پروژه کامپیوتری 5

سیگنال ها و سیستم ها

دکتر اخوان

پریسا محمدی 810101509

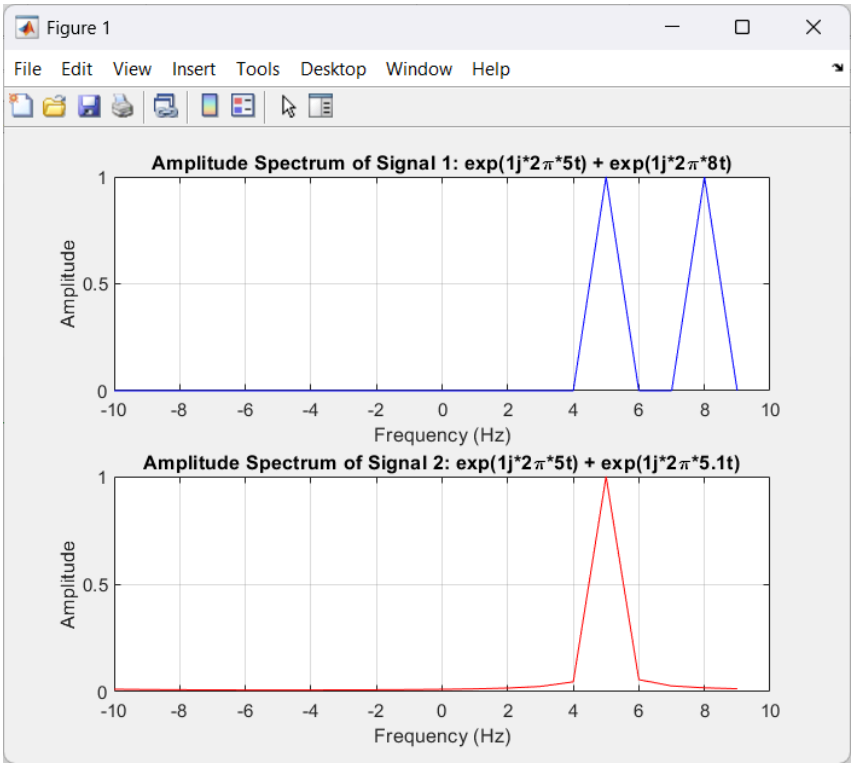
سامان دوچی طوسی 810101420

تمرین 1-0)

```
% parameters
t_start = 0;
t_end = 1;
fs = 20;
t = t_start:1/fs:t_end - 1/fs;

% number of samples
N = length(t);

f = -fs/2 : fs/N : fs/2 - fs/N;

signal1 = exp(1i*2*pi*5*t) + exp(1i*2*pi*8*t); % First signal
signal2 = exp(1i*2*pi*5*t) + exp(1i*2*pi*5.1*t); % Second signal

y1 = fft(signal1);
y2 = fft(signal2);

y1_shifted = fftshift(y1);
y2_shifted = fftshift(y2);

amplitude1 = abs(y1_shifted) / max(abs(y1_shifted));
amplitude2 = abs(y2_shifted) / max(abs(y2_shifted));

figure;

subplot(2, 1, 1);
plot(f, amplitude1, 'b');
xlabel('Frequency (Hz)');
ylabel('Amplitude');
title('Amplitude Spectrum of Signal 1: exp(1j*2\pi*5t) + exp(1j*2\pi*8t)');
grid on;

subplot(2, 1, 2);
plot(f, amplitude2, 'r');
xlabel('Frequency (Hz)');
ylabel('Amplitude');
title('Amplitude Spectrum of Signal 2: exp(1j*2\pi*5t) + exp(1j*2\pi*5.1t)');
grid on;
```

تصویر *1* - کد بررسی رزولوشن فرکانسی

در این سوال می خواهیم مفهوم رزولوشن فرکانسی را بررسی نماییم. در ابتدا بردار زمان را با استپ های fs/1 را تعریف می کنیم که سبب می شود سیگنال ورودی را به طول t سمپل برداری کنیم. برای بررسی رزولوشن فرکانسی باید توجه داشته باشیم که به نرخ نمونه برداری بستگی ندارد چرا که رزولوشن فرکانسی به صورت fs تقسیم بر N تعریف می شود و از آنجایی که fs با N متناسب است در نتیجه رزولوشن فرکانسی به N(نرخ نمونه برداری) بستگی ندارد. پس در واقع رزولوشن فرکانسی برابر با عکس طول زمانی سیگنال است و تا زمانی که طول زمانی سیگنال ثابت باشد، تغییری نمی کند. در این سوال نرخ نمونه برداری برابر با یک تقسیم بر یک است پس برای مشاهده تبدیل فوریه سیگنال اول به مشکلی نمی خوریم چرا که این تبدیل فوریه شامل دو تابع ضربه در فرکانس های 5 و 8 است و چون رزولوشن فرکانسی برابر با یک است می توانیم آنها را مشاهده کنیم. اما در سیگنال دوم، تبدیل فوریه شامل ضربه در فرکانس های 5 و 5.1 است و رزولوشن فرکانسی یک اجازه نمی دهد که اندازه تبدیل فوریه این سیگنال در 5.1 مشاهده شود در واقع این طول زمانی دقت لازم برای بررسی فرکانس هایی که مضرب یک نیستند را ندارد.و این به وضوح در نمودار اندازه تبدیل فوریه دو سیگنال مشاهده می شود.



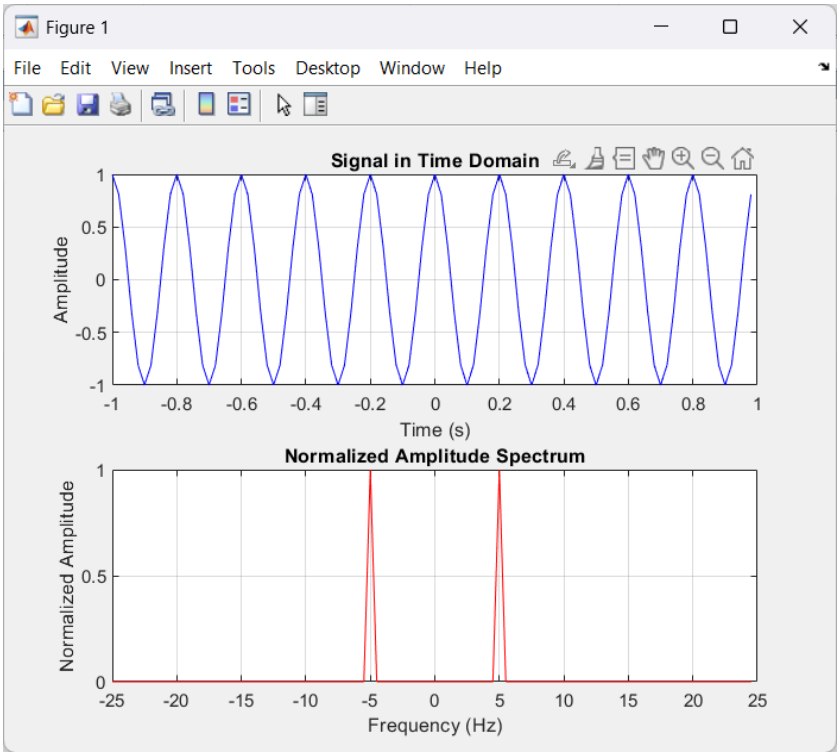
تصویر 2 - تاثیر رزولوشن فرکانسی در محاسبه تبدیل فوریه

تمرین 1-1)

```
% parameters
t_start = -1;
t_end = 1;
fs = 50;
t = t_start:1/fs:t_end - 1/fs;

x = cos(10*pi*t);

% the signal in time domain
figure;
subplot(2,1,1);
plot(t, x, 'b');
xlabel('Time (s)');
ylabel('Amplitude');
title('Signal in Time Domain');
grid on;

% number of samples
N = length(t);

f = -fs/2 : fs/N : fs/2 - fs/N;

y = fft(x);
y_shifted = fftshift(y);

amplitude = abs(y_shifted) / max(abs(y_shifted));

% the amplitude spectrum
subplot(2,1,2);
plot(f, amplitude, 'r');
xlabel('Frequency (Hz)');
ylabel('Normalized Amplitude');
title('Normalized Amplitude Spectrum');
grid on;
```

تصویر3 - کد رسم سیگنال در حوزه زمان و اندازه اش در حوزه فوریه

دراین کد ابتدا سیگنال ورودی را در حوزه زمان و بر روی بردار زمانی t رسم می کنیم. این بردار زمانی را طبق خواسته سوال دارای استپ های زمانی ts یا همان یک تقسیم بر fs است. در بخش بعدی خواسته شده که سیگنال را به حوزه فوریه (فرکانس) برده و اندازه تبدیل فوریه سیگنال را بر حسب بردار فرکانس که دارای استپ های فرکانسی یا رزولوشن فرکانسی fs تقسیم بر N است را رسم کردیم.
و نتایج با انتظارمان یکی بود چرا که تبدیل فوریه سیگنال بایستی دارای دو تابع ضربه در فرکانس های 5 و 5- را شامل می شد که خروجی نیز به همین صورت است.



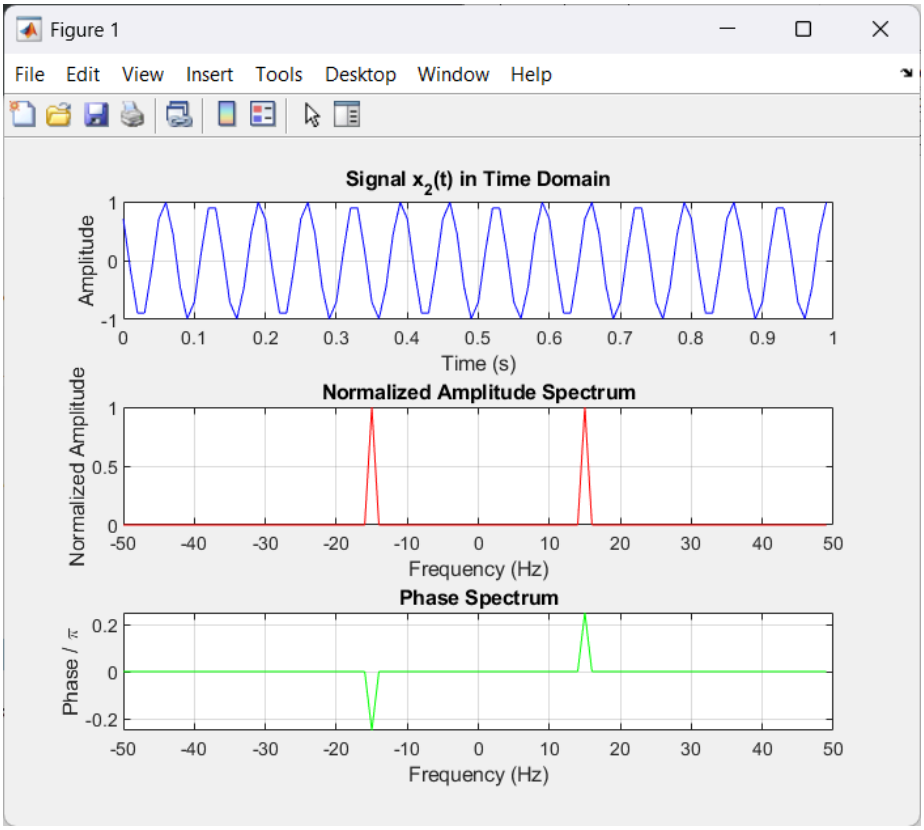
تصویر *4* - سیگنال ورودی کسینوس ای و اندازه تبدیل فوریه آن به صورت جمع دو تابع ضربه

تمرین 1-2)

```
% parameters
t_start = 0;
t_end = 1;
fs = 100;
t = t_start:1/fs:t_end - 1/fs;
x = cos(30*pi*t + pi/4);
% the signal in the time domain
figure;
subplot(3,1,1);
plot(t, x, 'b');
xlabel('Time (s)');
ylabel('Amplitude');
title('Signal x_2(t) in Time Domain');
grid on;
% number of samples
N = length(t);
f = -fs/2 : fs/N : fs/2 - fs/N;
y = fft(x);
y_shifted = fftshift(y);
amplitude = abs(y_shifted) / max(abs(y_shifted));
% the normalized amplitude spectrum
subplot(3,1,2);
plot(f, amplitude, 'r');
xlabel('Frequency (Hz)');
ylabel('Normalized Amplitude');
title('Normalized Amplitude Spectrum');
grid on;
% phase spectrum
tol = 1e-6;
y_shifted(abs(y_shifted) < tol) = 0; % Zero out low amplitudes
theta = angle(y_shifted);
subplot(3,1,3);
plot(f, theta/pi, 'g');
xlabel('Frequency (Hz)');
ylabel('Phase / \pi');
title('Phase Spectrum');
grid on;
```

تصویر 5 - کد بررسی سیگنال در حوزه زمان و اندازه و زاویه سیگنال در حوزه فوریه

در این بخش سیگنال ورودی به صورت یک تابع کسینوس ای می باشد که دارای مقدار فاز ثابتی نیز هست. ابتدا این سیگنال را در حوزه زمان طبق مشخصات شروع و پایان زمان رسم می کنیم. استپ های زمانی نیز ts می باشد که برابر با fs/1 می باشد. در ادامه بایستی اندازه سیگنال را در حوزه فوریه (فرکانس) رسم کنیم که بدین منظور از دستور fft , در ادامه از دستور fftshift استفاده می کنیم که دستور دوم تنها برای این است که بازه متقارن حول صفر را ایجاد کنیم. در نهایت نیز سوال از ما خواسته است تا زاویه یا فاز این سیگنال را در حوزه فوریه بر حسب pi رسم کنیم که بدین منظور ابتدا فاز را در فرکانس هایی که اندازه در آنها ناچیز است صفر کردیم و بعد از دستور angle در متلب برای یافتن تابع زاویه استفاده کردیم.



تصویر 6 - سیگنال ورودی در حوزه زمان و اندازه و زاویه سیگنال در حوزه فوریه

تمرین 2-1)

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

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

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

تصویر 7 - کد تولید کردن مپست

به ازای همه کاراکتر های خواسته شده توسط سوال مپست تشکیل داده شده است.

تمرین 2-2)

```
function [y,x_axis]=coding_freq(message,speed)
fs=100;
mapset= generateMapset;
message_len=length(message);
message_bin=cell(1,message_len);
for i=1:message_len
    ch=message(i);
    for j=1:32
        if ch==mapset{1,j}
            message_bin{i}=mapset{2,j};
        end
    end
end
binarymessage=cell2mat(message_bin);
binarymessage_len=length(binarymessage);
frequency=cell(1,5);
frequency{1,1}=[12,37];
frequency{1,2}=[5,16,27,38];
frequency{1,3}=[4,10,16,22,28,34,40,46];
frequency{1,4}=[2,5,8,11,14,17,20,23,26,29,32,35,38,41,44,47];
frequency{1,5}=[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];
y=[];
ts=1/fs;
time=0:ts:(1-ts);
for j=0:(binarymessage_len/speed)-1
    u=[];
    for t=1:speed
        q=(binarymessage((speed*j+1)+t-1));
        u=[u q];
    end
    andis=bin2dec(u)+1;
    x=cell2mat(frequency(speed));
    f=x(andis);
    attach=sin(2*pi*f*time);
    y=[y,attach];
end
x_axis=0:0.01:binarymessage_len/speed-0.01;
end
```

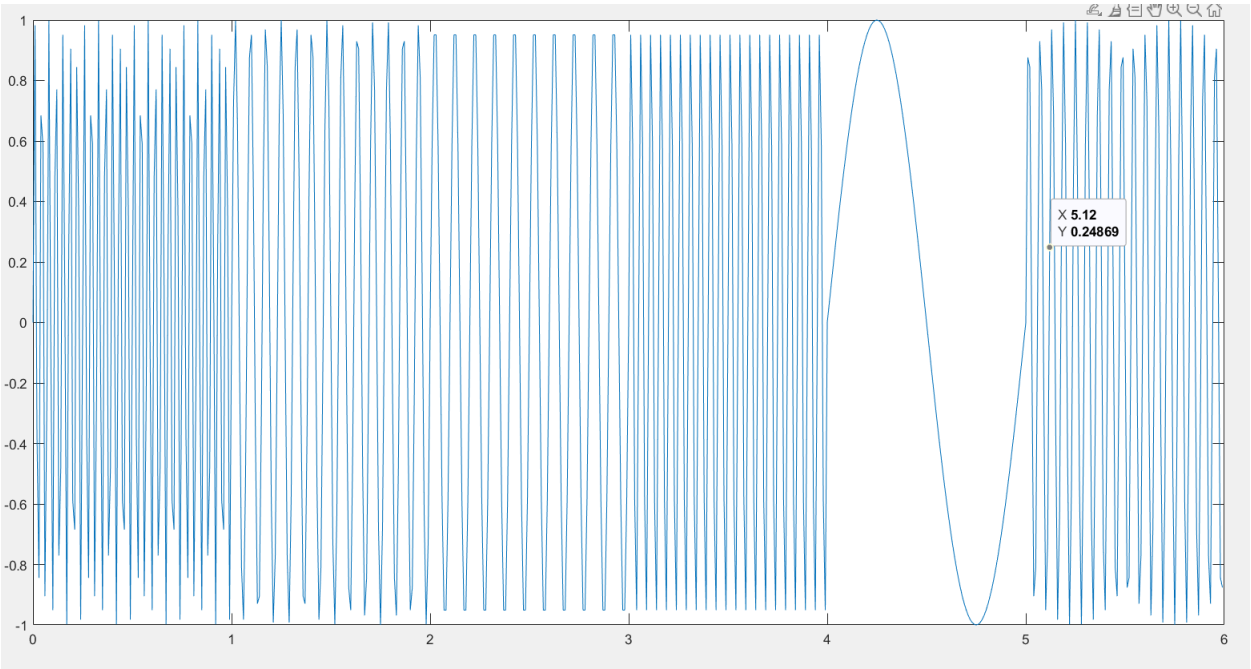
تصویر 8 - کد تابع *coding_freq*

توضیحات تابع:

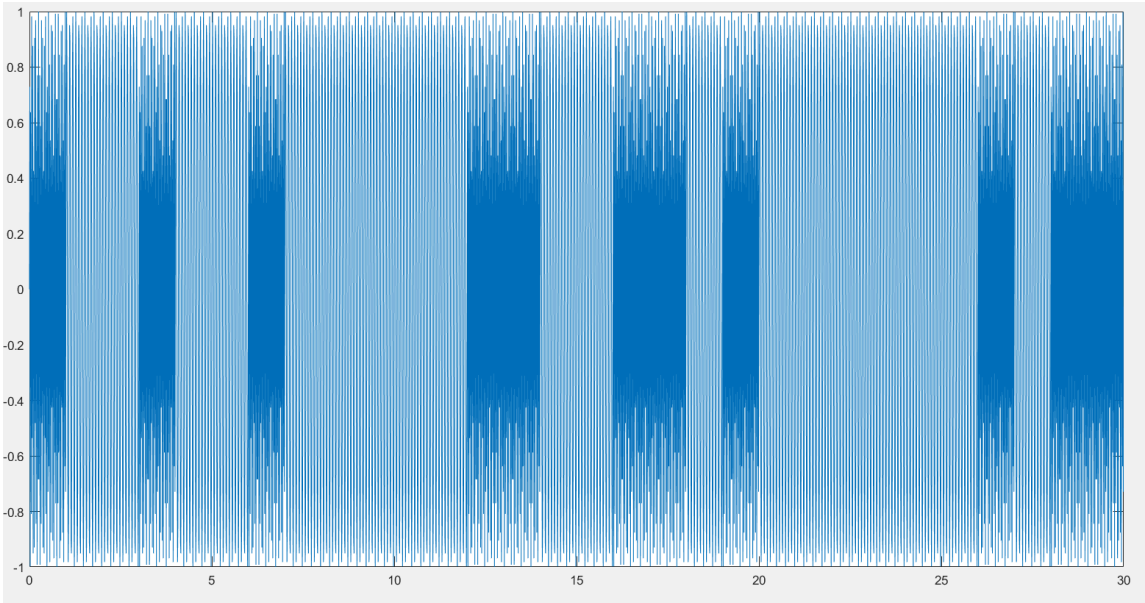
در این تابع ابتدا مپست را ایجاد میکنیم. سپس داخل حلقه for پیاممان را که در اینجا signal است، encode می‌کنیم. سپس فرکانس های مختلف برای بیت ریت های 1 تا 5 را تولید میکنیم. سپس تایم استپ ها را ایجاد میکنیم.

در حلقه تو در تو، با توجه به سرعت پیاممان را قسمت بندی می‌کنیم و با توجه به قسمت به دست آمده فرکانس آن را به دست می آوریم . شکل سینوسی آن را داخلی y ذخیره می کنیم و آن را بر می‌گردانیم.

تمرین 2-3)



تصویر *9* - خروجی حاصل از اجرای تابع ***coding_freq*** با سرعت *5*



تصویر *10* - خروجی حاصل از اجرای تابع ***coding_freq*** با سرعت *1*

مقایسه 2 تصویر: با افزایش پیدا کردن بیت ریت، تعداد فرکانس هایمان افزایش پیدا میکند درحالی که زمان آن کاهش پیدا میکند.

تمرین 2-4)

```
function DcodedMessageBin=decoding_freq(y,speed)
    s=zeros(length(y)/100,100);
    for r=1:(length(y)/100)
        for g=1:100
            s(r,g)=y((100*r-100)+g);
        end
    end
    len=length(s(:,1));
    index=zeros(1,len);
    for g=1:len
        y2=fftshift(fft(s(g,:)));
        out=y2/max(abs(y2));
        out=abs(out);
        positive=out(51:100);
        [~,I]=max(positive);
        index(g)=I-1;
    end
    frequency=cell(1,5);
    frequency{1,1}=[12,37];
    frequency{1,2}=[5,16,27,38];
    frequency{1,3}=[4,10,16,22,28,34,40,46];
    frequency{1,4}=[2,5,8,11,14,17,20,23,26,29,32,35,38,41,44,47];
    frequency{1,5}=[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];
    string=[];
    co=cell2mat(frequency(speed));
    thre=zeros(1,length(co)+1);
    thre(1)=(0+co(1)/2);
    for v=1:length(co)-1
```

تصویر *11* - بخش اول تابع *decoding_freq*

```
        thre(v+1)=(co(v)+co(v+1))/2;
    end
    thre(end)=(co(end)+49)/2;
    for r=1:len
            for k=1:length(thre)-1
                if index(r)>thre(k) && index(r)<thre(k+1)
                    z=k;
                    break
                end
            end
            bin=dec2bin(z-1,speed);
            string=[string bin];
    end
    DcodedMessageBin=[];
    ind=1;
    Alphabet ='abcdefghijklmnopqrstuvwxyz ,;?.!';
    for p=(1:length(string)/5)
        characterbin=zeros(1,5);
        for cont=1:5
            vals=string(ind);
            vals1=dec2bin(vals);
            characterbin(cont)=str2double(vals1(end));
            ind=ind+1;
        end
        num=sum(characterbin.*(2.^(4:-1:0)))+1;
        DcodedMessageBin=[DcodedMessageBin Alphabet(num)];
end
```

تصویر*12* - بخش دوم تابع *decoding_freq*

توضیحات کد: در حلقه با گام های 100 تایی از ماتریس y جدا میکنیم، و آن را داخل ماتریس S ذخیره میکنیم.(هر سطر برابر 1 ثانیه است.) اکنون از هر سطر با استفاده از حلقه تبدیل فوریه میگیریم. به دلیل متقارن بودن تبدیل فوریه تنها به بخش مثبت نیاز داریم، به همین دلیل 50 تای آخر را جدا می‌کنیم و داخل متغیر positive می‌ریزیم. حالا باید فرکانسی پیدا کنیم که داخل آن پیک رخ می‌دهد، آن را داخل متغیر index می‌ریزیم.(دلیل منهای 1،درست کردن اندیس ها داخل متلب است.)
اکنون معادل باینری هر فرکانس را پیدا کنیم. فرکانس ها رو مجددا تعریف میکنیم. برای آستانه ها حد میذاریم. ابتدا فرکانس ها را داخل co میریزیم و thre اول را برابر با میانگین 0 و 1 co میذاریم، و برای ترش هولد های میانی حلقه تعریف میکنیم تا میانگین را بگیرد و برای ترش هولد آخری با 49 جمع میکنیم و بر 2 تقسیم می‌کنیم. حالا نوبت رمز گشایی میرسد. در نهایت در یک حلقه تو در تو روی s حرکت میکنیم و در حلقه داخلی روی ترش هولد ها حرکت میکنم و اگر بینq (thre(k) and thre(k+1 باشد k ام انتخاب میشود.سپس داخل z آن را می‌ریزیم و اندیس را باینری میکنیم و برای هر یک ثانیه این الگوریتم را تکرار میکنیم و آن را به ماتریس string می‌چسبانیم، در انتها 5 تا 5 عملیات جداسازی ر انجام داده و حروف را پیدا میکنیم.

تصویر *13* - پیام دیکود شده با سرعت *1*

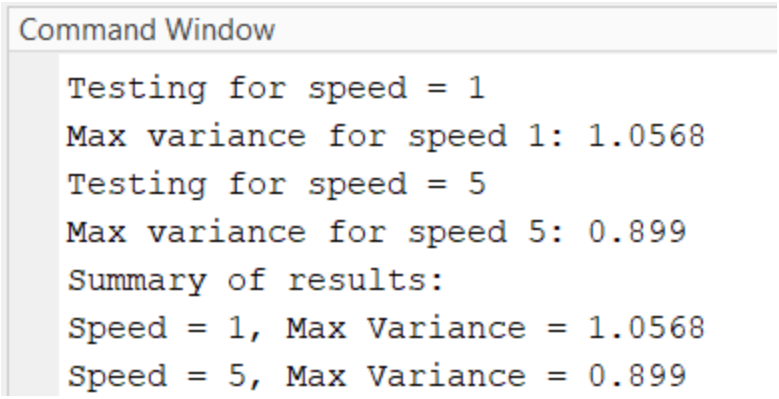
تصویر *14* - پیام دیکود شده با سرعت *5*

تمرین 2-(7,6,5)

```
my_message = 'signal';
speeds = [1, 3];
variance_values = 0:0.0001:2;
[y, x_axis] = coding_freq(my_message, speeds(1));

max_variance_matched = zeros(1, length(speeds));
for s_idx = 1:length(speeds)
    speed = speeds(s_idx);
    disp(['Testing for speed = ', num2str(speed)]);
    [encoded_signal, x_axis] = coding_freq(my_message, speed);
    for variance = variance_values
        noisy_signal = encoded_signal + sqrt(variance) * randn(size(encoded_signal));
        decoded_message = decoding_freq(noisy_signal, speed);
        if strcmp(decoded_message, my_message)
            max_variance_matched(s_idx) = variance;
        else
            break; % Stop when decoding fails
        end
    end
    disp(['Max variance for speed ', num2str(speed), ': ', num2str(max_variance_matched(s_idx))]);
end
% summary
disp('Summary of results:');
for s_idx = 1:length(speeds)
    disp(['Speed = ', num2str(speeds(s_idx)), ', Max Variance = ', num2str(max_variance_matched(s_idx))]);
end
```

تصویر *15* - کد افزودن نویز به سیگنال انکود شده و بررسی اثر نویز بر روی مقاومت بیت ریت های مختلف

در این بخش می خواهیم اثر نویز گوسی را بر روی سیگنال انکود شده و همچنین عملکرد تابع دیکود در حضور نویز بر روی سیگنال ورودی را بررسی کنیم. این آزمایش را بر روی سرعت ارسال اطلاعات با 1 و 5 بیت ریت بر ثانیه انجام می دهیم و در نهایت مشاهده می کنیم که کدام بیت ریت نسبت به نویز مقاوم تر است. به ازای هر بیت ریت ابتدا تابع انکودینگ را صدا می زنیم تا عبارت 'signal' در سیگنالی ،طبق تابع coding_freq که پیش تر معرفی شد، انکود شود. حال به این سیگنال انکود شده نویز گوسی با قدرت های متفاوت (واریانس که به عنوان ضریب در پشت سیگنال نویز ضرب می شود در حکم قدرت نویز است.) را با ان جمع می کنیم. به ازای هر یک از این سیگنال های آمیخته با نویز تابع دیکدینگ را صدا زده و عبارت خروجی را با عبارت اولیه مقایسه می کنیم و اگر برابر بودند ماکزیمم واریانس قابل تحمل برای دیکود شدن صحیح پیام را آپدیت میکنیم تا جایی این اپدیت شدن ادامه می یابد که دیگر سیگنال دیکود شده با پیام اولیه برابر نباشند و ان موقع از لوپ امتحان واریانس های مختلف خارج می شویم.
همان طور که در ابتدا مطرح شد این کار را یک بار برای سیگنالی که با سرعت 1 بیت بر ثانیه انکود شده و بار دیگر برای سیگنالی که با 5 بیت بر ثانیه انکود شده انجام می دهیم و با مقایسه ماکزیمم واریانس یا همان قدرت نویز قابل تحمل به این نتیجه میرسیم که سیگنالی که با سرعت 1 بیت بر ثانیه انکود شده نسبت به نویز مقاوم تر است و این با آنچه ما انتظار داشتیم هم همخوانی داشته چرا که با توجه به ثابت بودن پهنای باند، فرکانس های مرجع برای ارسال با یک بیت بر ثانیه اختلاف بیشتری با هم دارند پس وقتی نویز بر انها اثر می کند احتمال اینکه به صورت خطا مقدار را برابر مقداری دیگر ببینیم کم تر است.



تصویر*16* - ماکزیمم واریانس نویز قابل تحمل برای دیکود کردن صحیح پیام در بیت ریت های مختلف

تمرین 2-9)

اگر که تنها نرخ نمونه برداری فرکانس را تغییر دهیم و پهنای باند ثابت بماند نمی توان سرعت ارسال اطلاعات را به نویز مقاوم کرد چرا که اگر طول بردار زمانی ثابت باشد و fs را تنها تغییر دهیم ،رزولوشن فرکانسی نیز بی تغییر باقی می ماند پس بایستی پهنای باند زیاد شود تا فرکانس های انتخاب شده برای ارسال اطلاعات از یکدیگر به اندازه کافی فاصله داشته باشند تا با اضافه شدن نویز به سیگنال انکود شده، خطایی در دیکدینگ پیام رخ ندهد.