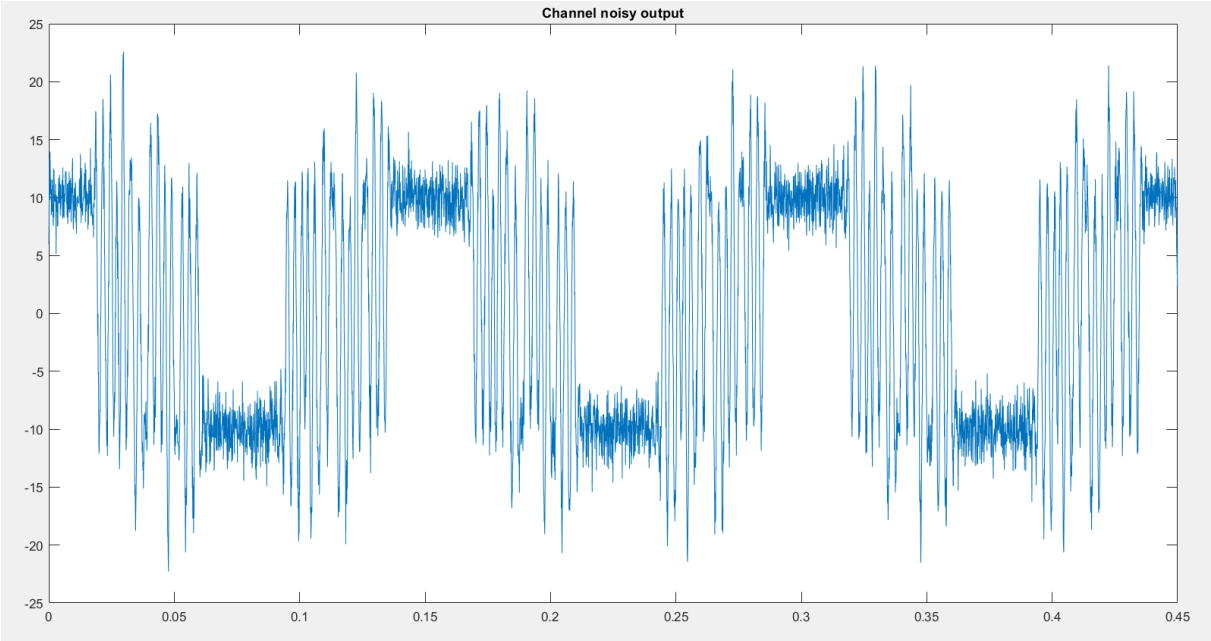
ابتدا سیگنال ورودی به کانال را از یک فیلتر پایین گذر با فرکانس قطع کانال عبور داده و سپس با استفاده از تابع randn متلب به صورت زیر نویز گوسی موردنظر را به سیگنال ورودی اضافه می کنیم.

```
function out = channel(in, B, sigma2, Fs, t2, plot)
   filtered_x = lowpass(in,B,Fs);
   noise = sqrt(sigma2) * randn(1,length(in));
   out = filtered_x + noise;
   if(plot)
      channel_plot(t2,out,noise)
   end
end
```

حال برای همان ورودی تست، خروجی ها زا نمایش می دهیم.

```
% Channel
B = baudRate/2;
N0 = 10e-4;
sigma2 = B*N0;
chnl_out = channel(x_lineCoded, B, sigma2, fs, t2, 1);
```





به خوبی اثر مخرب کانال بر سیگنال را مشاهده می کنیم.

LineDeecoder

با استفاده از بلوک lineDecoder باید با استفاده از تایمینگ سیگنال، داده های دیجیتال و symbol ها را از سیگنال به دست آمده از کانال استخراج کرد.

$$y(t) = \sum_{k=-\infty}^{\infty} a_k \tilde{p}(t - t_d - kD) + n(t) \qquad , \qquad \tilde{p} = p(t) * h_c(t)$$

$$y(t_K) = y(KD + t_d + ts)$$

کد متلب نوشته برای شبیه سازی این بلوک به صورت زیر است :

```
function bit = lineDecoder(input, t2, baudRate)
  d = round((1/baudRate / (t2(2) - t2(1))));
  index = 1 : d : length(t2);
  td = 3;
  ts = 1;
  index = index + td + ts;
  syms = input(index);
  bit = syms>0;
end
```

حال کد این بخش را به ازای ورودی تست، آزمایش کرده و درصد مطابقت داده های استخراج شده با داده های انکود شده را نمایش می دهیم.

```
% Decoder
detected_bits = lineDecoder(chnl_out,t2,baudRate);
data = reshape(detected_bits,[nu length(detected_bits)/nu])';
disp('Matching percentage of extracted data and encoded data :');
disp(sum(data==Q_encoded,'all')/length(Q_encoded)/nu*100);
```

```
Matching percentage of extracted data and encoded data :
   97.5556

fx >>
```

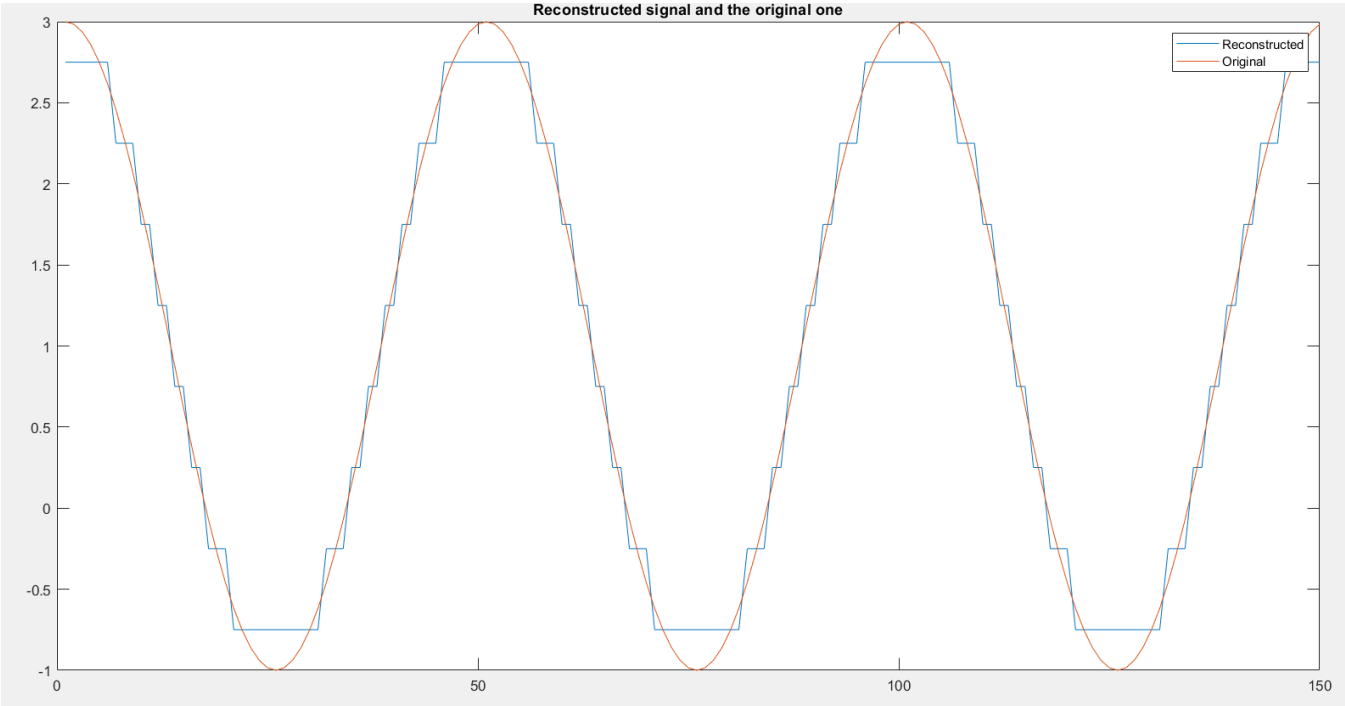
DAC

در این قسمت از سیستم، کافی است مقادیر symbol ها و داده های باینری استخراج شده را به مقادیر آنالوگ تبدیل کنیم.
برای بهبود نتیجه نهایی می توان از روش upsampling نیز استفاده کرد و بعد از آنکه توسط بلوک DAC سیگنال نمونه برداری شده را بازسازی کردیم، به کمک upsampling که در واقع معکوس فرایند نمونه برداری است، سیگنال اولیه را نیز به دست آورد.

```
function Upsampled_data = DAC(data,delta,min_x)
  A_data = bi2de(data, 'left-msb')*delta + delta/2 + min_x;
  A_data = A_data';
  diff = -[0 A_data] + [A_data 0];
  diff = diff(2:end-1);
  diff = upsample([diff 0], 4);
  A_data = upsample(A_data , 4);
  A_data1 = [A_data 0 0 0];
  A_data2 = [0, A_data + diff/4, 0, 0];
  A_data3 = [0 0, A_data + diff/2, 0];
  A_data4 = [0 0 0, A_data + 3*diff/4];
  res = A_data1 + A_data2 + A_data3 + A_data4;
  Upsampled_data = res(1:end-6);
  Upsampled_data = Upsampled_data';
end
```

```
% DAC
A_data = DAC(data,delta,min(x));
figure;
plot(A_data); hold on;
plot(x(1:fs/F:end));
title('Reconstructed signal and the original one');
```

```
        legend('Reconstructed','Original');
```



۲ Question b

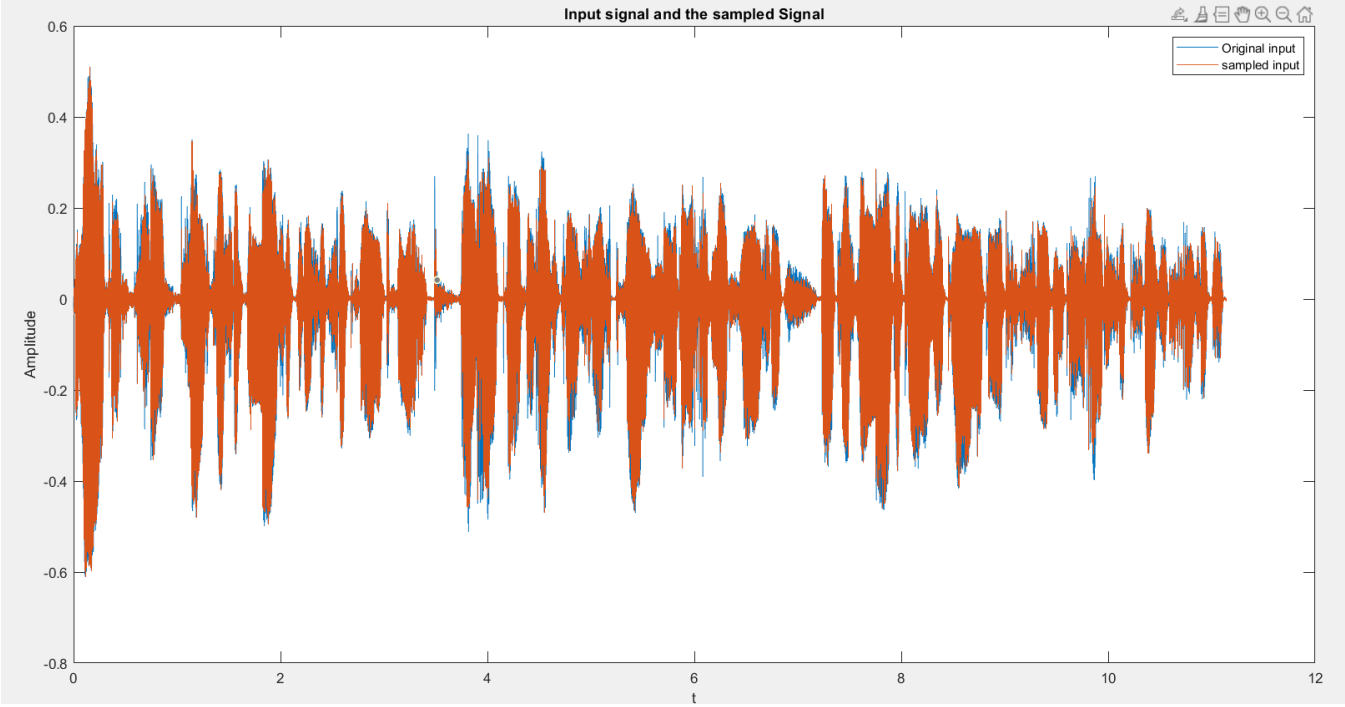
با استفاده از قطعه کد زیر از یک فایل صوتی به عنوان سیگنال ورودی استفاده کرده و نمودار های خروجی را نمایش می دهیم. همچنین می توان طبق قطعه کد زیر با استفاده از میکروفون صدا را ضبط کرده و خروجی ها را به دست می آوریم و صوت حاصل را پخش می کنیم.

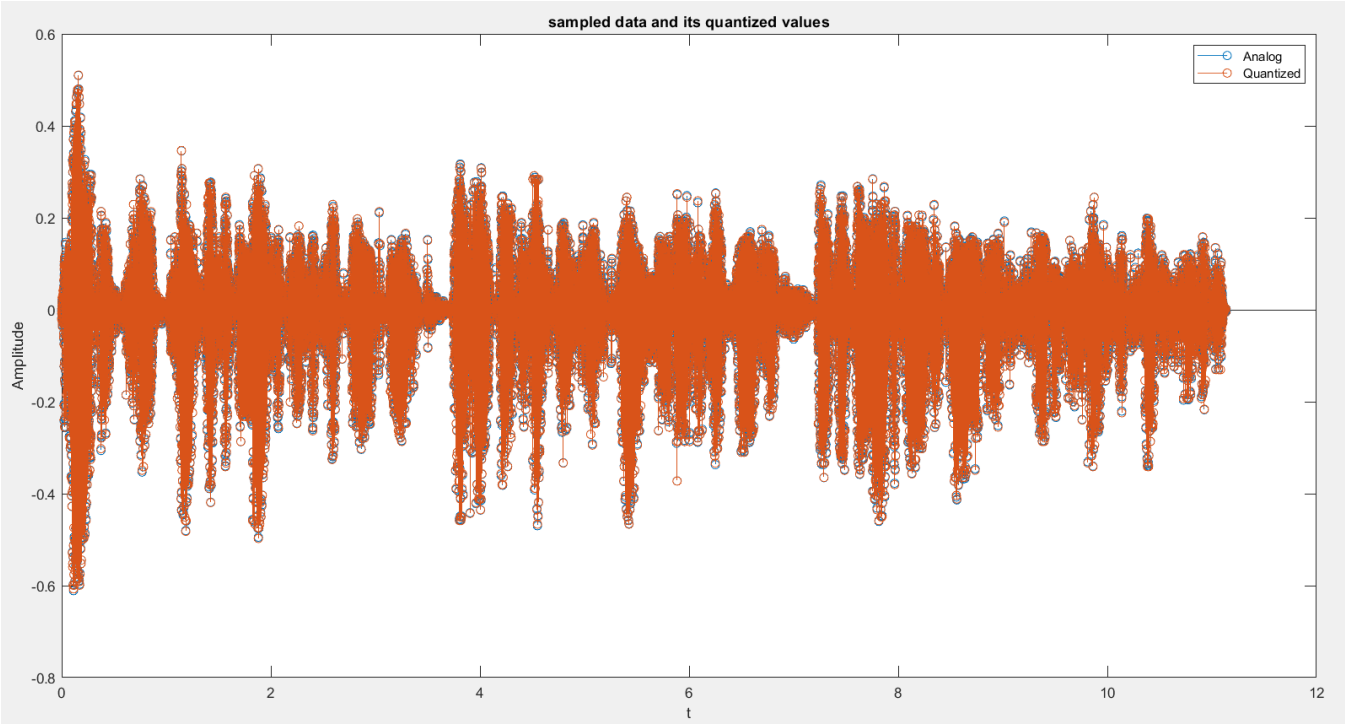
```
        ar = audiorecorder;

        duration = 10;
        record(ar,duration);
        pause(duration);
        x = getaudiodata(ar);
        F = ar.SampleRate;

        [x, F] = audioread('sound1.wav');
```

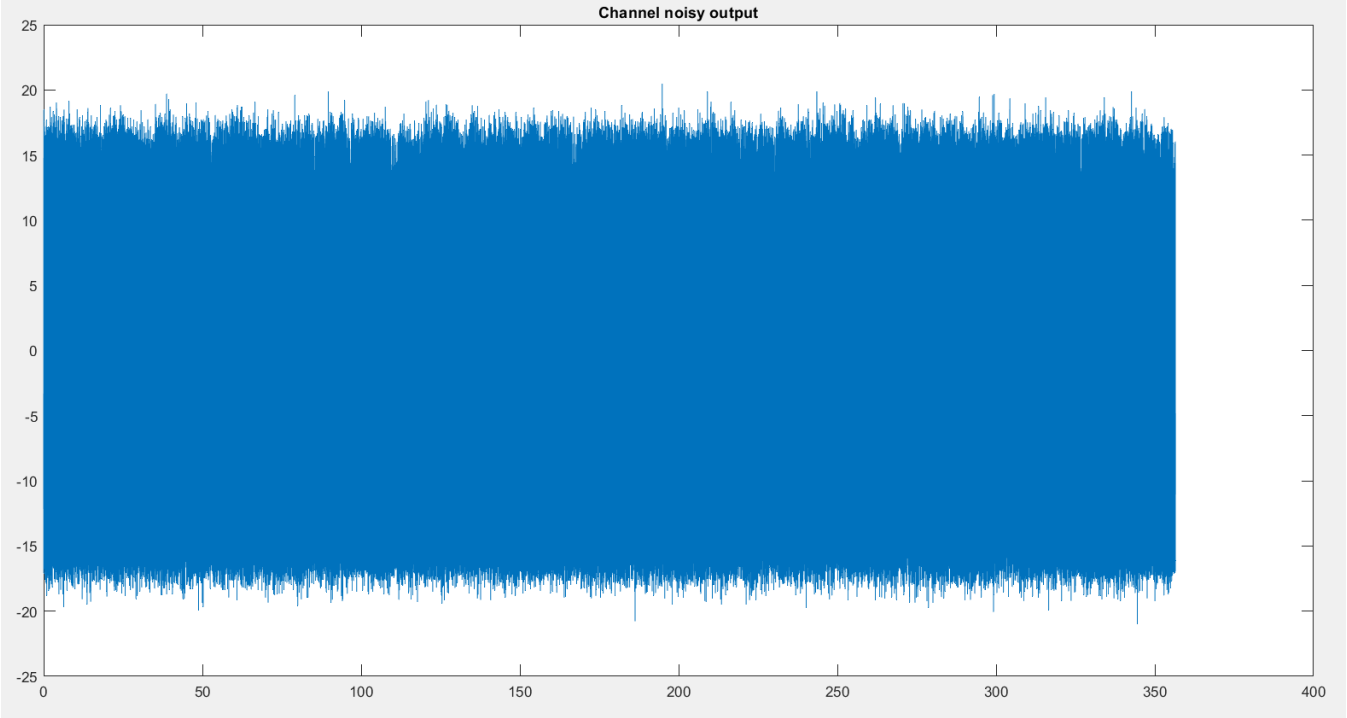
در آخر نیز پس از حاصل شدن سیگنال خروجی با استفاده از قطعه کد زیر صوت نهایی را با فرکانس مناسب پخش می کنیم و همچنین آن را در یک فایل صوتی ذخیره می کنیم.

```
         audiowrite('reconstructedSound.wav',A_data,F^2/fs);
        sound(A_data,F^2/fs);
```

نمودار های خروجی را برای یک نمونه صوت نمایش می دهیم.


Input signal and the sampled Signal
Original input
sampled input
Amplitude
t
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
2
4
6
8
10
12


sampled data and its quantized values
Analog
Quantized
Amplitude
t
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
2
4
6
8
10
12



با پخش فایل صوتی نهایی درمی یابیم که توسط بلوک های سمت گیرنده سیستم مخابراتی صوت ارسالی را با دقت مناسبی توانسته ایم بازسازی کنیم و صوت پخش شده تا حدی واضح می باشد اما در اثر عواملی مانند خطای کوانتیزاسیون، اضافه شدن نویز سفید گوسی بر روی سیگنال و خطای سنکرون سازی و پارامتر تاخیر decoder ، کیفیت سیگنال صوت تولیدی افت محسوسی می کند و با افزایش اثر این عوامل مخرب کیفیت سیگنال پایین تر می آید.

به منظور مشاهده اثر سطح نویز و پهنای باند کانال بر روی سیگنال نهایی از پارامتر SNR استفاده کرده و این مقدار را برای مقادیر مختلف سطح نویز و پهنای باند کانال به دست می آوریم.
ابتدا با ثابت در نظر گرفتن سطح نویز SNR را برای سیگنال خروجی به ازای مقادیر مختلف پهنای باند به دست می آوریم. همچنین در نظر داریم که برای جلوگیری از وقوع ISI باید رابطه $r \leq 2B$ برقرار باشد.

```
[x, F] = audioread('sound2.wav');
B = [500 600 700 800 900];
N0 = 10e-4;
o_SNR = zeros(1,5);
for i=1:5
   out = process_data(x,F,B(i),N0);
   o_SNR(i) = snr(out);
end
disp('   B        SNR');
disp([B; o_SNR]');
```

```
      B          SNR(dB)
   500.0000   -15.9462
   600.0000   -15.9462
   700.0000   -15.9468
   800.0000   -15.9915
   900.0000   -16.1855

fx >>
```

توجه داریم که نویز یک پدیده کاملا تصادفی است.
حال تاثیر چگالی نویز را مشاهده می کنیم :

```
[x, F] = audioread('sound2.wav');
N0 = [5*10e-5 10e-4 5*10e-4 10e-3 5*10e-3];
B = 500;
o_SNR = zeros(1,5);
for i=1:5
   out = process_data(x,F,B,N0(i));
   o_SNR(i) = snr(out);
end
disp('   N0        SNR(dB)');
disp([N0; o_SNR]');
```

```
    N0          SNR(dB)
    0.0005   -15.9462
    0.0010   -15.9462
    0.0050   -18.5403
    0.0100   -23.9623
    0.0500   -25.5922

fx >>
```

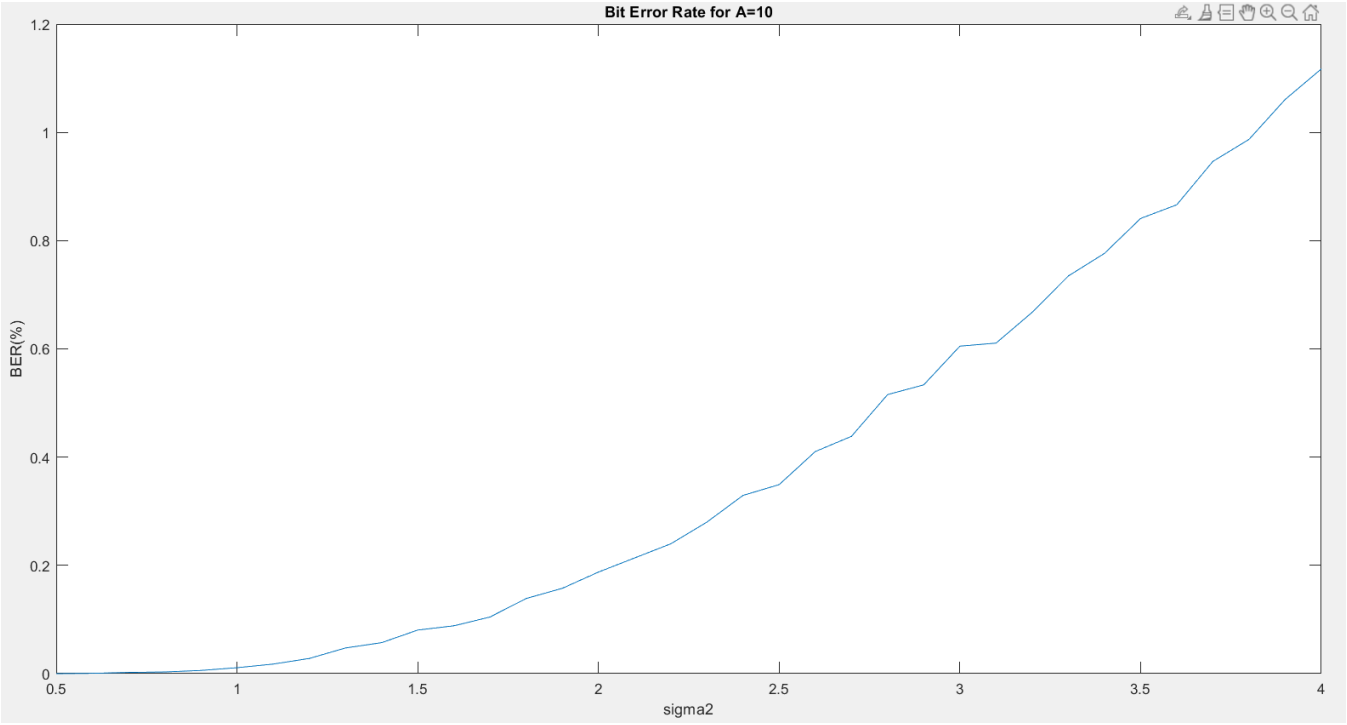
Question c ۳

برای محاسبه مقدار BER یا Bit Error Rate با مقایسه داده های انکود شده و ذخیره شده در ماتریس Q_encoded با داده های و بیت های استخراج شده و ذخیره شده در ماتریس data ، مقدار BER به دست می آید.

```
BER = 100 - (sum(data==Q_encoded,'all'))/(length(Q_encoded)*nu)*100;
```

حال مقادیر مختلف BER را به ازای مقادیر مختلف σ^2 به دست آورده و نمودار آن را رسم می کنیم.

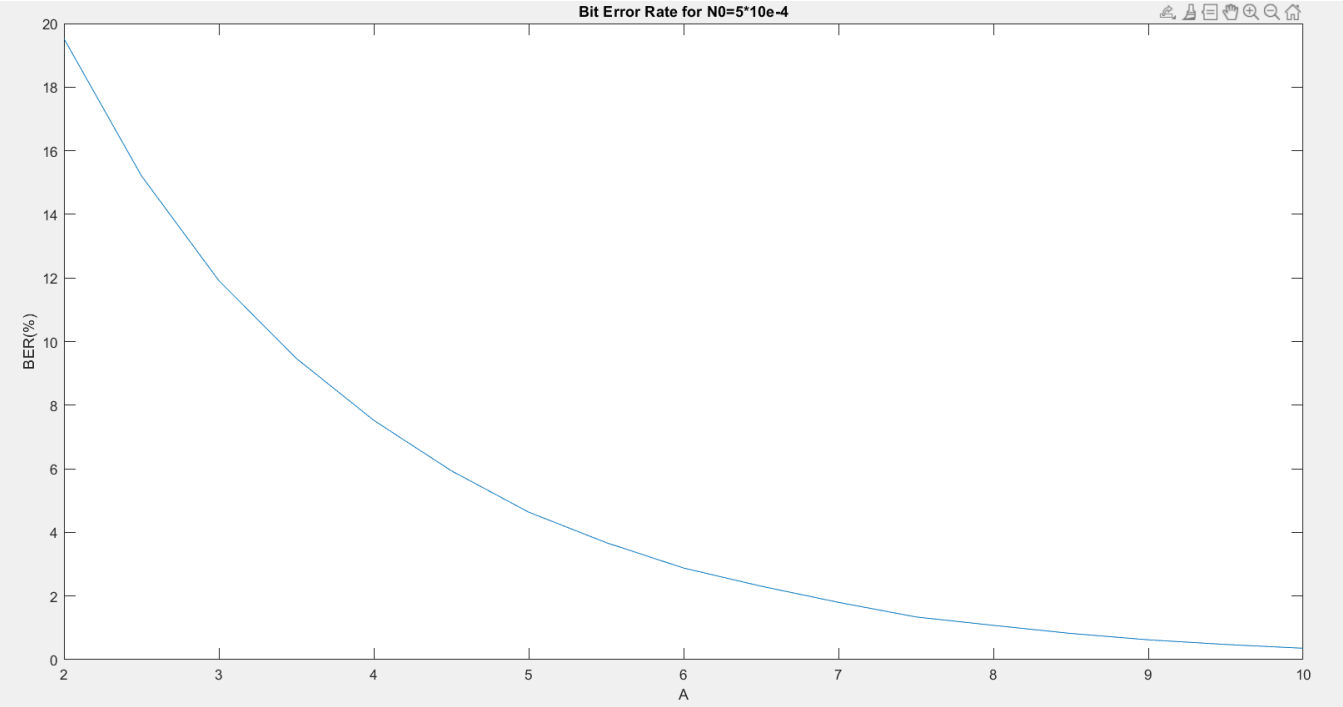
```
[x, F] = audioread('sound2.wav');
sigma2 = 0.5:0.1:4;
o_BER = zeros(1,length(sigma2));
for i=1:length(sigma2)
o_BER(i) = process_data(x,F,sigma2(i));
end
plot(sigma2,o_BER);
xlabel('sigma2');
ylabel('BER(%)');
title('Bit Error Rate for A=10');
```



به خوبی مشاهده می شود که با افزایش σ^2 و در نتیجه افزایش سطح نویز اضافه شده به سیگنال، احتمال تداخل بین سطوح نسبت داده شده به هر یک از صفر ها یا یک ها در linecode افزایش یافته و در نتیجه BER زیاد می شود.

حال اثر A را مشاهده می کنیم :

```
[x, F] = audioread('sound2.wav');
A = 2:0.5:10;
o_BER = zeros(1,length(A));
for i=1:length(A)
   o_BER(i) = process_data(x,F,A(i));
end
plot(A,o_BER);
xlabel('A');
ylabel('BER(%)');
title('Bit Error Rate for N0=5*10e-4');
```



مشاهده می شود که هرچه مقدار A کوچک تر باشد، احتمال تداخل بین سطوح نسبت داده شده به هر یک از صفر ها یا یک ها در linecode افزایش یافته و در نتیجه BER زیاد می شود.

۴ Question d

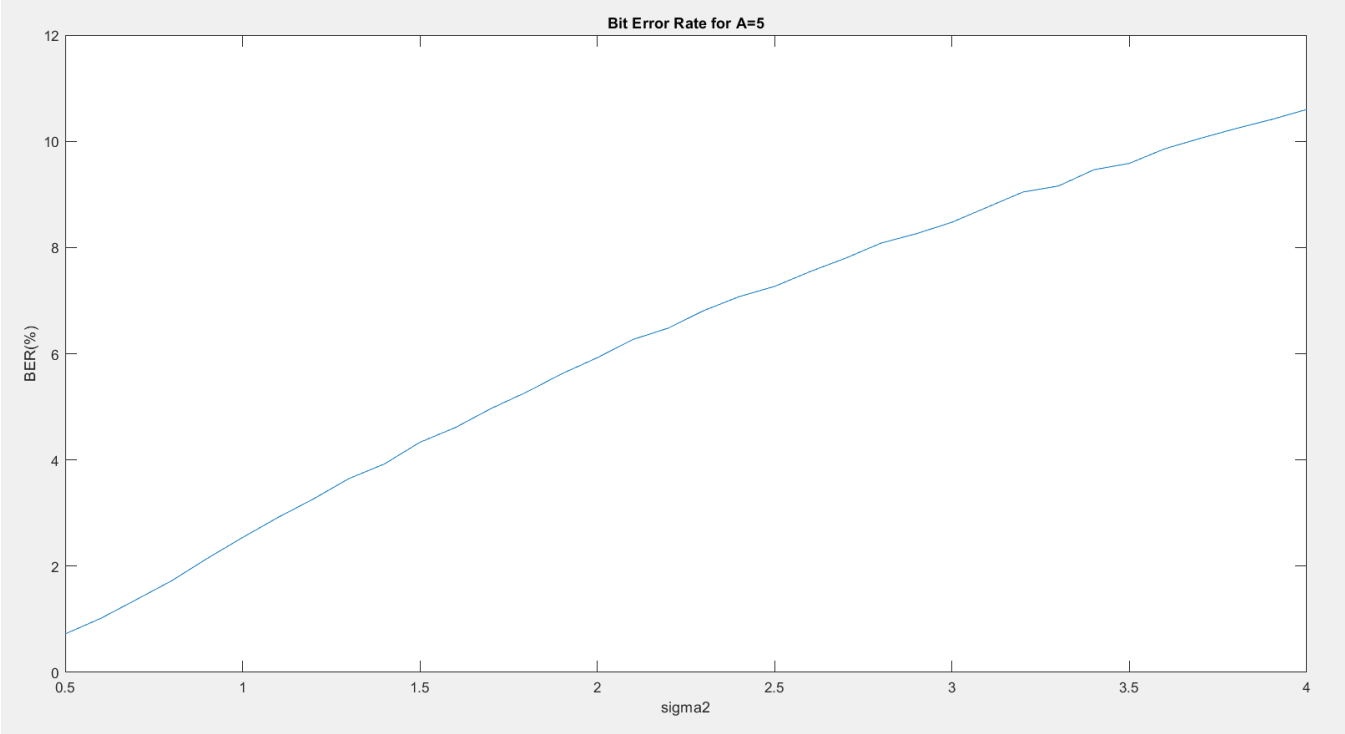
حال با فرض $r > 2B$، با ثابت در نظر گرفتن سطح نویز SNR را برای سیگنال خروجی به ازای مقادیر مختلف پهنای باند به دست می آوریم. توجه داریم که $baudRate = 1000$ و $N_0 = 10^{-3}$ می باشد.

```
Command Window
    B          SNR (dB)
  100.0000   -23.3706
  200.0000   -23.1818
  300.0000   -26.6933
  400.0000   -26.2418
  450.0000   -26.0135

fx >>
```

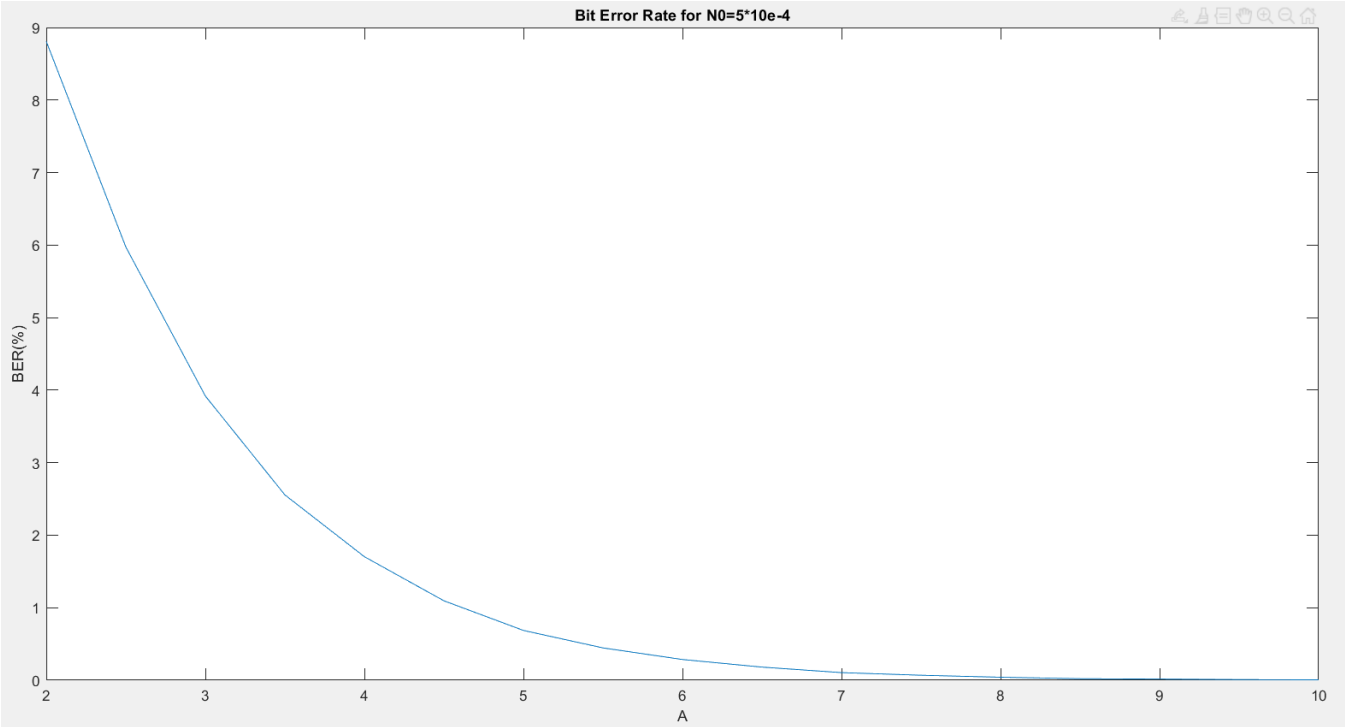
حال به ازای $B = 100$ مقدار snr را به ازای مقادیر مختلف N_0 به دست می آوریم.

```
Command Window
    N0          SNR (dB)
    0.0005   -15.9462
    0.0010   -15.9493
    0.0050   -19.9128
    0.0100   -22.7202
    0.0500   -24.7250

fx >>
```

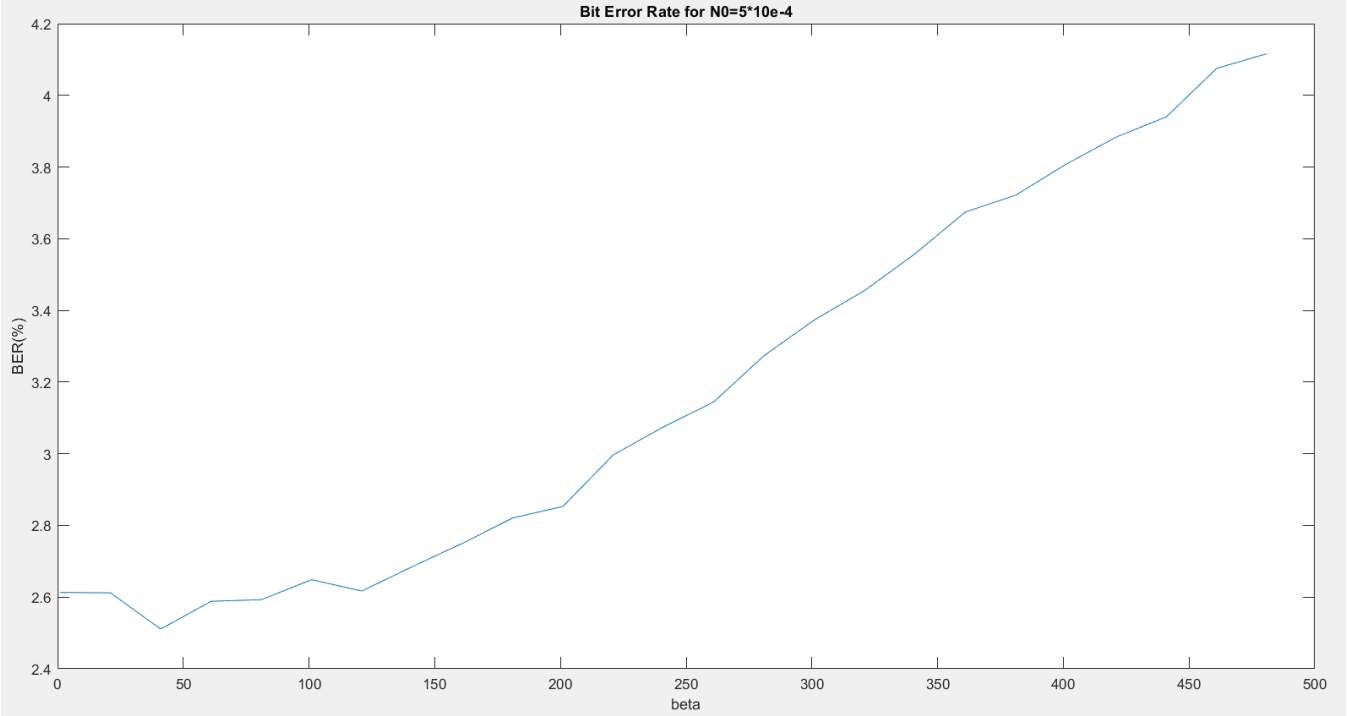
حال مقادیر مختلف BER را به ازای مقادیر مختلف σ^2 به دست آورده و نمودار آن را رسم می کنیم. توجه داریم که $B = 100$ می باشد.



حال اثر A را با ازای همان مقدار B مشاهده می کنیم :



حال برای مشاهده اثر β ، مقدار BER را به ازای مقادیر آن رسم می کنیم.



۵ Question e

برای مشاهده ی اثر فرکانس نمونه برداری بر روی صوت بازتولید شده، SNR و BER آن را به ازای فرکانس های مختلف محاسبه می کنیم.

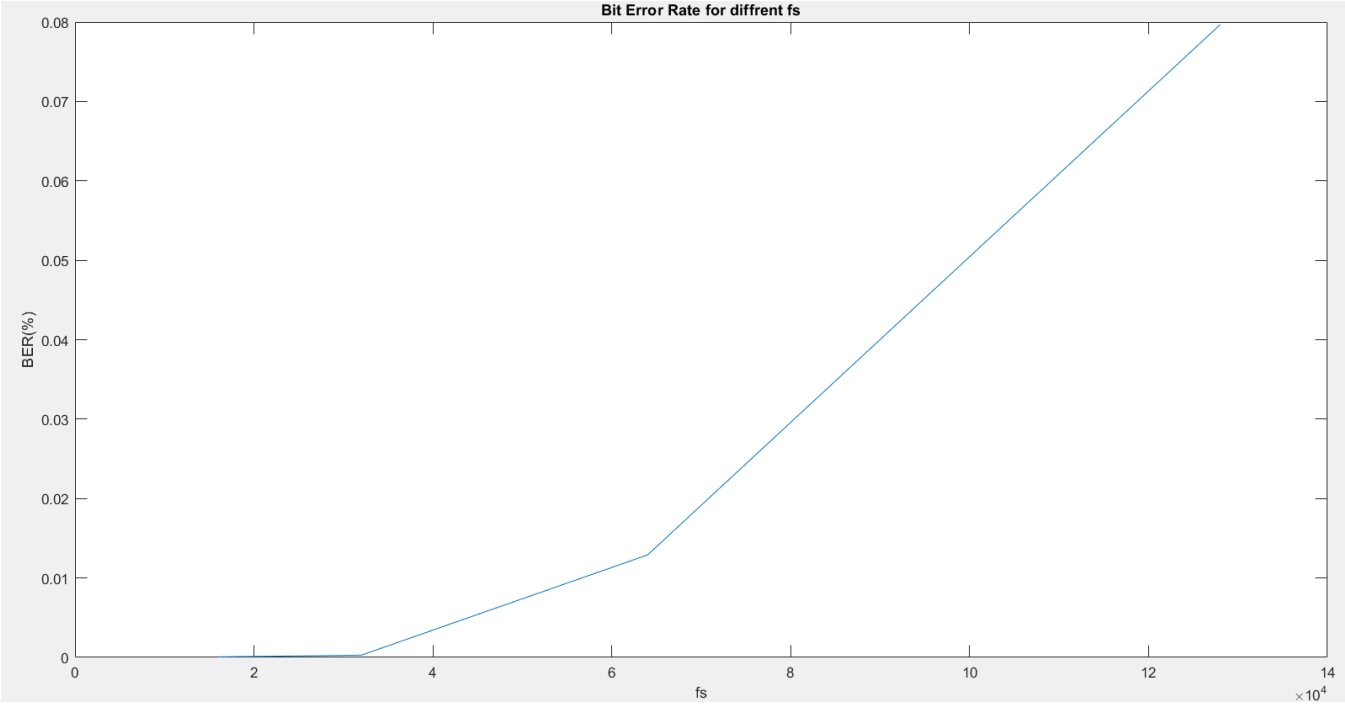
```
[x, F] = audioread('sound2.wav');
fs = [16000 32000 64000 128000];
o_SNR = zeros(1,4);
for i=1:4
   out = process_data(x,F,fs(i));
   o_SNR(i) = snr(out);
end
disp('   fs        SNR(dB)');
disp([fs; o_SNR]');
```

Command Window

```
      fs          SNR(dB)
   1.0e+05 *

    0.1600    -0.0002
    0.3200    -0.0002
    0.6400    -0.0002
    1.2800    -0.0002

fx >>
```

```
[x, F] = audioread('sound2.wav');
fs = [16000 32000 64000 128000];
```

```
o_BER = zeros(1,length(fs));
for i=1:length(fs)
o_BER(i) = process_data(x,F,fs(i));
end
plot(fs,o_BER);
xlabel('fs');
ylabel('BER(%)');
title('Bit Error Rate for diffrent fs');
```



برای مشاهده ی تاثیر تعداد بیت بر روی صوت بازتولید شده با استفاده از قطعه کدهای زیر BER و SNR را محاسبه می کنیم.

```
[x, F] = audioread('sound2.wav');
nu = [4 8 12 16];
o_SNR = zeros(1,4);
for i=1:4
   out = process_data(x,F,nu(i));
o_SNR(i) = snr(out);
end
disp('   nu        SNR(dB)');
disp([nu; o_SNR]');
```

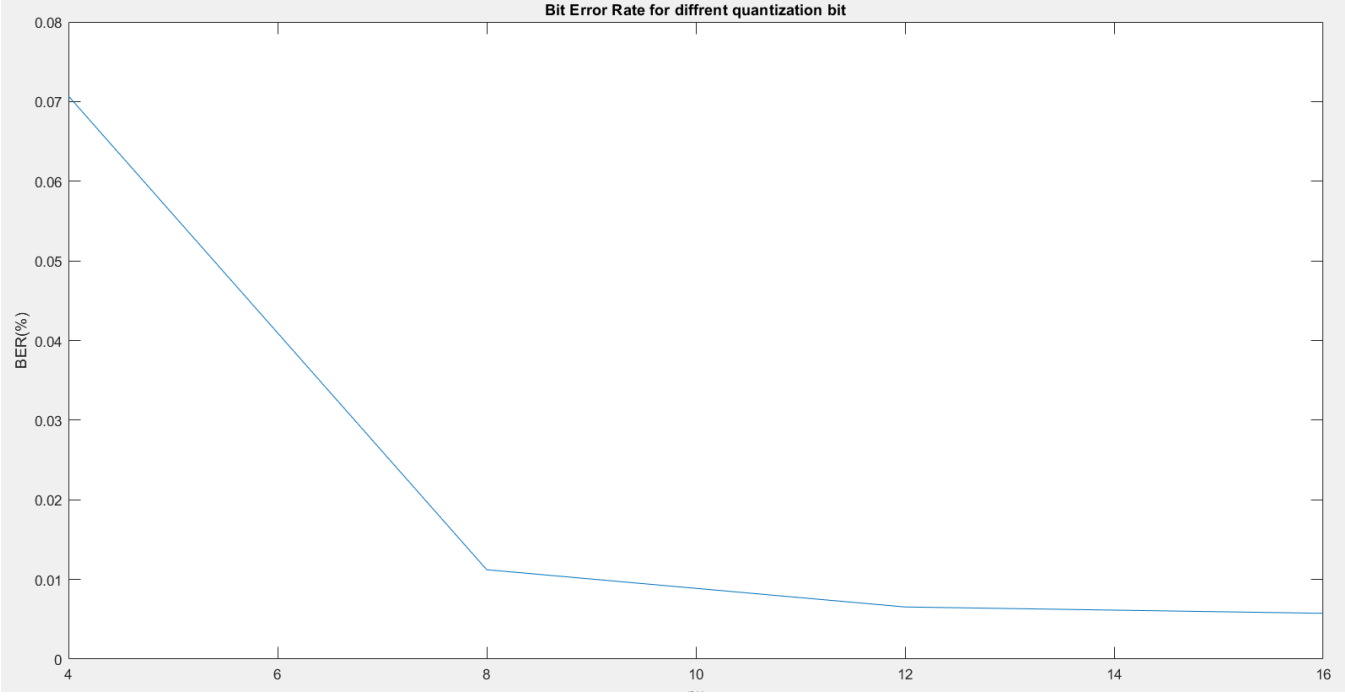
Command Window

nu	SNR(dB)
4.0000	-16.9099
8.0000	-15.8570
12.0000	-15.8683
16.0000	-15.5913

fx >>

نتایج فوق نشان می دهد که با افزایش تعداد بیت های کوانتیزاسیون ، SNR کاهش می یابد.

```
[x, F] = audioread('sound2.wav');
nu = [4 8 12 16];
o_BER = zeros(1,length(nu));
for i=1:length(nu)
   o_BER(i) = process_data(x,F,nu(i));
end
plot(nu,o_BER);
xlabel('nu');
ylabel('BER(%)');
title('Bit Error Rate for diffrent quantization bit');
```



همانطور که انتظار داشتیم و مشاهده می شود با افزایش تعداد بیت های کوانتیزاسیون، BER کاهش می یابد.

Question g ۶

با استفاده از قطعه کد زیر صوت ورودی از میکروفون را به صورت realtime پردازش کرده و صوت حاصل را پخش می کنیم.

```
%% Realtime Process
clear; clc; close all;

F = 60000;
fs = 4*F;
micReader = audioDeviceReader(F, F/2,"Driver","DirectSound");
speakerWriter = audioDeviceWriter(F, "Driver","DirectSound");

duration = 10;
tic;
while toc < 30
   audio = micReader();
   audio = process_data(audio, F);
   speakerWriter(audio);
end
```