

## تمرین ۱-۰:

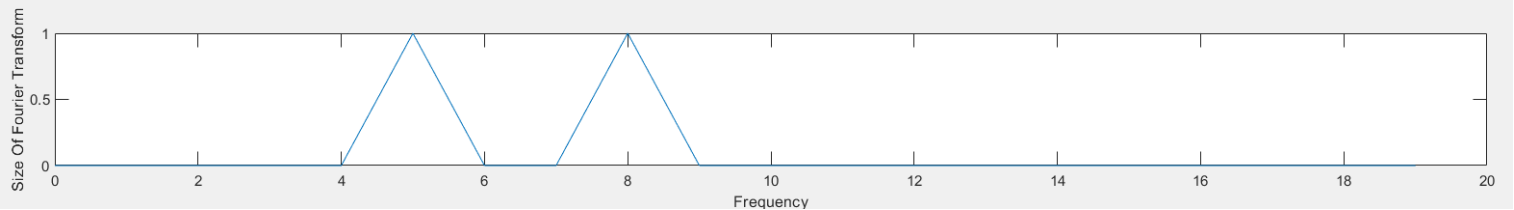
```
clc;
clear;
close all;

%0-1
%%
tstart=0;
tend=1;
fs=20;
ts=1/fs;

t=tstart:ts:tend-ts;
N=length(t);
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);
```

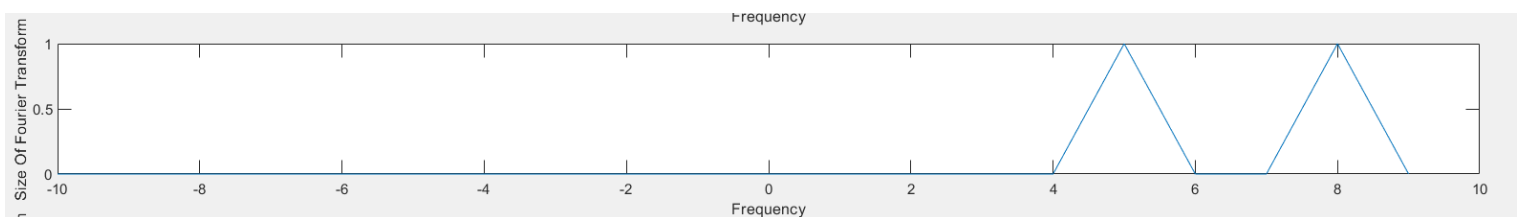
- رسم نمودار سیگنال  $x_1(t) = \exp(1j * 2\pi * 5 * t) + \exp(1j * 2\pi * 8 * t)$  با دستور fft :

```
subplot(4, 1, 1)
freq1=0:fs/N:(N-1)*fs/N;
x1f=fft(x1);
z1f=abs(x1f)/max(abs(x1f));
plot(freq1, z1f)
xlabel 'Frequency'
ylabel 'Size Of Fourier Transform'
```



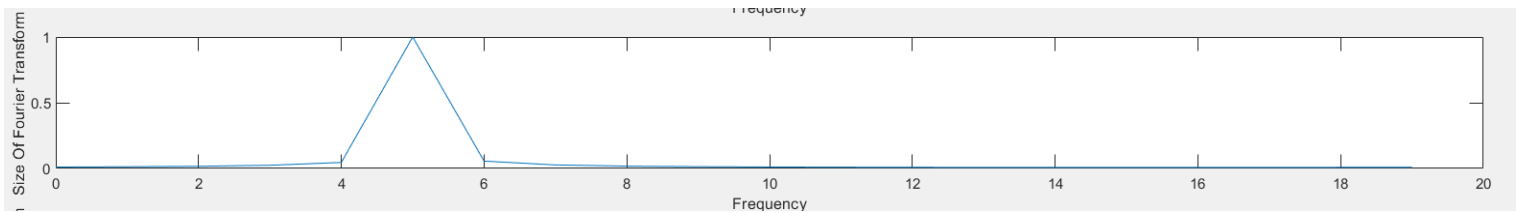
- رسم نمودار سیگنال  $x_1(t) = \exp(1j * 2\pi * 5 * t) + \exp(1j * 2\pi * 8 * t)$  با دستور fftshift :

```
subplot(4, 1, 2)
freq2=-fs/2:fs/N:fs/2-fs/N;
x2f=fftshift(x1f);
z2f=abs(x2f)/max(abs(x2f));
plot(freq2, z2f)
xlabel 'Frequency'
ylabel 'Size Of Fourier Transform'
```



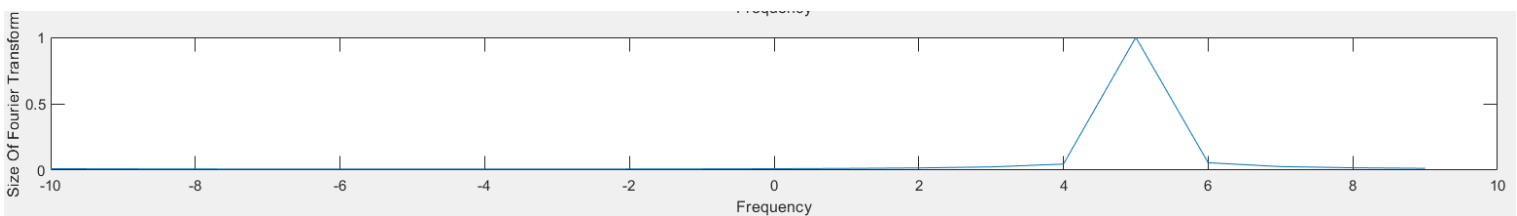
- رسم نمودار سیگنال  $x_2(t) = \exp(1j * 2\pi * 5 * t) + \exp(1j * 2\pi * 5.1 * t)$  با دستور fft :

```
subplot(4, 1, 3)
x3f=fft(x2);
z3f=abs(x3f)/max(abs(x3f));
plot(freq1, z3f)
xlabel 'Frequency'
ylabel 'Size Of Fourier Transform'
```



- رسم نمودار سیگنال  $x_2(t) = \exp(1j * 2\pi * 5 * t) + \exp(1j * 2\pi * 5.1 * t)$  با دستور fftshift :

```
subplot(4, 1, 4)
x4f=fftshift(x3f);
z4f=abs(x4f)/max(abs(x4f));
plot(freq2, z4f)
xlabel 'Frequency'
ylabel 'Size Of Fourier Transform'
%%
```



از آن جایی که اختلاف فرکانس دو سیگنال تک تُن کمتر از  $\delta_f = 1Hz$  است، فقط یک قله در فرکانس 5 هرتز داریم و توانایی تفکیک این دو سیگنال در حوزه فوریه را نداریم. پس می‌توان گفت  $\delta_f$  قدرت تفکیک پذیری فرکانسی را در حوزه فوریه نشان می‌دهد.

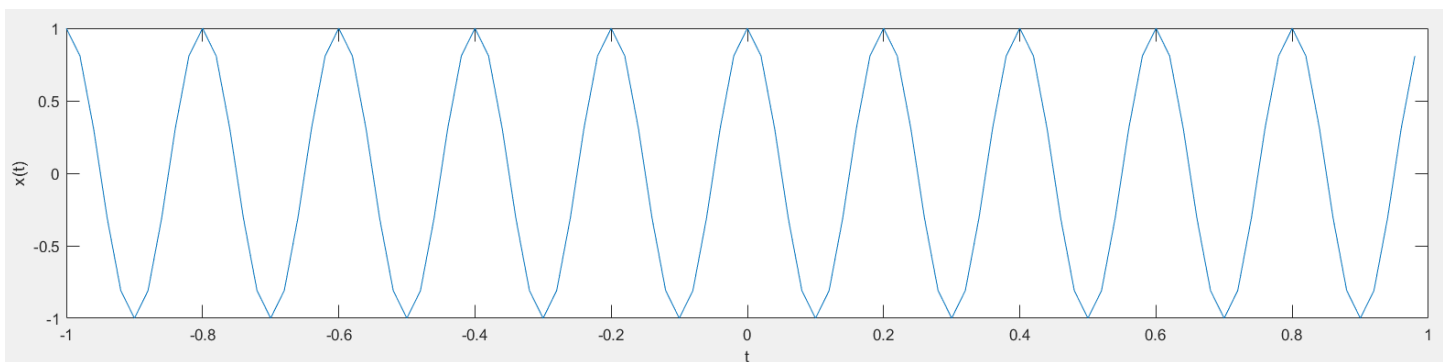
## تمرین ۱-۱:

الف) رسم نمودار سیگنال  $x_1(t) = \cos(10\pi t)$ :

```
%1-1
tstart=-1;
tend=1;
fs=50;
ts=1/fs;

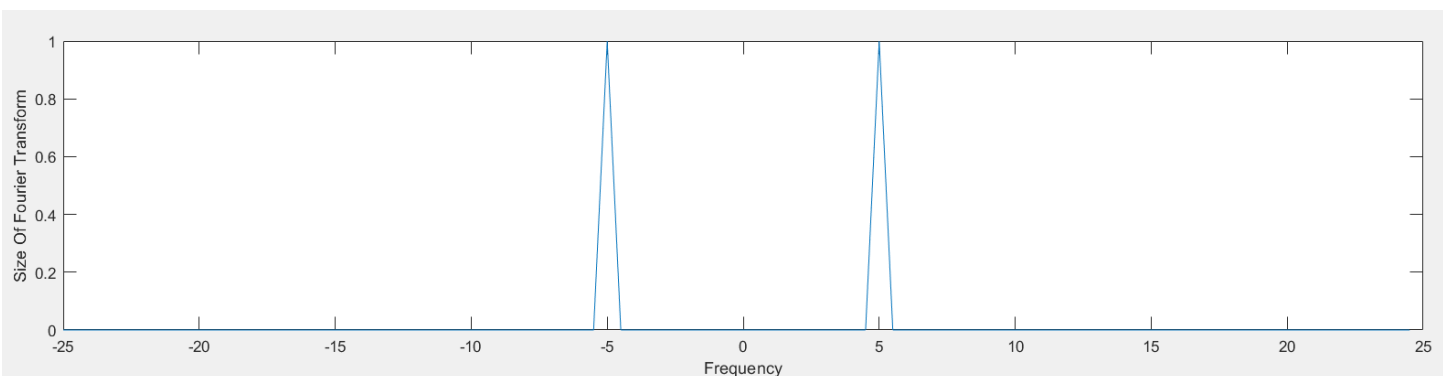
t=tstart:ts:tend-ts;
N=length(t);
x1=cos(2*pi*5*t);

subplot(2, 1, 1)
plot(t, x1)
xlabel 't'
ylabel 'x(t)'
```



ب) رسم نمودار اندازه تبدیل فوریه سیگنال  $x_1(t) = \cos(10\pi t)$ :

```
subplot(2, 1, 2)
freq=-fs/2:fs/N:fs/2-fs/N;
x1f=fftshift(fft(x1));
z1f=abs(x1f)/max(abs(x1f));
plot(freq, z1f)
xlabel 'Frequency'
ylabel 'Size Of Fourier Transform'
%%
```



ج) محاسبه تبدیل فوریه پیوسته سیگنال  $x_1(t) = \cos(10\pi t)$  به صورت تئوری :

از آن جایی که سیگنال  $x_1(t) = \cos(10\pi t)$  یک سیگنال متناوب است، پس تبدیل فوریه آن از رابطه زیر به دست می آید :

$$\hat{x}(\omega) = \sum_k 2\pi a_k \delta(\omega - \omega_k)$$

که  $a_k$  ضرایب سری فوریه  $x_1(t)$  هستند. از طرفی می دانیم سری فوریه  $x_1(t)$  به شکل زیر است :

$$x_1(t) = \frac{1}{2}e^{j10\pi t} + \frac{1}{2}e^{-j10\pi t}$$

هم چنین می دانیم  $\omega = 2\pi f$  پس داریم :

$$2\pi f_1 = 10\pi \Rightarrow f_1 = 5, 2\pi f_2 = -10\pi \Rightarrow f_2 = -5$$

پس تبدیل فوریه  $x_1(t)$  برابر است با :  $\pi(\delta(\omega - 10\pi) + \delta(\omega + 10\pi))$  دقت شود چون محور افقی، به جای  $\omega$ ،  $f$  است، پس تبدیل فوریه بر حسب  $f$  به این صورت می شود :  $\pi(\delta(f - 5) + \delta(f + 5))$  و چون نمودار  $\frac{|\hat{x}(\omega)|}{\max(|\hat{x}(\omega)|)}$  را رسم کرده ایم پس مقدار بیشینه نمودار به جای  $\pi$ ، 1 است.

## تمرین ۱-۲:

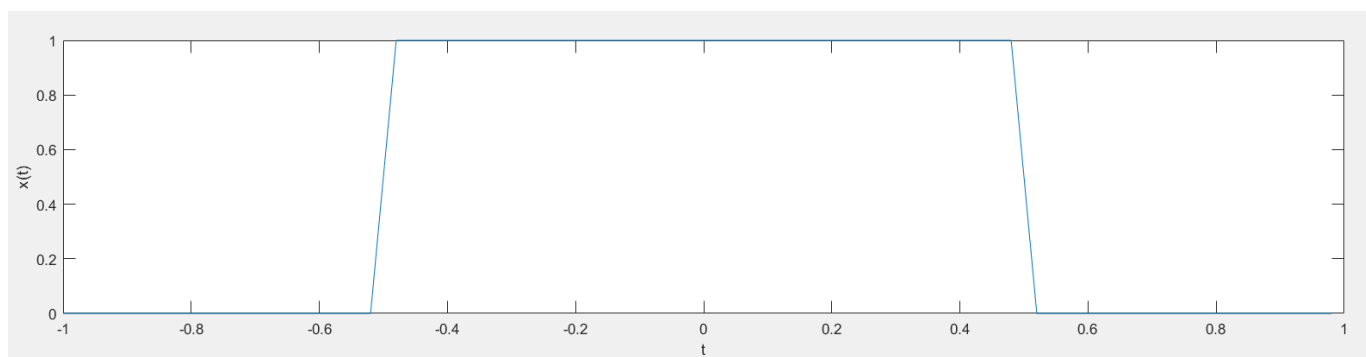
$$x_2(t) = \Pi(t) = \begin{cases} 1; & |t| < \frac{1}{2} \\ \frac{1}{2}; & |t| = \frac{1}{2} \\ 0; & \text{otherwise} \end{cases}$$

می دانیم که :

الف) رسم نمودار سیگنال  $x_2(t)$  :

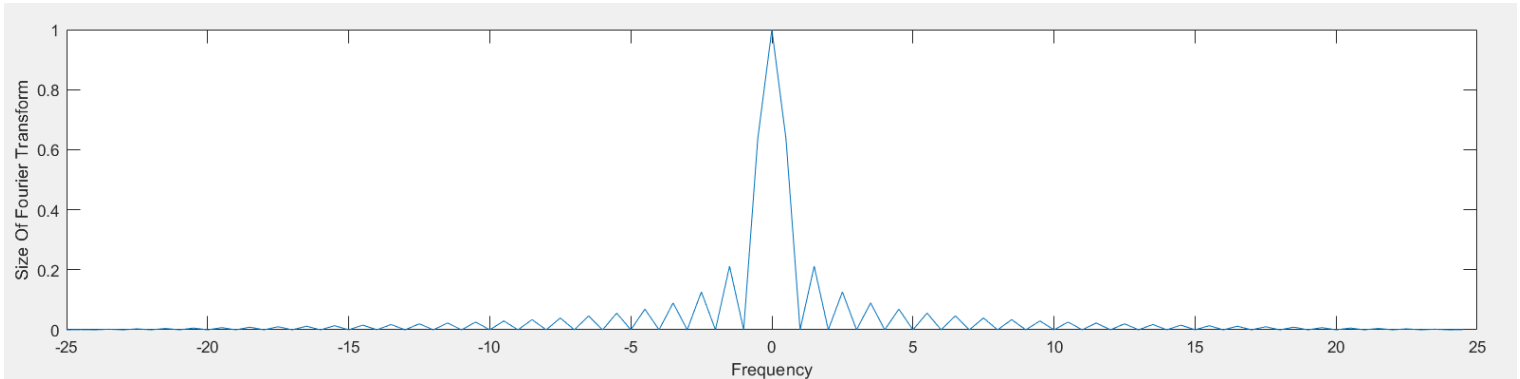
```
%1-2
tstart=-1;
tend=1;
fs=50;
ts=1/fs;

t=tstart:ts:tend-ts;
N=length(t);
x2=rectangularPulse(-1/2, 1/2, t);
subplot(2, 1, 1)
plot(t, x2)
xlabel 't'
ylabel 'x(t)'
```



ب) رسم نمودار اندازه تبدیل فوریه سیگنال  $x_2(t)$ :

```
subplot(2, 1, 2)
freq=-fs/2:fs/N:fs/2-fs/N;
x2f=fftshift(fft(x2));
z2f=abs(x2f)/max(abs(x2f));
plot(freq, z2f)
xlabel 'Frequency'
ylabel 'Size Of Fourier Transform'
%%
```



ج) محاسبه تبدیل فوریه پیوسته  $x_2(t)$  به شکل تئوری:

$$\hat{x}(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t} dt = \int_{-\frac{1}{2}}^{\frac{1}{2}} e^{-j\omega t} dt = \frac{e^{-j\omega t}}{-j\omega} \Big|_{-\frac{1}{2}}^{+\frac{1}{2}} = \frac{e^{j\frac{\omega}{2}} - e^{-j\frac{\omega}{2}}}{j\omega} = \frac{2 \sin\left(\frac{\omega}{2}\right)}{\omega}$$

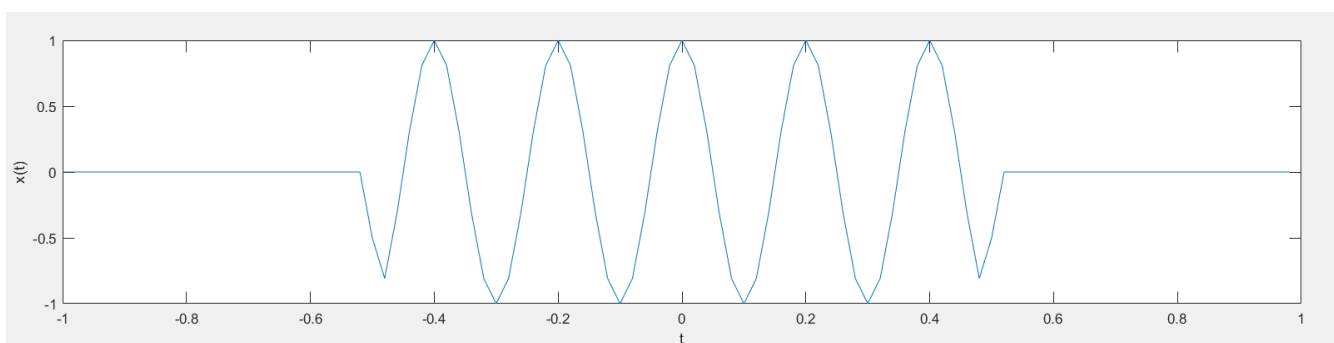
تمرین ۱-۳:

الف) رسم نمودار سیگنال

$$x_3(t) = x_1(t)x_2(t) = \cos(10\pi t)\Pi(t)$$

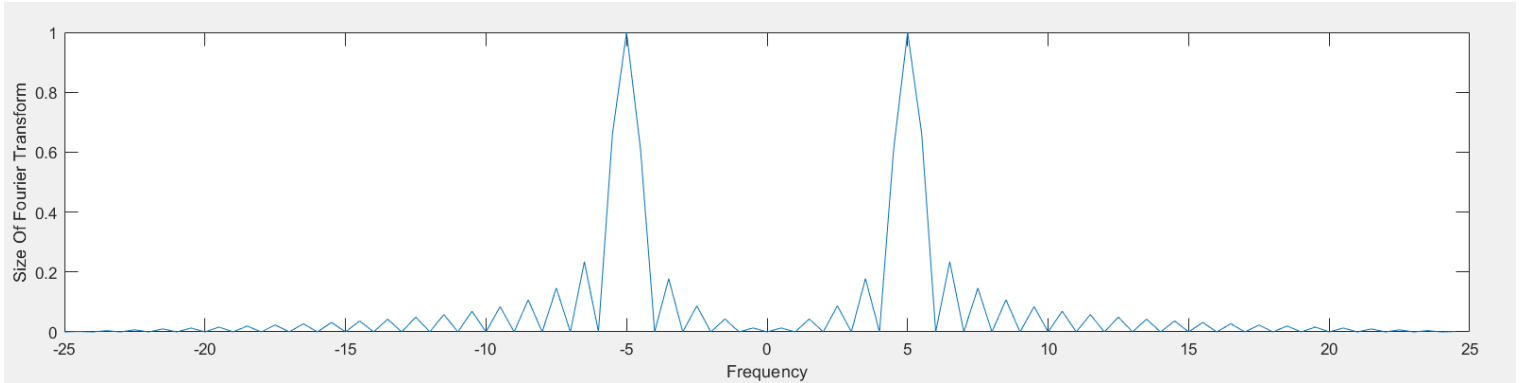
```
%1-3
tstart=-1;
tend=1;
fs=50;
ts=1/fs;

t=tstart:ts:tend-ts;
N=length(t);
x3=cos(2*pi*5*t).*rectangularPulse(-1/2, 1/2, t);
subplot(2, 1, 1)
plot(t, x3)
xlabel 't'
ylabel 'x(t)'
```



ب) رسم نمودار اندازه تبدیل فوریه سیگنال  $x_3(t)$ :

```
subplot(2, 1, 2)
freq=-fs/2:fs/N:fs/2-fs/N;
x3f=fftshift(fft(x3));
z3f=abs(x3f)/max(abs(x3f));
plot(freq, z3f)
xlabel 'Frequency'
ylabel 'Size Of Fourier Transform'
```



ج) محاسبه تبدیل فوریه پیوسته سیگنال  $x_3(t)$  به شکل تئوری:

با توجه به خواص تبدیل فوریه می‌دانیم تبدیل فوریه، ضرب دو تابع را به کانولوشن آن دو تابع تبدیل می‌کند:

$$\hat{x}_3(\omega) = \mathcal{F}\{x_3(t)\} = \mathcal{F}\{x_1(t)x_2(t)\} = \frac{1}{2\pi} (\mathcal{F}\{x_1(t)\} * \mathcal{F}\{x_2(t)\}) = \frac{1}{2\pi} \hat{x}_1(\omega) * \hat{x}_2(\omega)$$

از طرفی در دو سؤال قبلی مقادیر  $\hat{x}_1(\omega)$  و  $\hat{x}_2(\omega)$  محاسبه شده است و داریم:

$$\left. \begin{aligned} \hat{x}_1(\omega) &= \pi(\delta(\omega - 10\pi) + \delta(\omega + 10\pi)) \\ \hat{x}_2(\omega) &= \frac{2 \sin\left(\frac{\omega}{2}\right)}{\omega} \end{aligned} \right\} \Rightarrow \hat{x}_3(\omega) = \frac{2\pi}{2\pi} \int_{-\infty}^{+\infty} (\delta(\tau - \omega - 10\pi) + \delta(\tau - \omega + 10\pi)) \frac{\sin\left(\frac{\tau}{2}\right)}{\tau} d\tau$$

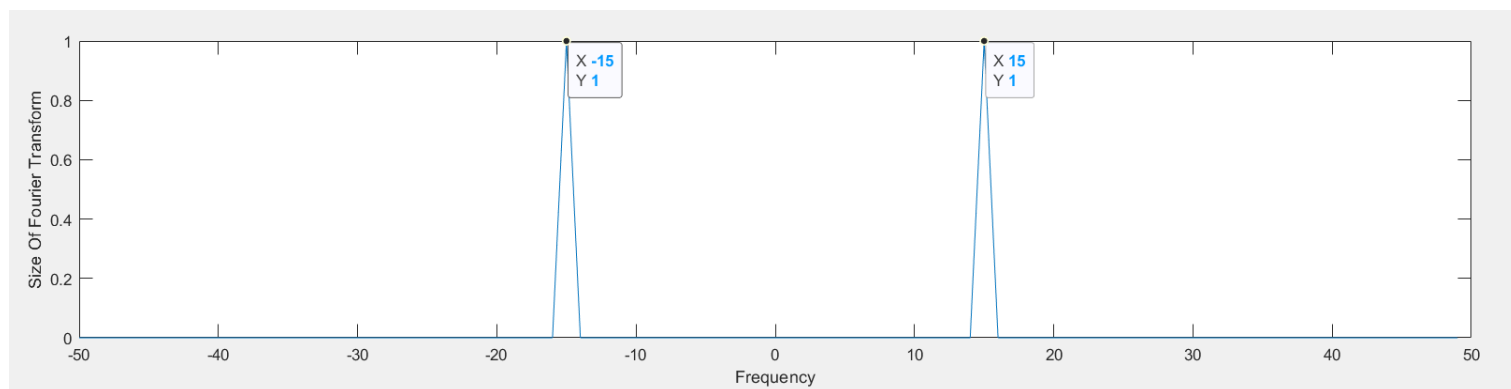
$$\Rightarrow \hat{x}_3(\omega) = \frac{\sin\left(\frac{\omega}{2} + 5\pi\right)}{\omega + 10\pi} + \frac{\sin\left(\frac{\omega}{2} - 5\pi\right)}{\omega - 10\pi} = \frac{2\omega \sin\left(\frac{\omega}{2}\right)}{100\pi^2 - \omega^2}$$

## تمرین ۱-۶ :

(الف) رسم نمودار اندازه تبدیل فوریه سیگنال  $x_4(t) = \cos(30\pi t + \frac{\pi}{4})$  :

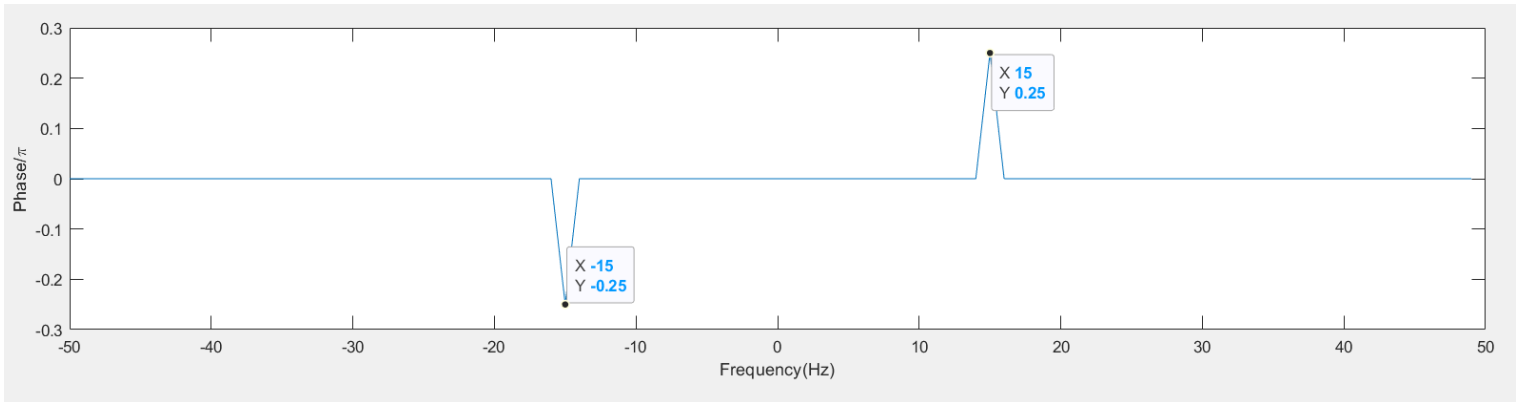
```
%1-4
tstart=0;
tend=1;
fs=100;
ts=1/fs;

t=tstart:ts:tend-ts;
N=length(t);
freq=-fs/2:fs/N:fs/2-fs/N;
x4=cos(2*pi*15*t+pi/4);
x4f=fftshift(fft(x4));
z4f=abs(x4f)/max(abs(x4f));
subplot(2, 1, 1)
plot(freq, z4f)
xlabel 'Frequency'
ylabel 'Size Of Fourier Transform'
```



(ب) رسم نمودار فاز تبدیل فوریه  $x_4(t)$  :

```
subplot(2, 1, 2)
tol=6;
x4f(abs(x4f)<tol)=0;
theta=angle(x4f);
plot(freq, theta/pi)
xlabel 'Frequency(Hz)'
ylabel 'Phase/\pi'
```



(ج) محاسبه تبدیل فوریه  $x_4(t)$  به صورت تئوری :

از آن جایی که سیگنال  $x_4(t) = \cos(30\pi t + \frac{\pi}{4})$  یک سیگنال متناوب است، پس تبدیل فوریه آن از رابطه زیر به دست می آید :

$$\hat{x}(\omega) = \sum_k 2\pi a_k \delta(\omega - \omega_k)$$

که ضرایب سری فوریه  $x_4(t)$  هستند. از طرفی می دانیم سری فوریه  $x_4(t)$  به شکل زیر است :

$$x_4(t) = \frac{1}{2} e^{j(30\pi t + \frac{\pi}{4})} + \frac{1}{2} e^{-j(30\pi t + \frac{\pi}{4})} = \frac{1}{2} (e^{\frac{j\pi}{4}} e^{j(30\pi t)} + e^{-\frac{j\pi}{4}} e^{-j(30\pi t)})$$

هم چنین می دانیم  $\omega = 2\pi f$  پس داریم :

$$2\pi f_1 = 30\pi \Rightarrow f_1 = 15, 2\pi f_2 = -30\pi \Rightarrow f_2 = -15$$

پس تبدیل فوریه  $x_1(t)$  برابر است با :  $\pi(\delta(\omega - 30\pi) + \delta(\omega + 30\pi))$ . دقت شود چون محور افقی، به جای  $\omega$ ،  $f$  است، پس تبدیل فوریه بر حسب  $f$  به این صورت می شود :  $\pi(\delta(f - 15) + \delta(f + 15))$  و چون نمودار  $\frac{|\hat{x}(\omega)|}{\max(|\hat{x}(\omega)|)}$  را رسم کرده ایم پس مقدار بیشینه نمودار به جای  $\pi$ ، 1 است. و فاز تابع  $x_4(t)$  بنا بر تبدیل فوریه اش که در بالا محاسبه شد، در فرکانس 15 هرتز برابر  $\frac{\pi}{4}$  و در فرکانس -15 هرتز برابر  $-\frac{\pi}{4}$  که چون نمودار  $\frac{phase}{\pi}$  را بر حسب فرکانس رسم کرده ایم،  $\frac{\pi}{4}$  و  $-\frac{\pi}{4}$  به  $\frac{1}{4}$  و  $-\frac{1}{4}$  تبدیل می شوند. مقادیری که به شکل تئوری محاسبه شده است با نمودارهای رسم شده در قسمت های الف و ب تطابق دارد.



## تمرین ۱-۵ :

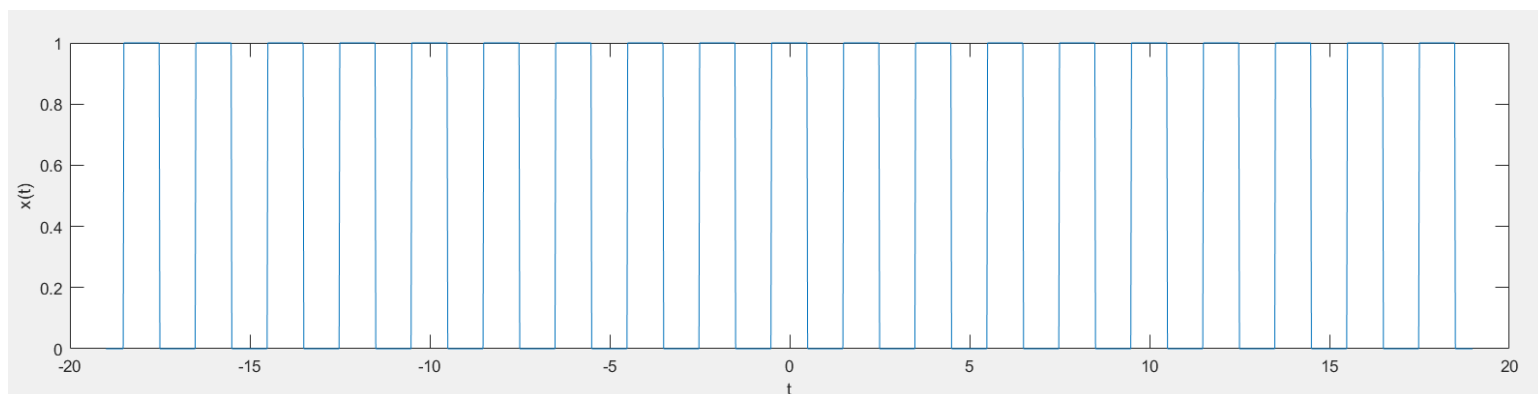
الف) رسم نمودار سیگنال  $x_5(t) = \sum_{k=-9}^{+9} \Pi(t - 2k)$  :

```
%1-5
tstart=-19;
tend=19;
fs=50;
ts=1/fs;
n=9;
t=tstart:ts:tend-ts;
N=length(t);

tmpx5=0;

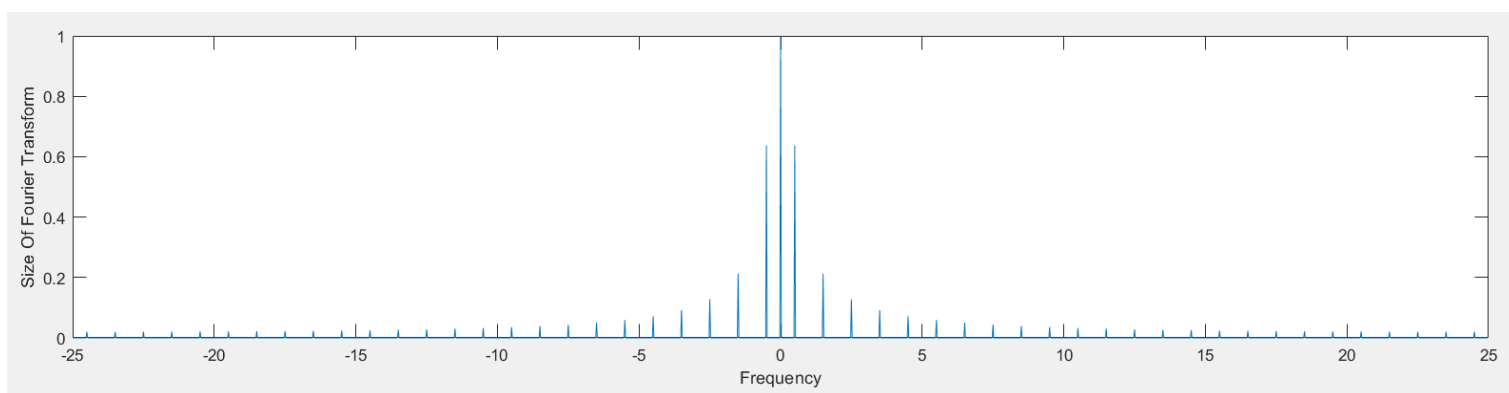
for k=-n:n
    tmpx5=tmpx5+rectpuls(t-2*k);
end

x5=tmpx5;
subplot(2, 1, 1)
plot(t, x5)
xlabel 't'
ylabel 'x(t)'
```



ب) رسم نمودار اندازه تبدیل فوریه سیگنال  $x_5(t)$ :

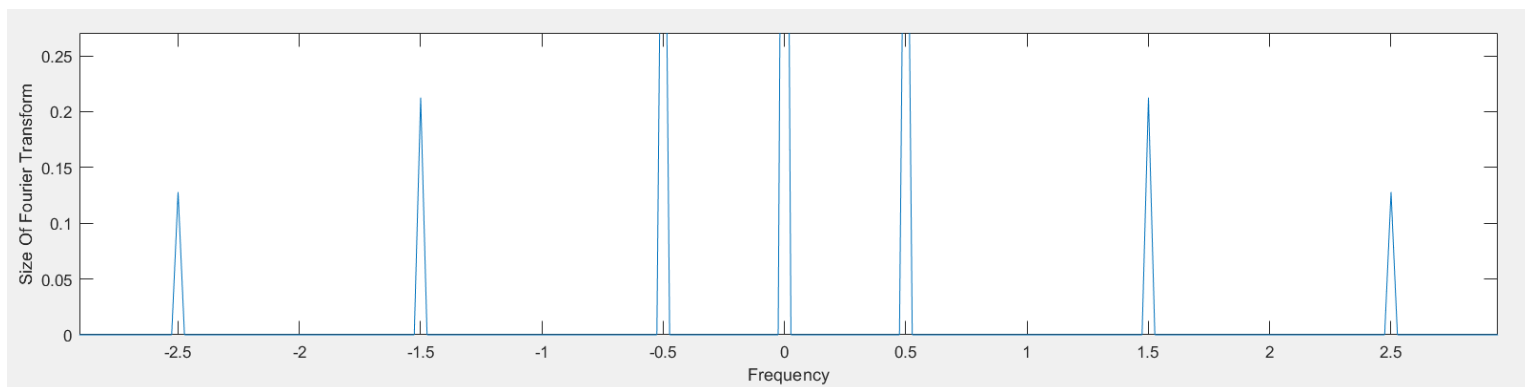
```
subplot(2, 1, 2)
freq=-fs/2:fs/N:fs/2-fs/N;
x5f=fftshift(fft(x5));
z5f=abs(x5f)/max(abs(x5f));
plot(freq, z5f)
xlabel 'Frequency'
ylabel 'Size Of Fourier Transform'
```



ج)

- به این علت تعدادی ضربه در نمودار اندازه تبدیل فوریه  $x_5(t)$  داریم که این سیگنال یک سیگنال متناوب است و تبدیل فوریه سیگنال‌های متناوب به شکل  $\hat{x}(\omega) = \sum_k 2\pi a_k \delta(\omega - \omega_k)$  است که ضرایب سری فوریه سیگنالند و همان طور که می‌بینیم تبدیل فوریه سیگنال‌های متناوب به شکل مجموع تعدادی تابع ضربه است.

- فواصل هر ضربه طبق شکل زیر که بزرگ نمایی شده نمودار اندازه تبدیل فوریه سیگنال  $x_5(t)$  است، برابر 1 است:

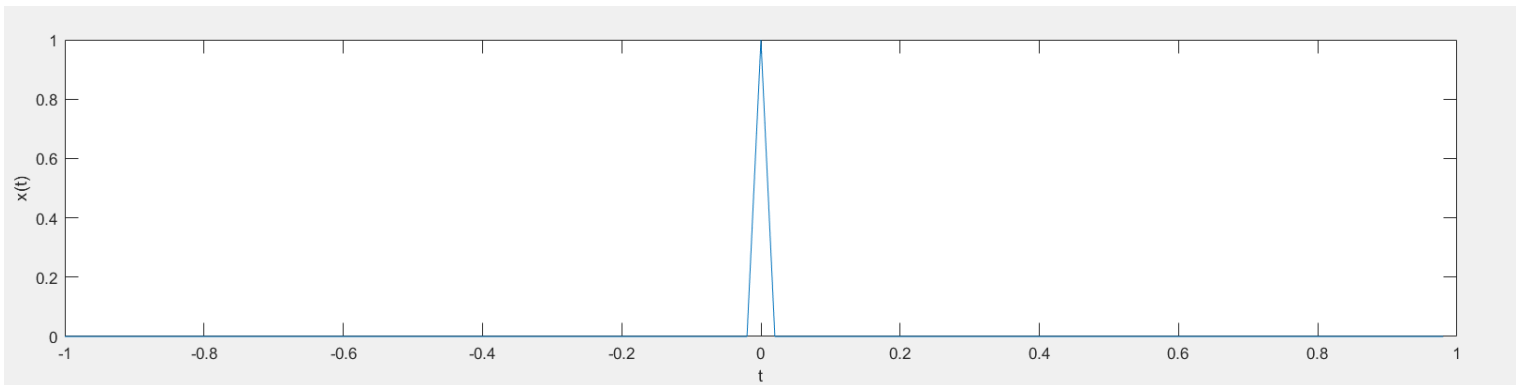


## تمرین ۱-۲ :

الف) رسم نمودار سیگنال  $x_6(t) = \delta(t)$  :

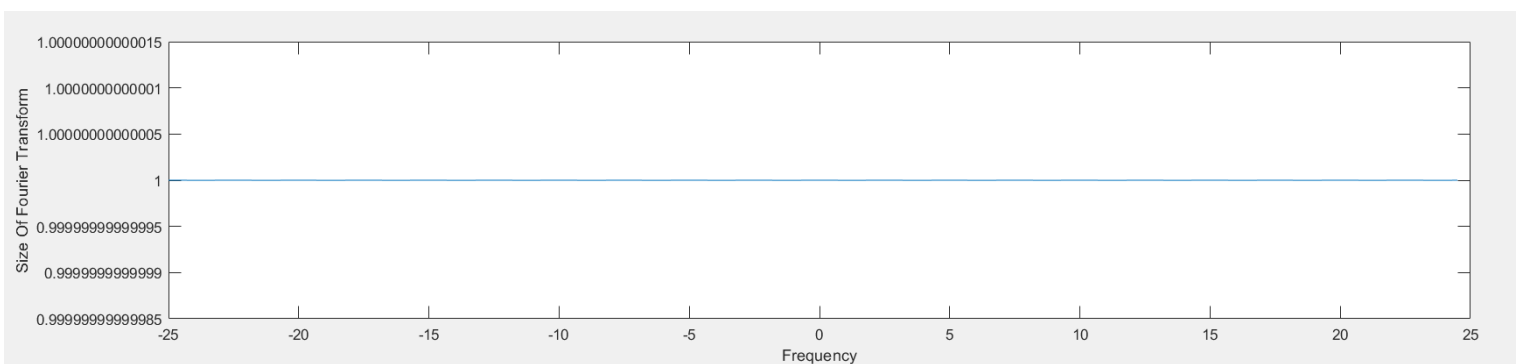
```
%2-1
tstart=-1;
tend=1;
fs=50;
ts=1/fs;

t=tstart:ts:tend-ts;
N=length(t);
x6=dirac(t)>0;
subplot(2, 1, 1)
plot(t, x6)
xlabel 't'
ylabel 'x(t)'
```



ب) رسم نمودار اندازه تبدیل فوریه سیگنال  $x_6(t)$  :

```
subplot(2, 1, 2)
freq=-fs/2:fs/N:fs/2-fs/N;
x6f=fftshift(fft(x6));
z6f=abs(x6f)/max(abs(x6f));
plot(freq, z6f)
xlabel 'Frequency'
ylabel 'Size Of Fourier Transform'
```



ج) محاسبه تبدیل فوریه  $x_6(t)$  به صورت تئوری :

$$\hat{x}(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t} dt = \int_{-\infty}^{+\infty} \delta(t)e^{-j\omega t} dt = e^{-j\omega 0} = 1$$

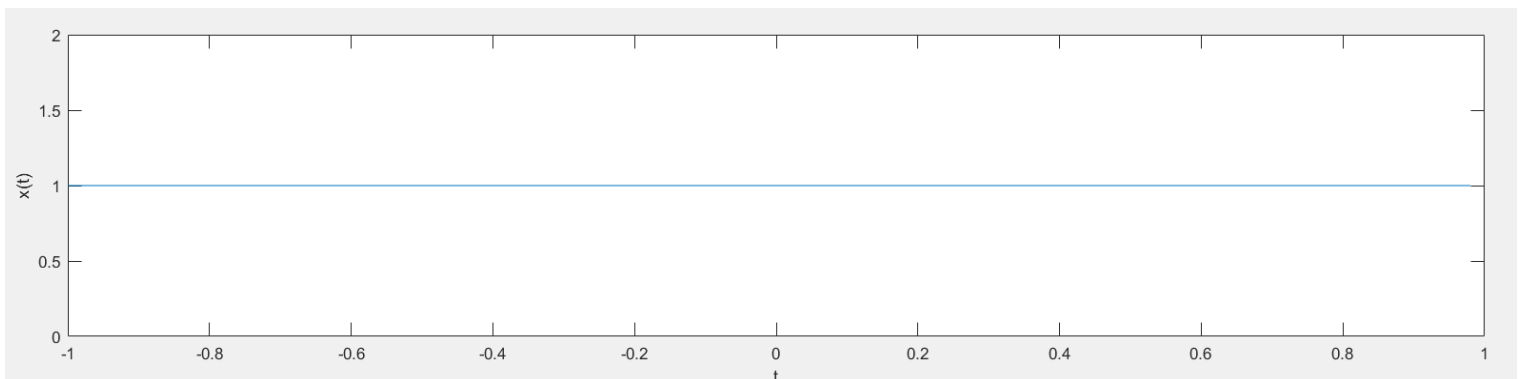
با توجه به خواص تبدیل فوریه می‌دانیم هر چقدر در حوزه زمان، انبساط بیشتری داشته باشیم، در حوزه فرکانس، انقباض (فشردگی) بیشتری خواهیم و فرکانس‌های کمتری در ساخت تبدیل فوریه مشارکت دارند. هم‌چنین هر چقدر در حوزه زمان، انقباض (فشردگی) بیشتری داشته باشیم در حوزه فرکانس، انبساط بیشتری خواهیم داشت و فرکانس‌های بیشتری در ساخت تبدیل فوریه مشارکت می‌کنند. تابع ضربه، در یک نقطه، فشرده شده است. پس بیشترین فشردگی ممکن را دارد. پس در حوزه فرکانس برای توصیف تابع ضربه به بیشترین انبساط نیاز داریم به همین علت تمامی فرکانس‌ها در ساخت تبدیل فوریه تابع ضربه مشارکت می‌کنند و تبدیل فوریه آن مقدار ثابت ۱ را دارد.

## تمرین ۲-۲ :

الف) رسم نمودار سیگنال  $x_7(t) = 1$  :

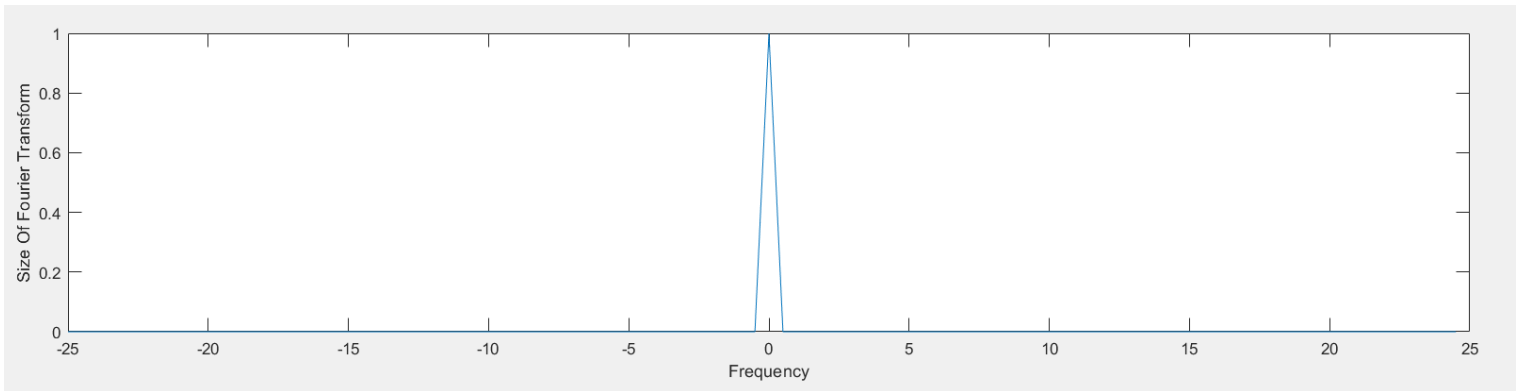
```
%2-2
tstart=-1;
tend=1;
fs=50;
ts=1/fs;

t=tstart:ts:tend-ts;
N=length(t);
x7=ones(1, N);
subplot(2, 1, 1)
plot(t, x7)
xlabel 't'
ylabel 'x(t)'
```



ب) رسم نمودار اندازه تبدیل فوریه سیگنال  $x_7(t)$  :

```
subplot(2, 1, 2)
freq=-fs/2:fs/N:fs/2-fs/N;
x7f=fftshift(fft(x7));
z7f=abs(x7f)/max(abs(x7f));
plot(freq, z7f)
xlabel 'Frequency'
ylabel 'Size Of Fourier Transform'
```



ج) محاسبه تبدیل فوریه  $x_7(t)$  به صورت تئوری :

$$\hat{x}(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t} dt = \int_{-\infty}^{+\infty} 1e^{-j\omega t} dt$$

در سؤال قبل محاسبه کردیم که تبدیل فوریه تابع ضربه برابر ۱ می شود پس می توان نوشت :

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \hat{x}(\omega)e^{j\omega t} d\omega \Rightarrow \delta(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} 1e^{j\omega t} d\omega \Rightarrow \int_{-\infty}^{+\infty} e^{j\omega t} d\omega = 2\pi\delta(t)$$

$$\Rightarrow \int_{-\infty}^{+\infty} e^{-j\omega t} dt = 2\pi\delta(-\omega) = 2\pi\delta(\omega) \text{ (چون تابع ضربه یک تابع زوج است)}$$

با توجه به خواص تبدیل فوریه می دانیم هر چقدر در حوزه زمان، انبساط بیشتری داشته باشیم، در حوزه فرکانس، انقباض (فشرده‌گی) بیشتری خواهیم و فرکانس‌های کمتری در ساخت تبدیل فوریه مشارکت دارند. هم چنین هر چقدر در حوزه زمان، انقباض (فشرده‌گی) بیشتری داشته باشیم در حوزه فرکانس، انبساط بیشتری خواهیم داشت و فرکانس‌های بیشتری در ساخت تبدیل فوریه مشارکت می کنند. تابع ثابت  $x_7(t) = 1$ ، یک تابع با بیشترین انبساط ممکن است. پس در حوزه فرکانس برای توصیف این تابع به کمترین انبساط نیاز داریم به همین علت کافی است یک فرکانس در ساخت تبدیل فوریه مشارکت کند و در نتیجه تبدیل فوریه آن برابر تابع ضربه می شود.

### تمرین ۱-۲ :

درست کردن سلول *Mapset* با استفاده از کد مقابل:

```
%3-1
charactersCount=32;
characters=['a':'z' ' ' ' ' '.' ',' '!' ';' ' "''];
Mapset=cell(2, charactersCount);
for i=0:charactersCount-1
    Mapset{1, i+1}=characters(i+1);
    Mapset{2, i+1}=dec2bin(i, 5);
end
```

### تمرین ۲-۲ :

کد مربوط به تابع `coding_amp` در صفحه بعد آورده شده است ولی ورودی ها و خروجی هایش تغییراتی نسبت به آن چه در صورت پروژه گفته شده دارد که در ادامه به توضیح این تغییرات می پردازیم:

- به ورودی ها *Mapset*، *noisePower* و پرچم *test* هم اضافه شده اند که :
- از *Mapset* برای تشخیص کد باینری هر کاراکتر استفاده می شود. (می توانستیم از *Mapset* به شکل سراسری هم استفاده کنیم ولی طبق هشدار خود متلب و قواعد برنامه نویسی بهتر است از متغیرهای سراسری استفاده نشود. به همین خاطر آن را به تابع پاس می دهیم.)
- اضافه کردن *noisePower* به ورودی ها به این علت بود که در قسمت های بعدی که می خواهیم سیگنال کد شده را *noisy* کنیم، خود تابع *coding\_amp* این کار را برای ما انجام دهد و وقتی نمی خواهیم سیگنال کد شده را *noisy* کنیم، کافی است *noisePower* را صفر کنیم.
- اضافه کردن *test* هم به این خاطر است که وقتی می خواهیم در بخش های بعدی آستانه واریانس را به دست آوریم، صد بار آزمایش می کنیم و خب در این صد بار نمی خواهیم نموداری بکشیم و با این پرچم به تابع اطلاع می دهیم که نمودار را بکشد یا خیر.
- به خروجی ها طول پیام و این که تعداد بیت های استفاده شده در پیام کد شده به سرعت ارسال بیت ها در هر ثانیه بخش پذیر است یا خیر اضافه شده است که از این دو ویژگی برای تشخیص پیام کد شده در تابع *decoding\_amp* به ازای هر سرعت ارسال بیت دلخواهی استفاده می شود. (برای آن که تابع *decoding\_amp* به درستی پیام را تشخیص دهد به طول پیام نیاز دارد از طرفی در هنگام کد کردن پیام چون در هر ثانیه تعداد بیت های ارسالی مقدار گسسته دارند، ممکن است تعداد بیت های تشکیل دهنده پیام به ۵ بخش پذیر نباشد. به عنوان مثال اگر همین کلمه *signal* پیام باشد. میدانیم با ۳۰ بیت کد می شود ولی اگر مثلاً سرعت ارسال بیت ها ۱۳ بیت در هر ثانیه باشد، برای کد کردن آن از ۳۹ بیت استفاده می کنیم. حال در تابع *decoding\_amp* برای تشخیص پیام به طول آن احتیاج داریم که اگر ۳۹ را به ۵ تقسیم کنیم، به طول اشتباهی می رسیم! به همین خاطر طول پیام ارسالی را در اینجا خروجی می دهیم و در تابع *decoding\_amp* از آن استفاده می کنیم.)

**شیوه کد کردن پیام:** با توجه به توضیحاتی که در مقدمه سوم آورده شده است، تابعی که پیام را کد می‌کند به شکل زیر است:

$$\sum_{i=0}^{tend} \frac{bin2dec(bits)}{2^{bitRate} - 1} \sin(2\pi t) [u(t-i) - u(t-i-1)]$$

$$tend = \left\lceil \frac{totalBits}{bitRate} \right\rceil$$

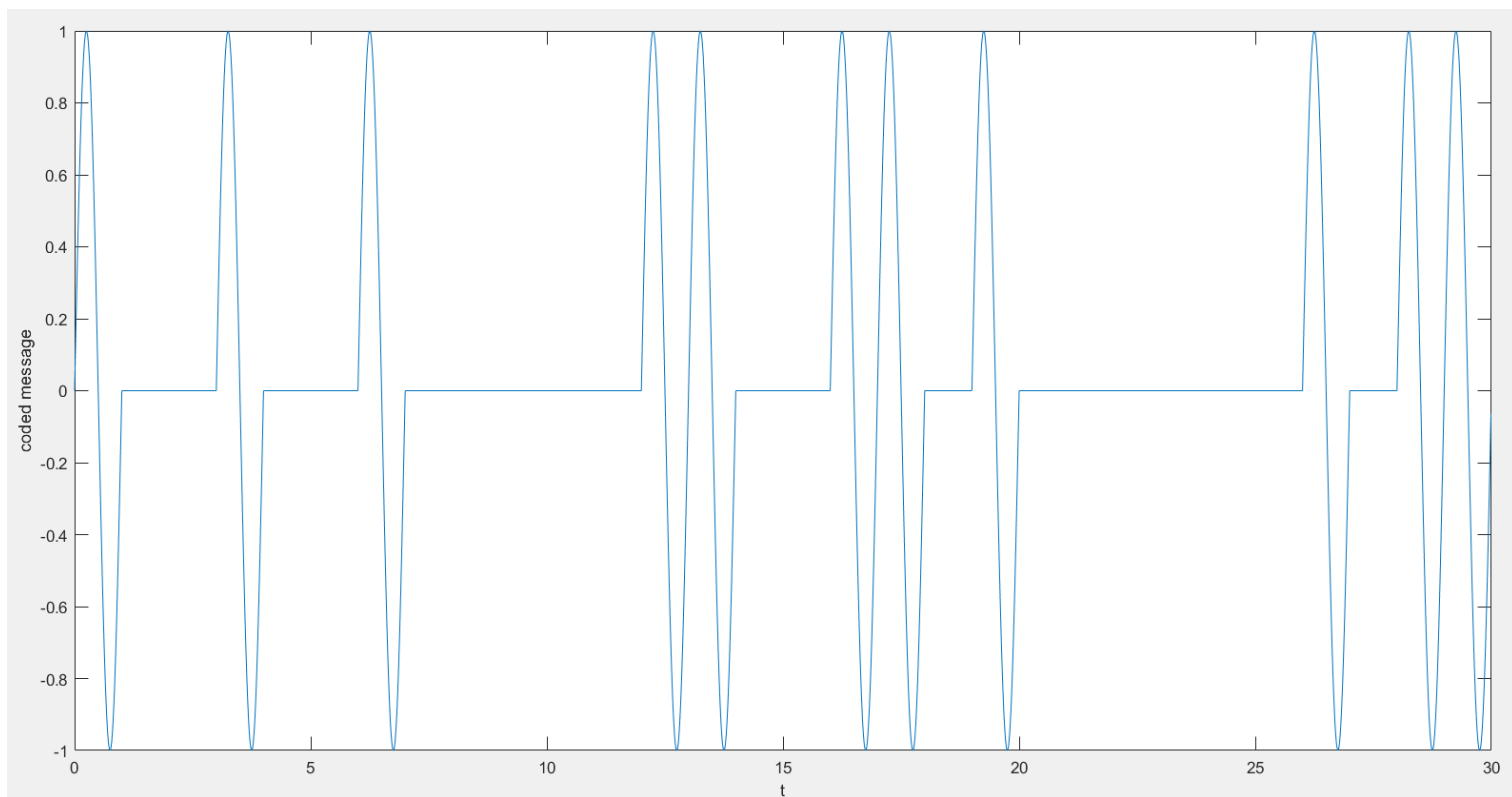
که *bits* در هر ثانیه روی ۵ بیت از تعداد بیت های تشکیل دهنده پیام کد شده جلو می رود. خطوط ۳۳ تا ۴۰ کد صفحه بعد پیاده سازی این تابع در متلب است.

---

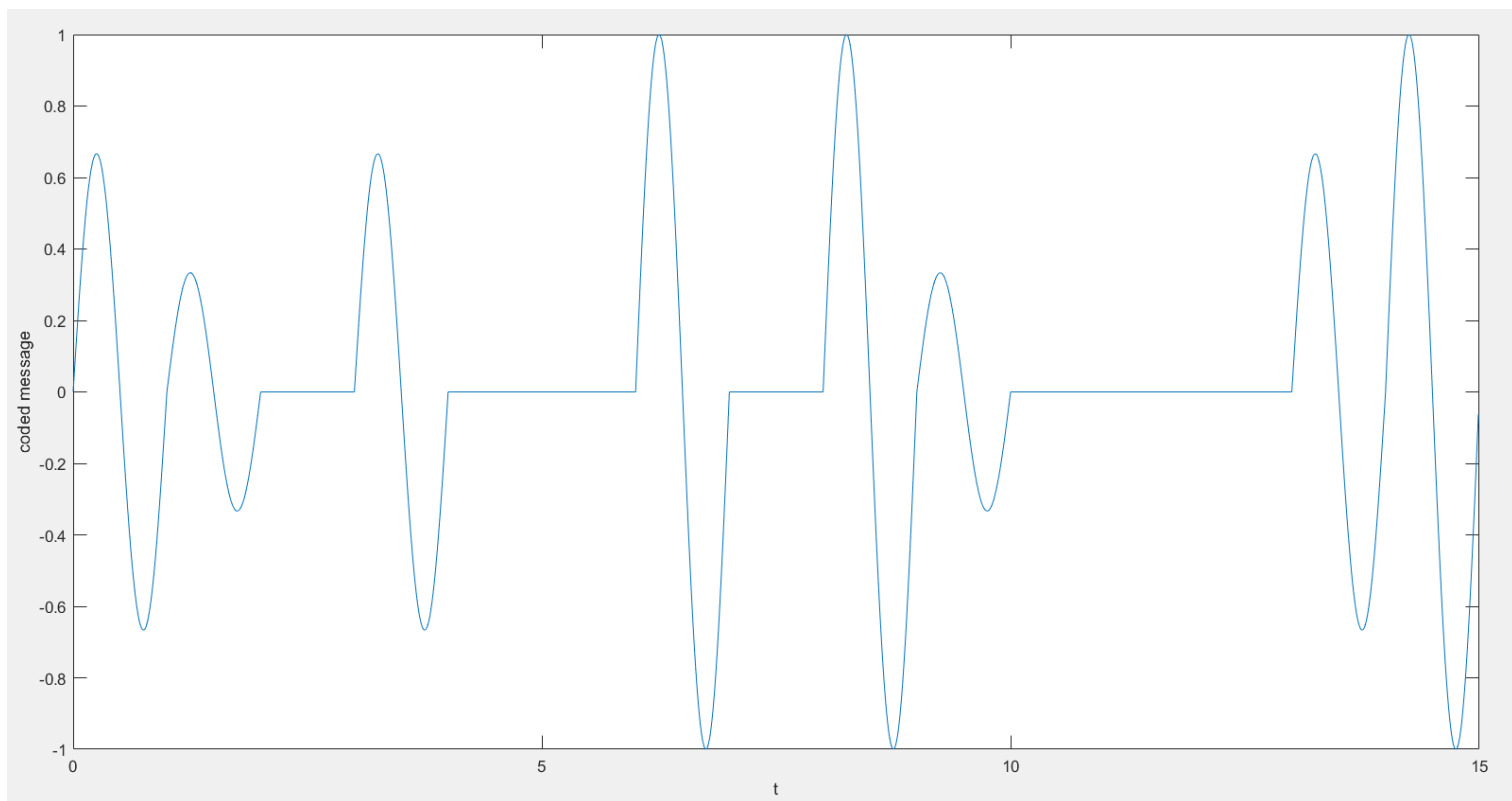
```
1 function [codedMessage, divisibleByBitRate, messageLength] = coding_amp(message, bitRate, Mapset, noisePower, test)
2     charactersCount = 32;
3     messageLength = strlen(message);
4     totalBitsCount = 5*messageLength;
5
6     if rem(totalBitsCount, bitRate)==0
7         divisibleByBitRate=true;
8     else
9         divisibleByBitRate=false;
10    end
11
12    charactersToBits = strings(1, messageLength);
13    signalsCount = 2^bitRate;
14
15    for i=1:messageLength
16        for j=1:charactersCount
17            if message(i) == Mapset{1, j}
18                charactersToBits(i) = Mapset(2, j);
19                break;
20            end
21        end
22    end
23
24    charactersToBits=join(charactersToBits, '');
25    charactersToBits=char(charactersToBits);
26    tstart=0;
27    tend=ceil(totalBitsCount/bitRate);
28    fs=100;
29    ts=1/fs;
30    t=tstart:ts:tend-ts;
31
32    codedMessage=0;
33    for i=0:tend-1
34        if (i+1)*bitRate<=totalBitsCount
35            bits=charactersToBits(i*bitRate+1:(i+1)*bitRate);
36        else
37            bits = charactersToBits(i*bitRate+1:totalBitsCount);
38        end
39        codedMessage=codedMessage+(bin2dec(bits)/(signalsCount-1))*sin(2*pi*t).*(heaviside(t-i)-heaviside(t-i-1));
40    end
41
42    codedMessage=codedMessage+noisePower*randn(1, length(codedMessage));
43
44    if ~test
45        plot(t, codedMessage)
46        xlabel 't'
47        ylabel 'coded message'
48    end
49
50 end
```

## تمرین ۳-۲ :

خروجی تابع *coding\_amp* برای کلمه *signal* با سرعت ارسال اطلاعات یک بیت بر ثانیه :

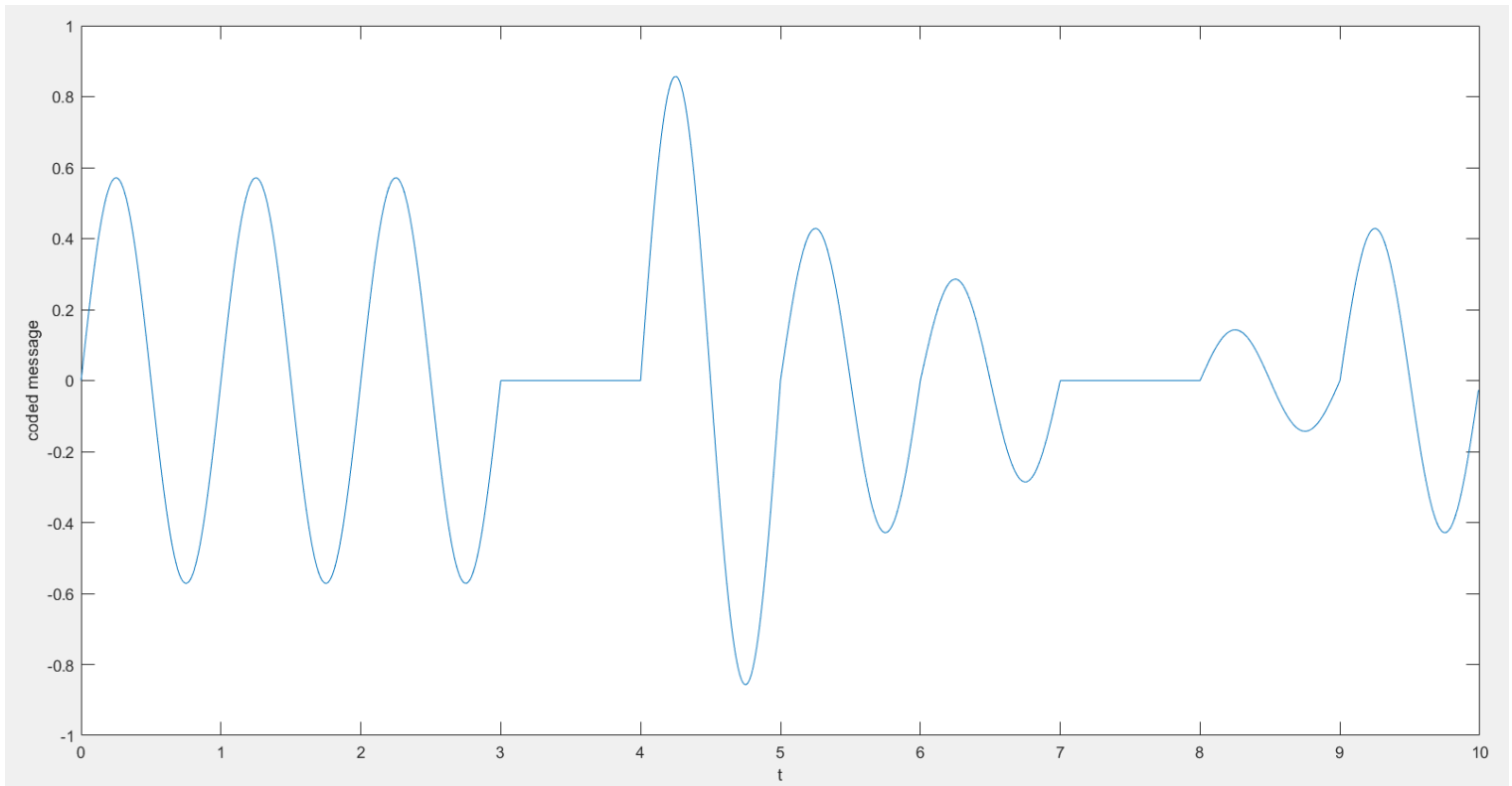


خروجی تابع *coding\_amp* برای کلمه *signal* با سرعت ارسال اطلاعات دو بیت بر ثانیه :





خروجی تابع `coding_amp` برای کلمه `signal` با سرعت اطلاعات سه بیت بر ثانیه :



### تمرین ۲-۴ :

کد مربوط به تابع `decoding_amp` در ۳ صفحه بعدی آورده شده است ولی ورودی ها تغییراتی نسبت به آن چه در صورت پروژه گفته شده دارد که به توضیح این تغییرات می پردازیم:

- به ورودی‌ها *Mapset*، *divisibleByBitRate*، *messageLength* و *noisePower* اضافه شده است. که مشابه با توضیحاتی که در تمرین ۲-۳ آورده شده است، اضافه شدن *messageLength* و *divisibleByBitRate* برای این است که بتوانیم سرعت ارسال بیت در ثانیه دلخواه داشته باشیم.
- اضافه کردن *Mapset* برای تشخیص پیام از روی بیت هاست.
- اضافه کردن *noisePower* برای این است که بدانیم پیام *noise* دارد یا خیر. چون در ازای *noise* داشتن یا نداشتن پیام تعریف *threshold* فرق می‌کند.

---

```
function decodedMessage = decoding_amp(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower)
    fs=100;
    tend=length(codedMessage)/fs;
    tstart=0;
    ts=1/fs;
    t=tstart:ts:tend-ts;
    coefficientsCount=2^bitRate;
    charactersCount=32;

    xt=2*sin(2*pi*t);

    product=codedMessage.*xt;
    correlationResults=zeros(1, tend);

    for i=1:tend
        correlationResults(i)=0.01*sum(product(((i-1)*fs)+1:i*fs));
    end

    coefficients=zeros(1, coefficientsCount);

    for i=0:coefficientsCount-1
        coefficients(i+1)=i/(coefficientsCount-1);
    end

    messageBits=strings(1, tend);
    decodedMessage=strings(1, messageLength);
```

```

if noisePower==0
    threshold=1/(((2^bitRate)-1)*2);
    for i=1:tend
        for j=1:coefficientsCount
            if abs(correlationResults(i)-coefficients(j))<threshold
                if ~divisibleByBitRate && i==tend
                    messageBits(i)=dec2bin(j-1, 5*messageLength-bitRate*(i-1));
                else
                    messageBits(i)=dec2bin(j-1, bitRate);
                end
                break;
            end
        end
    end
else
    for i=1:tend
        diff=1000;
        isEmpty=true;
        for j=1:coefficientsCount-1
            tmpdiff1=abs(correlationResults(i)-coefficients(j));
            tmpdiff2=abs(correlationResults(i)-coefficients(j+1));
            if tmpdiff2<tmpdiff1 && tmpdiff2<diff
                default=j;
                diff=tmpdiff2;
            end
            if tmpdiff1<tmpdiff2 && tmpdiff1<diff
                default=j-1;
                diff=tmpdiff1;
            end
            threshold=(coefficients(j)+coefficients(j+1))/2;
            if correlationResults(i)>=coefficients(j) && correlationResults(i)<=threshold
                if ~divisibleByBitRate && i==tend
                    messageBits(i)=dec2bin(j-1, 5*messageLength-bitRate*(i-1));
                else
                    messageBits(i)=dec2bin(j-1, bitRate);
                end
                isEmpty=false;
                break;
            end
            if correlationResults(i)<=coefficients(j+1) && correlationResults(i)>threshold
                if ~divisibleByBitRate && i==tend
                    messageBits(i)=dec2bin(j, 5*messageLength-bitRate*(i-1));
                else
                    messageBits(i)=dec2bin(j, bitRate);
                end
                isEmpty=false;
                break;
            end
        end
    end
    if isEmpty
        if ~divisibleByBitRate && i==tend
            messageBits(i)=dec2bin(default, 5*messageLength-bitRate*(i-1));
        else
            messageBits(i)=dec2bin(default, bitRate);
        end
    end
end
end
end

```

```

messageBits=join(messageBits, '');
messageBits=char(messageBits);

for i=1:messageLength
    for j=1:charactersCount
        if Mapset{2, j} == messageBits((i-1)*5+1:(i*5))
            decodedMessage(i)=Mapset{1, j};
            break;
        end
    end
end
end
end

```

خروجی تابع به ازای هر سه سرعت یک، دو و سه بیت بر ثانیه یکسان و صحیح بود:

$bitRate = 1$  -

```

bitRate=1;
noisePower=0;
message='signal';
test=false;

if ~test
    [codedMessage, divisibleByBitRate, messageLength]=coding_amp(message, bitRate, Mapset, noisePower, test);
    decodedMessage = decoding_amp(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %d\n', accuracyRate);
end

```

Command Window

```

decoded message is:
signal

```

*fx* >>

*bitRate* = 2 -

```
bitRate=2;
noisePower=0;
message='signal';
test=false;

if ~test
    [codedMessage, divisibleByBitRate, messageLength]=coding_amp(message, bitRate, Mapset, noisePower, test);
    decodedMessage = decoding_amp(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %d\n', accuracyRate);
end
```

Command Window

decoded message is:

signal

*fx* >>

*bitRate* = 3 -

```
bitRate=3;
noisePower=0;
message='signal';
test=false;

if ~test
    [codedMessage, divisibleByBitRate, messageLength]=coding_amp(message, bitRate, Mapset, noisePower, test);
    decodedMessage = decoding_amp(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %d\n', accuracyRate);
end
```

Command Window

decoded message is:

signal

*fx* >>

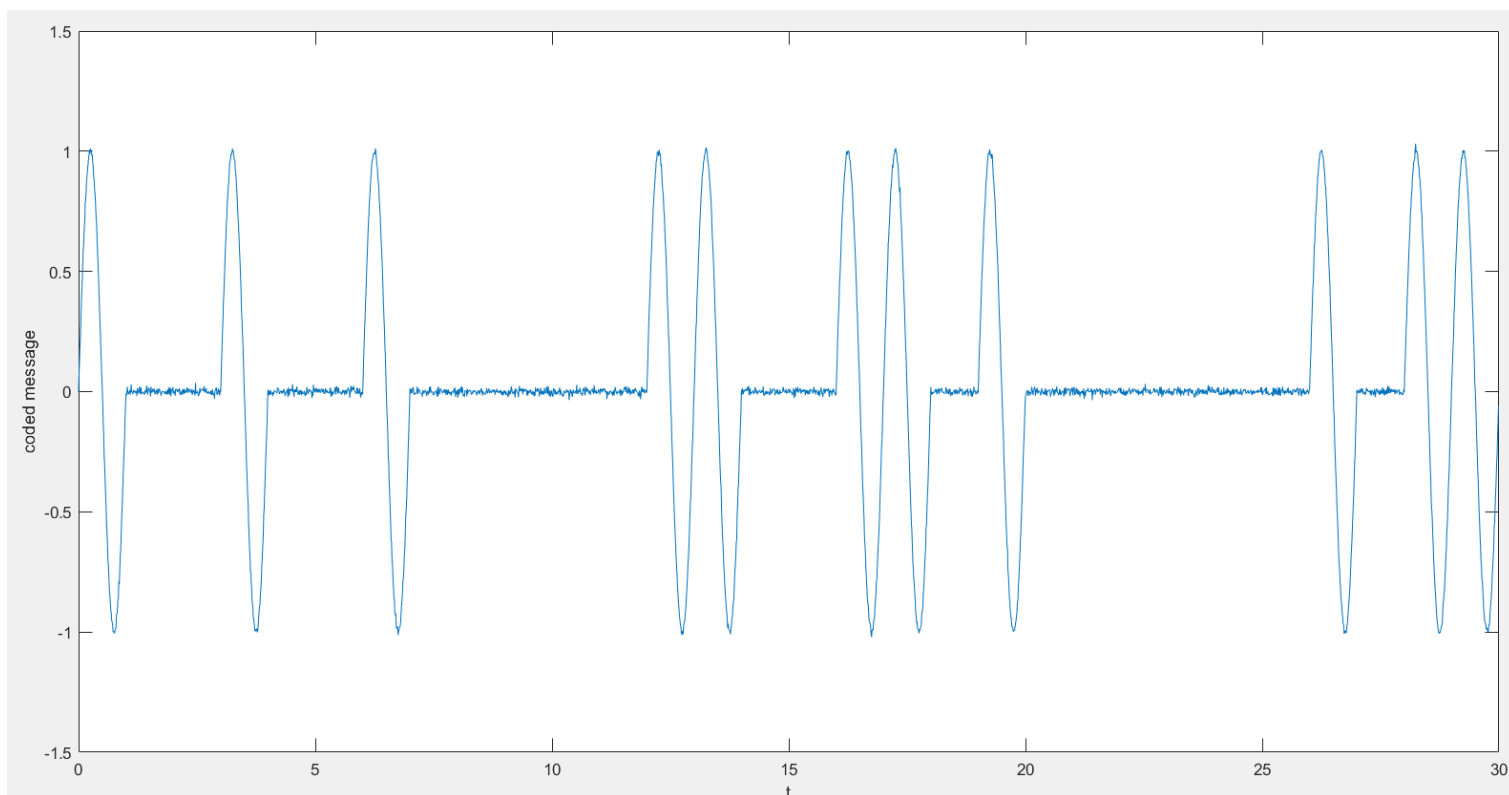
### تمرین ۳-۵:

اضافه کردن نویز گوسی با واریانس 0.0001 و میانگین 0 به ازای سرعت ارسال بیت در ثانیه:

$$bitRate = 1 \quad -$$

```
bitRate=1;
noisePower=0.01;
message='signal';
test=false;

if ~test
    [codedMessage, divisibleByBitRate, messageLength]=coding_amp(message, bitRate, Mapset, noisePower, test);
    decodedMessage = decoding_amp(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %d\n', accuracyRate);
end
```



Command Window

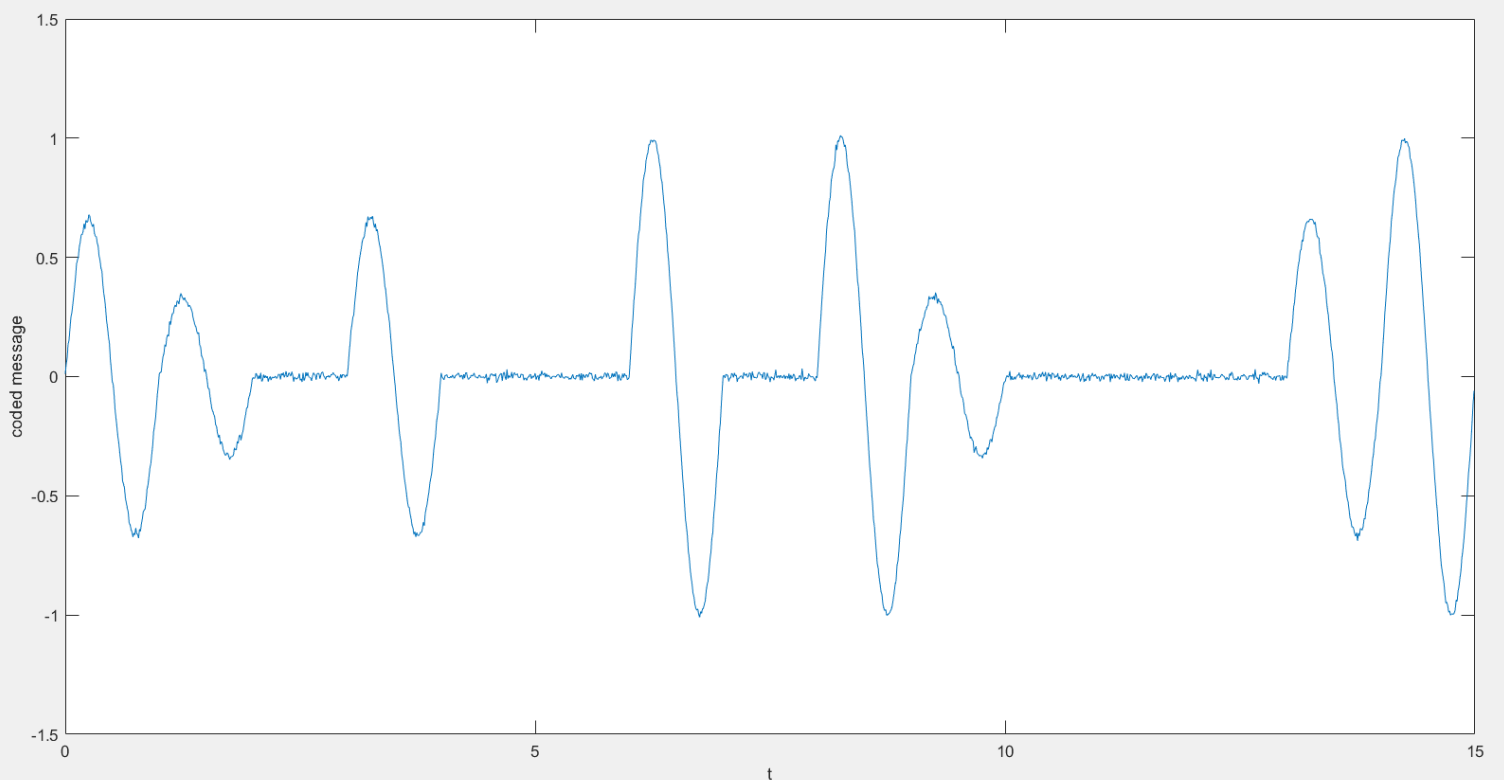
```
decoded message is:
signal
```

```
fx >>
```

$bitRate = 2$  -

```
bitRate=2;
noisePower=0.01;
message='signal';
test=false;

if ~test
    [codedMessage, divisibleByBitRate, messageLength]=coding_amp(message, bitRate, Mapset, noisePower, test);
    decodedMessage = decoding_amp(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %d\n', accuracyRate);
end
```



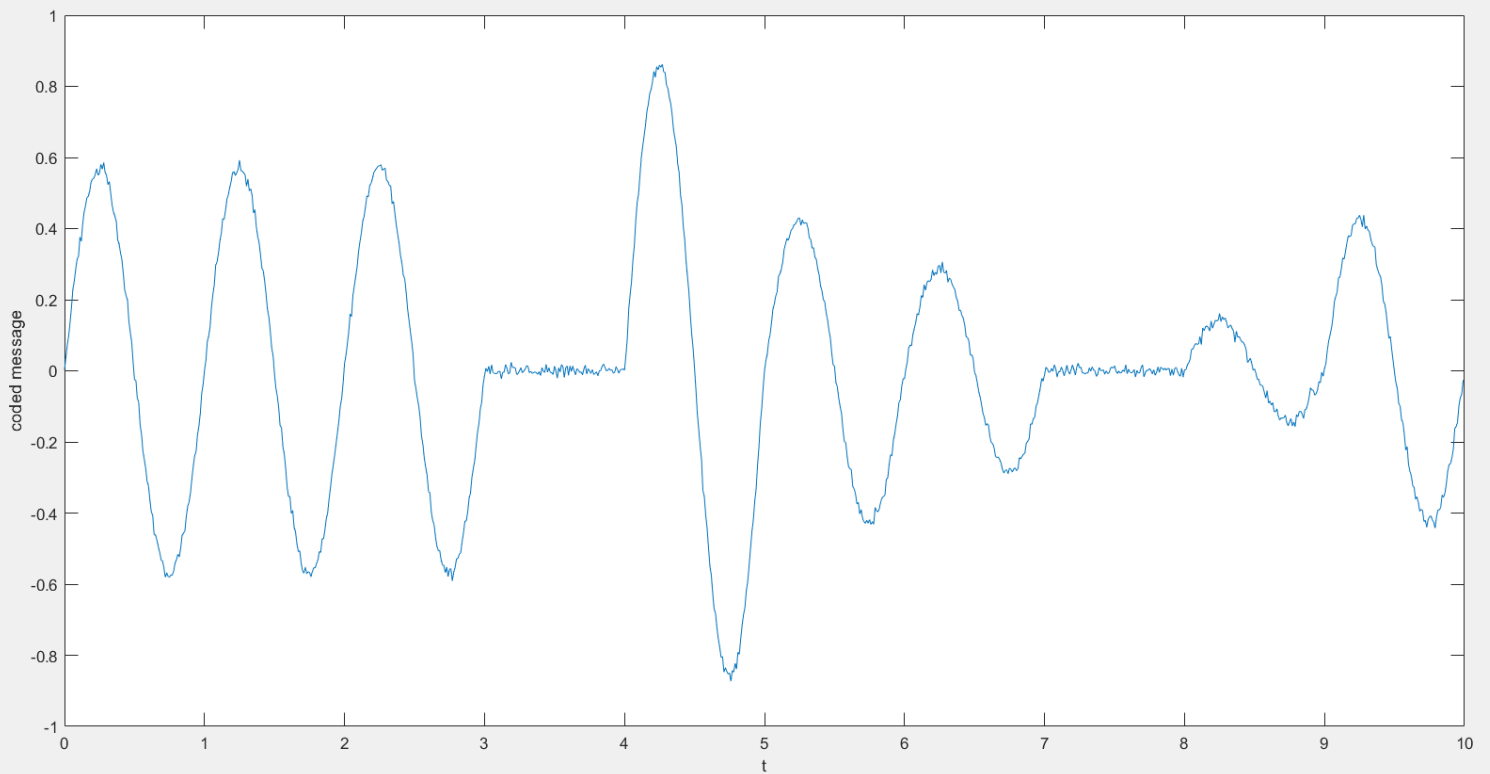
Command Window

```
decoded message is:
signal
fx >>
```

*bitRate = 3* -

```
bitRate=3;
noisePower=0.01;
message='signal';
test=false;

if ~test
    [codedMessage, divisibleByBitRate, messageLength]=coding_amp(message, bitRate, Mapset, noisePower, test);
    decodedMessage = decoding_amp(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %d\n', accuracyRate);
end
```



Command Window

```
decoded message is:
signal
```

```
fx >>
```



## تمرین ۳-۶ :

نحوه به دست آوردن مقدار بیشترین واریانس : در این قسمت به ازای یک مقدار  $noise$ ، ۱۰۰ بار کد را اجرا می‌کنیم تا ببینیم در این ۱۰۰ بار، چند بار پیام را به درستی تشخیص می‌دهد. ابتدا با یک  $noise$  کم شروع به آزمایش کردیم و در ادامه مقدار  $noise$  را زیاد کردیم تا وقتی که در ۱۰۰ بار اجرای کد، حداقل یک بار پیام به اشتباه تشخیص داده شد. این که در هر ۱۰۰ بار آزمایش چند بار پیام به اشتباه تشخیص داده می‌شود، توسط تابع  $calc\_accuracy$  انجام می‌شود و برای اجرای ۱۰۰ بار آزمایش همان طور که پیش‌تر در تمرین ۳-۲ اشاره شد، کافی است در ابتدای کد، پرچم  $test$  را  $true$  کنیم :

```
bitRate=1;
noisePower=0;
message='signal';
test=true;

if ~test
    [codedMessage, divisibleByBitRate, messageLength]=coding_amp(message, bitRate, Mapset, noisePower, test);
    decodedMessage = decoding_amp(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %d\n', accuracyRate);
end
```

```
function accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test)
    mistakesCount=0;
    for i=1:100
        [codedMessage, divisibleByBitRate, messageLength]=coding_amp(message, bitRate, Mapset, noisePower, test);
        decodedMessage = decoding_amp(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower);
        decodedMessage=join(decodedMessage, '');
        decodedMessage=char(decodedMessage);
        if decodedMessage~=message
            mistakesCount=mistakesCount+1;
        end
    end
    accuracyRate=(100-mistakesCount);
end
```

برای  $bitRate = 1$  در  $noisePower = 2.5$  به اولین جایی رسیدیم که  $accuracy \neq 100$  بود. (کد را به شکل دستی اجرا کردیم و در هر مرحله با شروع از  $noisePower = 0$ ،  $noisePower$  را ۰.۱ زیاد کردیم. به علت زیاد بودن تعداد مراحل، عکس آن‌ها را در گزارش نمی‌آوریم و اگر مایل بودید، خودتان می‌توانید تست کنید 😊) در ادامه  $accuracy$  را به ازای همین  $noisePower = 2.5$  برای  $bitRate$  های متفاوت به دست می‌آوریم تا نشان دهیم، هر چه سرعت ارسال بیت‌ها کمتر باشد، مقاومت نسبت به  $noise$  بیشتر خواهد بود و سرعت‌های بیشتر،  $accuracy$  کمتری دارند. هم‌چنین برای به دست آوردن مقدار بیشترین واریانس در دیگر سرعت‌ها از همین روش استفاده می‌کنیم. یعنی با شروع از  $noisePower = 0$ ،  $noisePower$  را در هر مرحله ۰.۱ زیاد می‌کنیم تا به  $accuracy$  کمتر از ۱۰۰ برسیم. دقت شود عددی که به عنوان بیشترین واریانس اعلام می‌شود تقریبی است و در آزمایش‌های متفاوت ممکن است به اعداد دیگری برسیم ولی با تقریب خوبی در همین حدود است.

#### Command Window

```
noise power = 2.50
bit rate = 1
accuracy = 99
```

**fx** >>

#### Command Window

```
noise power = 2.50
bit rate = 2
accuracy = 81
```

**fx** >>

#### Command Window

```
noise power = 2.50
bit rate = 3
accuracy = 47
```

**fx** >>

مطابق انتظار در سرعت های بالاتر، *accuracy* کم تر می شود و پیام های بیشتری اشتباه تشخیص داده می شوند.

در ادامه با استفاده از روشی که در پایین صفحه قبلی بیان شد، بیشترین واریانس ها را برای سرعت های دو و سه به دست می آوریم :

#### Command Window

```
noise power = 1.30
bit rate = 2
accuracy = 97
```

**fx** >>

#### Command Window

```
noise power = 0.40
bit rate = 3
accuracy = 98
```

**fx** >>

همان طور که مشاهده می شود و طبق انتظاری که با توجه به مقدمه داشتیم، *bitRate* یک نسبت به *noise* مقاوم تر است و نتایج به دست آمده با مقدمه همخوانی دارند. یعنی هر چه *bitRate* افزایش یابد، مقاومت نسبت به *noise* کمتر خواهد شد.

## تمرین ۳-۷ :

با توجه به محاسبات و نتایج به دست آمده از تمرین ۳-۶ می توان گفت بیشترین واریانس نویز برای هر *bitRate* به شکل زیر است:

$$\text{for } \text{bitRate} = 1 : \sigma_{\max} \cong 2.5 \Rightarrow \sigma_{\max}^2 = (2.5)^2 \cong 6.25$$

$$\text{for } \text{bitRate} = 2 : \sigma_{\max} \cong 1.3 \Rightarrow \sigma_{\max}^2 \cong (1.3)^2 \cong 1.69$$

$$\text{for } \text{bitRate} = 3 : \sigma_{\max} \cong 0.4 \Rightarrow \sigma_{\max}^2 \cong (0.4)^2 \cong 0.16$$

## تمرین ۳-۸ :

اگر قدرت فرستنده بیشتر بود، می توانستیم دامنه سیگنال بیشتری داشته باشیم. در نتیجه فاصله *tresold* هایی که برای تصمیم گیری در نظر گرفته بودیم، بیشتر می شد. پس حساسیت کمتری نسبت به *noise* ایجاد می شد. به عنوان مثال برای ارسال اطلاعات با سرعت  $2 \frac{\text{bit}}{\text{sec}}$ ، اگر قدرت فرستنده به گونه ای می بود که بیشترین دامنه سیگنال 3 می شد، شرایط بهتر می شد. در این صورت برای ارسال 00، سیگنال  $x_0(t) = 0$  را به مدت ۱ ثانیه، برای ارسال 01، سیگنال  $x_1(t) = \sin(2\pi t)$  را به مدت ۱ ثانیه، برای ارسال 10، سیگنال  $x_2(t) = 2\sin(2\pi t)$  را به مدت ۱ ثانیه و برای ارسال 11، سیگنال  $x_3(t) = 3\sin(2\pi t)$  را به مدت ۱ ثانیه می فرستادیم.

پس در روش کدگذاری دامنه اگر بخواهیم سرعت ارسال اطلاعات را افزایش دهیم، باید *power* بیشتری مصرف کنیم تا نسبت به *noise* مقاوم بمانیم.

### تمرین ۴-۱ :

کاملاً مشابه با تمرین ۳-۱ است.

```
%4-1
charactersCount=32;
characters=['a':'z' ' ' ' ' '.' ',' '!' ';' ' ' "'"];
Mapset=cell(2, charactersCount);
for i=0:charactersCount-1
    Mapset{1, i+1}=characters(i+1);
    Mapset{2, i+1}=dec2bin(i, 5);
end
```

### تمرین ۶-۶ :

کد مربوط به تابع `coding_freq` در دو صفحه بعد آورده شده است ولی ورودی ها و خروجی هایش تغییری نسبت به آن چه در صورت پروژه گفته شده دارد که در ادامه به توضیح این تغییرات می پردازیم:

- به ورودی ها *Mapset*، *noisePower* و پرچم *test* هم اضافه شده اند که :
- از *Mapset* برای تشخیص کد باینری هر کاراکتر استفاده می شود. (می توانستیم از *Mapset* به شکل سراسری هم استفاده کنیم ولی طبق هشدار خود متلب و قواعد برنامه نویسی بهتر است از متغیر های سراسری استفاده نشود. به همین خاطر آن را به تابع پاس می دهیم.)
- اضافه کردن *nosiePower* به ورودی ها به این علت بود که در قسمت های بعدی که می خواهیم سیگنال کد شده را *noisy* کنیم، خود تابع *coidng\_freq* این کار را برای ما انجام دهد و وقتی نمی خواهیم سیگنال کد شده را *noisy* کنیم، کافی است *noisePower* را صفر کنیم.
- اضافه کردن *test* هم به این خاطر است که وقتی می خواهیم در بخش های بعدی آستانه واریانس را به دست آوریم، صد بار آزمایش می کنیم و خب در این صد بار نمی خواهیم نموداری بکشیم و با این پرچم به تابع اطلاع می دهیم که نمودار را بکشد یا خیر.
- به خروجی ها طول پیام، این که تعداد بیت های استفاده شده در پیام کد شده به سرعت ارسال بیت ها در هر ثانیه بخش پذیر است یا خیر، فرکانس های انتخاب شده برای هر عدد و کد باینری پیام اضافه شده است که از سه ویژگی اول برای تشخیص پیام کد شده در تابع *decoding\_freq* به ازای هر سرعت ارسال بیت دلخواهی استفاده می شود و از کد باینری پیام برای سنجش دقت سرعت های مختلف ارسال بیت در تابع *calc\_accuracy* استفاده می شود. نکته قابل توجه در این قسمت این است که چون فرکانس ها را به شکل گسسته انتخاب می کنیم پس کلا ۵۰ تا فرکانس داریم. در نتیجه به ازای سرعت های ارسال بیت بیشتر از ۵ بیت در هر ثانیه، به ناچار به برخی اعداد، فرکانس یکسان نسبت داده خواهد شد. به عنوان مثال وقتی سرعت ارسال بیت ها ۶ بیت در هر ثانیه باشد، ۶۴ حالت داریم و ۵۰ فرکانس مختلف. پس به ناچار به ۱۴ عدد، فرکانس تکراری نسبت خواهیم داد که این کار باعث می شود در سرعت های بالاتر حتی اگر *noise* نداشته باشیم، نتوانیم همیشه پیام را به درستی تشخیص دهیم! (برای آن که تابع *decoding\_freq* به درستی پیام را تشخیص دهد به طول پیام نیاز دارد از طرفی در هنگام کد کردن پیام چون در هر ثانیه تعداد بیت های ارسالی مقدار گسسته دارند، ممکن است تعداد بیت های تشکیل دهنده پیام به ۵ بخش پذیر نباشد. به عنوان مثال اگر همین کلمه *signal* پیام باشد. میدانیم با ۳۰ بیت کد می شود ولی اگر مثلاً سرعت ارسال بیت ها ۴ بیت در هر ثانیه باشد، برای کد کردن آن از ۳۲ بیت استفاده می کنیم. حال در تابع *decoding\_freq* برای تشخیص پیام به طول آن احتیاج داریم که اگر ۳۲ را به ۵ تقسیم کنیم، به طول اشتباهی می رسیم! به همین خاطر طول پیام ارسالی را در اینجا خروجی می دهیم و در تابع *decoding\_freq* از آن استفاده می کنیم.)

**شیوه کد کردن پیام:** با توجه به توضیحاتی که در مقدمه چهارم آورده شده است، تابعی که پیام را کد می‌کند به شکل زیر است:

$$\sum_{i=0}^{tend} \sin(2\pi f_k t) [u(t-i) - u(t-i-1)], f_k = \text{bitsFrequencies}(\text{bin2dec}(\text{bits}) + 1)$$

$$tend = \left\lceil \frac{\text{totalBits}}{\text{bitRate}} \right\rceil$$

که  $\text{bits}$  در هر ثانیه روی ۵ بیت از تعداد بیت‌های تشکیل دهنده پیام کد شده جلو می‌رود. خطوط ۴۷ تا ۵۵ کد صفحه بعد پیاده‌سازی این تابع در متلب است.

**نحوه انتخاب فرکانس‌ها:**

اگر تعداد فرکانس‌های مورد نیاز برای کد کردن پیام کمتر از ۵۰ باشد ( $\text{bitRate} < 6$ ) از رابطه زیر برای انتخاب فرکانس‌ها انتخاب می‌کنیم:

$$\text{step} = \left\lfloor \frac{49}{\text{frequenciesCount}} \right\rfloor$$

که  $\text{frequenciesCount}$  همان تعداد فرکانس‌های مورد نیاز برای ساخت سیگنال مورد نظر است که از مشخصه از رابطه  $2^{\text{bitRate}}$  به دست می‌آید. در ادامه از رابطه زیر اولین فرکانس را انتخاب می‌کنیم و در هر مرحله برای انتخاب فرکانس بعدی، به اندازه  $\text{step}$  جلو می‌رویم.

$$\text{startFrequency} = \left\lfloor \frac{49 - (\text{frequenciesCount} - 1)\text{step}}{2} \right\rfloor$$

و در غیر این صورت، به هر عدد، فرکانس باقی مانده آن عدد به ۵۰ را نسبت می‌دهیم.

```
if frequenciesCount < 50
    step = floor(49 / frequenciesCount);
    startFrequency = floor((49 - (frequenciesCount - 1) * step) / 2);
    for i = 1:frequenciesCount
        bitsFrequencies(i) = startFrequency + (i - 1) * step;
    end
else
    for i = 1:frequenciesCount
        bitsFrequencies(i) = rem(i, 50);
    end
end
```

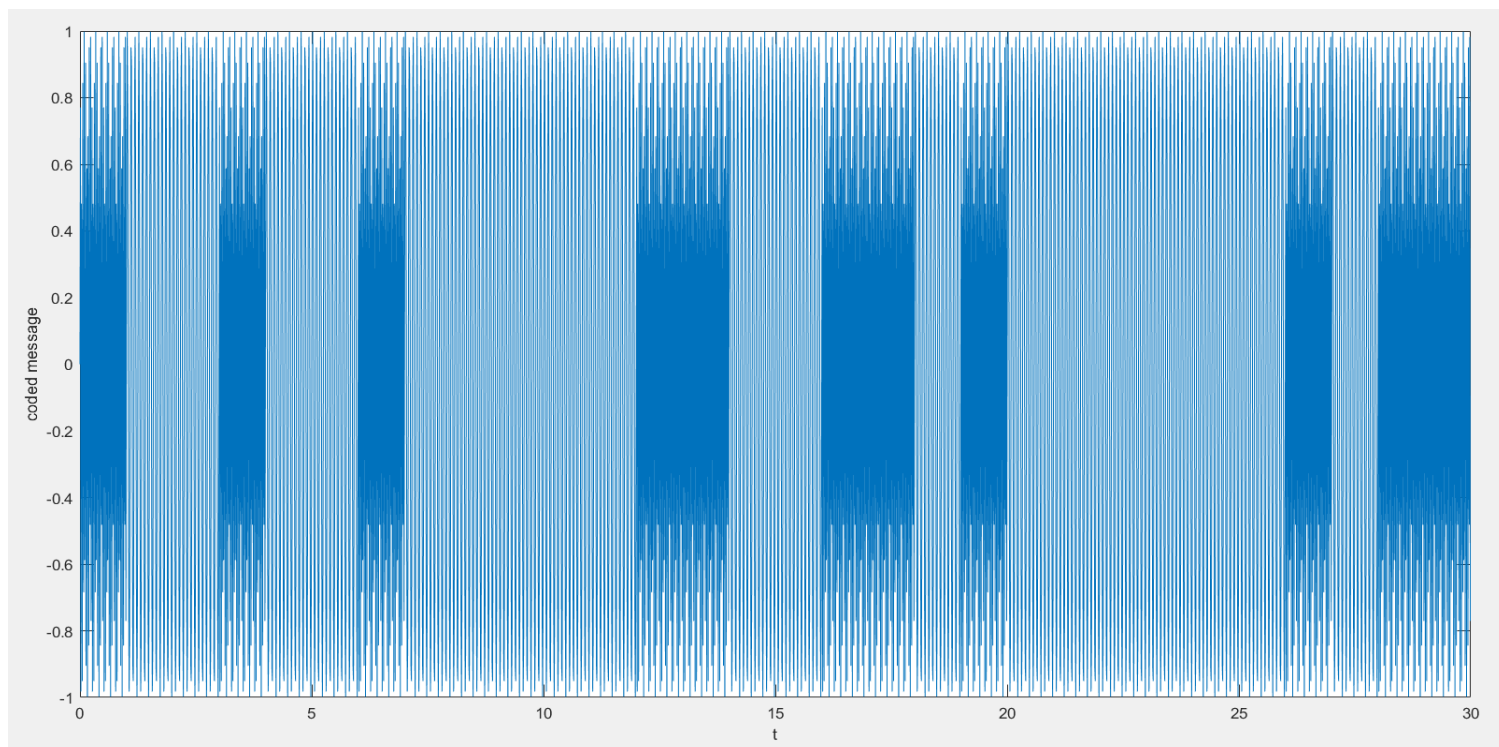
در ادامه کد تابع `coding_freq` را می‌آوریم.

---

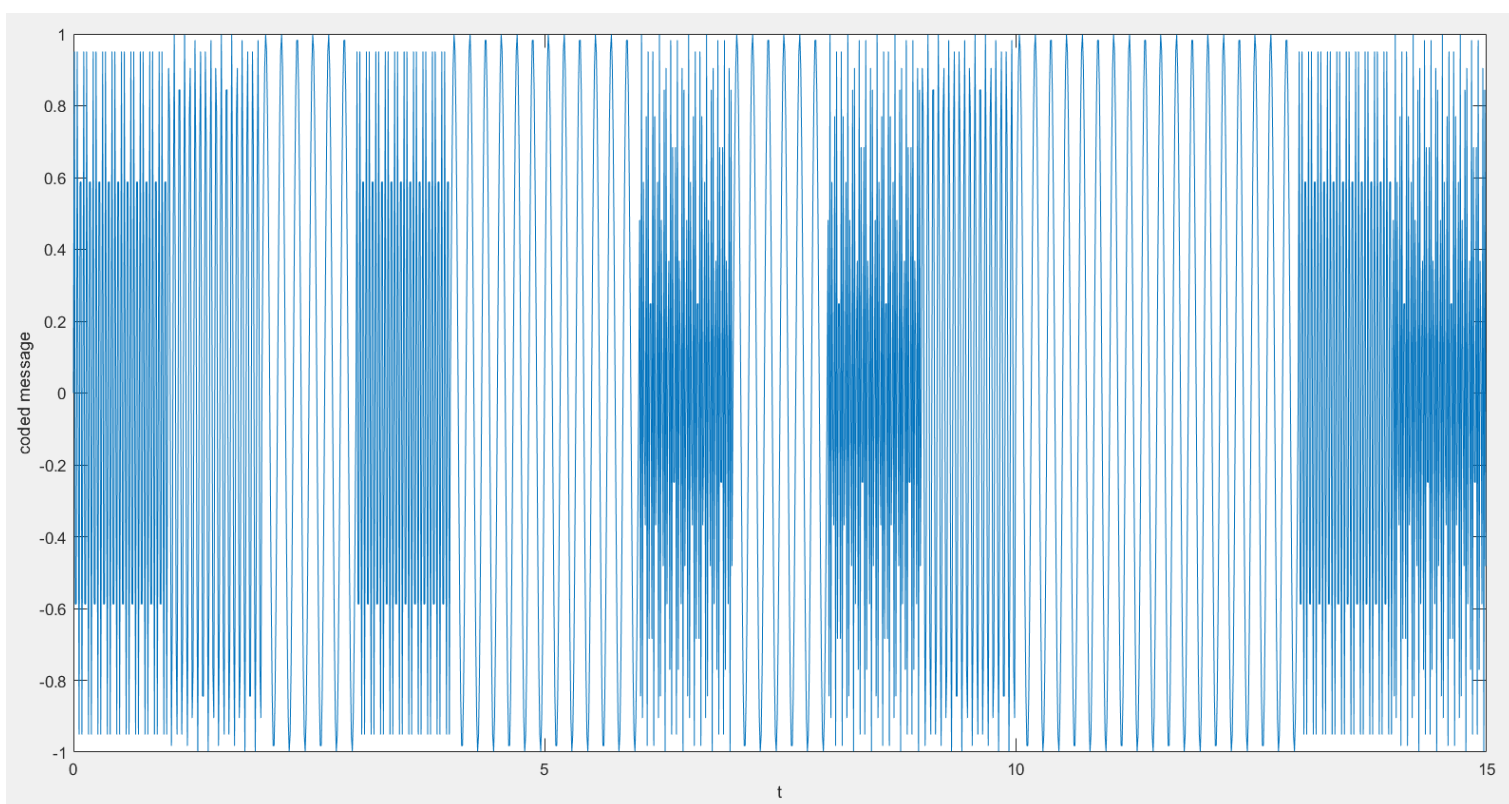
```
1 function [codedMessage, divisibleByBitRate, messageLength, bitsFrequencies, charactersToBits] = coding_freq(message, bitRate, Mapset, noisePower, test)
2     charactersCount = 32;
3     messageLength = strlen(message);
4     totalBitsCount = 5*messageLength;
5
6     if rem(totalBitsCount, bitRate)==0
7         divisibleByBitRate=true;
8     else
9         divisibleByBitRate=false;
10    end
11
12    charactersToBits = strings(1, messageLength);
13    frequenciesCount = 2^bitRate;
14
15    for i=1:messageLength
16        for j=1:charactersCount
17            if message(i) == Mapset{1, j}
18                charactersToBits(i) = Mapset{2, j};
19                break;
20            end
21        end
22    end
23
24    charactersToBits=join(charactersToBits, '');
25    charactersToBits=char(charactersToBits);
26    bitsFrequencies=zeros(1, frequenciesCount);
27
28    if frequenciesCount<50
29        step=floor(49/frequenciesCount);
30        startFrequency=floor((49-(frequenciesCount-1)*step)/2);
31        for i=1:frequenciesCount
32            bitsFrequencies(i)=startFrequency+(i-1)*step;
33        end
34    else
35        for i=1:frequenciesCount
36            bitsFrequencies(i)=rem(i, 50);
37        end
38    end
39
40    tstart=0;
41    tend=ceil(totalBitsCount/bitRate);
42    fs=100;
43    ts=1/fs;
44    t=tstart:ts:tend-ts;
45
46    codedMessage=0;
47    for i=0:tend-1
48        if (i+1)*bitRate<=totalBitsCount
49            bits=charactersToBits(i*bitRate+1:(i+1)*bitRate);
50        else
51            bits = charactersToBits(i*bitRate+1:totalBitsCount);
52        end
53        f=bitsFrequencies(bin2dec(bits)+1);
54        codedMessage=codedMessage+sin(2*pi*f*t).*(heaviside(t-i)-heaviside(t-i-1));
55    end
56
57    codedMessage=codedMessage+noisePower*randn(1, length(codedMessage));
58
59    if ~test
60        plot(t, codedMessage)
61        xlabel 't'
62        ylabel 'coded message'
63    end
64
65 end
```

## تمرین ۴-۴:

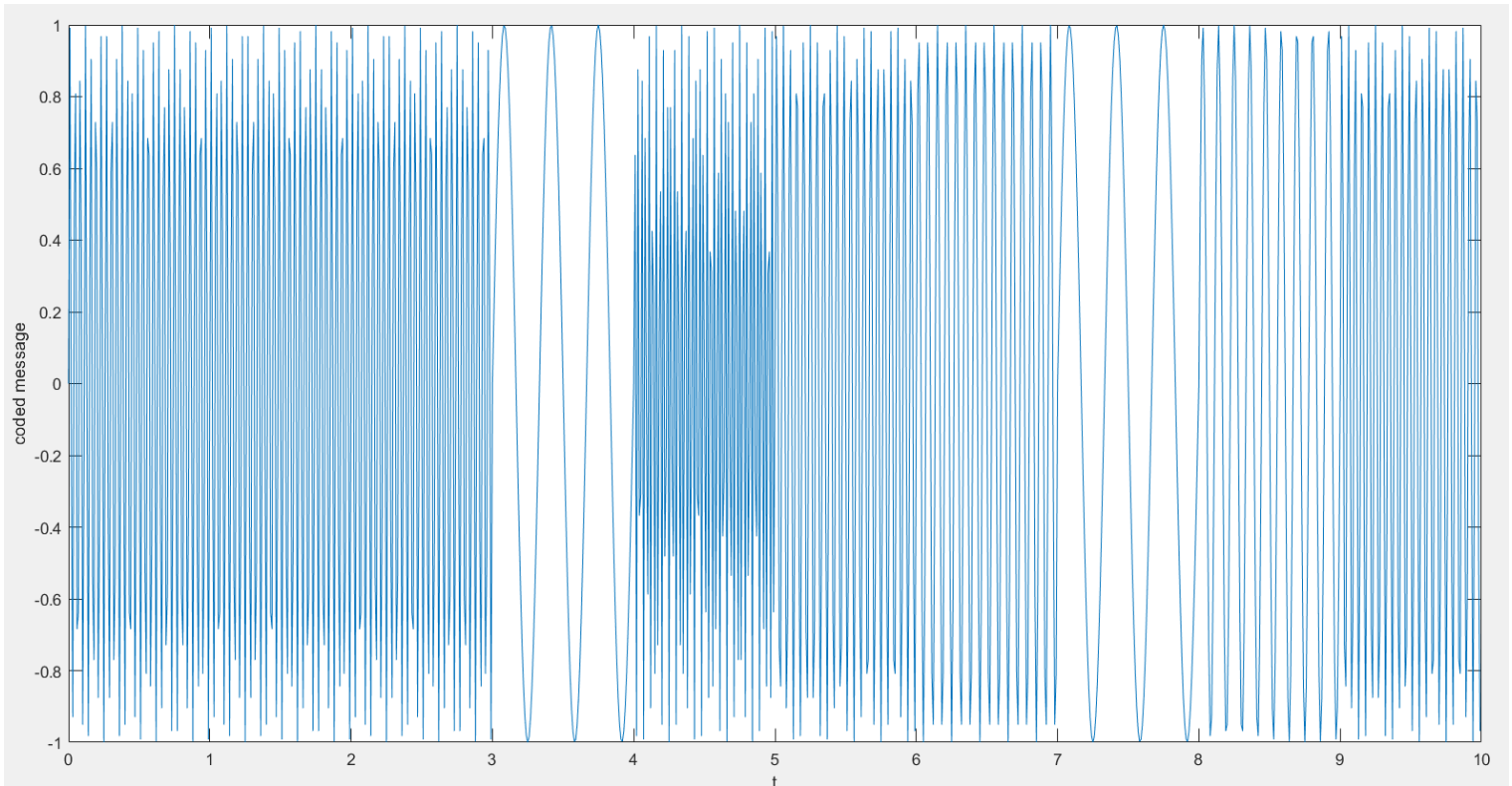
- خروجی تابع *coding\_freq* برای کلمه *signal* با سرعت ارسال اطلاعات یک بیت بر ثانیه:



- خروجی تابع *coding\_freq* برای کلمه *signal* با سرعت ارسال اطلاعات دو بیت بر ثانیه:



- خروجی تابع *coding\_freq* برای کلمه *signal* با سرعت ارسال اطلاعات سه بیت بر ثانیه:



## تمرین ۴-۴ :

کد مربوط به تابع *decoding\_freq* در ۳ صفحه بعدی آورده شده است ولی ورودی ها و خروجی ها تغییراتی نسبت به آن چه در صورت پروژه گفته شده دارد که به توضیح این تغییرات می پردازیم:

- به ورودی ها *Mapset*، *divisibleByBitRate*، *messageLength*، *noisePower* و *bitsFrequencies* اضافه شده است. که مشابه با توضیحاتی که در تمرین ۴-۲ آورده شده است، اضافه شدن *messageLength* و *divisibleByBitRate* برای این است که بتوانیم سرعت ارسال بیت در ثانیه دلخواه تا سقف ۵ بیت در ثانیه داشته باشیم.
- اضافه کردن *Mapset* برای تشخیص پیام از روی بیت هاست.
- اضافه کردن *noisePower* برای این است که بدانیم پیام *noise* دارد یا خیر. چون در ازای *noise* داشتن یا نداشتن پیام تعریف *threshold* فرق می کند.
- اضافه کردن *bitsFrequencies* برای این است که بدانیم به هر عدد در دامنه 0 تا  $2^{bitRate} - 1$  چه فرکانسی اختصاص داده شده است و از تعریف دوباره آن جلوگیری کنیم. (چون در تابع *coding\_freq* یک بار تعریفش کرده ایم).
- به خروجی ها *messageBits* (همان کد باینری پیام است بعد از این که پیام را *decode* کردیم) اضافه شده است که از *messageBits* برای سنجش دقت سرعت های مختلف ارسال بیت در تابع *calc\_accuracy* استفاده می شود.

---

```
function [decodedMessage, messageBits] = decoding_freq(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower, bitsFrequencies)
    fs=100;
    tend=length(codedMessage)/fs;
    frequenciesCount=2^bitRate;
    charactersCount=32;

    correlationResults=zeros(1, tend);

    for i=1:tend
        [maxfft, fmax]=max(abs(fftshift(fft(codedMessage((i-1)*fs+1:i*fs)))));
        correlationResults(i)=abs(fmax-51);
    end

    messageBits=strings(1, tend);
    decodedMessage=strings(1, messageLength);

    if noisePower==0
        for i=1:tend
            for j=1:frequenciesCount
                if correlationResults(i)==bitsFrequencies(j)
                    if ~divisibleByBitRate && i==tend
                        messageBits(i)=dec2bin(j-1, 5*messageLength-bitRate*(i-1));
                    else
                        messageBits(i)=dec2bin(j-1, bitRate);
                    end
                    break;
                end
            end
        end
    end
end
```



```

else
    for i=1:tend
        diff=1000000000000;
        isEmpty=true;
        for j=1:frequenciesCount-1
            tmpdiff1=abs(correlationResults(i)-bitsFrequencies(j));
            tmpdiff2=abs(correlationResults(i)-bitsFrequencies(j+1));
            if tmpdiff2<tmpdiff1 && tmpdiff2<diff
                default=j;
                diff=tmpdiff2;
            end
            if tmpdiff1<tmpdiff2 && tmpdiff1<diff
                default=j-1;
                diff=tmpdiff1;
            end
            threshold=(bitsFrequencies(j)+bitsFrequencies(j+1))/2;
            if correlationResults(i)>=bitsFrequencies(j) && correlationResults(i)<=threshold
                if ~divisibleByBitRate && i==tend
                    messageBits(i)=dec2bin(j-1, 5*messageLength-bitRate*(i-1));
                else
                    messageBits(i)=dec2bin(j-1, bitRate);
                end
                isEmpty=false;
                break;
            end
            if correlationResults(i)<=bitsFrequencies(j+1) && correlationResults(i)>threshold
                if ~divisibleByBitRate && i==tend
                    messageBits(i)=dec2bin(j, 5*messageLength-bitRate*(i-1));
                else
                    messageBits(i)=dec2bin(j, bitRate);
                end
                isEmpty=false;
                break;
            end
        end
        if isEmpty
            if ~divisibleByBitRate && i==tend
                messageBits(i)=dec2bin(default, 5*messageLength-bitRate*(i-1));
            else
                messageBits(i)=dec2bin(default, bitRate);
            end
        end
    end
end
end

```

```

messageBits=join(messageBits, '');
messageBits=char(messageBits);

for i=1:messageLength
    for j=1:charactersCount
        if Mapset{2, j} == messageBits((i-1)*5+1:(i*5))
            decodedMessage(i)=Mapset{1, j};
            break;
        end
    end
end
end
end
end

```

خروجی تابع به ازای هر سه سرعت یک، دو و سه بیت بر ثانیه یکسان و صحیح بود:

-  $bitRate = 1$

```

bitRate=1;
noisePower=0;
message='signal';
test=false;

if ~test
    [codedMessage, divisibleByBitRate, messageLength, bitsFrequencies, charactersToBits]=coding_freq(message, bitRate, Mapset, noisePower, test);
    [decodedMessage, messageBits] = decoding_freq(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower, bitsFrequencies);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %.2f\n', accuracyRate);
end

```

Command Window

```

decoded message is:
signal

```

 >>

: *bitRate* = 2 -

```
bitRate=2;
noisePower=0;
message='signal';
test=false;

if ~test
    [codedMessage, divisibleByBitRate, messageLength, bitsFrequencies, charactersToBits]=coding_freq(message, bitRate, Mapset, noisePower, test);
    [decodedMessage, messageBits] = decoding_freq(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower, bitsFrequencies);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %.2f\n', accuracyRate);
end
```

Command Window

```
decoded message is:
signal
```

*fx* >>

: *bitRate* = 3 -

```
bitRate=3;
noisePower=0;
message='signal';
test=false;

if ~test
    [codedMessage, divisibleByBitRate, messageLength, bitsFrequencies, charactersToBits]=coding_freq(message, bitRate, Mapset, noisePower, test);
    [decodedMessage, messageBits] = decoding_freq(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower, bitsFrequencies);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %.2f\n', accuracyRate);
end
```

Command Window

```
decoded message is:
signal
```

*fx* >>

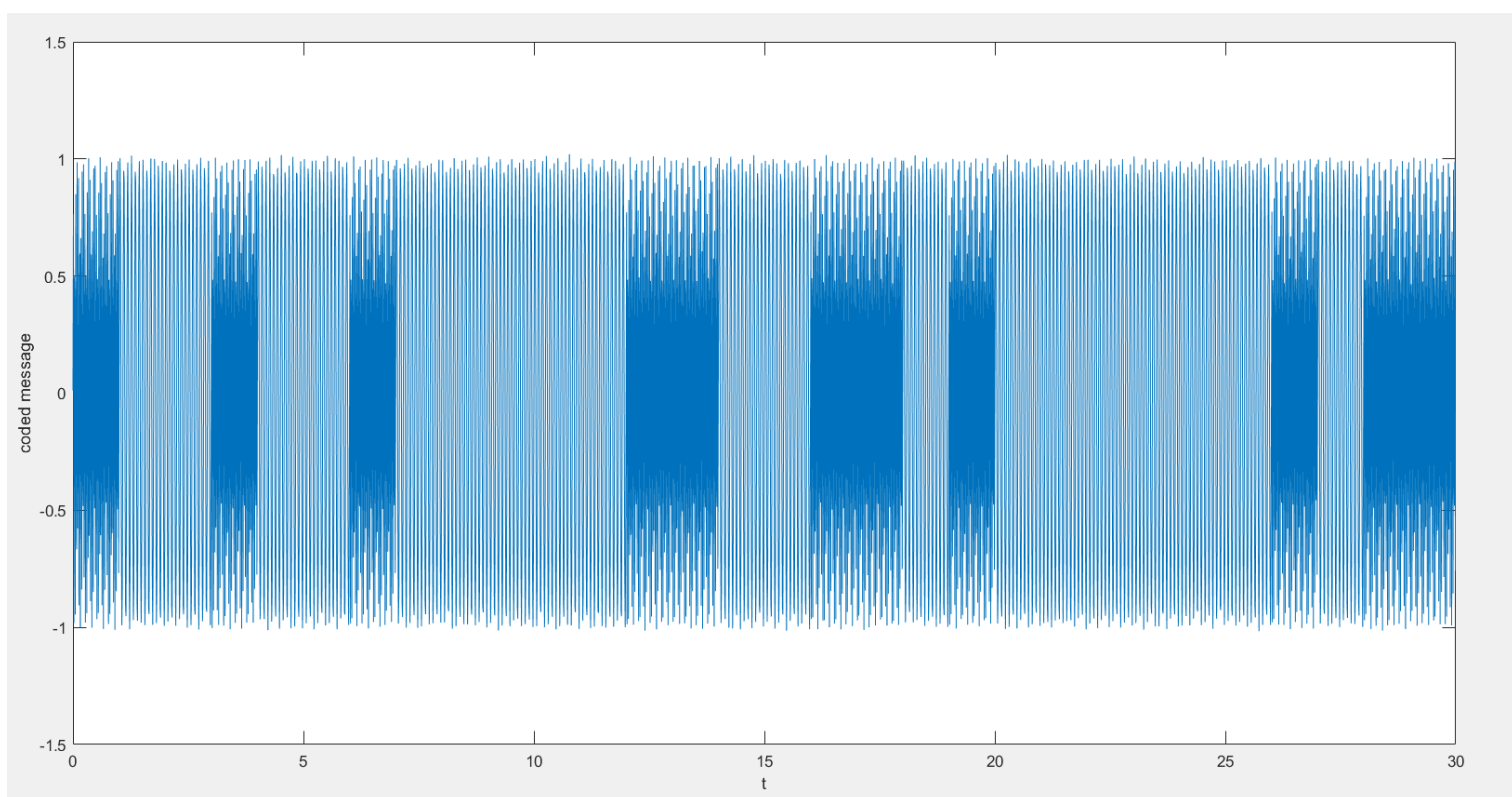
## تمرین ۴-۵:

اضافه کردن نویز گوسی با واریانس 0.0001 و میانگین 0 به ازای سرعت ارسال بیت در ثانیه:

-  $bitRate = 1$

```
bitRate=1;
noisePower=0.01;
message='signal';
test=false;

if ~test
    [codedMessage, divisibleByBitRate, messageLength, bitsFrequencies, charactersToBits]=coding_freq(message, bitRate, Mapset, noisePower, test);
    [decodedMessage, messageBits] = decoding_freq(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower, bitsFrequencies);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %.2f\n', accuracyRate);
end
```



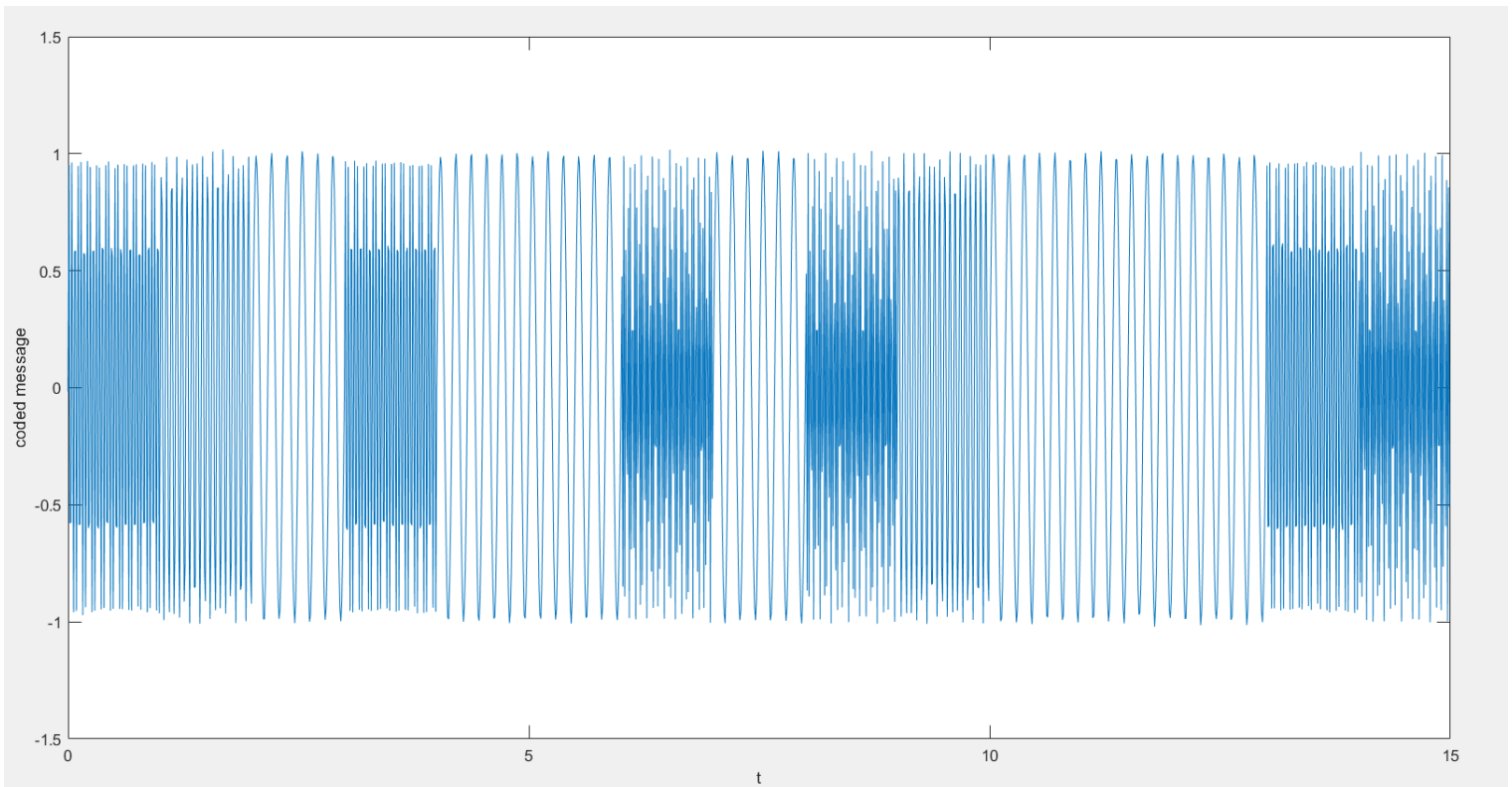
Command Window

```
decoded message is:
signal
fx >>
```

:  $bitRate = 2$  -

```
bitRate=2;
noisePower=0.01;
message='signal';
test=false;

if ~test
    [codedMessage, divisibleByBitRate, messageLength, bitsFrequencies, charactersToBits]=coding_freq(message, bitRate, Mapset, noisePower, test);
    [decodedMessage, messageBits] = decoding_freq(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower, bitsFrequencies);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %.2f\n', accuracyRate);
end
```



Command Window

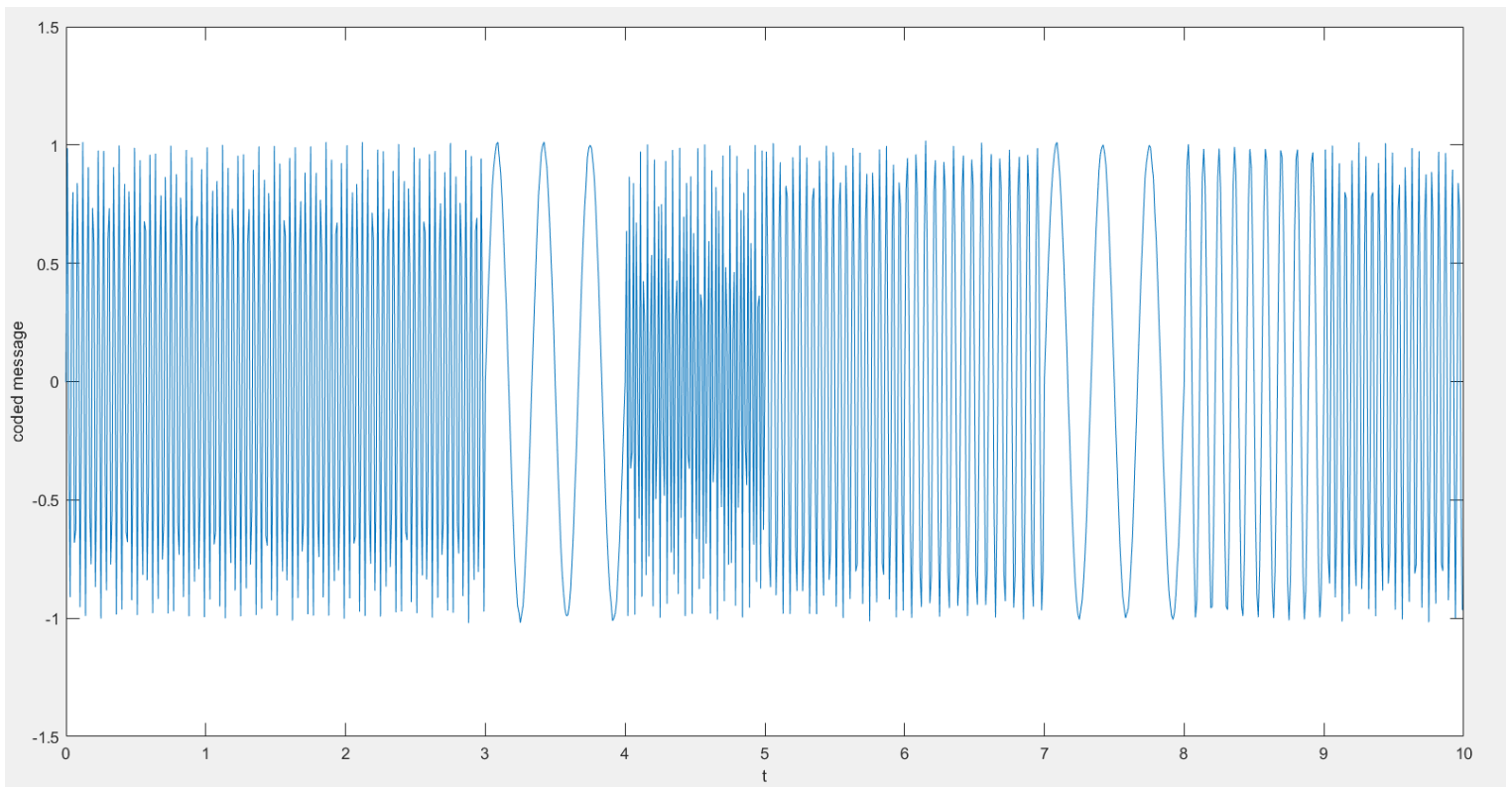
```
decoded message is:
signal
```

```
fz >>
```

:  $bitRate = 3$  -

```
bitRate=3;
noisePower=0.01;
message='signal';
test=false;

if ~test
    [codedMessage, divisibleByBitRate, messageLength, bitsFrequencies, charactersToBits]=coding_freq(message, bitRate, Mapset, noisePower, test);
    [decodedMessage, messageBits] = decoding_freq(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower, bitsFrequencies);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %.2f\n', accuracyRate);
end
```



Command Window

```
decoded message is:
signal
```

```
fx >>
```

## تمرین ۶-۶ :

نحوه به دست آوردن مقدار بیشترین واریانس : در این قسمت به ازای یک مقدار  $noise$ ، ۱۰۰ بار کد را اجرا می‌کنیم و در هر بار بررسی می‌کنیم که چه تعداد از بیت‌های کل پیام را به درستی تشخیص داده‌ایم. در ادامه میزان دقت را برابر میانگین تمام مراحل در نظر می‌گیریم. ابتدا با یک  $noise$  کم شروع به آزمایش کردیم و در ادامه مقدار  $noise$  را زیاد کردیم تا وقتی که در ۱۰۰ بار اجرای کد، حداقل یک بار پیام به اشتباه تشخیص داده شد. تابعی که این کارها را برای مان انجام می‌دهد، تابع  $calc\_accuracy$  است و برای اجرای ۱۰۰ بار آزمایش همان طور که پیش‌تر در تمرین ۴-۲ اشاره شد، کافی است در ابتدای کد، پرچم  $test$  را  $true$  کنیم :

```
bitRate=1;
noisePower=0;
message='signal';
test=true;

if ~test
    [codedMessage, divisibleByBitRate, messageLength, bitsFrequencies, charactersToBits]=coding_freq(message, bitRate, Mapset, noisePower, test);
    [decodedMessage, messageBits] = decoding_freq(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower, bitsFrequencies);
    fprintf('decoded message is:\n')
    fprintf('%s', decodedMessage{:});
    fprintf('\n')
else
    accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test);
    fprintf('noise power = %.2f\n', noisePower);
    fprintf('bit rate = %d\n', bitRate);
    fprintf('accuracy = %.2f\n', accuracyRate);
end
```

```
function accuracyRate = calc_accuracy(message, bitRate, Mapset, noisePower, test)
    accuraciesSum=0;
    for i=1:100
        mistakesCount=0;
        [codedMessage, divisibleByBitRate, messageLength, bitsFrequencies, charactersToBits]=coding_freq(message, bitRate, Mapset, noisePower, test);
        [decodedMessage, messageBits] = decoding_freq(codedMessage, bitRate, Mapset, divisibleByBitRate, messageLength, noisePower, bitsFrequencies);
        n=length(messageBits);
        m=length(charactersToBits);
        for j=1:min(n, m)
            if messageBits(j)~=charactersToBits(j)
                mistakesCount=mistakesCount+1;
            end
        end
        accuraciesSum=accuraciesSum+(n-mistakesCount)/n;
    end
    accuracyRate=accuraciesSum/100;
end
```

برای  $bitRate = 1$  در  $noisePower = 1.3$  به اولین جایی رسیدیم که  $accuracy \neq 1$  بود. (کد را به شکل دستی اجرا کردیم و در هر مرحله با شروع از  $noisePower = 0$ ،  $noisePower$  را 0.1 زیاد کردیم. به علت زیاد بودن تعداد مراحل، عکس آن‌ها را در گزارش نمی‌آوریم و اگر مایل بودید، خودتان می‌توانید تست کنید 😊) در ادامه  $accuracy$  را به ازای همین  $noisePower = 1.3$  برای  $bitRate$  های متفاوت به دست می‌آوریم تا نشان دهیم، هر چه سرعت ارسال بیت‌ها کمتر باشد، مقاومت نسبت به  $noise$  بیشتر خواهد بود و سرعت‌های بیشتر،  $accuracy$  کمتری دارند. هم‌چنین برای به دست آوردن مقدار بیشترین واریانس در دیگر سرعت‌ها از همین روش استفاده می‌کنیم. یعنی با شروع از  $noisePower = 0$ ،  $noisePower$  را در هر مرحله 0.1 زیاد می‌کنیم تا به  $accuracy$  کمتر از ۱ برسیم. دقت شود عددی که به عنوان بیشترین واریانس اعلام می‌شود تقریبی است و در آزمایش‌های متفاوت ممکن است به اعداد دیگری برسیم ولی با تقریب خوبی در همین حدود است.

Command Window

```
noise power = 1.30
bit rate = 1
accuracy = 0.99
```

fx >>

Command Window

```
noise power = 1.30
bit rate = 2
accuracy = 0.99
```

fx >>

Command Window

```
noise power = 1.30
bit rate = 3
accuracy = 0.99
```

fx >>

همان طور که مشاهده می شود این بار تفاوت *bitRate* های یک، دو و سه زیاد نیست که علت آن به خود کلمه *signal* مربوط می شود. پس آزمایش را برای *bitRate* های پنج، شش، هفت و هشت تکرار می کنیم تا ببینیم باز هم اوضاع همین است یا فرق می کند :

```
Command Window
noise power = 1.30
bit rate = 5
accuracy = 0.99
fx >>
```

```
Command Window
noise power = 1.30
bit rate = 6
accuracy = 0.89
fx >>
```

```
Command Window
noise power = 1.30
bit rate = 7
accuracy = 0.83
fx >>
```

```
Command Window
noise power = 1.30
bit rate = 8
accuracy = 0.70
fx >>
```

همان طور که مشاهده می شود و طبق انتظاری که با توجه به مقدمه داشتیم، *bitRate* های کمتر خطای کمتری دارند و به احتمال بیشتری به درستی پیام را تشخیص می دهند پس نسبت به *noise* مقاوم ترند. نتایج به دست آمده هم با مقدمه همخوانی دارند. یعنی هر چه *bitRate* افزایش یابد، مقاومت نسبت به *noise* کمتر خواهد شد. نکته قابل توجه در این بخش این است که قدرت مقاومت *bitRate* های دو، سه و پنج نسبت به حوزه زمان افزایش یافته و به یکدیگر نزدیک شده اند. که می توان گفت شاید علت آن این است که در حوزه فرکانس، فرکانس هایی که انتخاب می کنیم نسبت به حوزه زمان برای *bitRate* های دو، سه و پنج فاصله بیشتری دارند (فاصله فرکانس ها برای *bitRate* های دو، سه و پنج به ترتیب 12، 6 و 1 است. در حالی که در حوزه زمان این فاصله کمتر از 1 است!) در نتیجه احتمال خطا کمتر می شود.

## تمرین ۴-۶ :

با توجه به نمودار ها و نتایج تمرین ۴-۶ می توان گفت بیشترین واریانس نویز برای هر *bitRate* به شکل زیر است:

$$\text{for } bitRate = 1 : \sigma_{max} \cong 1.3 \Rightarrow \sigma_{max}^2 = (1.3)^2 \cong 1.69$$

$$\text{for } bitRate = 2 : \sigma_{max} \cong 1.3 \Rightarrow \sigma_{max}^2 \cong (1.3)^2 \cong 1.69$$

$$\text{for } bitRate = 3 : \sigma_{max} \cong 1.3 \Rightarrow \sigma_{max}^2 \cong (1.3)^2 \cong 1.69$$

$$\text{for } bitRate = 5 : \sigma_{max} \cong 1.3 \Rightarrow \sigma_{max}^2 \cong (1.3)^2 \cong 1.69$$

$$\text{for } bitRate = 6 : \sigma_{max} \cong 0 \Rightarrow \sigma_{max}^2 \cong (0)^2 \cong 0$$

$$\text{for } bitRate = 7 : \sigma_{max} \cong 0 \Rightarrow \sigma_{max}^2 \cong (0)^2 \cong 0$$

$$\text{for } bitRate = 8 : \sigma_{max} \cong 0 \Rightarrow \sigma_{max}^2 \cong (0)^2 \cong 0$$

دقت شود که چون در  $bitrate = 6$ ،  $bitrate = 7$  و  $bitrate = 8$  فرکانس های تکراری داریم از همان اول و به ازای *noise* های کم هم دقت 1 نداشتیم.

## تمرین ۴-۸ :

هر چقدر فاصله بین فرکانس های انتخابی بیشتر باشد، کدگذاری انجام شده نسبت به نویز مقاوم تر خواهد بود. پس هر چه پهنای باند بیشتری مصرف کنیم می توانیم با سرعت بیشتری اطلاعات را ارسال کنیم و در عین حال نسبت به نویز مقاوم باشیم.