گزارش تمرین کامپیوتری چهارم سیگنال سیستم دکتر اخوان

پاییز 1403

محمدمهدی صمدی - 810101465

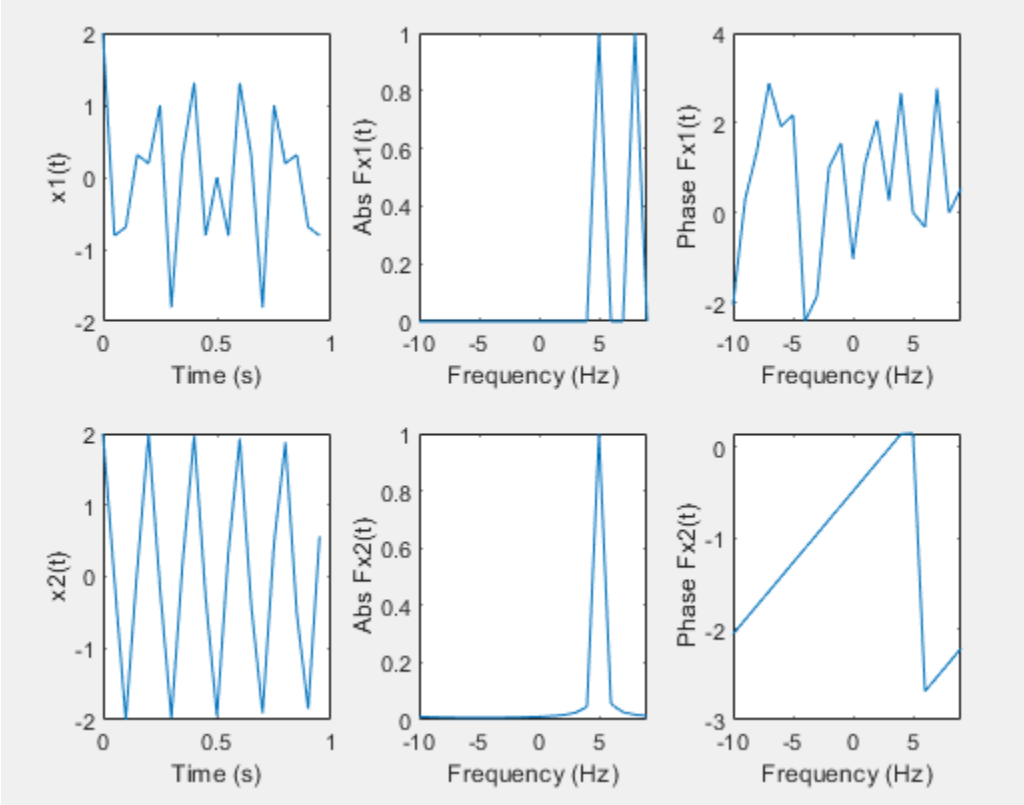
سپهر جمالی - 810101400

بخش اول:

در گزارش این بخش عکس کدها را نیاوردم زیرا نکته خاصی نداشتند.

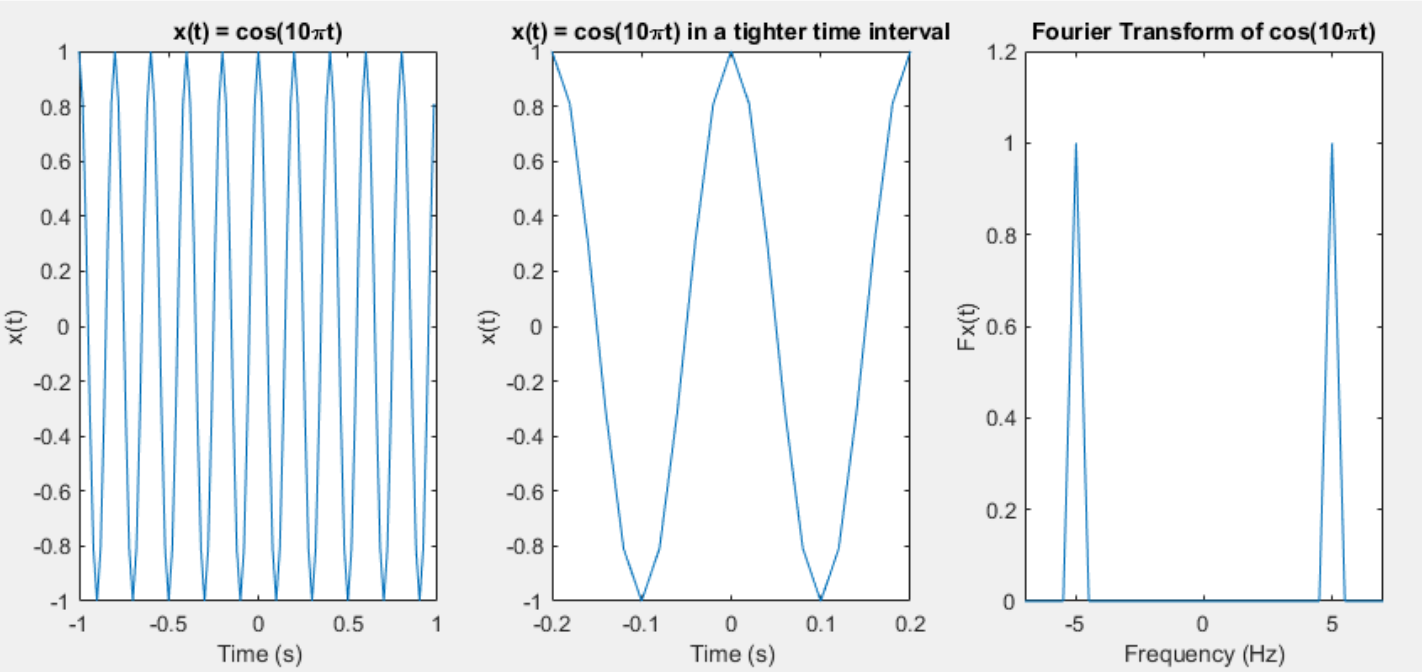
(0_1

دو سیگنال گفته شده را می‌سازیم و به ازای هر کدام، خود سیگنال دو حوزه زمان و اندازه و فاز آن در حوزه فوریه را رسم می‌کنیم. همین‌طور که مشخص است در سیگنال اول دو سمپل برداشته شده و در سیگنال دوم تنها یک سمپل برداشته شده.



(1_1

در ابتدا سیگنال $\cos(10\pi t)$ را در بازه گفته شده رسم کردیم. دوره تناوب این سیگنال $T = \frac{2\pi}{10\pi} = 0.2$ است. پس بازه $-1 \leq t \leq 1$ شامل 10 دوره تناوب است. سپس بازه را کوچک‌تر کرده و در دو دوره تناوب رسم کردیم. در نهایت هم تبدیل فوریه گرفته شده و طبق توضیحات صورت تمرین با تقسیم بر بیشینه مقدار، دامنه را به 1 می‌رسانیم.



حال تبدیل فوریه سیگنال $\cos(10\pi t)$ را حساب می‌کنیم تا تطابق با نتیجه متلب را بسنجیم:

$$F\{\cos(10\pi t)\} = \int_{-\infty}^{\infty} \cos(10\pi t) e^{-iwt} dt = \frac{1}{2}\int_{-\infty}^{\infty} (e^{i10\pi t} + e^{-i1o\pi t}) e^{-iwt} dt$$

$$= \frac{1}{2}\int_{-\infty}^{\infty} e^{-it(w-10\pi)} dt + \frac{1}{2}\int_{-\infty}^{\infty} e^{-it(w+10\pi)} dt$$

می‌دانیم:

$$\int_{-\infty}^{\infty} e^{iwt} dt = 2\pi\delta(w)$$

$$F\{cos(10\pi t)\} = \pi\delta(w - 10\pi) + \pi\delta(w + 10\pi)$$

همان‌طور که می‌دانیم $w = 2\pi f$ است. پس:

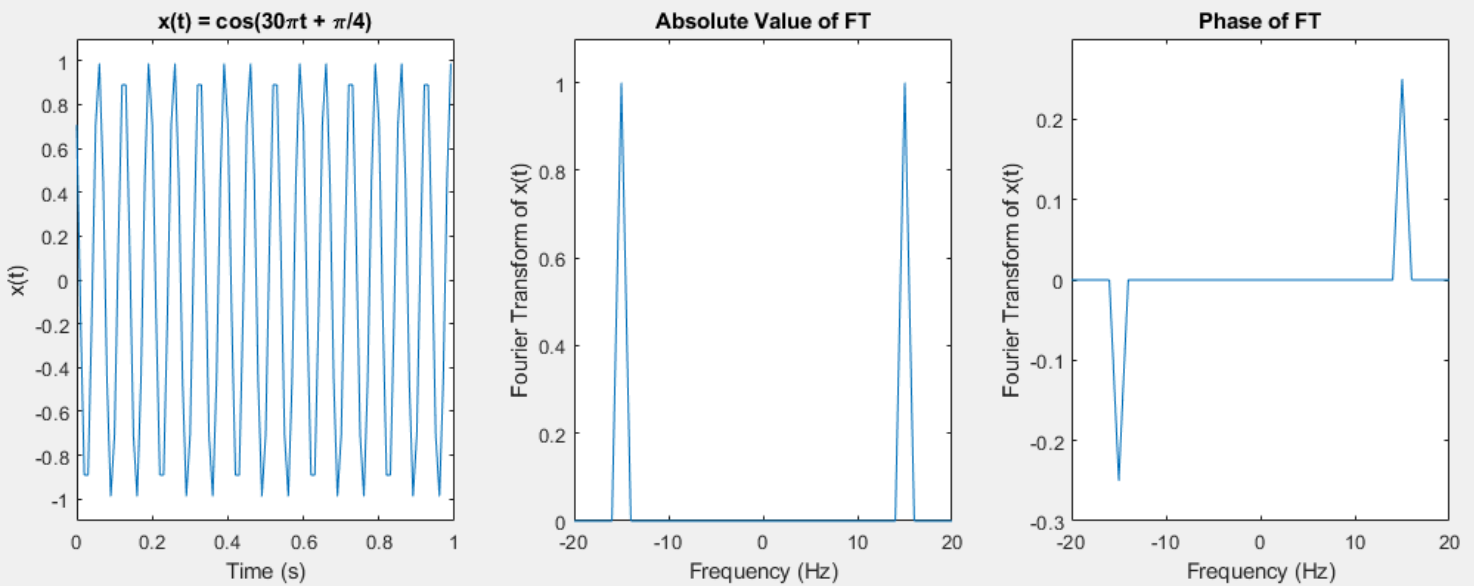
$$w_1 = 10\pi \rightarrow f_1 = 5$$

$$w_2 = -10\pi \rightarrow f_2 = -5$$

پس محاسبات با نتیجه متلب یکسان بود.

(2_1

در این بخش سیگنال $\cos(30\pi t + \frac{\pi}{4})$ که بر خلاف بخش قبل فاز دارد را رسم می‌کنیم، تبدیل فوریه می‌گیریم و در نهایت اندازه و فازش را بررسی می‌کنیم.



$$x_2(t) = \cos(30\pi t + \frac{\pi}{4}) = \frac{1}{2}(e^{i30\pi t}e^{i\frac{\pi}{4}} + e^{-i30\pi t}e^{-i\frac{\pi}{4}})$$

$$F\{x_2(t)\} = \frac{1}{2}\int_{-\infty}^{\infty}(e^{i30\pi t}e^{i\frac{\pi}{4}} + e^{-i30\pi t}e^{-i\frac{\pi}{4}})e^{-iwt}dt$$

$$= \frac{1}{2}e^{i\frac{\pi}{4}}\int_{-\infty}^{\infty}e^{-it(w-30\pi)}dt + \frac{1}{2}e^{-i\frac{\pi}{4}}\int_{-\infty}^{\infty}e^{-it(w+30\pi)}dt$$

$$F\{x_2(t)\} = \frac{\pi}{2}\delta(w - 30\pi t - \frac{\pi}{4})e^{i\frac{\pi}{4}} + \frac{\pi}{2}\delta(w + 30\pi + \frac{\pi}{4})e^{-i\frac{\pi}{4}}$$

$$\left|F\{x_2(t)\}\right| = \frac{\pi}{2}\delta(w - 30\pi t - \frac{\pi}{4}), \frac{\pi}{2}\delta(w + 30\pi + \frac{\pi}{4})$$

$$\angle F\{x_2(t)\} = \angle e^{i\frac{\pi}{4}}, \angle e^{-i\frac{\pi}{4}} = \frac{\pi}{4}, \frac{-\pi}{4}$$

خروجی اندازه تبدیل فوریه را بر بیشینه‌اش تقسیم کردیم و به این علت مقدار 1 دارد و نه $\frac{\pi}{2}$. همچنین برای اینکه فاز را بر حسب ضرایب π داشته باشیم بر آن تقسیمشان کردیم و در نتیجه روی نمودار فازهای $\pm\frac{1}{4}$ را گرفتیم.

همان‌طور که می‌دانیم $w = 2\pi f$ است. پس:

$$w_1 = 30\pi \rightarrow f_1 = 15$$
$$w_2 = -30\pi \rightarrow f_2 = -15$$

پس اندازه، فاز و فرکانس سیگنال تابع فوریه مطابق انتظار بوده‌اند.

بخش دوم:

2_1)

تابع زیر mapset را می‌سازد و در دو ردیف سل متلب ذخیره می‌کند.

```
function mapset=create_mapset(chars)
    num_chars = length(chars);
    coded_binary_length = ceil(log2(num_chars));
    mapset=cell(2, num_chars);
    for i=1:num_chars
        mapset{1,i}=chars(i);
        mapset{2,i}=dec2bin(i-1, coded_binary_length);
    end
end
```

چند خانه نمونه از خروجی تابع create mapset:

22	23	24	25	26
v	w	x	y	z
10101	10110	10111	11000	11001

2_2)

این تابع را به سه بخش کلی می‌توان تقسیم کرد.

- تعریف متغیرهایی مانند طول پیام، طول پیام باینری و تعداد بیت‌های نمایش هر کاراکتر.

```
function signal_containing_message = coding_freq(message, mapset, bit_rate)
    fs = 100;
    signal_containing_message = [];

    numof_chars = length(mapset);
    numof_each_char_bit = ceil(log2(numof_chars));
    message_len = strlength(message);
    binary_message_len = message_len * numof_each_char_bit;
    binary_message = blanks(binary_message_len);
```

- تبدیل پیام به فرم باینری با استفاده از مپ‌ست ساخته شده در بخش قبل. به ازای هر حرف پیام، سل متناظرش در مپ‌ست را پیدا کرده و 5 بیت نمایش‌دهنده آن را به پیام باینری اضافه می‌کنیم.

```
for i=1:message_len
    ch = message(i);
    for j=1:numof_chars
        if ch == mapset{1,j}
            coded_bits = mapset{2, j};
            for k=1:numof_each_char_bit
                binary_message(5*(j-1)+k) = coded_bits(k);
            end
        end
    end
end
```

- کد کردن پیام باینری در سیگنال حوزه فرکانس: در ابتدای هر دور حلقه، به اندازه bit rate بیت از پیام باینری جدا کرده و در متغیری می‌ریزیم. در این بخش فرض شده که سایز پیام باینری بر bit rate بخش‌پذیر است. البته اگر نباشد در اسکریپتی که توابع را ران می‌کند اروری می‌دهیم.

به اندازه 2^{br} فرکانس مختلف نیاز داریم که آن‌ها را در دورترین فواصل ممکن از هم باید بچینیم. در بازه 50 تایی، حداقل و حداکثر فرکانس را 5 و 44 ست کردیم و بازه بین این دو را به تعداد مورد نیاز متساوی‌الفاصله تقسیم می‌کنیم. به ازای هر bit rate چه فرکانس‌هایی برداشته می‌شوند؟

- $bit\ rate = 1 \rightarrow 5, 44$
- $bit\ rate = 2 \rightarrow 5, 18, 31, 44$
- $bit\ rate = 3 \rightarrow 5, 10, 15, 20, 25, 30, 35, 40$
- $bit\ rate = 4 \rightarrow 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35$
- $bit\ rate = 5 \rightarrow 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,$
  $21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36$

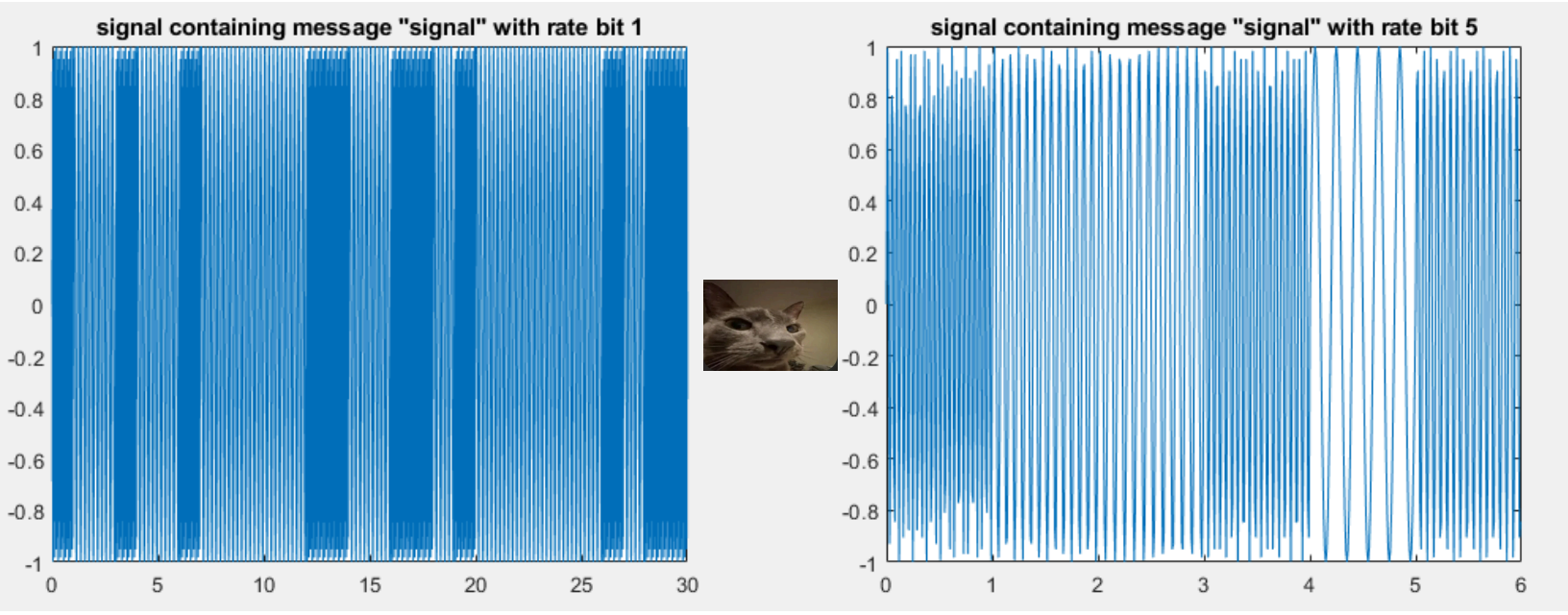
فرمولی که فرکانس هم سیگنال را مشخص می‌کند:

$$minf = 5\ and\ maxf = 44 \rightarrow maxf - minf = 39$$

$$f = minf + dec \times floor(\frac{maxf - minf}{num\ freqs - 1})$$

```
for i=1:bit_rate:binary_message_len
    bit_rate_consecutive_bits = binary_message(i : i + bit_rate - 1);
    dec = bin2dec(bit_rate_consecutive_bits);
    freq = (dec * (round(39 / ((2^bit_rate) - 1)))) + 5;
    t = (i/bit_rate) : (1/fs) : (i/bit_rate) + 1 - (1/fs);
    new_signal = sin(2 * pi * freq * (t-(i/bit_rate)));
    signal_containing_message = [signal_containing_message new_signal];
end
```

(3_2



(4_2

این تابع را به سه بخش کلی می‌توان تقسیم کرد:

- در ابتدا متغیرهای مورد نیاز برای تابع را تعریف می‌کنیم:
  - خط 3 تا 5: حداقل و حداکثر فرکانس مورد انتظار، اختلاف این دو عدد و تعداد فرکانس‌های متمایز.
  - خط 6 تا 9: تعریف آرایه خالی برای مسیج باینری و مسیج استرینگ سایز آن‌ها از روی سایز سیگنال و فرکانسی که می‌دانیم 100 است مشخص می‌شود.
  - خط 10 تا 14: در حلقه for آرایه middle frequencies را پر کرده‌ایم که نقطه وسط هر دو فرکانس متوالی را نشان می‌دهد. پس وقتی 2^{br} فرکانس ممکن از 5 تا 44 داریم، $2^{br} - 1$ نقطه وسط داریم که باید پر کنیم.

```
1  function decoded_message = decoding_freq(coded_signal, mapset, bit_rate)
2      fs = 100; N = 100;
3      min_freq = 5; max_freq = 44;
4      diff_freq = max_freq - min_freq;
5      num_freqs = 2^bit_rate;
6      numof_expected_bits = length(coded_signal) / fs;
7      binary_message = blanks(numof_expected_bits);
8      numof_each_char_bit = ceil(log2(length(mapset)));
9      decoded_message = blanks(numof_expected_bits / numof_each_char_bit);
10     midlle_freqs = blanks(num_freqs-1);
11     freq_dis = round(diff_freq / (num_freqs - 1));
12     for k=1:num_freqs-1
13         midlle_freqs(k) = min_freq + (k-1)*freq_dis + floor(freq_dis / 2);
14     end
```

- شکستن کد سیگنال: تبدیل فوریه گرفتن و شیفت دادنش معادل correlation گرفتن در تمرین قبلی است. سپس قله تبدیل فوریه را پیدا کرده و ادعا می‌کنیم بیت ذخیره شده در آن است. اگر پیام نویز نداشته باشد یا نویز کمی داشته باشد ادعای ما درست است اما با نویز زیاد ممکن است به اشتباه بیوفتیم. حالا از بین ارایه فرکانس‌ها، فرکانس قله را جدا کرده و آن را با اعداد داخل middle frequencies مقایسه می‌کنیم تا بفهمیم بین کدام دو عدد آن است. در انتهای این بخش پیام باینری را جدا کرده ایم.

```
freq = (-fs / 2) : (fs / N) : (fs / 2) - (fs / N);
for i = 0 : (length(coded_signal) / fs) - 1
    signal_part = coded_signal((i * fs) + 1:((i + 1) * fs));
    ft = abs( fftshift ( fft (signal_part) ) );
    [~, max_idx] = max(ft);
    selected_freq = abs( freq (max_idx) );
    for t=1:length(midlle_freqs)
        if selected_freq <= midlle_freqs(t)
            selected_freq = midlle_freqs(t) - (floor((freq_dis) / 2));
            break;
        elseif t == length(midlle_freqs)
            selected_freq = max_freq;
        end
    end
    b = dec2bin( ( (selected_freq - min_freq) / freq_dis), bit_rate);
    for j=1:length(b)
        binary_message(bit_rate*i + j) = b(j);
    end
end
```

- تبدیل پیام باینری به فرم اصلی: هر 5 بیت از پیام باینری را برداشته و در مپست حرف معادلش را انتخاب می‌کنیم. جواب نهایی تابع در decoded message قرار دارد.

```
for i = 1 : numof_expected_bits / numof_each_char_bit
    l = (i-1) *numof_each_char_bit + 1;
    r = i *numof_each_char_bit;
    message_to_check = binary_message(l:r);
    message_to_check = mapset{1,bin2dec(message_to_check)+1};
    decoded_message(i) = message_to_check;
end
```

```
decoded_message_rate_bit_1 =

    'signal'
```

خروجی برای دو سیگنال تولید شده بخش قبل:

```
decoded_message_rate_bit_5 =

    'signal'
```

5_2)

```
decoded_message_rate_bit_1 =

    'signal'


decoded_message_rate_bit_5 =

    'signal'
```

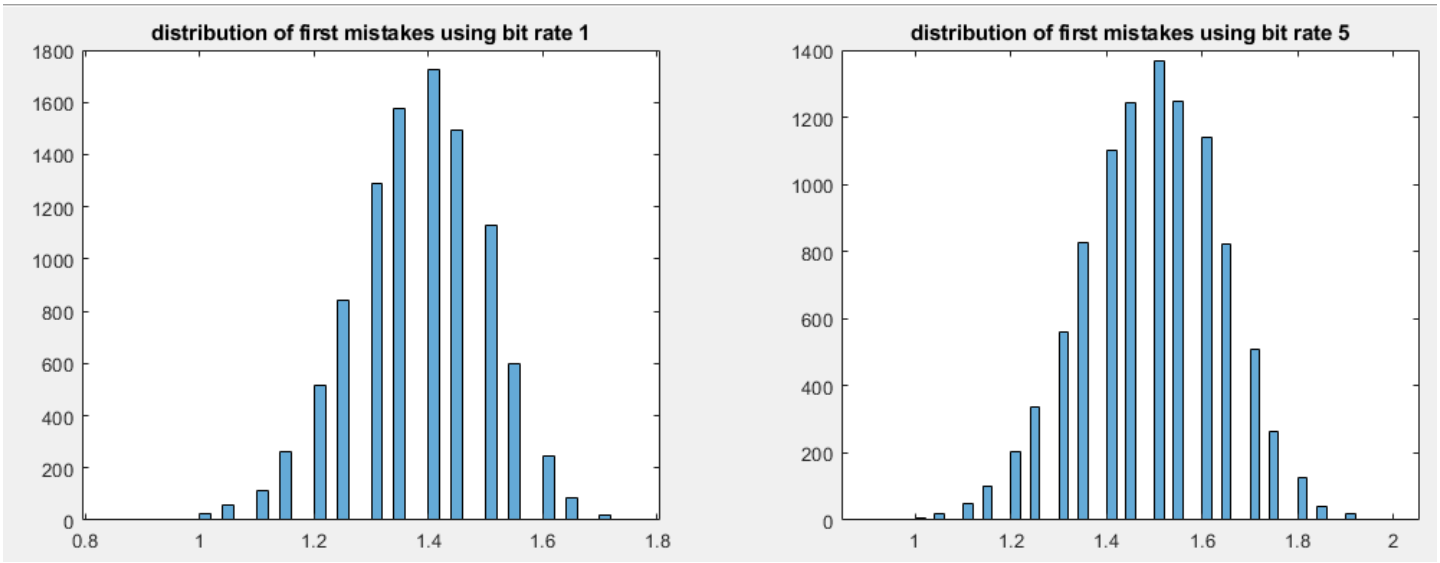
بله باز هم به درستی سیگنال decode شد.

6_2 و 7_2)

در کد زیر، به ازای std های مضرب 0.05 هر بار به سیگنال تولید شده نویز اضافه کرده و به تابع decode دادیم. اولین std ای که منجر به اشتباه شد را ثبت کردیم. این فرایند 10000 بار تکرار شده.

```
num_repeats = 10000;
first_mistake_1 = zeros(1, num_repeats);
first_mistake_5 = zeros(1, num_repeats);
for j=1:num_repeats
    check_1 = 1; check_5 = 1;
    for s=0.05:0.05:3
        if (check_1)
            noise1 = s*randn(1,length(signal_containing_message_bit_rate_1));
            noisy_signal_bit_rate_1 = signal_containing_message_bit_rate_1 + noise1;
            decoded_message_rate_bit_1 = decoding_freq(noisy_signal_bit_rate_1, mapset, 1);
            if (strcmp(decoded_message_rate_bit_1, message) == 0)
                first_mistake_1(j) = s;
                check_1 = 0;
            end
        end

        if (check_5)
            noise5 = s*randn(1,length(signal_containing_message_bit_rate_5));
            noisy_signal_bit_rate_5 = signal_containing_message_bit_rate_5 + noise5;
            decoded_message_rate_bit_5 = decoding_freq(noisy_signal_bit_rate_5, mapset, 5);
            if (strcmp(decoded_message_rate_bit_5, message) == 0)
                first_mistake_5(j) = s;
                check_5 = 0;
            end
        end
    end
end
```



سیگنال با bit rate برابر 1 به طور میانگین هر دفعه در std برابر 1.3782 دچار خطا شده است.

سیگنال با bit rate برابر 5 به طور میانگین هر دفعه در std برابر 1.4903 دچار خطا شده است.

نتیجه متناقض است با آنچه که در صورت پروژه بیان شد. در حالتی که $bit\ rate = 5$، ارسال پیام 0.6 ثانیه طول کشید (زیرا 6 سیگنال و هر کدام را به مدت زمان 0.1 ثانیه فرستادیم). اما در حالتی که $bit\ rate = 1$، ارسال پیام 3 ثانیه طول کشید (زیرا 30 سیگنال و هر کدام را به مدت زمان 0.1 ثانیه فرستادیم).

```
17    length(signal_containing_message_bit_rate_1)
18    length(signal_containing_message_bit_rate_5)
```

```
Command Window

ans =

        3000


ans =

   600
```

طبق تصویر روبه‌رو، سیگنال اول 3000 سمپل دارد و سیگنال دوم 600 سمپل. از آن‌جایی که نویز را به تعداد سمپل برای هر سیگنال تولید کردیم، احتمال اینکه یک اشتباه در سیگنال دارای نویز اول رخ دهد 5 برابر بیشتر است. زیرا اگر حتی یکی از نویزها از مقدار مقدار تبدیل فوریه بیشتر باشد، یعنی وقتی تبدیل فوریه می‌گیریم آن نویز به عنوان max تشخیص داده می‌شود و ما را گمراه می‌کند. پس به این علت سیگنال دوم مقاوم‌تر نشان داد. اما اگر نوع نویز دادن را عوض کنیم و به دو سیگنال به اندازه یکسان نویز دهیم مشاهده می‌کنیم که سیگنال دوم مقاوم‌تر خواهد بود.

2_8)

پاسخ صورت پروژه برای این بخش: هر چه فاصله ی فرکانس‌های انتخابی بیشتر باشند کدگذاری نسبت به نویز مقاوم‌تر می شود. بنابراین هر چه پهنای باند بیشتری مصرف کنیم می‌توانیم با سرعت بیشتری اطلاعات را ارسال کنیم و در عین حال نسبت به نویز مقاوم باشیم.

2_9)

با افزایش sampling rate، فرکانس‌های بیشتری هم در اختیار داریم. زیرا فرکانس‌های ما به صورت زیر هستند:

$$f = \frac{-f_s}{2} : \frac{f_s}{N} : \frac{f_s}{2} - \frac{f_s}{N}$$

اما صرفا فرکانس‌های بیشتر به معنای مقاومت بیشتر به نویز نخواهد بود. زیرا پهنای باند همچنان ثابت می‌باشد. در واقع برای مقاوم‌تر شدن به نویز باید رزولوشن فرکانسی را افزایش دهیم که در این حالت تغییری نکرده است.