# گزارش پروژه چهارم درس سیگنال سیستم‌ها

## تمرین 1_1)

در این بخش یک mapset برای Map کردن کاراکترها به رشته‌های باینری تعریف می‌کنیم. برای این کار از یک cell استفاده می‌کنیم که سطر اول آن را حروف و سطر دوم را رسته‌های باینری map شده تشکیل می‌دهند.

کد:

```
%% CREATE MAPSET AND LOAD IT
create_mapset();
load('mapset.mat')
mapset_len = length(mapset{2, 1});
```

```
function create_mapset()
Nch=32;
mapset=cell(2,Nch);
Alphabet ='abcdefghijklmnopqrstuvwxyz .,!";';
for i=1:Nch
    mapset{1,i}=Alphabet(i);
    mapset{2,i}=dec2bin(i-1,5);
end
save mapset.mat mapset;
end
```

پس از این کار، mapset به شکل روبه‌رو خواهد بود:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p |
| 2 | 00000 | 00001 | 00010 | 00011 | 00100 | 00101 | 00110 | 00111 | 01000 | 01001 | 01010 | 01011 | 01100 | 01101 | 01110 | 01111 |
| 3 | | | | | | | | | | | | | | | | |

## تمرین 1_2)

حال تابع coding_amp را می‌نویسیم تا پیام مورد نظر را کدگذاری کنیم.

این تابع پیام مورد نظر برای کدگذاری و bitrate را ورودی می‌گیرد. فرکانس نمونه‌برداری مطابق صورت سوال، 100 در نظر گرفته شده است.

در مرحله بعد با توجه به اینکه bitrate چند است، پیام به بخش‌هایی به طول bitrate تقسیم بندی می‌شود.

باید ضرایب هر بیت از bitrate یک عدد بین صفر و یک به خود بگیرند که مقادیر مربوط به آنها در coeffs مشخص شده است. (مثلا برای bitrate = 2، ضرایب `[0, 0.333, 0.667, 1]` خواهند بود.)

با توجه به صورت پروژه، برای ارسال 00، $x_0(t) = 0$، برای 01 $x_1(t) = \frac{1}{3}sin(2\pi t)$، برای 10، $x_2(t) = \frac{2}{3}sin(2\pi t)$ و برای 11، $x_3(t) = sin(2\pi t)$ ارسال می‌شود. برای پیدا کردن ضریب مورد نظر، مقدار آن بخش از سیگنال که در حال بررسی آن هستیم پیدا می‌کنیم تا ببینیم کدام یک از چهار حالت بالاست. سپس آن را یکی به

بالا شیفت می‌کنیم(چون ایندکس‌ها در متلب از 1 شروع می‌شوند). از ایندکس یافت شده برای پیدا کردن ضریب استفاده می‌کنیم.

برای بررسی پارت بعدی سیگنال، نقطه شروع و پایان را یکی جلو می‌بریم.

در نهایت تمامی بخش‌ها با هم مرج می‌شوند تا یک سیگنال خطی را به عنوان خروجی بسازند.

کد:

```
function signal = coding_amp(bin_msg, bitrate)
    fs = 100;
    bin_split = reshape(bin_msg, bitrate, []);

    coeffs = linspace(0, 1, pow2(bitrate));
    t = 0:1/fs:1 - 1/fs;
    num_segments = size(bin_split, 1);
    signal_parts = zeros(num_segments, fs);

    for i = 1:num_segments
        bin_value = bin_split(i, :);
        coeff = coeffs(bin2dec(bin_value) + 1);
        signal_parts(i, :) = coeff * sin(2 * pi * t);
    end

    signal = reshape(signal_parts', 1, []);
end
```

**تمرین 1_3)**

حال با کمک تابع coding_amp که در قسمت قبل تعریف کردیم، با bitrateهای مختلف پیام را ارسال می‌کنیم و پیام کدگذاری شده را رسم می‌کنیم.

کد:

```
%% SEND SIGNAL WITH BITRATE 1, 2, 3
fs = 100;
str = 'signal';
bin = str2bin(str, mapset);
for bitrate = 1:3
    x = coding_amp(bin, bitrate);
    t = 0:(1 / fs):(length(str) * mapset_len / bitrate - 1 / fs);

    figure;
    plot(t, x)
    title(['Bitrate = ', num2str(bitrate)])
end
```

خروجی:

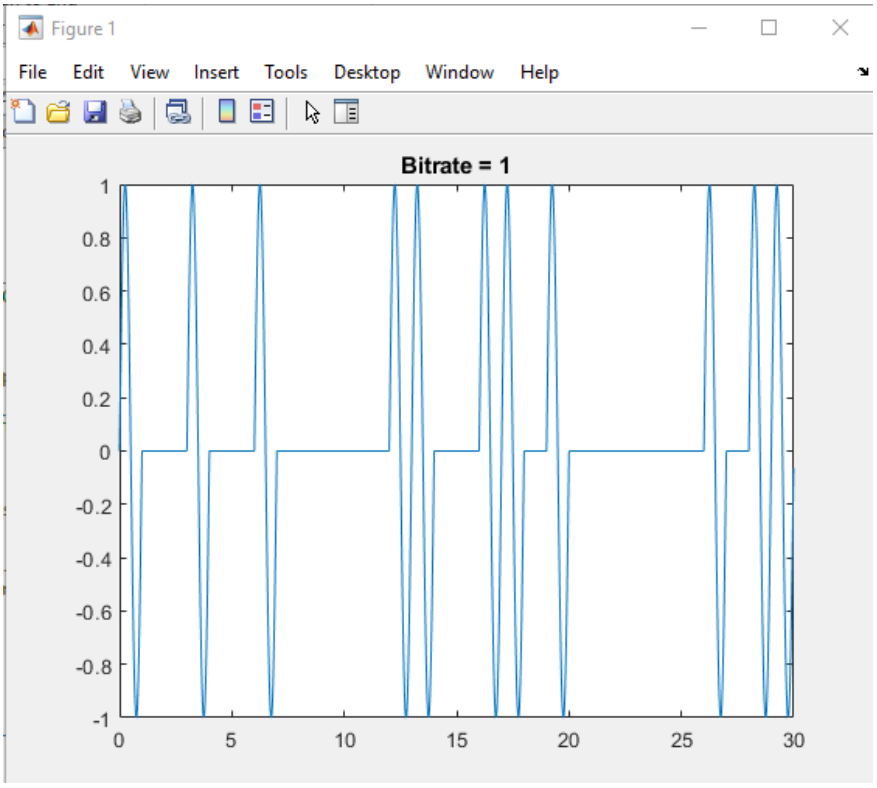


Figure 1
File Edit View Insert Tools Desktop Window Help
Bitrate = 1
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0 5 10 15 20 25 30

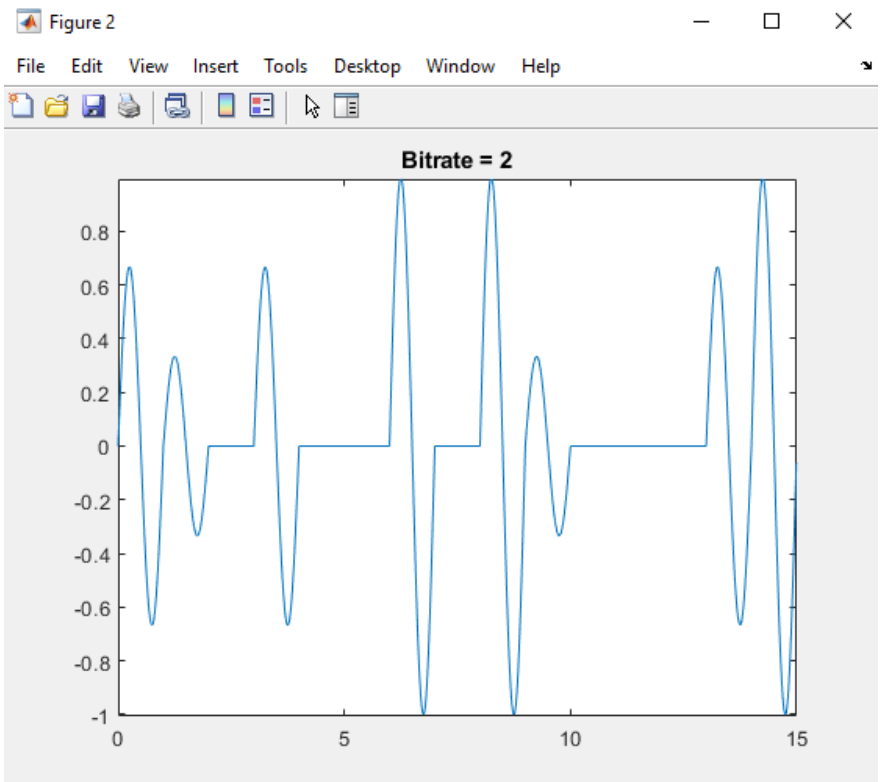
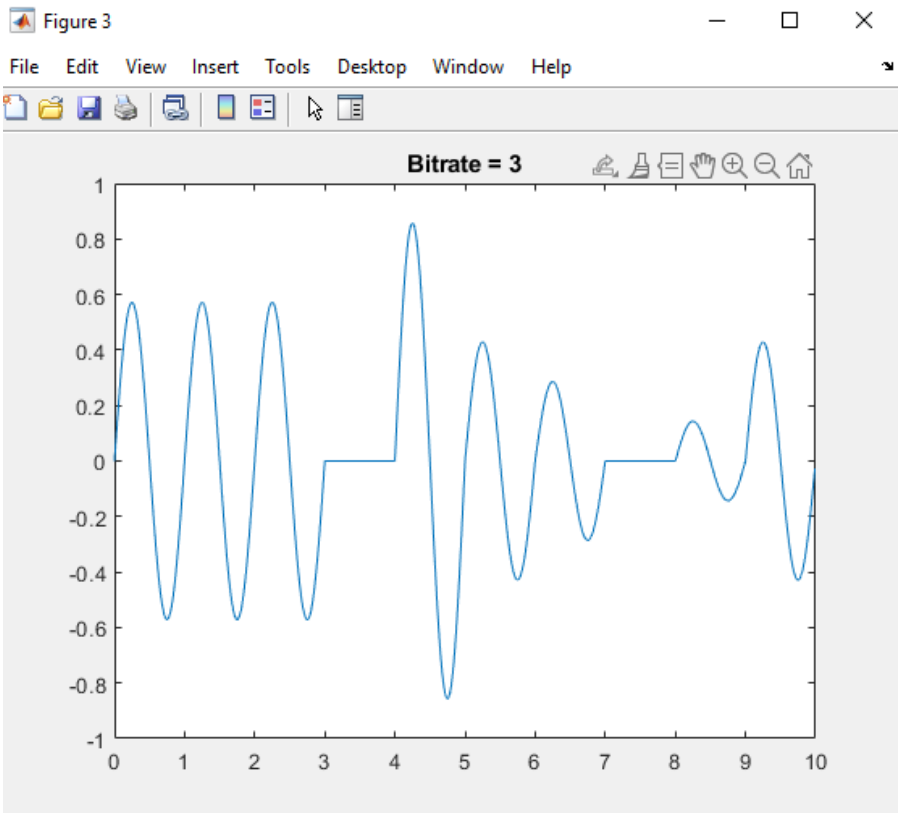


Figure 2
File Edit View Insert Tools Desktop Window Help
Bitrate = 2
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0 5 10 15




**تمرین 1_4)**

این تابع سیگنال کدگذاری شده به همراه bitrate ای که سیگنال بر اساس آن کد شده است را ورودی می‌گیرد و رشته باینری پیام را بر می‌گرداند.

با توجه به فرکانس نمونه‌برداری، تعداد بخش‌های سیگنال را که می‌خواهیم ذخیره‌سازی کنیم محاسبه می‌کنیم و یک آرایه برای سیگنال decode شده initialize می‌کنیم.

سپس بازه زمانی مورد نظر را با توجه به sampling frequency مشخص کرده و signal را با توجه به تعداد بخش‌های مورد نظر تقسیم‌بندی می‌کنیم.

حال روی هر بخش از signal، اول تمامی مقادیر بزرگ از 1 را به 1 و تمامی مقادیر کوچک‌تر از 1- را به 1- map می‌کنیم تا اثر داده‌های پرت حذف شود، بعد correlation را با سیگنال $y = 2sin(2\pi t)$ حساب می‌کنیم. برای این کار از تابع trapz استفاده کردیم و نهایتا برای Normalize کردن بر fs تقسیم کرده‌ایم.

از آنجایی که decoding در اینجا بر اساس مقدار است و علامت اهمیتی ندارد، از قدرمطلقش استفاده می‌کنیم.

با توجه به اینکه bitrate چند است، مقدار correlation را اسکیل می‌کنیم و به باینری تبدیل می‌کنیم و در رشته پیام قرار می‌دهیم، سپس سراغ بخش بعدی سیگنال می‌رویم.

کد:

```
function binary = decoding_amp(signal, bitrate)
    fs = 100;
    parts_count = length(signal) / fs;
    binary = blanks(parts_count * bitrate);

    signal_parts = reshape(signal, fs, parts_count);

    t = 0:1/fs:1 - 1/fs;
    reference_signal = 2 * sin(2 * pi * t);

    for i = 1:parts_count
        current_part = signal_parts(i, :);

        current_part = min(max(current_part, -1), 1);

        corr_integral = trapz(current_part .* reference_signal) / fs;
        corr_integral = abs(corr_integral);

        closest = round(corr_integral * (pow2(bitrate) - 1));
        binary_value = dec2bin(closest, bitrate);

        binary(bitrate * (i - 1) + 1:bitrate * i) = binary_value;
    end
end
```

برای تست کردن عملکرد decoding_amp با توجه به bitrateهای مختلف، یک تابع test_without_noise می‌نویسیم که پیام را با توجه به bitrate مورد نظر code می‌کند و آن را به decoding_amp می‌دهد و نتیجه را چاپ می‌کند.

کد:

```
function test_without_noise(str, bitrates, mapset)
    bin_send = str2bin(str, mapset);

    num_bitrates = length(bitrates);
    result = cell(num_bitrates, 1);

    for idx = 1:num_bitrates
        bitrate = bitrates(idx);

        signal_send = coding_amp(bin_send, bitrate);
        bin_receive = decoding_amp(signal_send, bitrate);

        str_receive = bin2str(bin_receive, mapset);

        result{idx} = sprintf('Received (bitrate=%d) Received message: %s', bitrate, str_receive);
    end

    for i = 1:num_bitrates
        disp(result{i});
    end
end
```

کد:

```
%% DECODE SIGNAL WITH BITRATE 1, 2, 3
str = 'signal';
bitrates = 1:3;
test_without_noise(str, bitrates, mapset);
```

خروجی:

```
Recieved (bitrate=1) Recieved message: signal
Recieved (bitrate=2) Recieved message: signal
Recieved (bitrate=3) Recieved message: signal
```

**تمرین 1_5)**

در این قسمت می‌خواهیم مطمئن شویم نویز ساخته شده توسط randn، ویژگی‌های یک نویز گاوسی با میانگین صفر و واریانس یک را داراست.

برای این منظور تابع check_properties را می‌نویسیم که برای بررسی گاوسی بودن، هیستوگرام مربوط به تابع با توزیع نرمال و میانگین صفر و واریانس یک را رسم می‌کند و با Noise ورودی مقایسه می‌کند، سپس میانگین و واریانس نویز را چاپ می‌کند.

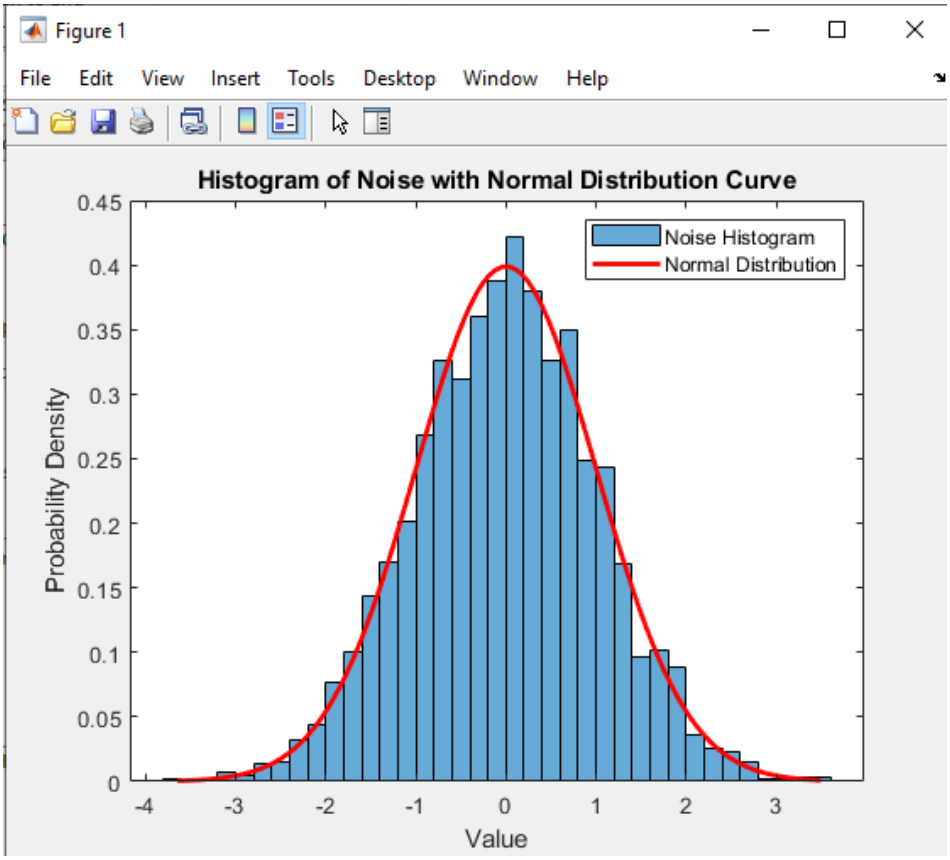
کد:

```
function check_properties(noise)

figure;
histogram(noise, 'Normalization', 'pdf');
hold on;
x = linspace(min(noise), max(noise), 100);
y = normpdf(x, 0, 1);
plot(x, y, 'r', 'LineWidth', 2);
title('Histogram of Noise with Normal Distribution Curve');
xlabel('Value');
ylabel('Probability Density');
legend('Noise Histogram', 'Normal Distribution');
hold off;

mean_noise = mean(noise);
fprintf('Mean of the noise: %.4f\n', mean_noise);

variance_noise = var(noise);
fprintf('Variance of the noise: %.4f\n', variance_noise);
end
```

کد:

```
%% CHECK NOISE PROPERTIES
noise = randn(1, 3000);
check_properties(noise)
```

خروجی:




```
Mean of the noise: 0.0008
Variance of the noise: 1.0304
```

همانطور که می‌بینیم، نویز ویژگی‌های یک توزیع گاوسی را داراست.

## بخش 1-6)

در این بخش یک تابع مخصوص ایجاد می کنیم چون تنها می خواهیم چک کنیم که آیا پیام استخراج شده با پیام فرستاده شده برابر است یا نه

```
%% TEST CODING AND DECODING FOR A SIGNAL WITH NOISE 0.0001
send_message = 'signal';
bitrates = 1:3;
noise = 0.01;
disp("With noise 0.0001");
p1_6(send_message, bitrates, noise, mapset);
```

برای این کار تابع p1_6 را تعریف می کنیم در این جا به سیگنال ارسالی به اندازه خواسته شده نویز اضافه می کنیم با توجه به اینکه با ضرب عدد n مقدار واریانس نهایی $n^2$ برابر می شود پس برای رسیدن به واریانس نویز 0.0001 کافیست تابع randn را در عدد 0.01 ضرب کنیم، چون نمی خواهیم میانگین تغییر کند عدد نویز به دست آمده را تغییری نمی دهیم چون همانطور که در قسمت قبل مشاهده کردیم میانگین به صورت پیش فرض برروی عدد 0 قرار گرفته است.

```
function p1_6(str, bitrates, noise, mapset)
bin_send = str2bin(str, mapset);
for i = 1:length(bitrates)
    bitrate = bitrates(i);
    signal_send = coding_amp(bin_send, bitrate);
    noise_signal=noise* randn(size(signal_send));
    signal_receive = signal_send +noise_signal;
    bin_message = decoding_amp(signal_receive, bitrate);
    message_receive = bin2str(bin_message, mapset);
    fprintf("the bitrate is %d with noise is resulted: %s\n",bitrate,message_receive)
end
end
```

همانطور که در شکل بالا قابل مشاهده است ما پیام signal را دریافت می کنیم و آن را code می کنیم در قدم بعدی به این سیگنال نویز اضافه کرده و آن را تحت عنوان signal دریافتی تعریف می کنیم

حال با استفاده از تابع decode پیام را استخراج کرده و به ازای هر bitrate آن را چاپ می کنیم :

```
the bitrate is 1 with noise is resulted: signal
the bitrate is 2 with noise is resulted: signal
the bitrate is 3 with noise is resulted: signal
```

همانطور که مشخص است در هر 3 تا bitrate مدنظر پیام به درستی رمزگشایی شده است و این مسئله نشان می دهد که این میزان از نویز نمی تواند در خروجی ما تاثیر چندانی بگذارد و نتایج همچنان درست هستند

**تمرین 1_7)**

حال می‌خواهیم مقاومت را نسبت به مقادیر مختلف نویز بسنجیم.

کد:

```
function test_with_noise(str, bitrates, noise, mapset)
    bin_send = str2bin(str, mapset);

    num_bitrates = length(bitrates);
    result = cell(num_bitrates, 1);

    for idx = 1:num_bitrates
        bitrate = bitrates(idx);

        signal_send = coding_amp(bin_send, bitrate);

        noisy_signal = signal_send + noise * randn(size(signal_send));

        bin_receive = decoding_amp(noisy_signal, bitrate);

        str_receive = bin2str(bin_receive, mapset);

        result{idx} = sprintf('Received (bitrate=%d, noise=%.2f): %s', bitrate, noise, str_receive);
    end

    for i = 1:num_bitrates
        disp(result{i});
    end
end
```

کد:

```
%% TEST CODING AND DECODING FOR MULTIPLE VALUES OF NOISE
str = 'signal';
bitrates = 1:3;

noise = 0.1;
test_with_noise(str, bitrates, noise, mapset);
fprintf('\n')

noise = 0.4;
test_with_noise(str, bitrates, noise, mapset);
fprintf('\n')

noise = 0.7;
test_with_noise(str, bitrates, noise, mapset);
fprintf('\n')

noise = 1;
test_with_noise(str, bitrates, noise, mapset);
fprintf('\n')

noise = 1.2;
test_with_noise(str, bitrates, noise, mapset);
fprintf('\n')
```

خروجی:

```
Received (bitrate=1, noise=0.10): signal
Received (bitrate=2, noise=0.10): signal
Received (bitrate=3, noise=0.10): signal

Received (bitrate=1, noise=0.40): signal
Received (bitrate=2, noise=0.40): signal
Received (bitrate=3, noise=0.40): sgfnad

Received (bitrate=1, noise=0.70): signal
Received (bitrate=2, noise=0.70): seejah
Received (bitrate=3, noise=0.70): nwfncl

Received (bitrate=1, noise=1.00): sicnal
Received (bitrate=2, noise=1.00): kieiag
Received (bitrate=3, noise=1.00): ogffck

Received (bitrate=1, noise=1.20): qagnaj
Received (bitrate=2, noise=1.20): keefag
Received (bitrate=3, noise=1.20): iuliab
```

همانطور که می‌بینیم، BITRATE = 1، نسبت به نویز از همه مقاوم تر بود و تا نویز 0.7 نتایج درستی ارائه می‌دهد. از طرفی این نتایج با مقدمه نیز همخوانی دارد، چون با افزایش bitrate مقاومت نسبت به Noise کاهش یافته است.

## بخش 1-8)

برای این بخش نیاز است تا تابع دیگری تعریف کنیم تا بتوانیم بیشترین واریانسی که له آن مقاوم هستیم را پیدا کنیم ( لازم به ذکر است که دلیل تعریف یک تابع برای این بخش این است که نویز یک ماهیت تصادفی دارد پس نیاز است تا ما چندین بار این حلقه را به ازای نویز های مختلف اجرا کنیم تا اولین نقطه ای که پیام درست decode نمی شود را پیدا کنیم )

```
function val_threshold = var_threshold(send_message, bitrate, mapset)
    bin_send = str2bin(send_message, mapset);
    signal_send = coding_amp(bin_send, bitrate);
    val_threshold = 2;
    number_step = 0.02;
    for noise = number_step:number_step:2
        for i = 1:50
            signal_noise= noise * randn(size(signal_send));
            signal_receive = signal_send + signal_noise;
            bin_message = decoding_amp(signal_receive, bitrate);
            message_receive = bin2str(bin_message, mapset);
            if ~strcmp(send_message, message_receive)
                val_threshold = noise - number_step;
                return
            end
        end
    end
end
```

در اینجا تابع ما به شکل زیر عمل می کند:

ابتدا برحسب bitrate داده شده یک سیگنال ایجاد می کنیم در طی این فرآیند سیگنال ارسالی ثابت می ماند تنها سیگنال دریافتی دستخوش تغییرات می شود

حالا در قدم بعدی از 0.02 شروع می کنیم و تا عدد 2 با گام های 0.02 جلو می رویم پس به طور کلی 100 مقدار را برای نویز در این بازه امتحان می کنیم، همانطور که گفتیم نویز ماهیت تصادفی داد پس لازم است به ازای هر قدرت نویز یک حلقه 50 تایی را اجرا کنیم

در این حلقه ما مقدار تصادفی را برحسب قدرت اعلام شده ایجاد می کنیم و سپس به سیگنال ورودی اضافه می کنیم پس از این کار این سیگنال decode می شود اگر پیام خروجی با پیام اصلی برابر بود پس نسبت به این مقدار تصادفی نویز مقاوم بوده ایم و حلقه را ادامه می دهیم اگر این حلقه 50 تایی تمام شود یعنی نسبت به این قدرت نویز مقاوم بوده ایم پس به سراغ قدرت بعدی برای چک کردن می رویم.

حال اگر مقدار استخراج شده با مقدار اصلی برابر یعنی قدرت نویز قبلی اخرین قدرت نویزی بود که نسبت به آن مقاوم بوده ایم پس این قدرت را برمی گردانیم و در تابع اصلی آن را چاپ می کنیم:

```
%%    Thereshold For Variance
send_message = 'signal';
for bitrate = 1:3
    var_hold = var_threshold(send_message, bitrate, mapset);
    fprintf("the threshold is %.4f for bitrate %d\n",var_hold,bitrate)
end
```

```
Noise threshold (bitrate=1): 0.66
Noise threshold (bitrate=2): 0.3
Noise threshold (bitrate=3): 0.12
```

این نتیجه انحراف معیار را گزارش می کند و واریانس نهایی برابر است با

Bit rate=1 the variance is 0.660

Bit rate=2 the variance is 0.300

Bit rate=3 the variance is 0.120

## بخش 1-9)

افزایش bit rate باعث می شود که فرکانس های انتخابی ما بسیار به یکدیگر نزدیک شوند همانند انچه داشتیم برابر 1/3 و 2/3 شود اگر میزان نویز افزایش یابد ما فرکانس را اشتباه تشخیص می دهیم چرا که با اضافه شدن نویز ممکن است مقدار 1/3 به 2/3 رسیده و مقدار اشتباهی تشخیص داده شود .

در صورتی که بتوانیم فرکانس های نمونه برداری را افزایش دهیم در این صورت محدوده مورد نظر برای هر مقدار باینری افزایش می یابد و این فرکانس ها از هم دورتر می شوند لذا خطای تشخیص اشتباه فرکانس با اضافه شدن نویز در این سیستم کمتر شده و در حقیقت ما با دورتر کردن فرکانس ها از هم، پهنای باند را افزایش داده ایم که دقت را افزایش داده است..

## بخش 1-10)

در حیطه تئوری می توان اعلام کرد که تا بی نهایت می توان این bit rate را افزایش داد اما در حیطه عملی با مشکلاتی مواجه هستیم

1- پهنای باند (Bandwidth):

در سیستم ها ماکسیمم bit rate که در یک کانال می توان انتقال داد با توجه به Nyquist theorem تعیین می شود که عبارت است از :

$$R = 2 * B * \log_2 M$$

در این مثال B همان bandwidth است و M تعداد بیت هایی است که برای encode درنظر می گیریم و در نهایت R نیز bit rate برای یک سیستم بدون نویز است.

کانال‌های فیزیکی پهنای باند محدودی دارند. حتی بدون نویز، نمی‌توانیم مقدار نامحدودی از داده‌ها را در یک محدوده فرکانسی محدود جای دهیم (قضیه شانون-هارتلی حتی اگر عبارت نویز صفر باشد، همچنان اعمال می‌شود). نرخ‌های بیت بالاتر به پهنای باند بالاتر نیاز دارند.

2- تداخل بین نمادها (Inter-symbol interference یا ISI):

حتی بدون نویز،پالس‌های نشان‌دهنده بیت‌ها اگر خیلی به هم نزدیک بسته‌بندی شوند، می‌توانند همپوشانی داشته باشند. این تداخل بین نمادها، سیگنال دریافتی را تحریف می‌کند و نرخ بیت قابل دستیابی را محدود می‌کند. تکنیک‌های برابرسازی پیچیده می‌توانند ISI را کاهش دهند، اما به طور کامل از بین نمی‌برند.

3- محدودیت های سخت افزاری (Hardware Limitations):

دستگاه‌های فیزیکی (فرستنده‌ها، گیرنده‌ها و غیره) محدودیت‌های عملی در سرعت دارند. آن‌ها نمی‌توانند داده‌ها را با نرخ‌های دلخواه پردازش و ارسال کنند.

همگام‌سازی دقیق بین فرستنده و گیرنده برای ارتباط قابل اعتماد بسیار مهم است. در نرخ‌های بیت بسیار بالا، حفظ این همگام‌سازی به طور فزاینده‌ای دشوار می‌شود.

بخش 1-11)

خیر، ضرب کردن سیگنال دریافتی در $10\sin(2\pi t)$ به جای $2\sin(2\pi t)$ به طور کلی سیستم را در برابر نویز مقاوم‌تر نمی‌کند. در واقع، این عمل می‌تواند سیستم را حساس‌تر به نویز کند.

* افزایش دامنه:

ضرب کردن سیگنال در 10 به جای 2، دامنه سیگنال را پنج برابر افزایش می‌دهد. این عمل به خودی خود هیچ تاثیری بر نسبت سیگنال به نویز (SNR) ندارد. SNR نسبت قدرت سیگنال به قدرت نویز است. افزایش دامنه (سیگنال و نویز) را به یک میزان افزایش می دهد.

* افزایش حساسیت به نویز: در حالتی که نویز اضافه شود، افزایش دامنه سیگنال دریافتی به معنی افزایش دامنه نویز ضرب شده نیز است. از آنجا که دامنه نویز افزایش می‌یابد، نسبت سیگنال به نویز کاهش یافته و سیستم حساس تر به نویز می شود.

برای افزایش مقاومت در برابر نویز، به طور معمول از تکنیک‌های دیگری مانند:

* کدگذاری خطا: روش هایی برای افزودن اطلاعات اضافی به سیگنال برای تشخیص و تصحیح خطاهای ناشی از نویز.

* فیلترینگ: حذف نویز با استفاده از فیلترهای مناسب.

* مدولاسیون: روش های مدولاسیون که به طور خاص برای مقابله با نویز طراحی شده اند.

استفاده می‌شوند. عمل ضرب کردن سیگنال در یک تابع سینوسی با دامنه بزرگتر، یک رویکرد مناسب برای افزایش مقاومت در برابر نویز نیست.

بخش 1-12)

در اینترنت های خانگی سرعت ارسال اطلاعات از 1 مگابایت بر ثانیه تا 24 مگابایت بر ثانیه متنوع است.


که در این شرایط سرعت ارسال اطلاعات مگابایت بر ثانیه می باشد در حالی که در این تمرین سرعت ارسال اطلاعات نهایتا به 3 بایت بر ثانیه رسید .