

پروژه کامپیوتری 4

سیگنال ها و سیستم ها

دکتر اخوان

پریسا محمدی 810101509

سامان دوچی طوسی 810101420

تمرین 1-1

در این بخش تابعی به نام `generateMapset()` را نوشته که هدف اش ایجاد `Mapset` مورد نظر سوال است. در این تابع ابتدا، 32 کاراکتر را طبق خواسته سوال در آرایه ای به `characterList()` قرار می دهیم. سپس سلولی، با 2 ردیف و تعداد ستون هایی در ابعاد تعداد کاراکترهای مان، را به `Mapset` نسبت می دهیم. برای پر کردن این سلول در یک حلقه `for` ستون های این سلول را طی کرده و در ردیف اول این ستون ها، کاراکتر ها را یکی یکی قرار داده و در ردیف دوم کد باینری 5 بیتی دلخواهی که از صفر شروع می شود را به هر کاراکتر نسبت می دهیم.

```
function Mapset = generateMapset()
    characterList = ['a':'z', ' ', ',', '.', '!', '"', ';'];

    Mapset = cell(2, length(characterList));

    for k = 1:length(characterList)
        Mapset{1, k} = characterList(k);
        Mapset{2, k} = dec2bin(k-1, 5);
    end
end
```

تصویر 1 - تابع ایجاد مپ ست (نسبت دادن کد 5 بیتی به هر یک از 32 کاراکتر)

تمرین 2-1

در این بخش می خواهیم تابعی به نام `coding_amp()` بنویسیم که هدف اش کد کردن پیامی در یک سیگنال خروجی است. این تابع در ابتدا `Mapset` ای را به عنوان معیاری برای کد کردن ایجاد می کنیم. می خواهیم در مرحله اول `message` را طبق `Mapset` به معادل باینری اش تبدیل کنیم. برای این کار یک `string` خالی (`binary_coded_message`) را در ابتدا تعیین می کنیم. در گام بعدی، در حلقه (خارجی) روی تک تک حرف ها و کاراکتر های پیام امان لوپ می زنیم و در هر مرحله حرف فعلی را در یک متغیر ذخیره می کنیم. در حلقه داخلی روی کاراکتر های ردیف اول `Mapset` امان لوپ می زنیم تا بتوانیم کاراکتر فعلی پیام را بین آنها پیدا کنیم و وقتی که آن را پیدا کردیم، کد متناظر اش را به `binary_coded_message` اضافه می کنیم و از حلقه داخلی، بیرون می آییم. با خروج از حلقه خارجی در متغیر `binary_coded_message` تمام پیام به صورت باینری کد شده و ذخیره شده است. در مرحله بعدی باید طبق خواسته سوال، آن را به صورت مضربی از سیگنال های سینوسی که در طی زمان در کنار یکدیگر قرار گرفته اند کد کنیم. برای این کار روی کد باینری شده مان لوپ می زنیم و هر بار به تعداد `bitRate` رقم باینری از پیام جدا می کنیم و آن را در متغیر `curr_code` ذخیره می کنیم. می دانیم که مقدار `curr_code` (مقدار ددهی اش) دامنه سیگنال سینوسی متناظر را تعیین می کند. ولی ما طبق خواسته سوال (محدودیت توان منبع فرستنده سیگنال) بایستی دامنه سیگنال هر کد را بین صفر و یک قرار دهیم. در نتیجه از آنجایی که حداکثر مقدار هر کد باینری جدا شده از `binary_coded_message` برابر با 2 به توان تعداد بیت هایش (`bitRate`) منهای یک است، بایستی این مقدار را در مخرج شان قرار دهیم. حال، برای هر کد یک سیگنال سینوسی با دامنه متناظر را توانستیم ایجاد کنیم. اکنون باید این سیگنال را به بخشی از سیگنالی که در نهایت شامل همه سیگنال های کد شده می شود اضافه کنیم. بدین منظور، نقطه شروع و پایانی را به ازای هر کد تعیین کرده و در آن بازه مقدار سیگنال خروجی برابر با سیگنال کد

شده قرار داده می شود و طبق خواسته مسئله طول هر یک از این بازه ها (طول هر سیگنال کد شده) برابر با f_s (یک ثانیه) است. در نهایت این تابع، سیگنال کد شده را خروجی می دهد.

```
function output_signal = coding_amp(message, bitRate)

    Mapset = generateMapset;
    binary_coded_message = '';
    messege_len = strlen(message);

    for p = 1:messege_len
        curr_letter = extract(message,p);
        for q = 1:length(Mapset(1,:))
            if strcmp(curr_letter, Mapset{1, q})
                binary_coded_message = [binary_coded_message ,Mapset{2, q}];
                break;
            end
        end
    end

    binary_length = strlen(binary_coded_message);
    num_func = 2^bitRate;
    f_s = 100;
    t = 0:(1/f_s):1-(1/f_s);
    out_values = zeros(1, f_s * binary_length/bitRate);

    for n = 1:(bitRate):(binary_length - bitRate + 1)
        curr_code = extractBetween(binary_coded_message,n,n+bitRate-1);

        decimal_code = bin2dec(curr_code);

        start_idx = ((n-1)/bitRate) * f_s + 1;
        end_idx = start_idx + f_s - 1;

        out_values(start_idx:end_idx) = (decimal_code/(num_func -1 )) * sin(2*pi*t);
    end
    output_signal = out_values;
end
```

تصویر 2 - تابع *decode* کردن پیامی در یک سیگنالی تکه ای مضرب سینوسی

تمرین 3-1

```

clc;
close all
clear

message = 'signal';
bitRates = [1, 2, 3];
f_s = 100;

for k = 1:length(bitRates)
    bitRate = bitRates(k);
    output_signal = coding_amp_1(message,bitRate);
    t = (0:length(output_signal) - 1) / f_s;

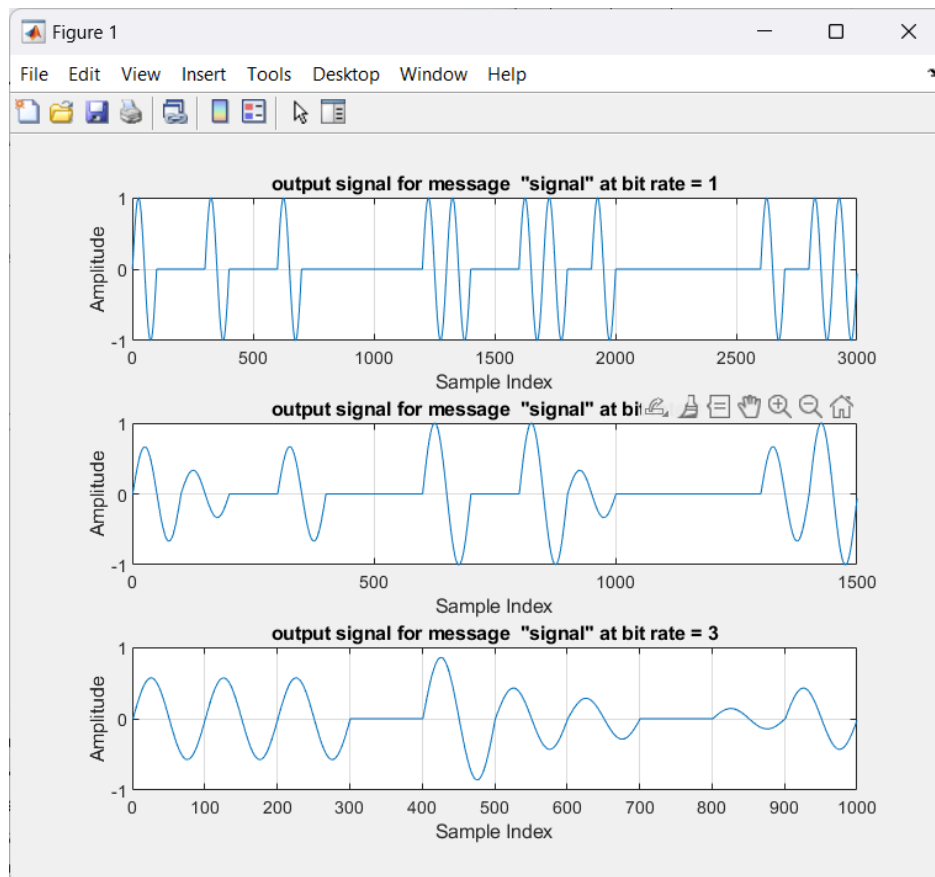
    subplot(3, 1, k);
    plot(output_signal);
    xlabel('Sample Index');
    ylabel('Amplitude');
    title(['output signal for message "',message,'" at bit rate = ', num2str(bitRate)]);
    grid on;

    decoded_message = decoding_amp(output_signal, bitRate);
    fprintf('Original message: "%s", Decoded message at bit rate %d: "%s"\n', ...
        message, bitRate, decoded_message);

end

```

تصویر 3 - اسکریپت برای پیاده سازی تابع *encoding_amp* با 3 بیت ریت مختلف



تصویر 4- خروجی پیاده سازی تابع *encoding_amp* با 3 بیت ریت مختلف

تمرین 4-1

این بخش شامل تابعی می شود که هدف اش decode کردن سیگنالی است که پیامی در آن طبق روش مرحله قبل encode شده است. در ابتدا string خالی ای (binary_coded_message) را تعیین می کنیم. در این string در ابتدا می خواهیم پیامی را به صورت باینری decode کرده را ذخیره کنیم تا سپس با استفاده از Mapset کاراکترمرتبط با هر 5 بیت باینری را پیدا کنیم. می دانیم که در مرحله encode به هر bitRate تا بیت، یک سیگنال سینوسی اختصاص دادیم که concat شده آنها را در سیگنال ورودی وجود دارد. در نتیجه بایستی این سیگنال ورودی را به segment هایی با طول $f_s * \text{bitRate}$ تقسیم کرده تا با بررسی دامنه هر segment، به مقدار encode شده در آن پی ببریم. برای به دست آوردن مقدار دامنه از روش correlation بین segment سینوسی با سیگنال سینوسی دیگری استفاده می کنیم. این کار را در تابعی به نام correlation_with_sin پیاده سازی کرده ایم. در این تابع انتگرال حاصل ضرب دو تابع سینوسی را به علت گسسته بودن مقادیر سیگنال ورودی با کمک تابع trapz متلب انجام دادیم. برای ساده تر شدن محاسبات در مراحل بعدی، حاصل انتگرال را با 2sint محاسبه کردیم. در تابع decoding از حاصل این انتگرال round می گیریم چرا که می خواهیم حاصل را به تابع dec2bin بدهیم که حتما ورودی صحیح می خواهد. در نهایت نیز مقدار decode شده را به binary_coded_message اضافه می کنیم. حال که توانستیم پیام را به صورت باینری decode کنیم، لازم است با استفاده از Mapset کاراکتر های مرتبط را بیابیم تا در خروجی پیام را به صورت text داشته باشیم. برای این کار بر روی binary_coded_message لوپ می زنیم و از سمت چپ 5 بیت را جدا کرده و آن را با بیت های کد شده بر روی Mapset مقایسه کرده و در صورت یافتن کدی مشابه، کاراکتر متناظر را به پیام امان اضافه می کنیم و بعد از لوپ داخلی خارج شده و به decode کردن 5 بیت بعدی می رویم.

Command Window

```
Original message: "signal", Decoded message at bit rate 1: "signal"  
Original message: "signal", Decoded message at bit rate 2: "signal"  
Original message: "signal", Decoded message at bit rate 3: "signal"
```

fx >>

تصویر 5 - خروجی پیاده سازی تابع decoding با 3 بیت ریت مختلف

```

function message = decoding_amp(received_signal, bitRate)
    f_s = 100;
    start_idx = 1;
    end_idx = f_s;

    binary_coded_message = '';
    num_signal = length(received_signal) / f_s;
    num_func = 2^bitRate;

    for n = 1:num_signal

        segment_values = received_signal(start_idx:end_idx);
        corr_value = correlation_with_sin(segment_values);

        decimal_code_5x = round(corr_value * (num_func - 1));
        decimal_code = decimal_code_5x/5;

        binary_code = dec2bin(decimal_code, bitRate);
        binary_coded_message = [binary_coded_message ,binary_code];

        start_idx = start_idx + f_s;
        end_idx = end_idx + f_s;

    end

    message = '';
    Mapset = generateMapset();
    bin_message_len = strlen(binary_coded_message);
    for p = 1:5:(bin_message_len)
        curr_letter = extractBetween(binary_coded_message,p,p+5-1);
        for q = 1:length(Mapset(1,:))
            if strcmp(curr_letter, Mapset{2, q})
                message = [message ,Mapset{1, q}];
                break;
            end
        end
    end
end
end

```

تصویر 6 - تابع *decoding_amp* برای استخراج کردن پیام از سیگنال ورودی با توجه به روش *encoding*

تمرین 5-1

در این بخش می خواهیم تابع *randn* را برای ایجاد نویز بررسی کنیم. می دانیم که این تابع به صورت رندوم مقادیری را می دهد که از توزیع نرمال با میانگین صفر و واریانس یک تبعیت می کنند و اساسا تابعی برای شبیه سازی توزیع گوسی است. همچنین ما می توانیم با دادن پارامترهایی به این تابع، میانگین و واریانس را نیز به صورت دلخواه تعیین کنیم. برای بررسی صحت اینکه مقادیر خروجی این تابع از توزیع نرمال با میانگین صفر و واریانس یک پیروی می کنند دو روش را انجام دادیم :

- 1) استفاده از توابع *mean* , *variance* : چون تعداد سمپل هایمان زیاد است می توانیم از توابع *mean* , *variance* متلب استفاده کرده و با خطای کمی می بینیم که خروجی با انتظارمان تطابق دارد.
- 2) رسم نمودار توزیع : با استفاده از دستور *histogram* می توانیم ببینیم که در هر بازه ای از مقادیر خروجی، تابع *randn* چه تعداد سمپل تولید کرده است و بدین صورت نمودار توزیع را رسم می کنیم. همچنین برای مقایسه اینکه این

توزیع با توزیع گوسی با میانگین صفر و واریانس یک تقریباً برابر است نمودار این توزیع را هم در کنارش رسم می کنیم.

```
function generateGaussianNoise(signal_lenght, noise_amp)
    noise_signal = randn(1, 1000);

    calculated_mean = mean(noise_signal);
    calculated_variance = var(noise_signal);

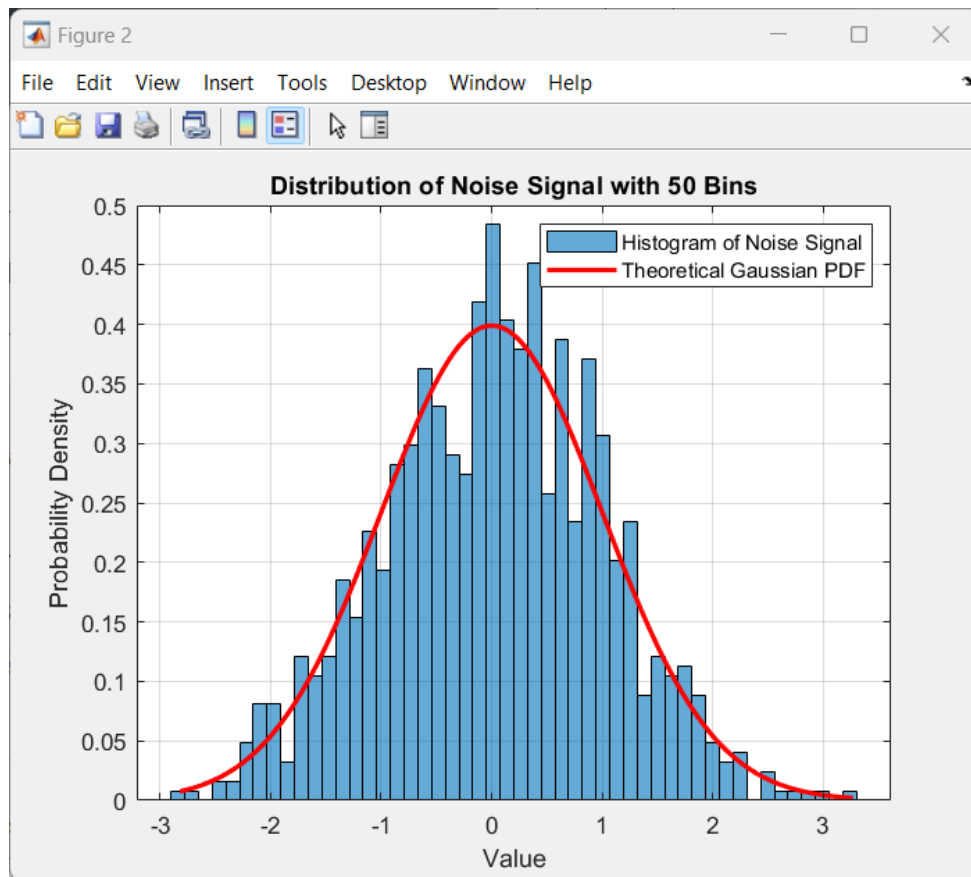
    fprintf('Calculated Mean: %.4f\n', calculated_mean);
    fprintf('Calculated Variance: %.4f\n', calculated_variance);

    num_bins = 50;
    figure;
    histogram(noise_signal, 'NumBins', num_bins, 'Normalization', 'pdf'); % Normalized to PDF
    hold on;

    x = linspace(min(noise_signal), max(noise_signal), 1000);
    gaussian_pdf = (1 / (sqrt(2 * pi) * noise_amp)) * exp(-(x - 0).^2 / (2 * noise_amp^2));
    plot(x, gaussian_pdf, 'r-', 'LineWidth', 2);

    xlabel('Value');
    ylabel('Probability Density');
    title(['Distribution of Noise Signal with ', num2str(num_bins), ' Bins']);
    legend('Histogram of Noise Signal', 'Theoretical Gaussian PDF');
    grid on;
    hold off;
end
```

تصویر 7 - تابعی برای رسم توزیع خروجی *randn* فیت شده با تابع گوسی متناظر



تصویر 8 - توزیع خروجی *randn* فیت شده با تابع گوسی متناظر

```
Calculated Mean: 0.0273  
Calculated Variance: 0.9907
```

```
fx >>
```

تصویر 9 - محاسبه واریانس و میانگین سمپل های خروجی از تابع *randn* برای ایجاد نویز به کمک تابع های متلب

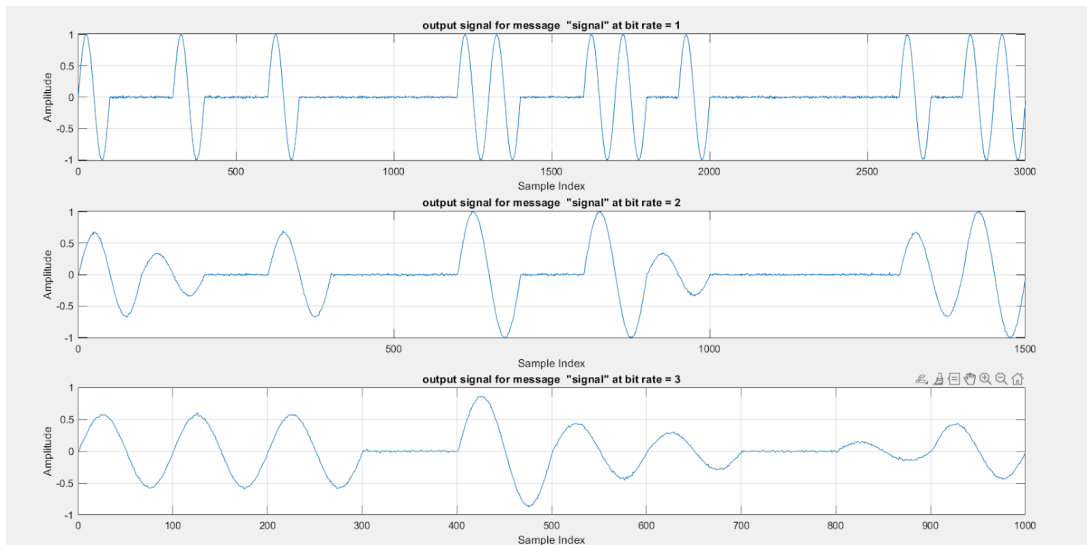
تمرین 6-1

```
31 % Loop for each bit rate and adding noise into the signal  
32 for k = 1:length(bitRates)  
33     bitRate = bitRates(k);  
34     output_signal = coding_amp_1(message, bitRate);  
35  
36     % Generate Gaussian noise  
37     noisePower = 0.0001; % Define noise power  
38     noise = sqrt(noisePower) * randn(1, length(output_signal)); % Gaussian noise  
39     noisy_signal = output_signal + noise; % Add noise to the signal  
40  
41     % Plot the signal  
42     t = (0:length(noisy_signal) - 1) / f_s;  
43     subplot(3, 1, k);  
44     plot(noisy_signal);  
45  
46     xlabel('Sample Index');  
47     ylabel('Amplitude');  
48     title(['output signal for message "', message, '" at bit rate = ', num2str(bitRate)]);  
49     grid on;  
50  
51     % Decode each noisy signal  
52     decoded_message = decoding_amp(noisy_signal, bitRate);  
53     disp(['Decoded message at bit rate ', num2str(bitRate), ': ', decoded_message]);  
54 end
```

تصویر 10- کد برای اضافه کردن نویز به سیگنال ها

توضیحات کد:

ابتدا بر روی بیت ریت ها لوپ میزنیم و یه نویز گوسی ایجاد میکنیم و آن را به سیگنالمان اضافه میکنیم. سپس پلات مربوط به سیگنال را رسم میکنیم و در ادامه با استفاده از تابع *decoding_amp* پیاممان را دیکود میکنیم و آن را در کامند نمایش میدهم.



تصویر 11- تصاویر سیگنال های نویز دار

```
100100100000110011010000001011
Decoded message at bit rate 1: signal
```

تصویر 12- دیکود برای سیگنال با بیت ریت 1

```
100100100000110011010000001011
Decoded message at bit rate 2: signal
```

تصویر 13- دیکود برای سیگنال با بیت ریت 2

```
100100100000110011010000001011
Decoded message at bit rate 3: signal
```

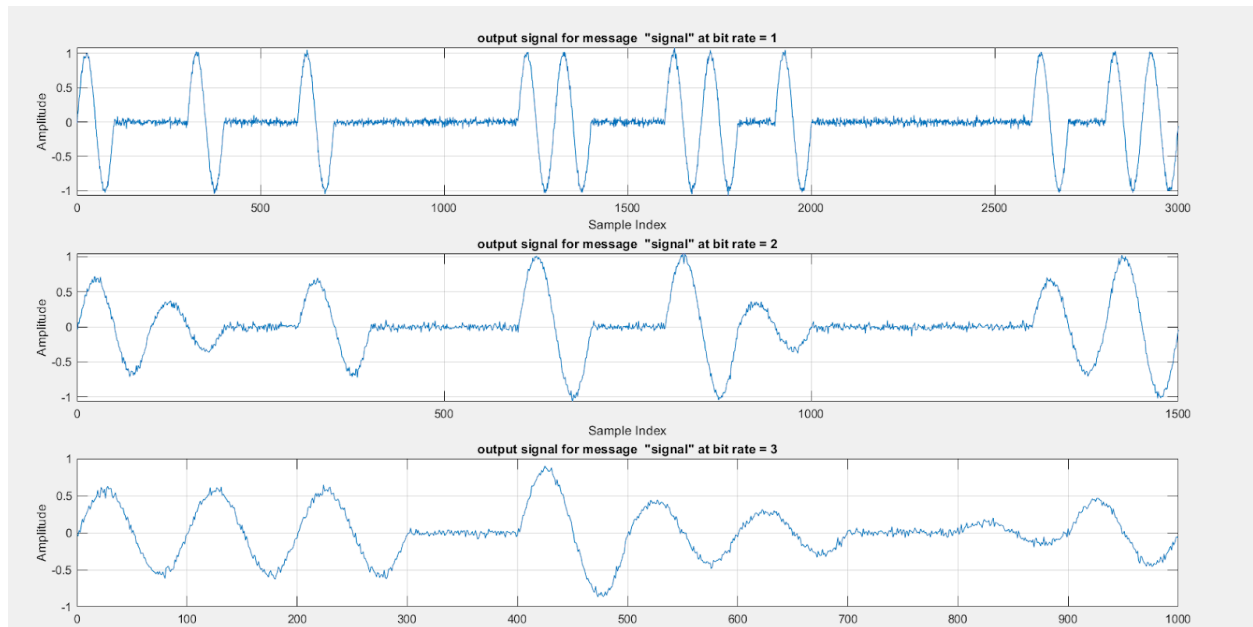
تصویر 14- دیکود برای سیگنال با بیت ریت 3

همان طور که در تصویر مشخص است با وجود اضافه شدن سیگنال نویز در فرایند دیکود خللی ایجاد نشد.

تمرین 7-1

برای تغییر قدرت نویز باید به متغیر noisePower را تغییر دهیم.

واریانس استاندارد = 0.001



تصویر 15- تصاویر سیگنال های نویز دار

```
100100100000110011010000001011
Decoded message at bit rate 1: signal
```

تصویر 16- دیکود برای سیگنال با بیت ریت 1

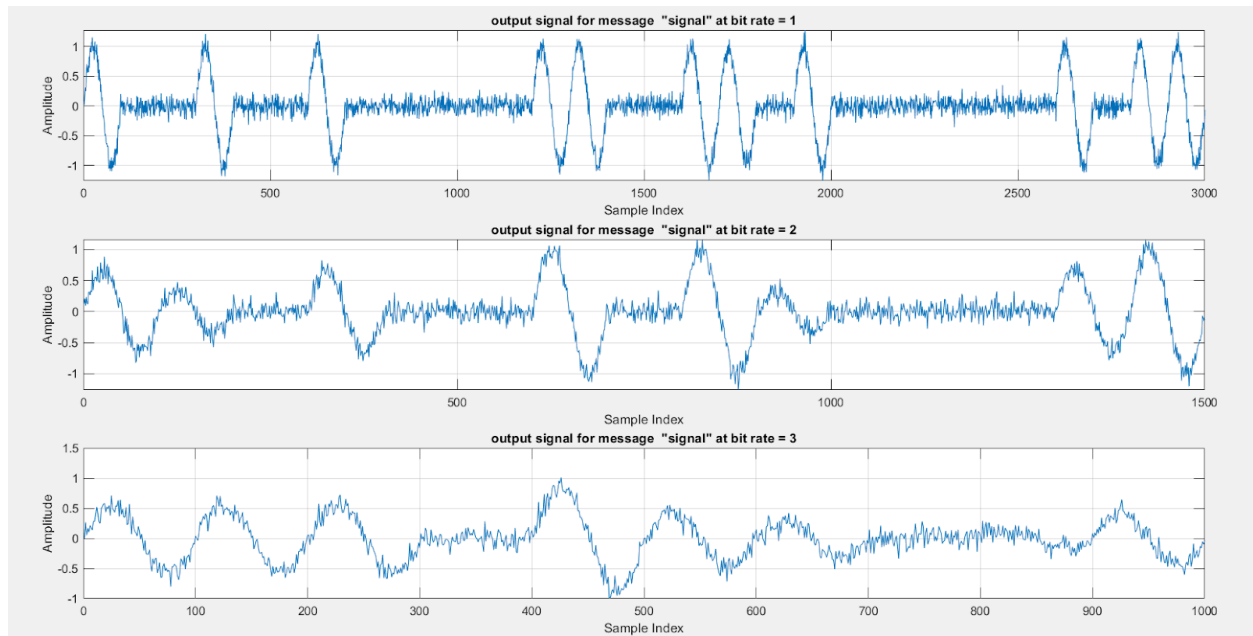
```
100100100000110011010000001011
Decoded message at bit rate 2: signal
```

تصویر 17- دیکود برای سیگنال با بیت ریت 2

```
100100100000110011010000001011
Decoded message at bit rate 3: signal
```

تصویر 18- دیکود برای سیگنال با بیت ریت 3

واریانس استاندارد = 0.01



تصویر 19- تصاویر سیگنال های نویز دار

```
100100100000110011010000001011
Decoded message at bit rate 1: signal
```

تصویر 20- دیکود برای سیگنال با بیت ریت 1

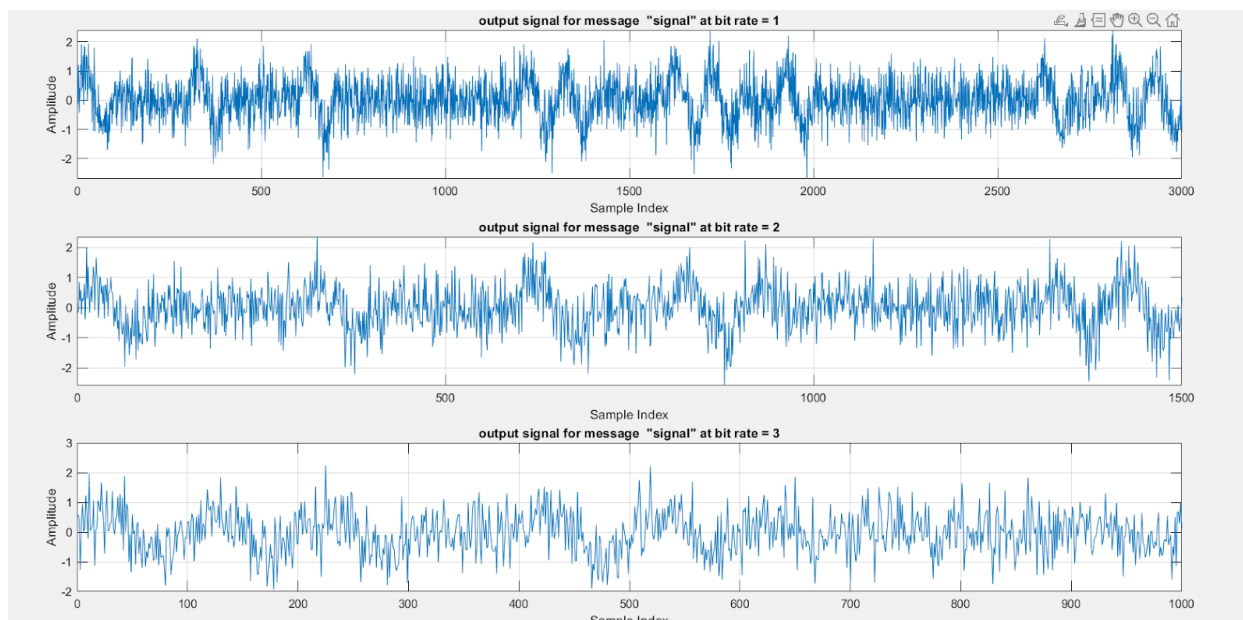
```
100100100000110011010000001011
Decoded message at bit rate 2: signal
```

تصویر 21- دیکود برای سیگنال با بیت ریت 2

```
100100100000110011010000001011
Decoded message at bit rate 3: signal
```

تصویر 22- دیکود برای سیگنال با بیت ریت 3

واریانس استاندارد = 0.4



تصویر 23- تصاویر سیگنال های نویز دار

```
100100100000110011010000001011
Decoded message at bit rate 1: signal
```

تصویر 24- دیکود برای سیگنال با بیت ریت 1

```
100000100000110011010000001011
Decoded message at bit rate 2: qignal
```

تصویر 25- دیکود برای سیگنال با بیت ریت 2

```
100100100000101100010001000010
Decoded message at bit rate 3: sifrrc
```

تصویر 26- دیکود برای سیگنال با بیت ریت 3

همان طور که در تصویر مشخص است به دلیل بالا بودن نویز ما با بیت ریت های 2 و 3 نتوانستیم به درستی سیگنال را دیکود کنیم.

تمرین 8-1

این داده ها بر اساس آزمایش گزارش شده اند و امکان خطا داخل آن ها وجود دارد.

```

9      tolerance = 0;

61
62      % Loop over each bit rate to find maximum noise power
63      for k = 1:length(bitRates)
64          bitRate = bitRates(k);
65          output_signal = coding_amp_1(message, bitRate); % Generate the clean output signal
66          noisePower = 0.01; % Start with zero noise
67
68          while true
69              % Generate Gaussian noise with the current noise power
70              noise = sqrt(noisePower) * randn(1, length(output_signal));
71              noisy_signal = output_signal + noise; % Add noise to the signal
72
73              try
74                  decoded_message = decoding_amp(noisy_signal, bitRate);
75                  % Check if decoding was successful or acceptable
76                  if strcmp(decoded_message, message) || acceptable_error(decoded_message, message, tolerance)
77                      noisePower = noisePower + 0.0001; % Continue to increase noise
78                  else
79                      max_noise_power(k) = noisePower - 0.0001;
80                      break;
81                  end
82              catch ME
83                  disp(['Decoding failed due to: ', ME.message]);
84                  max_noise_power(k) = noisePower - 0.0001;
85                  break;
86              end
87          end
88          disp(['Max noise power for bit rate ', num2str(bitRate), ': ', num2str(max_noise_power(k))]);
89          pause(10);
90      end

92      % Define acceptable error function if tolerance is used
93      function success = acceptable_error(decoded, original, tolerance)
94          % Count character differences
95          errors = sum(decoded ~= original);
96          success = errors <= tolerance;
97      end

```

تصویر 27- تابعی برای یافتن max_noise_power برای هر بیت ریت

توضیحات کد:

متغیر $tolerance$ ، برای اینکه آیا پیام دیکود شده به طور کامل تطابق دارد یا خیر. قسمت جدید این تیکه کد اضافه شدن قسمت `try and catch` می باشد. که در آن اگر پیام دیکود شده با پیام اصلی یکی باشد قدرت نویز را افزایش می دهیم در غیر این صورت پیغام مناسب را چاپ کرده و پیش از رفتن سراغ بیت ریت بعدی 10 ثانیه صبر میکنیم. تابع `success` برای این تعریف شده است که اگر خواستیم با تقریب خاصی پیام دیکود رو مقایسه کنیم، کارمان را ادامه دهیم.

Max noise power for bit rate 1: 0.8047

Max noise power for bit rate 2: 0.1023

Max noise power for bit rate 3: 0.0304

تمرین 9-1

همان طور که در صورت پروژه توضیح داده شد، با داشتن قدرت بیشتر در فرستنده ما دامنه سیگنال بیشتری داریم و فاصله بین threshold های برای تصمیم گیری بیشتر میشود و حساسیت کمتری نسبت به نویز پیدا میشود.

تمرین 10-1

اگر که حتی نویز هم نداشته باشیم، به علت نزدیک بودن مقادیر اعشاری حاصل از correlation به هم و خطای محاسبات، با محدودیت فرستادن سیگنال تا مقدار بیت ریت معین مواجه هستیم. برای بررسی این پدیده، پیامی را در سیگنالی با بیت ریت های مختلف encode می کنیم و بعد پیام decode شده را با پیام encode شده مقایسه می کنیم. باید توجه داشته باشیم که پیامی به طول n می تواند bitrate هایی که مقسوم علیه طبیعی $n*5$ را فقط، طبق صورت پروژه، داشته باشد. برای یافتن bitrate هایی با این ویژگی تابعی به نام findDivisors را نوشته این که تابع با لوپ زدن بر روی تمام اعداد طبیعی کمتر از $n*5$ ، مقسوم علیه های طبیعی آن را با بررسی صفر بودن باقی مانده $n*5$ بر هر یک از آنها می یابد. در ادامه برای یافتن max_bitRate ای که خروجی درستی را بدهد، از تعداد بیت ریت کم شروع کرده و در هر مرحله تعداد بیت ریت را افزایش می دهیم و اگر که پیام ها با هم همخوانی داشتند که آن بیت ریت را به عنوان max_bitRate قرار می دهیم. وگرنه، اولین بیت ریتی که پیام decode شده و encode شده اش تفاوت داشت سبب خروج از لوپ می شود. و در نهایت نیز max_bitRate را چاپ می کنیم.

```
clc;
close all
clear

message = 'signal';

len_binary_coded_message = length(message) * 5;
acceptable_bitRates = findDivisors(len_binary_coded_message);

maxBitRate = 0;

for k = 1:length(acceptable_bitRates)
    bitRate = acceptable_bitRates(k);
    output_signal = coding_amp_1(message,bitRate);
    decoded_message = decoding_amp(output_signal,bitRate);
    if strcmp(decoded_message, message)

        maxBitRate = bitRate;
    else
        break;
    end
end

fprintf('The max limit of bitRate for deociding correctly is %f\n', maxBitRate);
```

تصویر 28- محاسبه واریانس و میانگین سمپل های خروجی از تابع randn برای ایجاد نویز به کمک تابع های متلب

Command Window

```
The max limit of bitRate for decoding correctly is 10.000000  
fx >>
```

تصویر 29 - خروجی *max_bitRate* برای پیام *signal*

تمرین 1-11

خیر. با افزایش 5 برابری ضریب در محاسبه *correlation* مقاومتی در برابر نویز ایجاد نخواهد شد، چرا که ما در نهایت بایستی حاصل *correlation* را به 5 تقسیم کنیم تا به مقادیر واقعی بین 0، 1 ای که در دامنه سیگنال (توسط منبع سیگنال این محدودیت برقرار شده) وجود دارد برسیم. و در این نوع مدل سازی تنها راه برای افزایش مقاومت در برابر نویز، کاهش بیت ریت یا افزایش دامنه سیگنال توسط منبع است.

```
function integral_value = correlation_with_sin(recieverd_signal)  
    f_s = 100;  
  
    t = (0:length(recieverd_signal) - 1)/f_s;  
  
    product_values = recieverd_signal .* (5*2*sin(2 * pi * t));  
  
    integral_value = trapz(t,product_values);  
  
end
```

تصویر 30 - محاسبه *correlation* بین سگمنتی از سیگنال *encode* شده و سیگنال معیار سینوسی با دامنه 5 برابر

```

function message = decoding_amp(received_signal, bitRate)
    f_s = 100;
    start_idx = 1;
    end_idx = f_s;

    binary_coded_message = '';
    num_signal = length(received_signal) / f_s;
    num_func = 2^bitRate;

    for n = 1:num_signal

        segment_values = received_signal(start_idx:end_idx);
        corr_value = correlation_with_sin(segment_values);

        decimal_code_5x = round(corr_value * (num_func - 1));
        decimal_code = decimal_code_5x/5;

        binary_code = dec2bin(decimal_code, bitRate);
        binary_coded_message = [binary_coded_message ,binary_code];

        start_idx = start_idx + f_s;
        end_idx = end_idx + f_s;

    end

```

تصویر 31 - تقسیم کردن خروجی *correlation* به 5 و ایجاد دامنه ای مطابق با توان منبع تامین کننده سیگنال *encode* شده

```

Command Window

The max limit of bitRate for deociding correctly is 10.000000
fx >>

```


تصویر 11 - خروجی ثابت *max_bitRate*

تمرین 12-1

سرعت اینترنت خانگی برابر با 10 تا 15 مگابیت بر سکند می باشد.

 سرعت آپلود
 0 Kbps

 سرعت دانلود
 9.16 Mbps

 زمان پاسخ
 11 ms
 Jitter: 30ms