گزارش پروژه پنجم درس سیگنال و سیستم‌ها

پریا پاسه‌ورز 810101393

کوثر شیری جعفرزاده 810101456

بخش اول:

تمرین 1-0)

فایل متناظر این بخش p1_0 است.

در این قسمت مثال مطرح شده در بخش توضیحات را نمایش می دهیم:

در این مثال باید نشان داده شود که در صورتی که فرکانس ها از رزولوشن فرکانسی کمتر باشند دیگر نمی توان آن ها را در حوزه ی فرکانسی تفکیک کرد.

```
fs = 20;
time_start=0;
time_end=1;
step=1/fs;
t = time_start:steps:time_end - steps;
N = length(t);
f = 0:(fs / N):((N - 1) * fs / N);
```

همانند آنچه به عنوان مثال آمده است ابتدا نرخ نمونه برداری را برابر 20 هرتز قرار می دهیم.

سپس باید بازه ی زمانی را طبق فرمول داده شده در نظر بگیریم بازه ی زمانی ما بین 0 و 1 قرار گرفته است و فاصله زمانی بین هردو سمپل برابر t_S است که از رابطه $t_s = \frac{1}{f_s} = \frac{1}{20} = 0.05$ به دست می آید.

حالا بازه ی 0 تا 1 را با فاصله زمانی 0.05 تقسیم می کنیم و تعداد سمپل هارا نیز در متغیر N ذخیره می کنیم. طبق فرمول بعدی در دستور پروژه (هایلایت سبز رنگ) فرکانس را نیز به دست می آوریم.

```
x1 = exp(1j * 2 * pi * 5 * t) + exp(1j * 2 * pi * 8 * t);
x2 = exp(1j * 2 * pi * 5 * t) + exp(1j * 2 * pi * 5.1 * t);
y1 = fft(x1);
y2 = fft(x2);
magnitude_1=abs(y1) / max(abs(y1));
magnitude_2=abs(y2) / max(abs(y2));
```

در تصویر بالا ما دو تابع ذکر شده در مثال را رسم می کنیم :

$$x_1(t) = e^{j*2\pi*5*t} + e^{j*2\pi*8*t}$$

$$x_2(t) = e^{j*2\pi*5*t} + e^{j*2\pi*5.1*t}$$

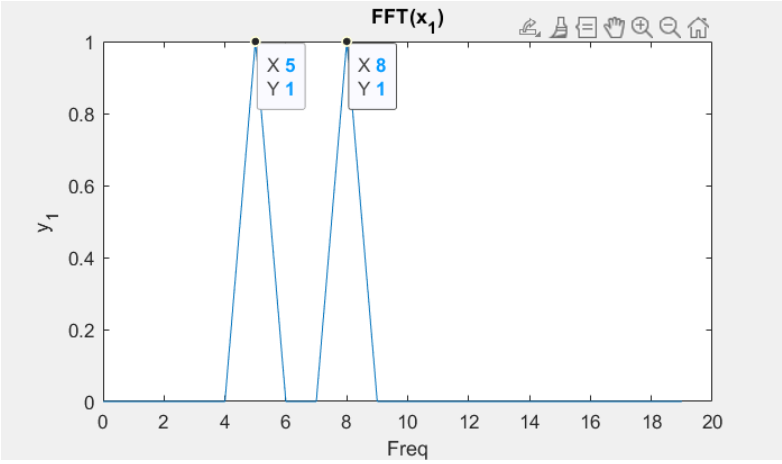
سپس در ابتدا سیگنال را به حوزه ی فوریه میبریم و درقدم بعدی برای محاسبه اندازه این سیگنال همانطور که گفته شده absolute value(y) را بر max(abs(y)) تقسیم می کنیم.

```
figure()
plot(f,magnitude_1)
xlabel('Freq')
ylabel('y_1')
title('FFT(x_1)')

figure()
plot(f, magnitude_2)
xlabel('Freq')
ylabel('y_2')
title('FFT(x_2)')
```

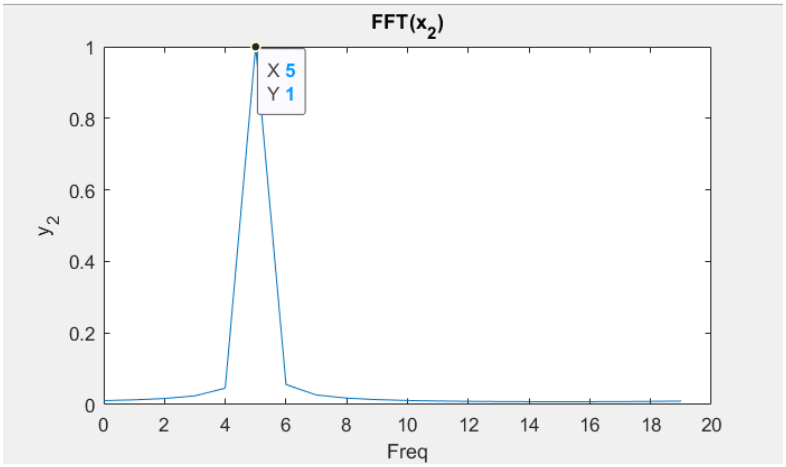
حال با استفاده از کد بالا اندازه سیگنال را در حوزه ی فوریه رسم می کنیم.

سیگنال اول :



همانطور که مشاهده می شود چون فاصله 5 و8 برابر 3 است و از رزولوشن فرکانس بیشتر می باشد پس 2 قله در تصویر قابل مشاهده می باشد.

سیگنال دوم:



همانطور که مشاهده می شود به دلیل اینکه فاصله 5 با 5.1 که برابر 0.1 می باشد و از رزولوشن فرکانس کمتر می باشد فقط یک قله قابل مشاهده است.

تمرین 1-1)

فایل متناظر این بخش p1_1 است.

الف)

```
fs = 50;
time_start=-1;
time_end=1;
steps=1/fs;
t = time_start:steps:time_end - steps;
N = length(t);
f = (-fs / 2):(fs / N):(fs / 2 - fs / N);
```

همانند آنچه در قسمت قبل انجام دادیم در اینجا نیز ابتدا نرخ نمونه برداری را برابر 50 هرتز قرار می دهیم و در قدم بعدی نقطه شروع و پایان بازه ی زمانی را تعیین می کنیم که برابر 1- تا 1 می باشد این بازه با گام های $t_s = \frac{1}{f_S} = \frac{1}{50} = 0.02$ تقسیم بندی می شود.

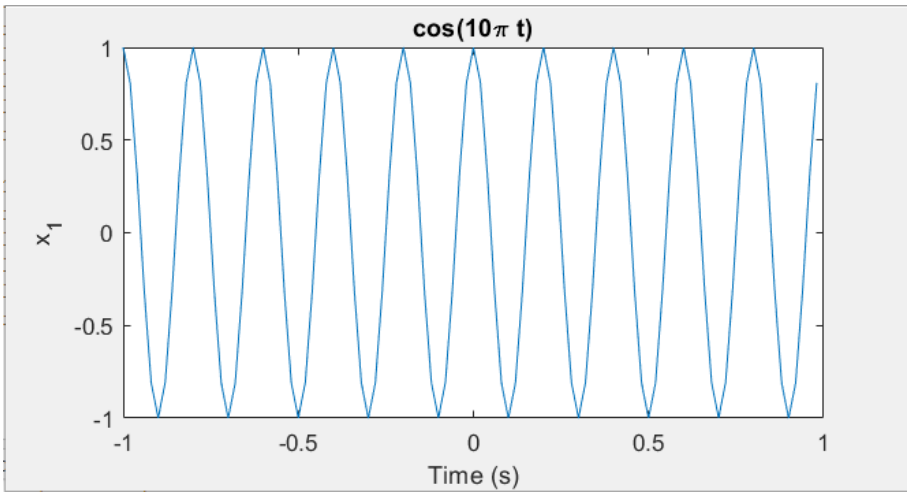
حال بازه زمانی t را برحسب گام های 0.02 شکل داده و در مرحله بعد تعداد کل سمپل را به دست می آوریم. با استفاده از فرمول صورت پروژه (هایلایت سبز رنگ) فرکانس را نیز به دست می آوریم.

```
x1 = cos(2 * pi * 5 * t);
figure()
plot(t, x1)
xlabel('Time (s)')
ylabel('x_1')
title('cos(10\pi t)')
```

در این مرحله تابع زیر را در بازه 1- تا 1 تعریف میکنیم:

$$x_1(t) = \cos(10\pi t)$$

و آن را دربازه ی مورد نظر رسم می کنیم (نهایت مقدار عبارت cos در لیمیت داده شده برای رسم سیگنال ها صدق می کند).



ب)

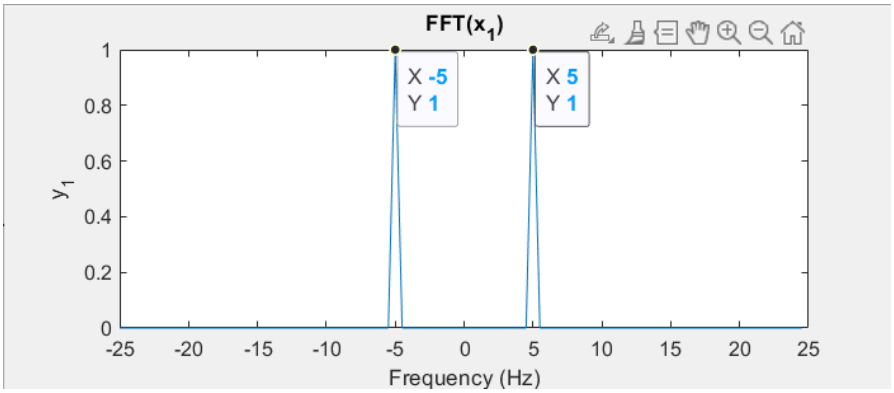
در این مرحله باید اندازه این سیگنال را در حوزه ی فوریه به دست بیاوریم:

```
y1 = fftshift(fft(x1));
magnitude=abs(y1) / max(abs(y1));
figure()
plot(f, magnitude)
xlabel('Frequency (Hz)')
ylabel('y_1')
title('FFT(x_1)')
```

همانند آنچه در صورت پروژه ذکر شده بود برای رسم اندازه سیگنال در حوزه ی فوریه باید حتما به شکل زیر خروجی رسم شود:

$$magnitude = \frac{fftshift(fft(x))}{\max(abs(y))}$$

در نهایت شکل خروجی را رسم می کنیم:



همانطور که مشاهده می شود شکل نهایی دارای دو قله در فرکانس های 5 و 5- است.

حال به صورت تئوری این موضوع را محاسبه می کنیم:

تبدیل فوریه سیگنال $\cos(w_0 t)$ به شرح زیر است :

$$\mathcal{F}(\cos(w_0 t)) = \pi\delta(w - w_0) + \pi\delta(w + w_0)$$

در این تمرین نمودار ها براساس فرکانس رسم شده اند در نتیجه در فرمول داده شده را تغییر متغیر می دهیم و برای نرمال سازی ضریب π را حذف کرده ایم:

$$w_0 = 10\pi \quad then\ f_0 = \frac{w_0}{2\pi} = \frac{10\pi}{2\pi} = 5$$

$$\mathcal{F}(\cos(f_0 t)) = \delta(f - f_0) + \delta(f + f_0)$$

$$\mathcal{F}(\cos(5t)) = \delta(f - 5) + \delta(f + 5)$$

همانطور که مشاهده می شود ما انتظار داریم که دو ضربه در فرکانس های 5 و -5 ببینیم که با مقادیر رسم شده تطابق دارد.

تمرین 1-2)

فایل متناظر این بخش p1_2 است.

الف)

همانند آنچه در دو قسمت قبل انجام داده ایم initialize کردن مقادیر را در ابتدا انجام می دهیم تا نرخ نمونه برداری، شروع و پایان بازه ی زمانی، فاصله ی زمانی بین دو سمپل و در نهایت فرکانس ها را به دست بیاوریم.

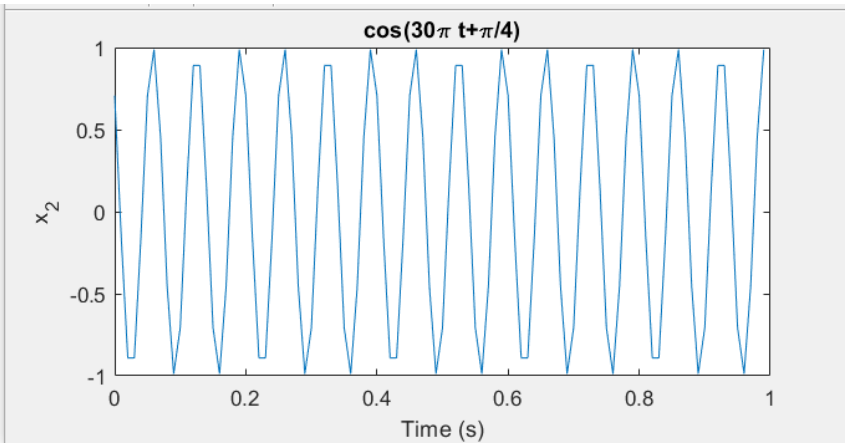
```
fs = 100;
time_start=0;
time_end=1;
steps=1/fs;
t = time_start:steps:time_end - steps;
N = length(t);
f = 0:(fs / N):((N - 1) * fs / N);
```

حال در قدم بعدی باید سیگنال زیر را رسم کنیم :

$$x_2(t) = \cos\left(30\pi t + \frac{\pi}{4}\right)$$

```
x2 = cos(2 * pi * 15 * t + 0.25 * pi);
figure()
plot(t, x2)
xlabel('Time (s)')
ylabel('x_2')
title('cos(30\pi t+\pi/4)')
```

شکل تابع رسم شده به صورت زیر خواهد بود:

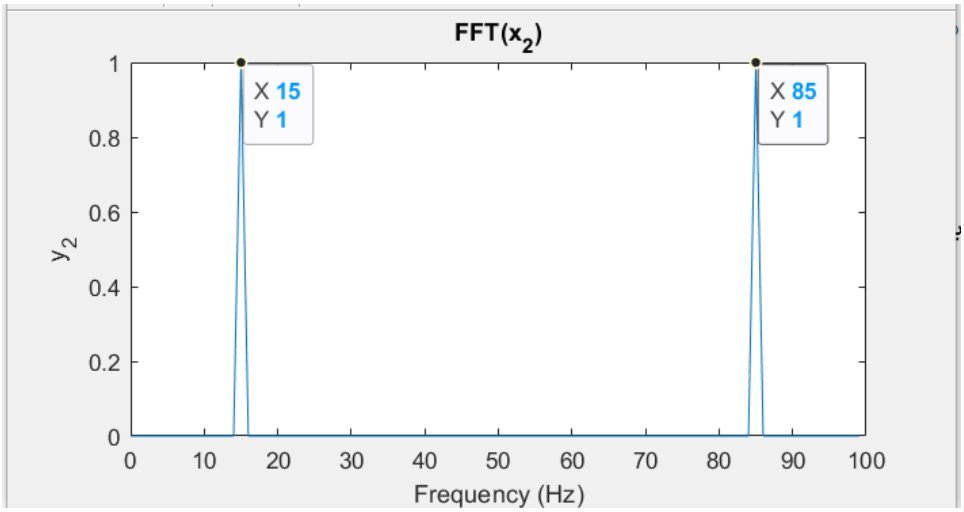


ب)

در این مرحله اندازه ی سیگنال را در حوزه ی فوریه رسم می کنیم:

```
y2 = fft(x2);
magnitude= abs(y2) / max(abs(y2));
figure()
plot(f,magnitude)
xlabel('Frequency (Hz)')
ylabel('y_2')
title('FFT(x_2)')
```

قطعه کد بالا اندازه سیگنال را در حوزه ی فوریه رسم می کند که به شکل زیر می باشد:



همانطور که دیده می شود دو قله در اعداد 15 و 85 دیده می شود.

حال به محاسبه تئوری این بخش می پردازیم تا از صحت اعداد مطمئن شویم:

$$\mathcal{F}(\cos(w_0 t)) = \pi\delta(w - w_0) + \pi\delta(w + w_0)$$

همانطور که می دانیم می توانیم از خاصیت شیفت استفاده کنیم :

$$\mathcal{F}\big(x(t - t_0)\big) = e^{-jwt_0}x(w)$$

پس می توانیم بنویسیم:

$$\mathcal{F}(\cos(w_0 t - t_0)) = \left(e^{-jwt_0}\pi\delta(w - w_0) + e^{jwt_0}\pi\delta(w + w_0)\right)$$

$$w_0 = 30\pi \quad then\ f_0 = \frac{w_0}{2\pi} = \frac{30\pi}{2\pi} = 15$$

به منظور نرمال سازی ضریب π را حذف می‌کنیم:

$$F(cos\ (f_0\ t)\ ) = \delta(f - f_0\ ) + \delta(f + f_0)$$

$$\mathcal{F}\left(\cos\left(30\pi t - \frac{\pi}{4}\right)\right) = \left(e^{-\frac{jw\pi}{4}}\delta(f + 15) + e^{\frac{jw\pi}{4}}\delta(f - 15)\right)$$

باتوجه به اینکه بازه ما به صورت متقارن نیست و از 0 تا 1 درنظر گرفته شده است پس باید متناظر ضربه ای که در 15- رخ می دهد را پیدا کنیم:

$$\mathcal{F}\left(\cos\left(30\pi t-\frac{\pi}{4}\right)\right)=\left(e^{-\frac{jw\pi}{4}}\delta(f+15-100)+e^{\frac{jw\pi}{4}}\delta(f-15)\right)$$

$$\mathcal{F}\left(\cos\left(30\pi t-\frac{\pi}{4}\right)\right)=\left(e^{-\frac{jw\pi}{4}}\delta(f-85)+e^{\frac{jw\pi}{4}}\delta(f-15)\right)$$

همانطور که در نتیجه دیده می شود در صورتی که بازه را متقارن در نظر نگیریم دو ضربه در 15 و 85 رخ می دهد که به دلیل آنکه عبارت $e^{-\frac{jw\pi}{4}}$ دارای اندازه 1 است پس در اندازه توابع ضربه تاثیری نداشته و مقدار ضربه همان 1 باقی می ماند.

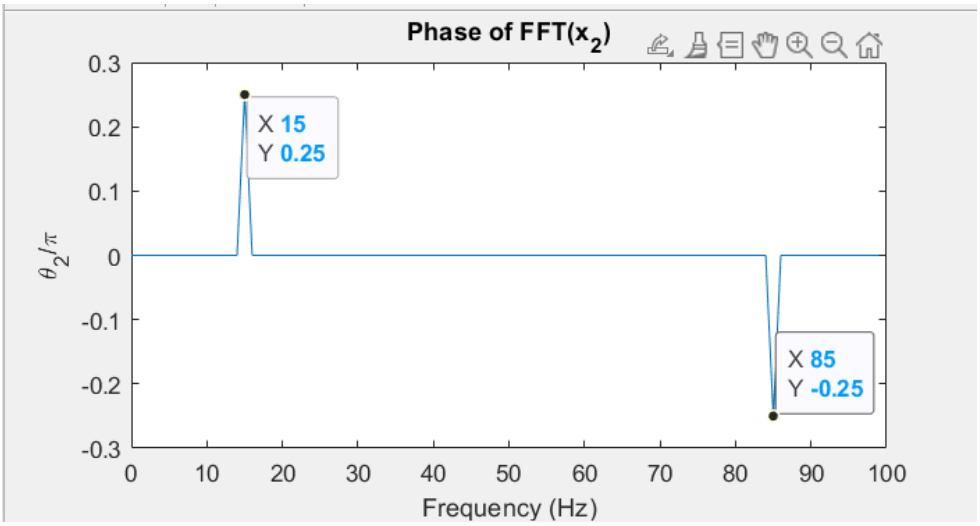
نتیجه بدست آمده در حالت تئوری با حالت رسم شده در متلب مطابقت دارد.

ج)

همانند آنچه در صورت پروژه ذکر شده است فاز این سیگنال را در حوزه ی فوریه رسم می کنیم:

```
smallValue = 1e-6;
y2(abs(y2) < smallValue) = 0;
theta2 = angle(y2);
figure()
plot(f, theta2 / pi)
xlabel('Frequency (Hz)')
ylabel('\theta_2/\pi')
title('Phase of FFT(x_2)')
```

شکل خروجی به شرح زیر خواهد بود:



همانطور که مشاهده می شود دو قله در15 با فاز 0.25 و در 85 با فاز 0.25- دیده می شود.

حال باید فاز این سیگنال را در حوزه ی فوریه به دست بیاوریم:

$$\mathcal{F}\left(\cos\left(30\pi t-\frac{\pi}{4}\right)\right)=\left(e^{-\frac{jw\pi}{4}}\delta(f-85)+e^{\frac{jw\pi}{4}}\delta(f-15)\right)$$

همانطور که می دانیم فاز e^{-jwx} برابر x می باشد پس در عبارت بالا نیز در نقطه 15 باید فاز 0.25 را مشاهده کنیم و در نقطه 85 باید فاز 0.25- را مشاهده کنیم.

که نتیجه بدست آمده در شکل رسم شده با مقادیر حساب شده تطابق دارد.

بخش دوم:

تمرین 1-2)

برای ساخت این mapset از قطعه کد زیر استفاده می کنیم:

```
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 mapset;
end
```

این قطعه کد ابتدا یک سلول با ابعاد 2*32 می سازد و سپس در سطر اول آن یکی از کارکتر های موجود در alphabet را قرار می دهد و در قدم بعدی در سطر دوم یک عدد باینری 5 بیتی قرار می دهد(چون ایندکس های متلب از 1 شروع می شوند ولی ما میخواهیم اعداد ما از رنج 0 تا 31 باشند پس عدد i-1 را به باینری تبدیل می کنیم لازم به ذکر است پارامتر دوم در dec2bin مشخص می کند که عدد باینری چند بیتی باشد).

در نهایت پس از ذخیره تمامی 32 کارکتر سلول Mapset را ذخیره می کنیم.

کد متناظر این بخش تابع create_mapset() می باشد.

در نهایت mapset ساخته شده را load می‌کنیم:

```
%% LOADING MAPSET

load('mapset.mat')
char_bin_len = length(mapset{2, 1});
```

تمرین 2-2)

کد:

```
function coded_signal = coding_freq(binary_msg, bitrate)

    fs = 100;
    freq_map = {[12, 37], ...
                [5, 16, 27, 38], ...
                [4, 10, 16, 22, 28, 34, 40, 46], ...
                [2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47], ...
                [1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19, 20, 22, 23, ...
                 25, 26, 28, 29, 31, 32, 34, 35, 37, 38, 40, 41, 43, 44, 46, 47]};
```

تابعی به نام coding_freq تعریف می‌کنیم که دو ورودی پیام باینری و bitrate (نرخ بیت) دارد و یک سیگنال خروجی تولید می‌کند.

همانطور که در صورت سوال بیان شده، برای ارسال اطلاعات با توجه به هر بیت‌ریت خاص باید فرکانس‌هایی را برای ارسال اطلاعات اطلاعات انتخاب کنیم. چون در صورت سوال تا بیت‌ریت 5 از ما خواسته شده، پس در آرایه دو بعدی freq_map، تا این بیت‌ریت خاص فرکانس‌های مورد نظر را مشخص کرده‌ایم. انتخابها باید به گونه‌ای باشد که فرکانس ها بیشترین فاصله ی ممکن را از هم داشته باشند تا وقتی داده نویزی می شود در تصمیم‌گیری دچار اشتباه کمتری شویم.

```
frequencies = cell2mat(freq_map(bitrate));
signal_length = length(binary_msg);
num_segments = signal_length / bitrate;
decoded_values = zeros(1, num_segments);
coded_signal = [];
```

حال فرکانس‌های مورد نظر را با توجه به مقدار bitrate ورودی جدا می‌کنیم. طول پیام با توجه به پیام باینری ورودی و تعداد بخشهایی که می‌خواهیم پیام را استخراج کنیم با کمک bitrate به دست می آوریم. آرایه coded signal جهت ذخیره‌سازی پیام نهایی تعریف شده است.

```
    for idx = 1:bitrate:signal_length
        segment_idx = (idx - 1) / bitrate + 1;
        binary_segment = binary_msg(idx:idx + bitrate - 1);
        decoded_values(segment_idx) = bin2dec(num2str(binary_segment));
        time_step = (segment_idx - 1):1/fs:segment_idx - 1/fs;
        freq_signal = sin(2 * pi * frequencies(decoded_values(segment_idx) + 1) * time_step);
        coded_signal = [coded_signal, freq_signal];
    end
end
```

در این بخش پیام را به قسمت‌هایی به طول بیت‌ریت تقسیم کرده و مقدار موجود در هر segment را به decimal تبدیل می‌کنیم. با توجه به این مقدار، فرکانس مورد نظر از frequencies انتخاب می‌شود.

زمان مورد نظر برای هر *segment* با توجه به فرکانس کدگذاری مشخص شده و از روی اطلاعات به دست آمده و به کمک رابطه $x(t) = sin(2\pi ft)$، سیگنال کد شده برای این بخش را می‌سازیم و به بخش‌های قبلی الحاق می‌کنیم.

تمرین 2-3)

در این قسمت خروجی تابع coding_freq را برای بیت‌ریت‌های 1 و 5 رسم می‌کنیم:

کد:

```
%% CODE MESSAGE WITH DIFFERENT BITRATES

fs = 100;
str = 'signal';

bin = message_to_binary(str, mapset);

bitrates = [1, 5];

num_bitrates = length(bitrates);
for i = 1:num_bitrates
    bitrate = bitrates(i);
    x = coding_freq(bin, bitrate);

    t = 0:(1 / fs):(length(str) * char_bin_len / bitrate - 1 / fs);
    figure;
    plot(t, x);
    title(['Bitrate = ', num2str(bitrate)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
end

grid on;
```

ابتدا پیام ورودی را به کمک تابع message_to_binary() و mapset ساخته شده به پیامی باینری تبدیل می‌کنیم:

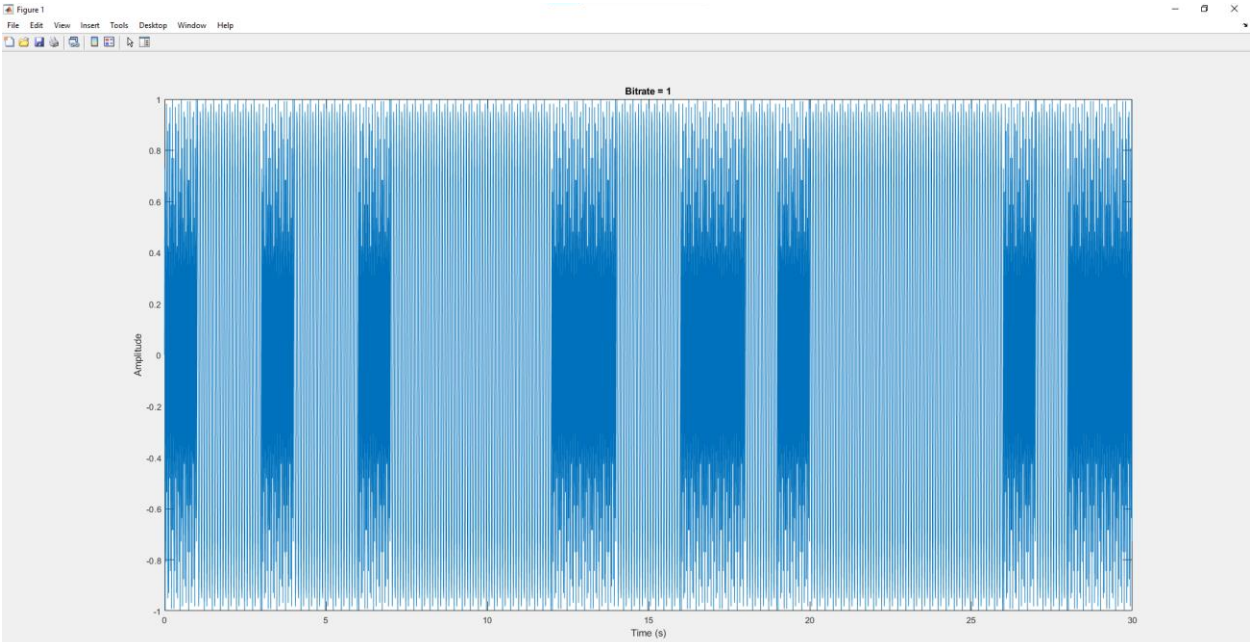
کد:

```
function bin = message_to_binary(str, mapset)
    bin = arrayfun(@(c) mapset{2, strcmp(mapset(1, :), c)}, str, 'UniformOutput', false);
    bin = cell2mat(bin);
end
```

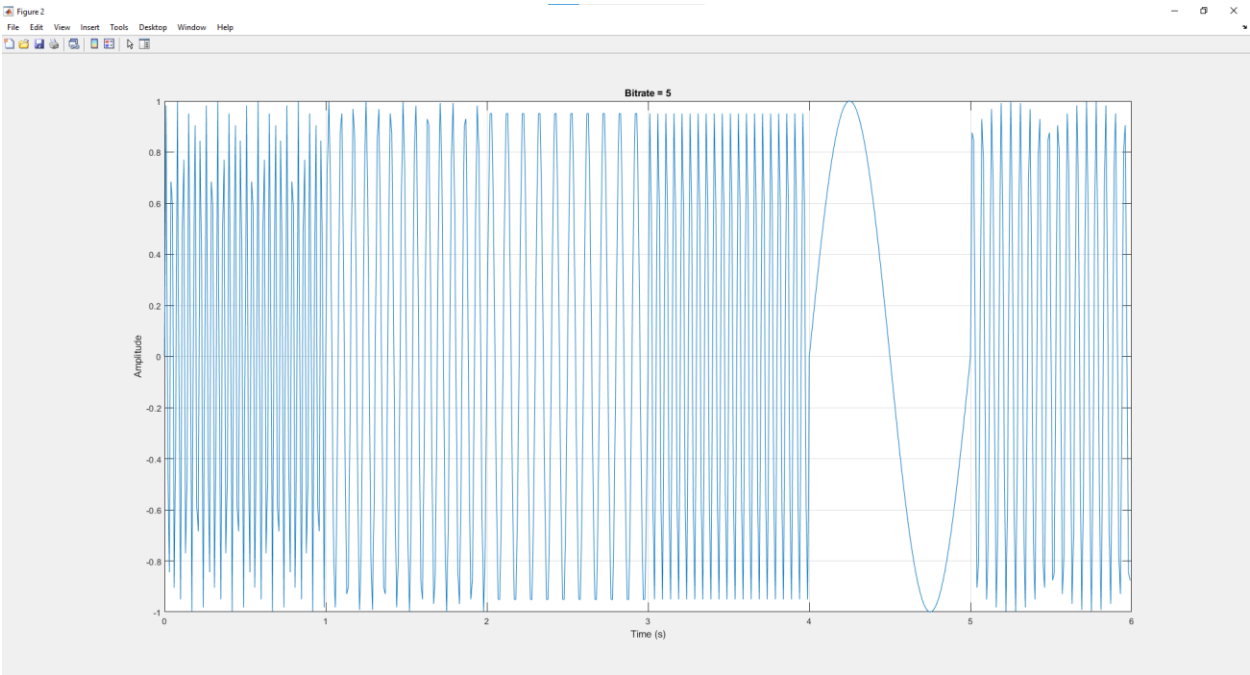
سپس برای هر یک ازbitrate های داده شده، آن را به کمک تابع coding_freq کد می‌کنیم و بر حسب زمان رسم می‌کنیم.

خروجی:

Bitrate = 1:



Bitrate = 5:



تمرین 2-4)

در این قسمت می‌خواهیم تابعی بنویسیم که پیام code شده را مجددا decode کند.

کد:

```
%% DEODING SIGNAL MESSAGE WITH DIFFERENT BITRATES, WIITHOUT NOISE

str = 'signal';
bitrates = [1, 5];
decoding_without_noise(str, bitrates, mapset);
```

برای این منظور ابتدا پیام را باینری کرده، آن را به تابع coding_freq دادیم و خروجی‌اش را به تابع decofing_freq دادیم و نهایتا پیام decode شده را چاپ کردیم.

کد:

```
function decoding_without_noise(str, bitrates, mapset)
bin_send = message_to_binary(str, mapset);
    result = cell(length(bitrates), 1);

    for i = 1:length(bitrates)
        bitrate = bitrates(i);
        signal_send = coding_freq(bin_send, bitrate);
        bin_receive = decoding_freq(signal_send, bitrate);
        str_receive = binary_to_message(bin_receive, mapset);
        result{i} = ['Recieved (bitrate=', num2str(bitrate),'): ', str_receive];
    end

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

کد:

```
function decoded_signal = decoding_freq(signal, bitrate)
    fs = 100;
    ts = 1 / fs;
    N = 1 / ts;
    freq_map = {[12, 37], ...
                [5, 16, 27, 38], ...
                [4, 10, 16, 22, 28, 34, 40, 46], ...
                [2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47], ...
                [1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19, 20, 22, 23, ...
                 25, 26, 28, 29, 31, 32, 34, 35, 37, 38, 40, 41, 43, 44, 46, 47]};

    frequencies = freq_map{bitrate};
    signal_length = length(signal);
    decoded_signal = [];
```

تابعی به نام decoding_freq تعریف می‌کنیم که دو ورودی سیگنال کد شده و bitrate (نرخ بیت) دارد و یک پیام خروجی تولید می‌کند.

مقداردهی‌های اولیه این تابع مشابه تابع coding_freq است.

کد:

```
for start_idx = 1:fs:signal_length
    segment = signal(start_idx:start_idx + fs - 1);
    normalized_fft = abs(fftshift(fft(segment)) / max(abs(fft(segment))));
    freq_indices = arrayfun(@(freq) ...
        normalized_fft(round(fs / 2 * fs / N - freq * fs / N + 1)), frequencies);
    [~, max_idx] = max(freq_indices);
    decoded_signal = [decoded_signal, dec2bin(max_idx - 1, bitrate)];
end
```

حال سیگنال را به بخش‌هایی به طول فرکانس نمونه‌برداری تقسیم می‌کنیم. برای هر بخش، ابتدا تبدیل فوریه آن را محاسبه می‌کنیم و تبدیل fftshift را انجام می‌دهیم تا حول صفر آورده شود.

در بخش بعدی، هدف این است که از خروجی تبدیل فوریه مشخص کنیم کدام فرکانس از میان مجموعه فرکانس‌های تعیین‌شده (frequencies) بیشترین سهم را دارد. این کار به کمک انتخاب فرکانس غالب و یافتن اندیس آن در آرایه انجام می‌شود.

خروجی:

```
Recieved (bitrate=1): signal
Recieved (bitrate=5): signal
```

تمرین 2-5)

کد:

```
%% DEODING SIGNAL MESSAGE WITH DIFFERENT BITRATES, WIITH NOISE

str = 'signal';
bitrates = [1, 5];
noise = 0.01;
decoding_with_noise(str, bitrates, noise, mapset);
```

کد:

```
function result = decoding_with_noise(str, bitrates, noise, mapset)
    bin_send = str2bin(str, mapset);
    result = cell(length(bitrates), 1);

    for i = 1:length(bitrates)
        bitrate = bitrates(i);
        signal_send = coding_freq(bin_send, bitrate);
        signal_receive = signal_send + noise * randn(size(signal_send));
        bin_receive = decoding_freq(signal_receive, bitrate);
        str_receive = bin2str(bin_receive, mapset);
        result{i} = ['Recieved (bitrate=', num2str(bitrate), ', noise=', num2str(noise), '): ', str_receive];
    end

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

کد این بخش مشابه قسمت قبل است، با این تفاوت که از تابع randn برای تولید noise استفاده کرده‌ایم و برای داشتن نویزی با واریانس 0.0001 باید این تابع را در 0.01 ضرب کنیم.

خروجی:

```
Recieved (bitrate=1, noise=0.01): signal
Recieved (bitrate=5, noise=0.01): signal
```

تمرین 2-6)

کد:

```
%% DEODING SIGNAL MESSAGE WITH DIFFERENT BITRATES, WIITH DIFFERENT NOISES

send_message = 'signal';
bitrates = [1, 5];

noise = 0.1;
decoding_with_noise(send_message, bitrates, noise, mapset);
fprintf('\n')

noise = 0.4;
decoding_with_noise(send_message, bitrates, noise, mapset);
fprintf('\n')

noise = 0.7;
decoding_with_noise(send_message, bitrates, noise, mapset);
fprintf('\n')

noise = 1;
decoding_with_noise(send_message, bitrates, noise, mapset);
fprintf('\n')

noise = 1.2;
decoding_with_noise(send_message, bitrates, noise, mapset);
fprintf('\n')

noise = 1.5;
decoding_with_noise(send_message, bitrates, noise, mapset);
fprintf('\n')

noise = 1.8;
decoding_with_noise(send_message, bitrates, noise, mapset);
fprintf('\n')
```

در این قسمت کم‌کم noise را افزایش دادیم و اثر آن را روی بیت‌ریت‌های 1 و 5 بررسی کردیم.

خروجی:

```
Recieved (bitrate=1, noise=0.01): signal
Recieved (bitrate=5, noise=0.01): signal
Recieved (bitrate=1, noise=0.1): signal
Recieved (bitrate=5, noise=0.1): signal

Recieved (bitrate=1, noise=0.4): signal
Recieved (bitrate=5, noise=0.4): signal

Recieved (bitrate=1, noise=0.7): signal
Recieved (bitrate=5, noise=0.7): signal

Recieved (bitrate=1, noise=1): signal
Recieved (bitrate=5, noise=1): signal

Recieved (bitrate=1, noise=1.2): signal
Recieved (bitrate=5, noise=1.2): signal

Recieved (bitrate=1, noise=1.5): signal
Recieved (bitrate=5, noise=1.5): signal

Recieved (bitrate=1, noise=1.8): signal
Recieved (bitrate=5, noise=1.8): signax
```

همانطور که مشاهده می‌شود، بیت‌ریت 1 نسبت به نویز مقاوم‌تر بوده و این مسئله با آنچه در مقدمه گفته شد سازگار است.

تمرین 2-7)

کد:

```
function thold = find_noise_threshold(str, bitrate, mapset)
    bin_send = message_to_binary(str, mapset);
    signal_send = coding_freq(bin_send, bitrate);

    thold = 2;
    nStep = 0.02;

    noise_values = nStep:nStep:2;
    for noise = noise_values
        if check_message_with_noise(signal_send, noise, bitrate, mapset, str)
            thold = noise - nStep;
            return
        end
    end
end
```

در این تابع، می‌خواهیم بررسی کنیم که هر بیت‌ریت تا چه نویزی مقاوم است و پیام را به درستی تشخیص می‌دهد. به این منظور از صفر شروع کردیم و 0.02، 0.02 به مقدار نویز اضافه کردیم و چک کردیم که آیا با این مقدار نویز پیام به درستی تشخیص داده می‌شود یا خیر. اگر اینطور بود سراغ مقدار نویز بالاتری می‌رویم.

کد:

```
function is_corrupted = check_message_with_noise(signal_send, noise, bitrate, mapset, original_str)
    for i = 1:100
        signal_receive = signal_send + noise * randn(size(signal_send));
        bin_receive = decoding_freq(signal_receive, bitrate);
        str_receive = binary_to_message(bin_receive, mapset);
        if ~strcmp(original_str, str_receive)
            is_corrupted = true;
            return;
        end
    end
    is_corrupted = false;
end
```

این تابع در یک حلقه 100 تایی، نویز با قدرت مشخص شده می‌سازیم و به سیگنالمان اضافه می‌کنیم. اگر پیام decode شده در هر صد حالت صحیح بود، true برمی‌گردانیم و در غیر این صورت، false.

خروجی:

```
Noise threshold (bitrate=1): 1.18
Noise threshold (bitrate=5): 0.98
```

همانطور که انتظار می‌رفت، بیت‌ریت 1 نسبت به نویز مقاوم‌تر است.

تمرین 2-9)

اگر نرخ نمونه برداری ارا افزایش دهیم ولی پهنای باند مصرفی را ثابت نگه داریم عملاً به این معناست که ما به وضوح بیشتری در زمان نمونه برداری دست پیدا می‌کنیم اما چون بهنای باند ثابت است، این افزایش نرخ نمونه برداری به معنای افزایش توان سیستم برای ارسال اطلاعات بیشتر نخواهد بود.

در صورتی که نرخ نمونه برداری افزایش یابد و پهنای باند ثابت بماند سیستم مجبور خواهد بود که داده های بیشتری را در همان پهنای باند انتقال دهد. به عبارت دیگر میتواند اطلاعات بیشتری را در زمان کمتری منتقل کند. اما چون پهنای باند محدود است ممکن است این افزایش نرخ نمونه برداری باعث ایجاد تداخل یا کاهش کیفیت سیگنال شود.