

# Computer Assignment 4

مصطفی کرمانی نیا 810101575

امیر نداف فهمیده 810101540

## (1-1) تمرین

```
%% Part 0
close all;
clc;
clear;
```

- پیش از هر کاری تمام فیگورهای باز را بسته، متغیرها و ترمینال را هم کاملاً پاک می‌کنیم.

```
%% Part 1
% Define our char array which contains alphabet and additional characters
chars = ['a':'z', ' ', '.', ',', '!', "'", ";"];
% chars_len = length(chars); % its 32

% Creating a mapset with 2 rows and columns :
% Row 1 is the characters, Row 2 is the binary coded value of each character
Mapset = cell(2, 32);

% Set the values into Mapset
for i = 1:32
    % Store the character in the first row of Mapset
    Mapset{1, i} = chars(i);

    % Add the binary coded value for each character (i-1 in binary)
    Mapset{2, i} = dec2bin(i - 1, 5);
end
```

- حالا ابتدا تمام کاراکترهای مدنظر در سوال را در یک رشته به طول 32 بصورت زیر می‌بینیم:

chars      'abcdefghijklmnopqrstuvwxyz .!";'

- سپس یک سلول 2 در 32 برای نگهداری از کاراکترها و کد 5 بیتی متناظر آن ها می سازیم به نام

### .Mapset

- حالا در یک حلقه، روی تک تک ستون های Mapset حرکت کرده و هر بار در ردیف اول، یک کاراکتر گذاشته و در ردیف دوم هم کد 5 بیتی متناظر آن را قرار می دهیم که این کد شامل اعداد باینری 0 تا 31 است.

- نهایتاً Mapset بصورت زیر خواهد بود:

9	10	11	12	13	14	15
i	j	k	l	m	n	o
01000	01001	01010	01011	01100	01101	01110

## (2-1) تمرین

```
%% part 2
function Coded_signal=coding_amp(message, BitRate, mapset, show_plt)

    fs=100;
    % Convert message to binary sequence using mapset
    message_bin = '';
    for k = 1:length(message)
        char_index = find(strcmp(mapset(1,:), message(k))); % Find the character index in mapset
        if isempty(char_index)
            error(['Character ', message(k), ' not found in mapset.']);
        end
        char_bin = mapset{2, char_index}; % Get binary representation of each character
        message_bin = [message_bin, char_bin]; % Append to the binary message sequence
    end
    Coded_signal=[];
    message_bin_len = length(message_bin);

    t=0.01:(1/fs):1;
    a = [];
    a = [a,zeros(1,length(t))];
```

- در ابتدا فرکانس نمونه برداری تعریف شده و متغیر message\_bin برای ذخیره کردن پیام باینری شده‌ی نهایی استفاده می‌کنیم و یک حلقه می‌زنیم و تک تک کاراکتر های پیام ابتدایی را برداشته و تلاش می‌کنیم آن را در ردیف اول Mapset پیدا کنیم. اگر پیدا

نشد که ارور بر می‌گردانیم و اگر پیدا شد، معادل باینری آن کاراکتر را از ردیف دوم Mapset message\_bin برداشته و به اضافه می‌کنیم.

- سپس بردار زمان های نمونه برداری را با فرکانس تعیین شده (100) می‌سازیم. برای ساخت سیگنال های مپ شده با هر قطعه به طول bitrate هم از یک بردار a استفاده می‌کنیم که هر صد عنصر آن، نشانگر سیگنال معادل با یک رشته به طول bitrate است و در ابتدا یک سیگنال که تماماً صفر است بعنوان عضو اول آن گذاشته و سایر عناصر را با دادن یک ضریب به سینوس، نسبت به مقدار bitrate در حلقه‌ای که در ادامه توضیح میدهیم، تولید می‌کنیم.

```
for i=1:((2^BitRate)-1)
    a = [a, i/((2^BitRate)-1)*sin(2*pi*t)];
end
for i=1:BitRate:message_bin_len
    code_number = bin2dec(message_bin(i:i+BitRate-1));
    Coded_signal=[Coded_signal,a((code_number)*100 + 1 : (code_number+1)*100)];
end

if (show_plt == 1)
    x_axis=zeros(1,message_bin_len/BitRate*100);
    for i=1:message_bin_len/BitRate*100
        x_axis(i)=i/100;
    end
    figure;
    plot(x_axis,Coded_signal)
end
end
```

- در ادامه در یک حلقه a را به این شکل می‌سازیم که اگر مثلاً بخواهیم به جای هر 2 بیت یک سیگنال ارسال کنیم، باید بتوانیم  $2^2 = 4$  سیگنال مختلف برای حالات مختلف رشته‌ی 2 بیتی (0010 و 0110 و 1001 و 1101) داشته باشیم، پس در حالت کلی اگر بخواهیم به جای هر bitrate بتاتا بیت، یک سیگنال بفرستیم، نیازمند 2 به توان bitrate سیگنال مختلف هستیم که یکی از آن‌ها را پیش از این با افزودن سیگنال 0 به لیست، داریم، پس حلقه را از 1 تا 2 به توان bitrate منهای 1 می‌زنیم و در هر با پیمایش، با ضرب کردن ضریب  $1/2^{bitrate-1}$  در سینوس، به یک سیگنال جدید می‌رسیم.

- حالا که یک بردار  $a$  حاوی تمام سیگنال های مطلوبمان داریم، به سراغ پیام باینزی شده رفته و هر بار به اندازه  $i$  bitrate از آن را جدا کرده و باید از داخل  $a$  سیگنال معادل با آن را پیدا کنیم. این کار را با تبدیل آن عدد از باینزی به دسیمال و سپس ضرب کردن آن در 100 و برداشتن قطعه ای 100 تایی از همان بخش  $a$  انجام می شود چون مثلا فرض کنید سیگنال معادل با 11 را میخواهیم، طبق حلقه  $i$  قبلی، سیگنال معادل با این عدد که 3 است، در چهارمین بازه  $i$  100 تایی از  $a$  ذخیره شده که میشود بخش بین عنصر

.400 تا 301

- نهایتا برحسب مقدار bitrate و بطور کاملا flexible، هر bitata بیت موجود در پیام باینزی مان را به یک سیگنال تبدیل کرده و در coded\_signal ریختیم.

- نهایتا اگر flag موجود برای نمایش نمودار، 1 بود، صرفا آن سیگنال را برحسب زمان plot میکنیم.

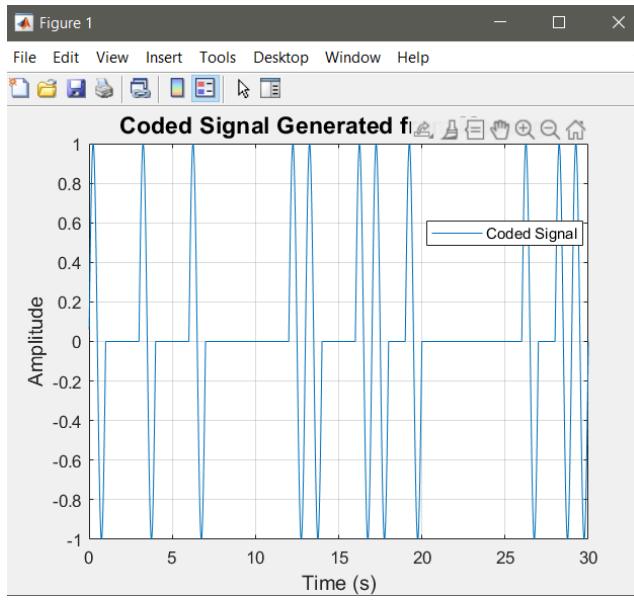
### تمرين (3-1)

- در وله ای اول باینزی شده ای کلمه  $i$  signal به صورت زیر است:

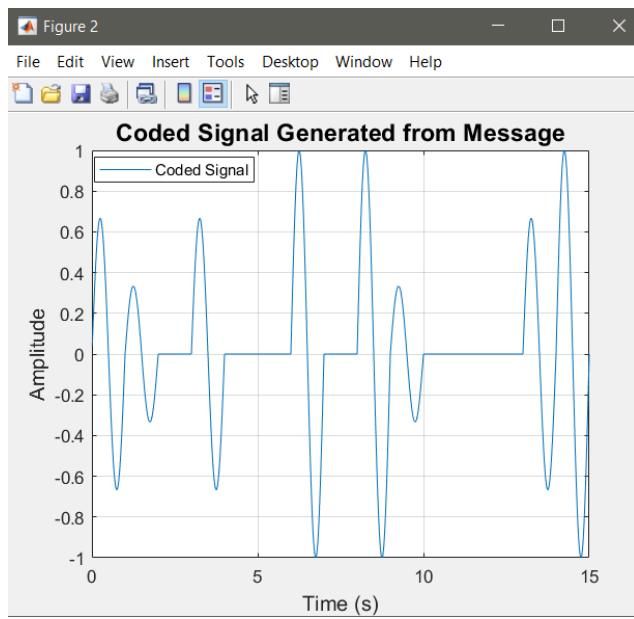
message	'signal'
message_bin	'10010010000011001101000001011'
message_bin_len	30

- با توجه به طول پیام که 30 تا است:

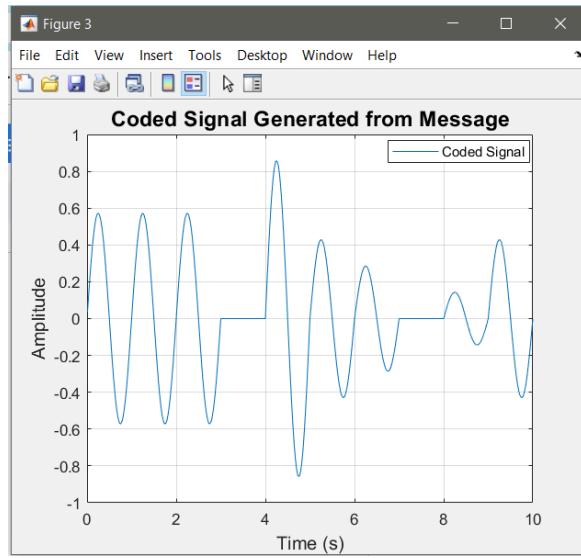
• اگر bitrate یک باشد انتظار داریم 30 ثانیه طول بکشد سیگنال ارسالی، و همین اتفاق هم می افتد:



اگر bitrate دو باشد انتظار داریم 15 ثانیه طول بکشد سیگنال ارسالی، و همین اتفاق  
• هم می افتد:



اگر bitrate سه باشد انتظار داریم 10 ثانیه طول بکشد سیگنال ارسالی، و همین اتفاق  
• هم می افتد:



## تمرين (4-1)

```

%% part 4
function decoded_message=decoding_amp(coded_signal, BitRate, mapset, show_plt)
    fs = 100;
    t = 0.01 : 0.01 : 1;
    time_of_message = (length(coded_signal)) / fs;

    correlation = zeros(1, time_of_message);

    Y = 2*sin(2*pi*t);
    for i = 1 : time_of_message
        X = coded_signal(fs*i-(fs-1):fs*i);
        correlation(i) = sum(X.*Y)/fs;
    end
    if show_plt == 1
        t = (0:length(correlation) - 1);
        figure;
        plot(t, correlation);
        title('Convolution of Signal with One Period of 2*sin(2*pi*t)');
        xlabel('Time (s)');
        ylabel('Amplitude');
    end
end

```

- ورودی های تابع شامل : سیگنال کدگذاری شده که باید رمزگشایی شود، تعداد بیت‌هایی که در هر ثانیه ارسال می‌شود، کاراکترها و معادل باینری آنها است و پارامتر منطقی 1 (یا 0) برای نمایش نمودار است. و خروجی آن decoded\_message یا همان پیام رمزگشایی شده به صورت رشته‌ای است.

- 
- $fs$  فرکانس نمونه‌برداری به ازای هر ثانیه است که برابر با 100 نمونه در هر ثانیه تنظیم شده است. بردار  $t$  زمان را برای یک بازه یک ثانیه با 100 نمونه تعریف می‌کند. متغیر  $time\_of\_message$  مدت زمان کل سیگنال دریافتی را بر حسب ثانیه محاسبه می‌کند که با تقسیم طول سیگنال بر فرکانس نمونه‌برداری به دست می‌آید.
  - سپس، آرایه‌ای به نام  $correlation$  با اندازه  $time\_of\_message$  ایجاد می‌شود تا مقادیر همبستگی برای هر ثانیه از سیگنال ذخیره شود و سیگنال مرجع ۷ تعریف می‌شود که سیگنالی است که با آن سیگنال دریافتی همبستگی گرفته می‌شود.
  - در ادامه، یک حلقه  $for$  برای محاسبه همبستگی بین سیگنال دریافتی و سیگنال مرجع برای هر ثانیه اجرا می‌شود. در هر تکرار حلقه، 100 نمونه مربوط به یک ثانیه از سیگنال کدگذاری شده استخراج می‌شود و با سیگنال مرجع ۷ ضرب و سپس جمع می‌شود و نتیجه بر  $fs$  تقسیم می‌گردد (طبق توضیحات صورت سوال) تا مقدار همبستگی برای آن بازه زمانی محاسبه شود.
  - نهایتاً اگر پارامتر  $show\_plt$  برابر با ۱ باشد، نمودار همبستگی نمایش داده می‌شود. محور  $x$  زمان بر حسب ثانیه و محور  $y$  دامنه همبستگی را نشان می‌دهد.

```

% Find thresholds
a = [];
a = [a,0];
for i=1:((2^BitRate)-1)
    a = [a, i/((2^BitRate)-1)];
end
threshold = [];
for i=1:((2^BitRate)-1)
    threshold = [threshold, (a(i)+a(i+1))/2];
end

binarydecoded = [];
for i=1:time_of_message
    if correlation(i) < threshold(1)
        binarydecoded = [binarydecoded dec2bin(0, BitRate)];
    end
    if correlation(i) > threshold((2^BitRate)-1)
        binarydecoded = [binarydecoded dec2bin(((2^BitRate)-1), BitRate)];
    end

    for j = 1 : ((2^BitRate)-1)-1
        if threshold(j) < correlation(i) && threshold(j+1) > correlation(i)
            binarydecoded = [binarydecoded dec2bin(j, BitRate)];
        end
    end
end
end

```

- سپس، thresholdهای لازم برای تصمیم‌گیری در مورد بیت‌های دریافتی تعیین می‌شوند. در ابتدا، آرایه a با مقدار 0 شروع می‌شود و سپس به تعداد حالات ممکن برای نمایش بوسیله ی سپس، آستانه‌ها با محاسبه میانگین هر جفت متوالی از مقادیر a تعیین می‌شوند. این آستانه‌ها برای تشخیص بیت‌های دریافتی بر اساس مقدار همبستگی استفاده می‌شوند.

- پس از تعیین آستانه‌ها، مرحله دیکود کردن باینری آغاز می‌شود. در این بخش، برای هر مقدار همبستگی محاسبه شده در هر ثانیه، بررسی می‌شود که آیا این مقدار کمتر از اولین آستانه است یا بیشتر از آخرین آستانه. اگر کمتر باشد، بیت‌های معادل صفر اضافه می‌شوند و اگر بیشتر از آخرین آستانه باشد، بیت‌های معادل بیشترین مقدار ممکن (2 به توان bitrate منهای 1) اضافه می‌شوند. در غیر این صورت، حلقه داخلی بررسی می‌کند که مقدار همبستگی در کدام بازه آستانه‌ها قرار دارد و بیت‌های معادل آن بازه به رشتہ باینری رمزگشایی شده اضافه می‌شوند.

```

    decoded_message = [];
    lenbin = length(binarydecoded);
    for i = 1 : lenbin/5
        ch = binarydecoded(5*i-4 : 5*i);
        number = bin2dec(ch);
        decoded_message = [decoded_message mapset{1, number+1}];
    end
end

```

- در نهایت، پیام اصلی از رشته باینری رمزگشایی شده بازسازی می‌شود. حلقه for هر 5 بیت را استخراج کرده و با استفاده از تابع bin2dec به عدد دسیمال تبدیل می‌کند. سپس با استفاده از این عدد، کاراکتر معادل از مجموعه نگاشت (mapset) انتخاب شده و به پیام رمزگشایی شده اضافه می‌شود. توجه داشته باشید که به دلیل اینکه اندیس‌ها در MATLAB از 1 شروع می‌شوند، به عدد تبدیل شده 1 اضافه می‌شود تا به اندیس صحیح در mapset دست یابیم

```

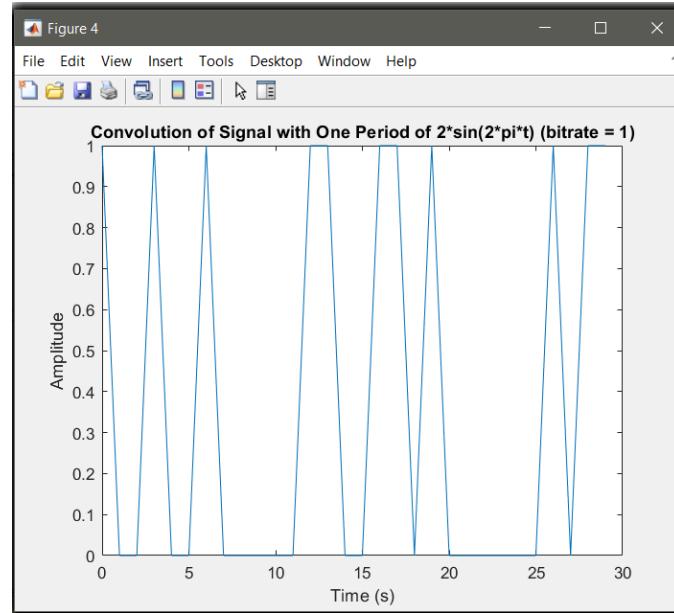
%% Part 4 test
decoded_message1=decoding_amp(coded_signal1, 1, Mapset, 1);
decoded_message2=decoding_amp(coded_signal2, 2, Mapset, 1);
decoded_message3=decoding_amp(coded_signal3, 3, Mapset, 1);

disp(decoded_message1);
disp(decoded_message2);
disp(decoded_message3);

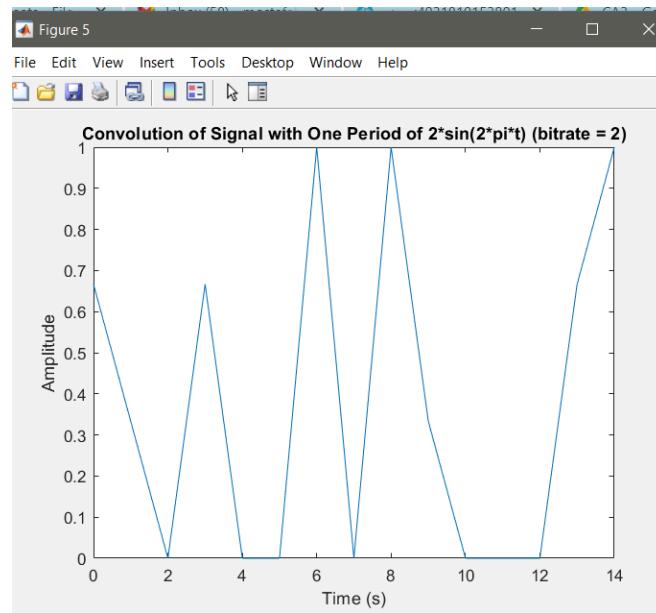
```

- در بخش تست تابع هم سه سیگنال کدگذاری شده با نرخ بیت‌های 1، 2 و 3 بیت بر ثانیه به ترتیب به تابع decoding\_amp ارسال می‌شوند و پیام‌های رمزگشایی شده در متغیرهای decoded\_message3 و decoded\_message1، decoded\_message2 سپس این پیام‌ها با استفاده از disp نمایش داده می‌شوند. پaramتر show\_plt برابر با 1 قرار داده شده است تا نمودار همبستگی برای هر تست نمایش داده شود:

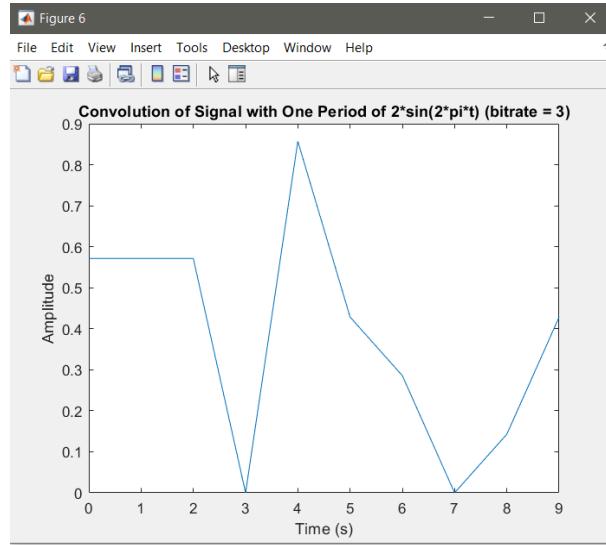
- در bitrate مساوی یک که مقدار کانولوشن طبق انتظارمان یا 0 است و یا 1:



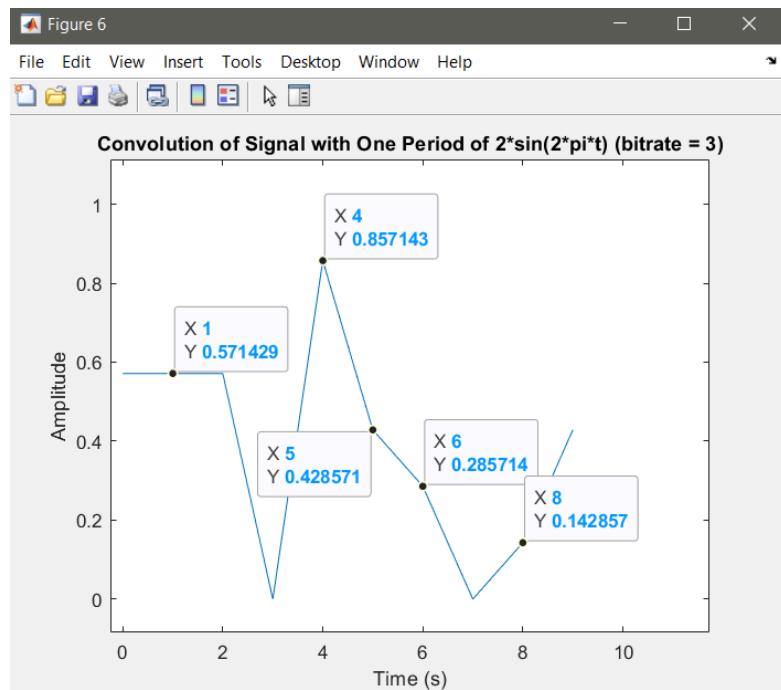
در مساوی دو هم مقدار کانولوشن طبق انتظارمان یا ۰ است و یا ۱ و یا  $\frac{1}{2}$  و یا  $\frac{2}{3}$  :



در مساوی سه هم مقدار کانولوشن طبق انتظارمان یا ۰ است و یا ۱ و یا  $\frac{1}{7}$  و یا  $\frac{2}{7}$  و یا  $\frac{3}{7}$  :



با دقت در نمودار آخر میتوان دید که دقیقاً اگر مقداری گرفته شده است، به فرم  $7/8$  بوده است:



و در نهایت هم هر سه پیام به درسی دیکود شده و چاپ شدند:

```
signal  
signal  
signal
```

## تمرین 1 (5-1)

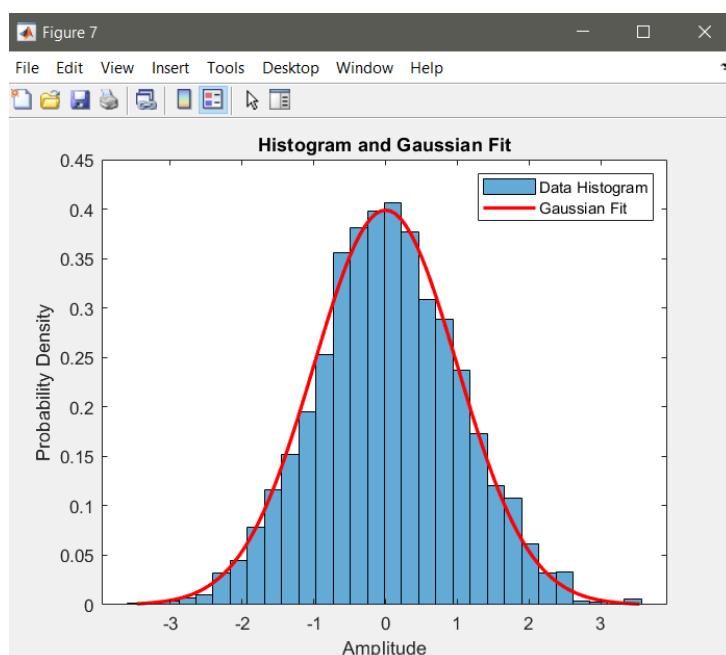
```
%% part 5  
noise = randn(1,3000);  
  
figure;  
histogram(noise, 30, 'Normalization', 'pdf');  
hold on;  
x = linspace(min(noise), max(noise), 100);  
plot(x, normpdf(x, 0, 1), 'r', 'LineWidth', 2);  
title('Histogram and Gaussian Fit');  
xlabel('Amplitude');  
ylabel('Probability Density');  
legend('Data Histogram', 'Gaussian Fit');
```

- ابتدا با دستور `(noise = randn(1,3000);`, یک بردار شامل 3000 نمونه از توزیع نرمال استاندارد (با میانگین صفر و واریانس یک) تولید می شود. این نویز می تواند به سیگنال اصلی اضافه شود تا تاثیر نویز در دریافت سیگنال شبیه سازی گردد.
- سپس با استفاده از دستور `figure`; یک پنجره جدید برای رسم نمودار باز می شود. در این پنجره، ابتدا هیستوگرام نویز تولید شده با 30 bin رسم می شود و با گزینه 'Normalization','pdf" نرمال سازی می گردد تا هیستوگرام به تابع چگالی احتمال تبدیل شود. این نرمال سازی اجازه می دهد تا هیستوگرام به راحتی با تابع چگالی نرمال استاندارد مقایسه شود.
- بعد از رسم هیستوگرام، با دستور `hold on`; اجازه می دهیم تا نمودارهای بعدی روی همان پنجره رسم شوند بدون اینکه هیستوگرام پاک شود. سپس محور x را با استفاده از `linspace` تعریف می کنیم که 100 نقطه مساوی فاصله بین حداقل و حداکثر مقدار نویز تولید شده ایجاد می کند. این نقاط برای رسم تابع چگالی نرمال استاندارد استفاده می شوند.
- با دستور `(plot(x, normpdf(x, 0, 1), 'r', 'LineWidth', 2);`تابع چگالی احتمال توزیع نرمال استاندارد (با میانگین صفر و واریانس یک) روی هیستوگرام رسم می شود. خط قرمز با عرض دو

برابر نشان‌دهنده تابع نرمال استاندارد است که باید با شکل هیستوگرام داده‌های نویز مطابقت داشته باشد.

- این کد به ما امکان می‌دهد تا بررسی کنیم که نویز تولید شده با استفاده از `randn` واقعاً دارای توزیع نرمال استاندارد است یا خیر. اگر هیستوگرام داده‌ها به خوبی با خط قرمز که نشان‌دهنده تابع چگالی نرمال استاندارد است، همخوانی داشته باشد، نشان می‌دهد که نویز تولید شده گوسی بوده، میانگین آن نزدیک به صفر و واریانس آن نزدیک به یک است.

در نهایت، با اجرای این کد، نمودار زیر مشاهده می‌شود:



پس هیستوگرام داده‌های نویز تولید شده به همراه تابع چگالی نرمال استاندارد (دارای میانگین 0 و واریانس 1) رسم شده است. و این نمایش مار را مطمئن می‌کند که نویز تولید شده دارای ویژگی‌های مورد نظر (گوسی بودن، میانگین صفر و واریانس یک) می‌باشد.

## (6-1) تمرین

---

```

%% part 6

message = 'signal';
for i=1:3
    coded_signal = coding_amp(message, i, Mapset, 1);
    coded_signal = coded_signal + 0.01*randn(1, length(coded_signal));
    x_axis=zeros(1,length(coded_signal));
    for j=1:length(coded_signal)
        x_axis(j)=j/100;
    end
    figure;
    plot(x_axis,coded_signal)
    title('coded signal with noise');
    decoded_message=decoding_amp(coded_signal, i, Mapset, 1);

    disp([decoded_message, ' for BitRate = ', num2str(i)]);
end

```

- ابتدا، پیام اصلی که در اینجا رشته 'signal' تعریف شده است، برای هر کدام از bitrate های ۱، ۲ و ۳ با استفاده از تابع coding\_amp کدگذاری می‌شود. این تابع سیگنال کدگذاری شده را بر اساس نرخ بیت مشخص و مجموعه مپینگ (Mapset) تولید می‌کند.
- سپس به سیگنال کدگذاری شده، نویز گوسی با میانگین صفر و واریانس ۱..... اضافه می‌شود. این نویز با استفاده از دستور randn تولید شده و با ضرب در ۱... مقیاس‌بندی می‌شود تا واریانس آن به ۱..... کاهش یابد. افزودن این نویز به سیگنال کدگذاری شده، شرایط انتقال سیگنال در محیط‌های واقعی‌تر با حضور نویز را شبیه‌سازی می‌کند.
- بعد از اضافه کردن نویز، محور زمان برای رسم سیگنال نویزی ایجاد می‌شود که با استفاده از یک حلقه for هر نمونه سیگنال به زمان متناظر خود تقسیم بر ۱۰۰ تعیین می‌شود تا مقیاس زمانی مناسبی برای نمودار فراهم شود. سپس سیگنال نویزی با استفاده از دستور plot رسم می‌شود
- پس از رسم سیگنال نویزی، این سیگنال به تابع decoding\_amp ارسال می‌شود که پیام اصلی را از سیگنال نویزی دیکد می‌کند. نتیجه دیکدینگ، که پیام استخراج شده است، با استفاده از دستور disp به همراه نرخ بیت مورد استفاده نمایش داده می‌شود. این فرآیند برای هر یک از نرخ بیت‌های ۱، ۲ و ۳ به صورت جداگانه تکرار می‌شود تا تاثیر نرخ بیت مختلف بر دقت دیکدینگ

---

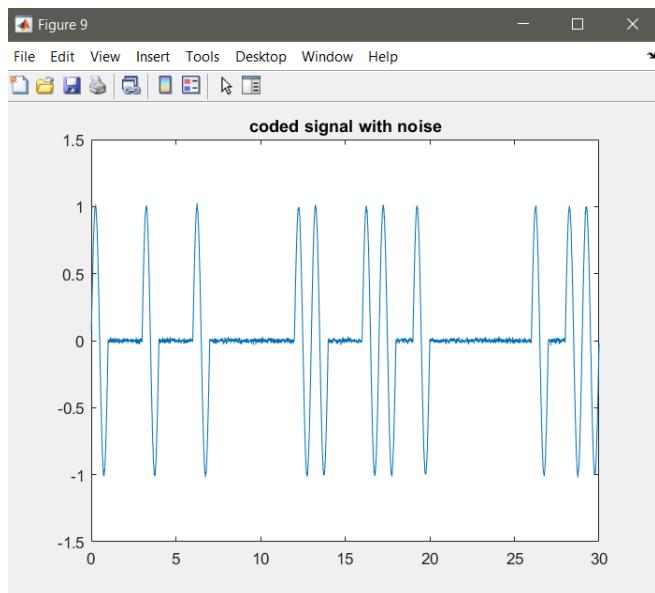
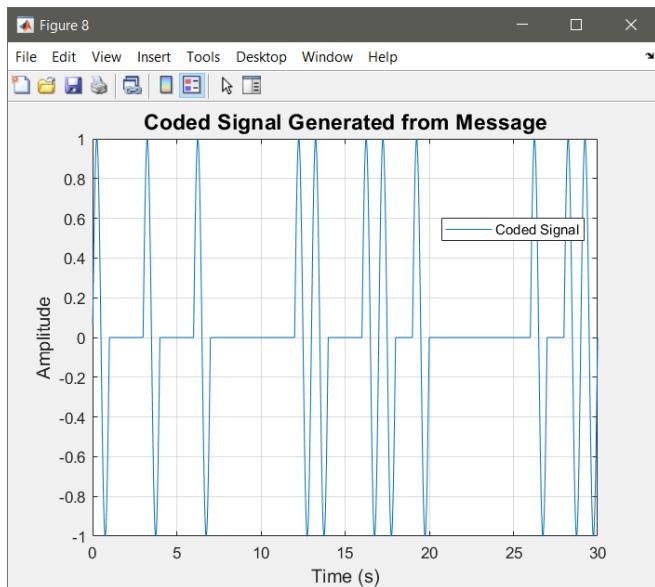
در حضور نویز بررسی شود. با اجرای این کد، برای هر نرخ بیت، سیگنال کدگذاری شده با نویز رسم شده و پیام دیدک شده در پنجره فرمان نمایش داده می‌شود.

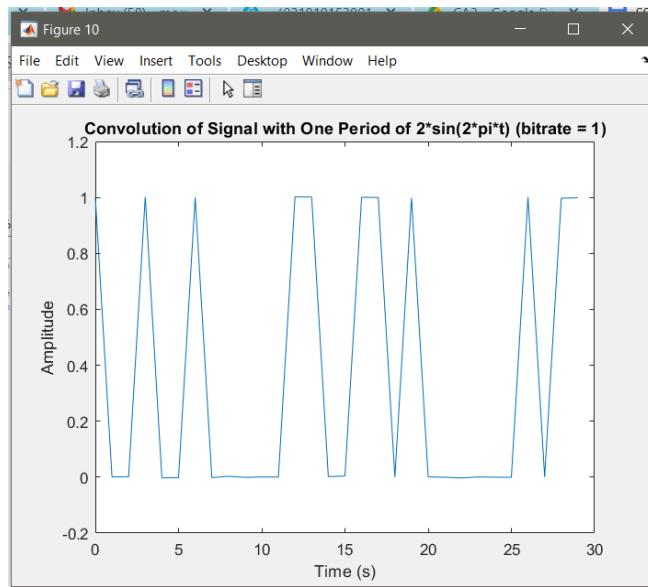
- پس دیدیم که چگونه افزودن نویز با واریانس کم بر قابلیت استخراج صحیح پیام اصلی تاثیر می‌گذارد و بسته به bitrate انتخاب شده، دقت دیکدینگ متفاوت خواهد بود. به طور کلی، با افزایش bitrate حساسیت سیستم به نویز افزایش می‌یابد و احتمال اشتباه در دیکدینگ بیشتر می‌شود، در حالی که bitrates پایین‌تر معمولاً مقاومت بیشتری در برابر نویز دارند و احتمال استخراج صحیح پیام بالاتر است. این تحلیل به درک بهتری از تعادل بین bitrate و مقاومت در برابر نویز در سیستم‌های کدگذاری و دیکدینگ کمک می‌کند.

- در نهایت تمام سیگنال‌ها به خوبی دیدک شدند:

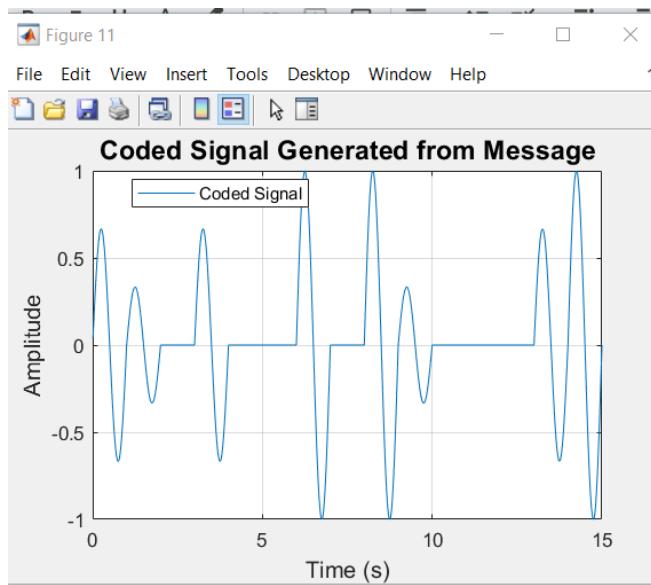
```
signal for BitRate = 1  
signal for BitRate = 2  
signal for BitRate = 3
```

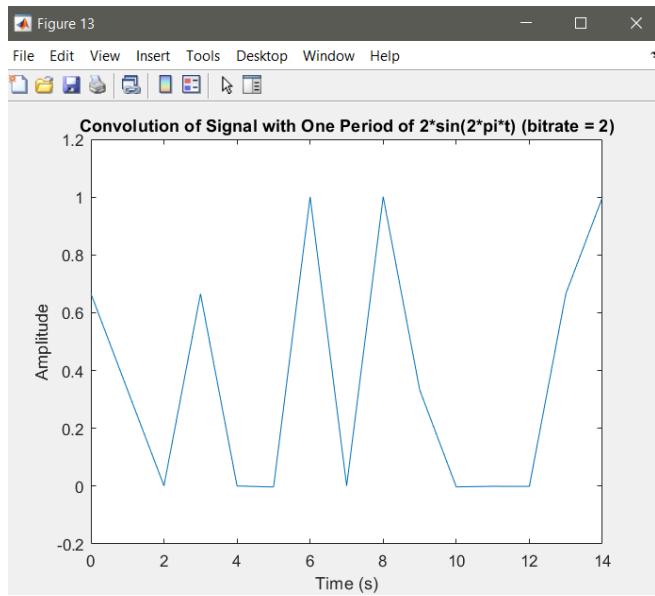
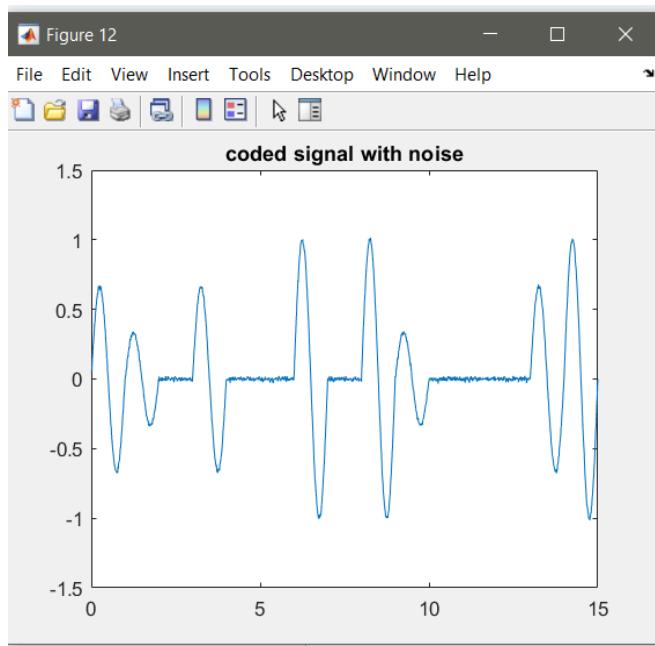
- در اینجا سیگنال‌های تولید شده را در هر bitrate بدون نویز و با نویز و نتیجه‌ی کانولوشن گرفته شده در حین دیکود مشاهده می‌کنیم:  
برای  $bitrate = 1$  داریم -



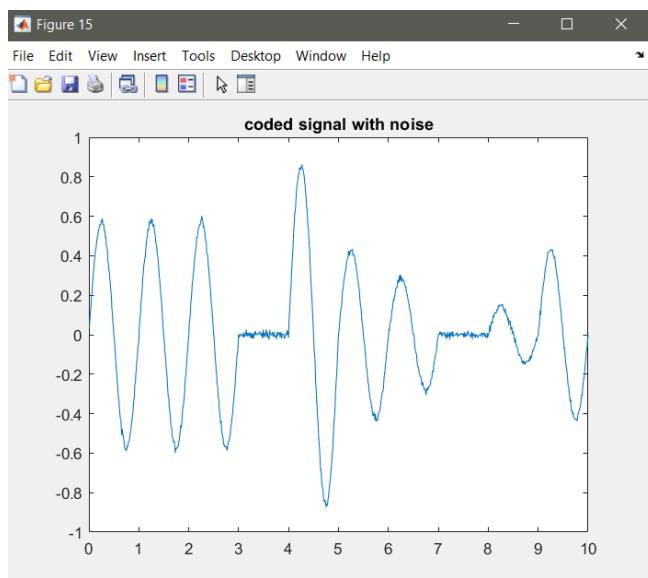
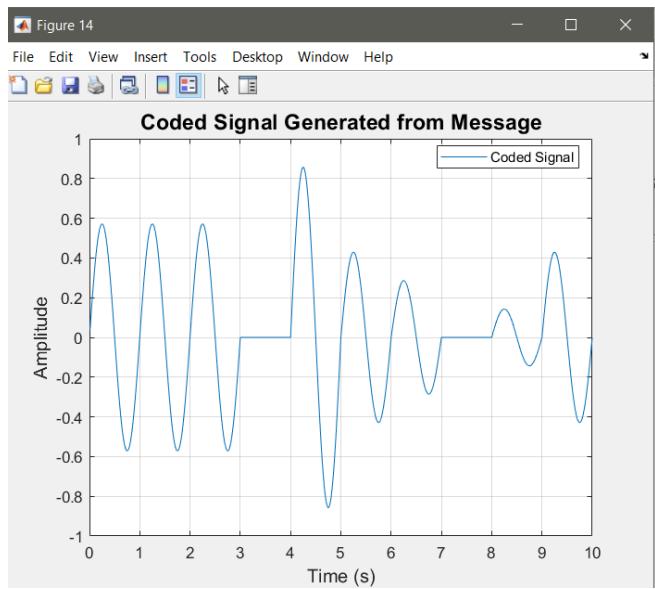


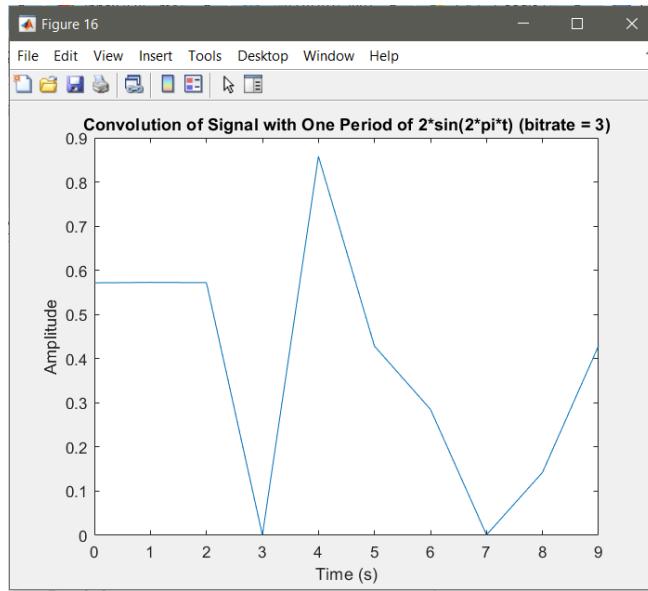
برای bitrate = 2 داریم :-





برای bitrate = 3 داریم -





## تمرين 1 و 7-8

```

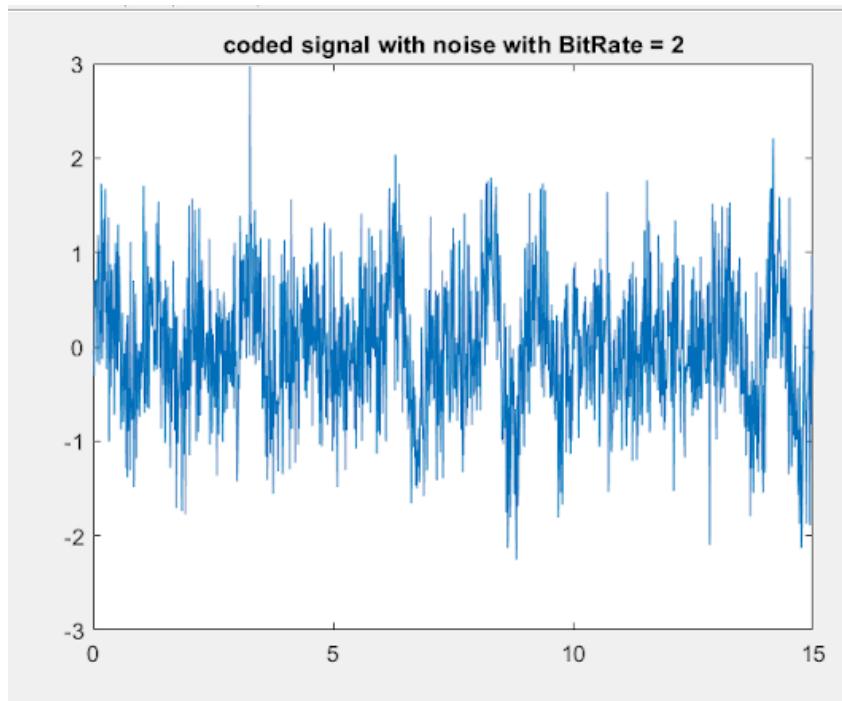
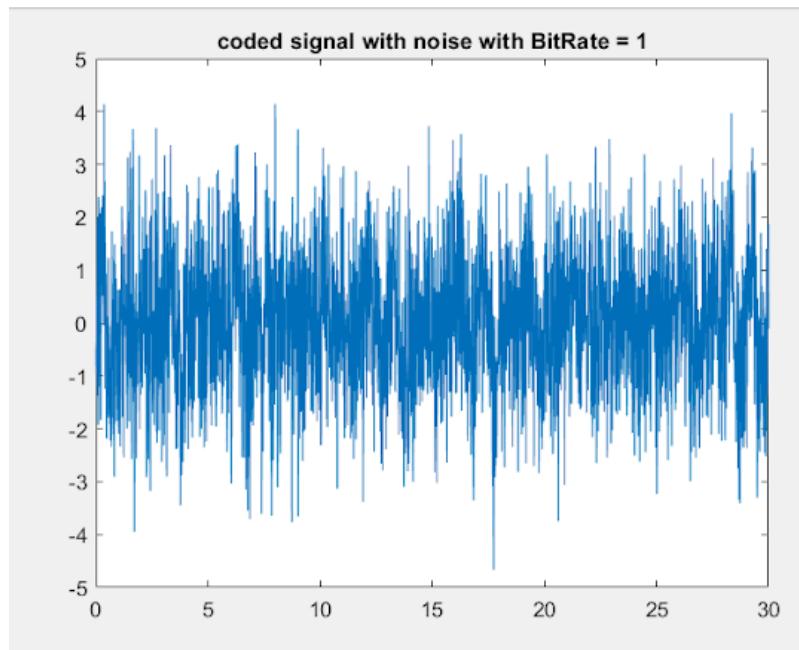
61 %% part 7 and 8
62
63 message = 'signal';
64 for BitRate=1:3
65     sum_var = 0;
66     max_var = 0;
67     for i=1:1000
68         std = 0.01;
69         decoded_message = 'signal';
70         while message == decoded_message
71             coded_signal = coding_amp(message, BitRate, Mapset, 0);
72             coded_signal = coded_signal + std*randn(1, length(coded_signal));
73             decoded_message=decoding_amp(coded_signal, BitRate, Mapset, 0);
74             if message == decoded_message
75                 std = std + 0.01;
76             end
77         end
78         sum_var = sum_var + (std^2);
79         if max_var < (std-0.01)^2
80             max_var = (std-0.01)^2;
81         end
82     end
83     x_axis=zeros(1,length(coded_signal));
84     for j=1:length(coded_signal)
85         x_axis(j)=j/100;
86     end
87     figure;
88     plot(x_axis,coded_signal)
89     title(['coded signal with noise with BitRate = ', num2str(BitRate)]);
90     disp([decoded_message, ' for BitRate = ', num2str(BitRate), ' mean noise variance = ', num2str(sum_var/1000)]);
91     disp(['Max variance for BitRate = ', num2str(BitRate), ' decodes correctly is ', num2str(max_var)]);
92 end

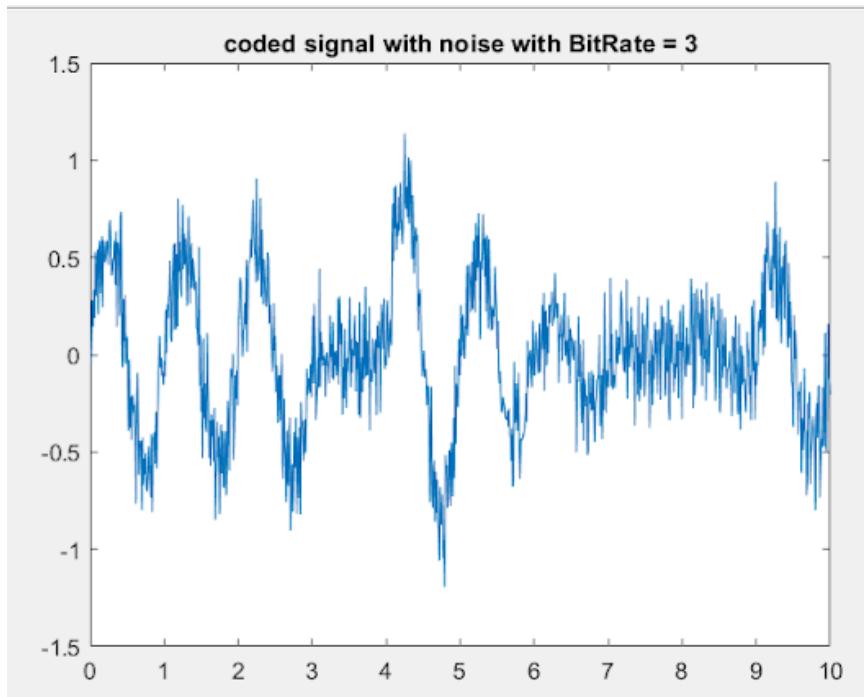
```

---

در این تکه کد ابتدا message را به signal ست می‌کنیم. سپس در حلقه for، مقدار BitRate را از 1 تا 3 تغییر می‌دهیم. برای هر BitRate، به تعداد 1000 بار پیدا می‌کنیم که در چه واریانس نویزی، سیگنال کد شده به درستی decode نمی‌شود. به این منظور ابتدا انحراف معیار را 0.01 ست می‌کنیم. سپس در یک حلقه while پیام مورد نظر را به وسیله coding\_& decoding\_& std را یک صدم یک صدم افزایش شده در می‌آوریم، نویز را اضافه می‌کنیم، سیگنال با نویز را به تابع decoding\_& می‌دهیم و در آخر تا زمانی که message برابر با decoded\_message باشد، سیگنال کد گذاری می‌دهیم. زمانی که بالاخره نویز باعث شد که پیام به درستی decode نشود، مقدار واریانس را با مجموع واریانس‌های قبلی جمع می‌کنیم (اینکار را برای این می‌کنیم که در انتها میانگین را گزارش کنیم) سپس ماکسیمم واریانسی که پیام به درستی decode شده را نیز آپدیت می‌کنیم. سپس آخرین سیگنال نویز دار را که پلات می‌کنیم و ماکسیمم واریانس و میانگین واریانس را گزارش می‌کنیم. خروجی به صورت زیر است.

```
siwnal for BitRate = 1 mean noise variance = 1.5397
Max variance for BitRate = 1 decodes correctly is 2.4964
sigoal for BitRate = 2 mean noise variance = 0.23
Max variance for BitRate = 2 decodes correctly is 0.4096
sigmql for BitRate = 3 mean noise variance = 0.052774
Max variance for BitRate = 3 decodes correctly is 0.0961
```





**تمرین 9-1)** با افزایش دامنه سیگنال فرستاده شده، فاصله ترشلدها از هم بیشتر می‌شود و در این صورت برای decode کردن سیگنال حساسیت کمتری نسبت به نویز نشان می‌دهیم.

**تمرین 10-1)** اگر مشکل نویز وجود نداشت ما می‌توانستیم، BitRate را به اندازه کل تعداد بیت پیام (زمانی که به باینری تبدیل کردیم) افزایش دهیم و کل پیام را در یک ثانیه ارسال کنیم. در این صورت هرچقدر تعداد بیت پیام بالاتر باشد، ترشلدها نزدیک تر به هم خواهند بود اما چون نویزی وجود ندارد که باعث شود مقدار کوولیشن تغییر کند، ترشلدها دقیق خواهند بود و پیام به درستی رمزگشایی می‌شود.

**تمرین 11-1)** تابع  $2\sin(2\pi t)$  را مثل تابع  $\text{decoding\_amp2}$  می‌سازیم ولی به جای  $t$  از  $10\sin(2\pi t)$  استفاده می‌کنیم. همچنین مقادیر تابع کانولوشن 5 برابر می‌شوند پس مقادیر  $a$  را نیز 5 برابر می‌کنیم تا ترشلدها را نیز به مقدار صحیح ست کنیم.

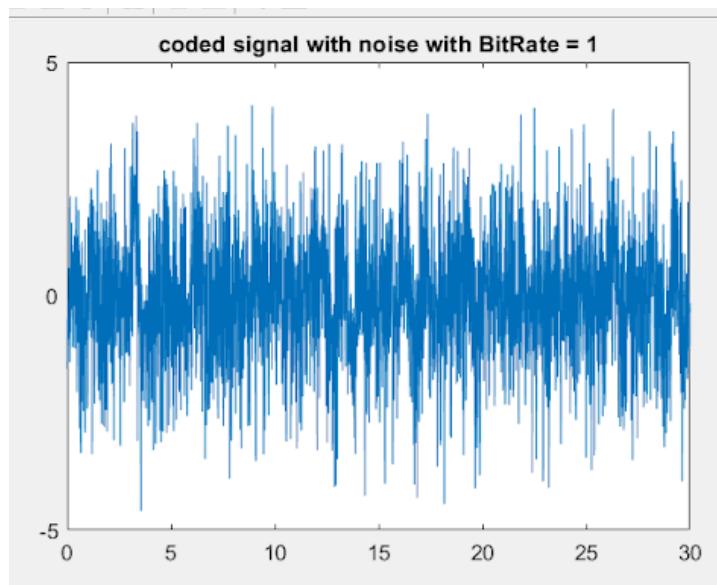
```
228 -      Y = 10*sin(2*pi*t);
```

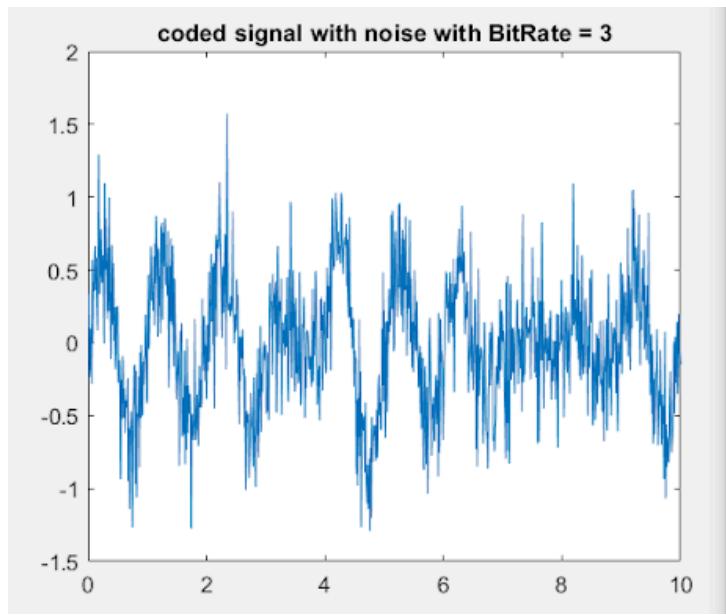
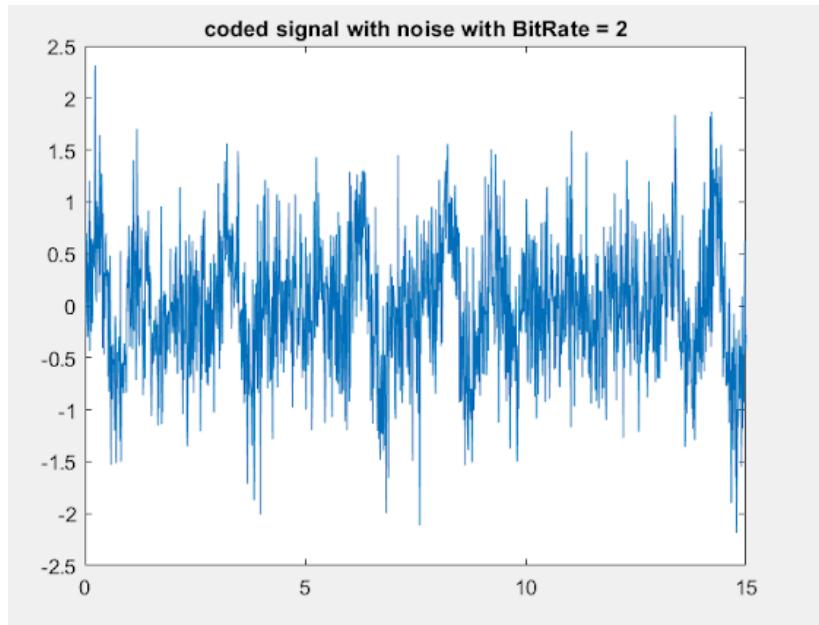
```
241 % Find thresholds
242 a = [];
243 a = [a,0];
244 for i=1:(2^BitRate)-1)
245     a = [a, 5*i/((2^BitRate)-1)];
246 end
247 threshold = [];
248 for i=1:(2^BitRate)-1)
249     threshold = [threshold, (a(i)+a(i+1))/2];
250 end
```

سپس از کد بخش 7 استفاده می‌کنیم اما این سری از decoding\_amp2 برای استخراج پیام استفاده می‌کنیم. نتایج به صورت زیر می‌شود که سه خط اول نتایج بخش 7 و سه خط دوم نتایج این بخش است که همانطور که دیده می‌شود تقریباً با هم برابرند.

```
siwnal for BitRate = 1 mean noise variance = 1.5397
Max variance for BitRate = 1 decodes correctly is 2.4964
sigoal for BitRate = 2 mean noise variance = 0.23
Max variance for BitRate = 2 decodes correctly is 0.4096
sigmql for BitRate = 3 mean noise variance = 0.052774
Max variance for BitRate = 3 decodes correctly is 0.0961
    ignal for BitRate = 1 mean noise variance = 1.5433
Max variance for BitRate = 1 decodes correctly is 2.56
sienal for BitRate = 2 mean noise variance = 0.23039
Max variance for BitRate = 2 decodes correctly is 0.4096
siorqm for BitRate = 3 mean noise variance = 0.053031
Max variance for BitRate = 3 decodes correctly is 0.1024
```

**fx** >> |





همانطور که دیده می‌شود، با تغییر 2 به 10 تغییر نمی‌بینیم چون به همان نسبت مقادیرتابع کورلیشن 5 برابر می‌شوند و ترشلهای هم 5 برابر می‌شوند و تاثیر خاصی ندارد (نویز همان تاثیر قبلی خودش را می‌گذارد انگار که قدرت نویز هم 5 برابر شده) اما اگر دامنه سیگنال کد گذاری شده بیشتر شود، نویز قوی تری نیاز است تا باعث شود پیام به درستی رمزگشایی نشود، (بخش 9) و در این حالت اگر دامنه  $n$  برابر شود، تاثیر نویز بسیار کم می‌شود زیرا در این حالت قدرت نویز  $n$  برابر نمی‌شود.

---

**تمرین 12-1**) سرعت اینترنت های adsl معمولاً بین 8 تا 16 مگابیت بر ثانیه هستند یعنی حدودا $10^6$ \*8 بیت بر ثانیه ولی ما در این تمرین 1 و 2 و 3 بیت بر ثانیه ارسال کردیم.