گزارش تمرین کامپیوتری 6 درس سیگنال‌ها و سیستم‌ها

بابک حسینی محتشم 810101408

محمدسینا پرویزی مطلق 810101394

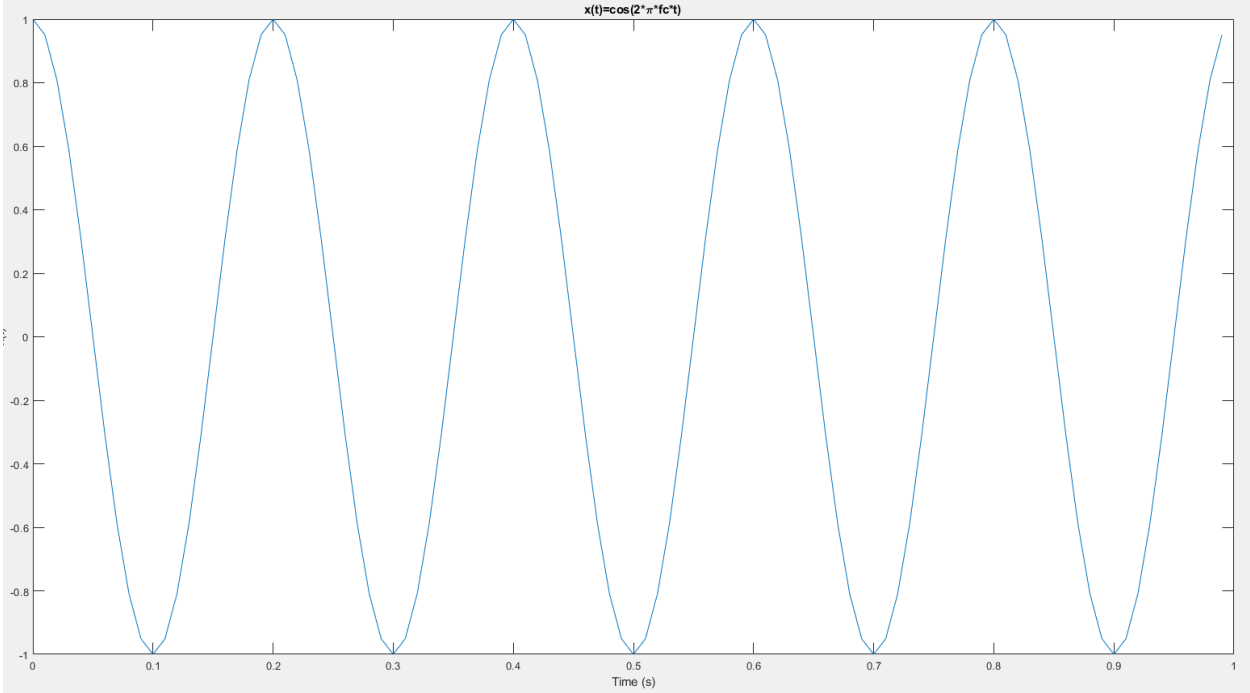
1403/10

بخش اول:

۱-۱) همان طور که خواسته شد نمودار سیگنال ارسالی توسط رادار را با کمک اسکریپت زیر با ثابت‌های داده شده رسم می‌کنیم:

```
fs=100;
ts=1/fs;
t_start=0;
t_end=1;
T=t_end-t_start;
N = fs * T;
t = t_start:ts:t_end-ts;
f = -fs/2:fs/N:fs/2-fs/N;
fc=5;
x=cos(2*pi*fc*t);
plot(t,x);
title('x(t)=cos(2*\pi*f\_c*t)');
xlabel('Time (s)');
ylabel 'x(t)';
```

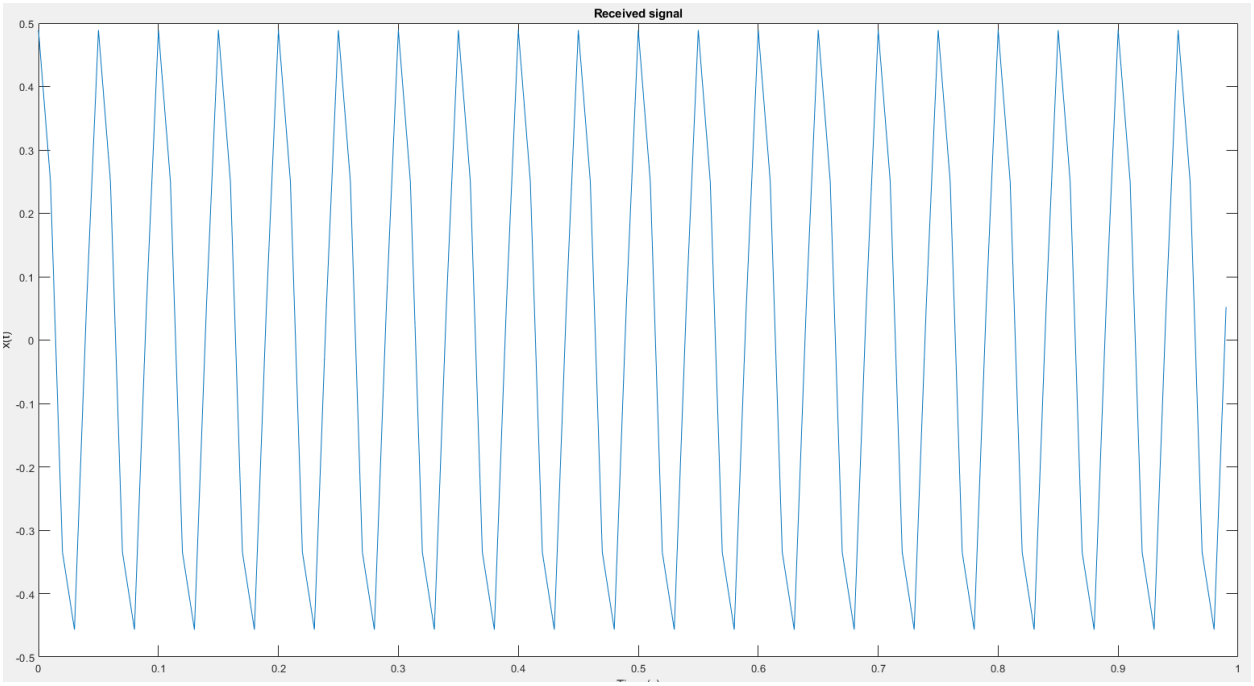
تصویر خروجی همان طور که انتظار داریم، شکل کسینوسی دارد:



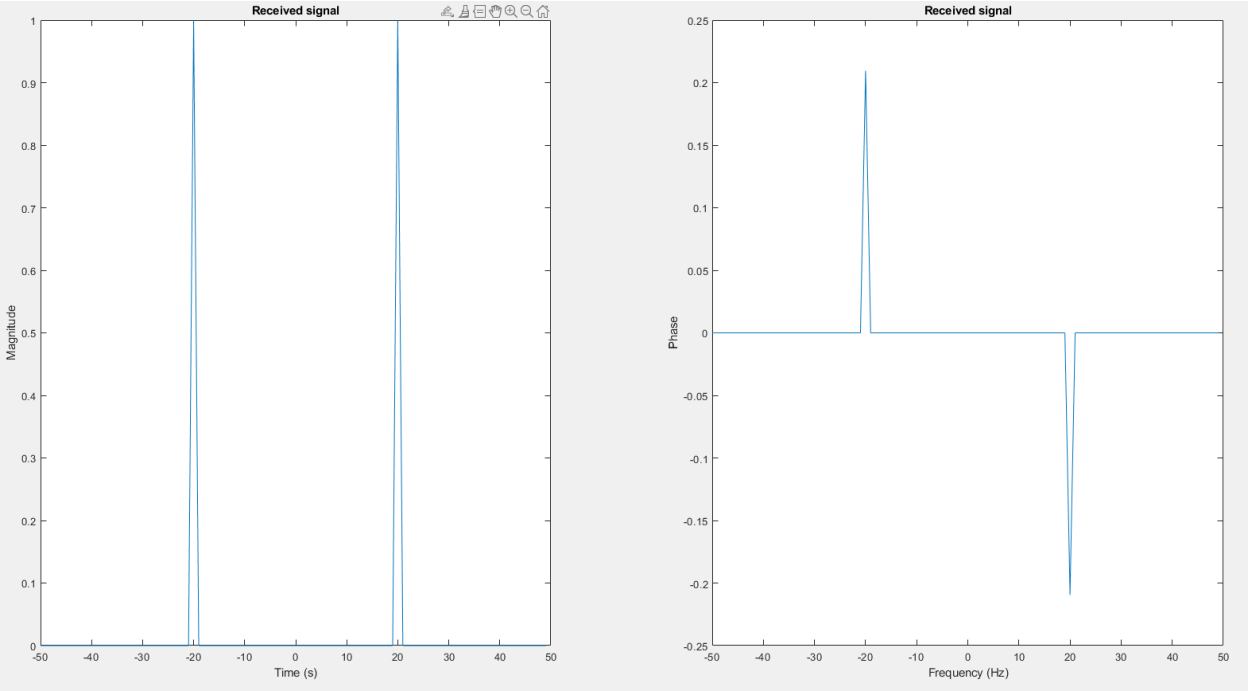
۲-۱) حال با داشتن بقیه ثابت‌ها، ابتدا تمام ثابت ها را به واحد SI تنبدی کرده و سپس سیگنال دریافت شده پس از ارسال را نیز تولید و رسم می‌کنیم:

```
fs=100;
ts=1/fs;
t_start=0;
t_end=1;
T=t_end-t_start;
N = fs * T;
t = t_start:ts:t_end-ts;
f = -fs/2:fs/N:fs/2-fs/N;
fc=5;
alpha=0.5;
beta=0.3;
R=250000;
V=180/3.6;
fd=beta*V;
C=3e8;
Ro=2/C;
td=Ro*R;
x=alpha*cos(2*pi*(fc+fd)*(t-td));
plot(t,x);
title('Received signal');
xlabel('Time (s)');
ylabel 'x(t)';
```

تصویر سیگنال:



۳-۱) برای حساب کردن فاصله و سرعت نیاز است اندازه فاز و فرکانس سیگنال دریافتی را پیدا کنیم که با بردن سیگنال دریافتی به فضای فرکانس می‌توانیم این دو را پیدا کنیم. پس از بردن سیگنال به حوزه فرکانس، اندازه آن را بررسی کرده و نقطه ای که پس از fc قرار دارد و اندازه در آن بیشینه است را برابر فرکانس در نظر میگیریم چون که می‌دانیم فرکانس سیگنال حداقل به اندازه fc است(در صورتی که fd مینیمم یعنی صفر باشد). با داشتن فرکانس، فاز را نیز می‌توانیم به دست آوریم. از این دو می‌توانیم fd و td را با روابط داده شده پیدا کنیم. تصویر سیگنال در حوزه فرکانس:



سپس با داشتن td و fd فاصه و سرعت را به دست می‌آوریم و همین طور که میبینم هر دو به درستی به دست می‌آیند:

```
X=fftshift(fft(x));
X=X/max(abs(X));
subplot(1,2,1);
plot(f,abs(X));
title('Received signal');
xlabel('Time (s)');
ylabel 'Magnitude';
tol = 1e-6;
X(abs(X) < tol) = 0;
theta = angle(X);
subplot(1,2,2);
plot(f,theta);
title('Received signal');
xlabel 'Frequency (Hz)';
ylabel 'Phase';
[M,new_freq]=max(X(length(X)/2+2+fc:length(X)));
new_freq=new_freq+fc;
new_phase=abs(angle(X(new_freq+length(X)/2+1)));
estimated_fd=new_freq-fc;
estimated_td=new_phase/(2*pi*(fc+estimated_fd));
estimated_V=estimated_fd/beta
estimated_R=estimated_td/Ro
```

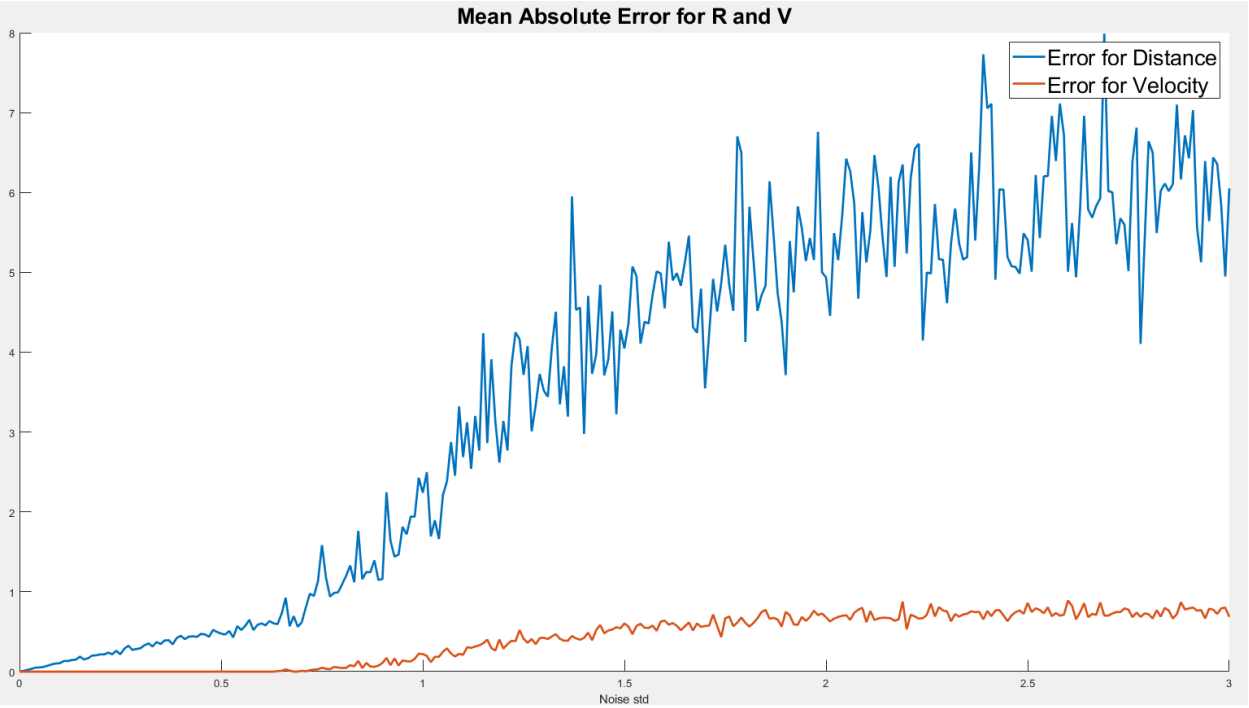
```
ommand Window
R =

      250000


V =

    50


estimated_V =

    50


estimated_R =

   2.5000e+05
```

۴-۱) به ازای نویز با انحراف معیار ۰.۰۱ تا ۳، نویز را به سیگنال ورودی اضافه کرده و سپس مانند قبل، از سیگنال حاصل، فاصله و سرعت را حساب می‌کنیم. این کار را ۱۰۰ بار برای هر مقدار نویز انجام می‌دهیم و هر بار نسبت خطای مطلق را حساب می‌کنیم و نهایتا میانگین میگیریم.

تصویر اسکریپت:

```
t_start=0;
t_end=1;
T=t_end-t_start;
N = fs * T;
t = t_start:ts:t_end-ts;
f = -fs/2:fs/N:fs/2-fs/N;
fc=5;
alpha=0.5;
beta=0.3;
R=250000;
V=180/3.6;
fd=beta*V;
C=3e8;
Ro=2/C;
td=Ro*R;
x_no_noise=alpha*cos(2*pi*(fc+fd)*(t-td));
std=0:0.01:3;
n_test=100;
error_R=zeros(size(std));
error_V=zeros(size(std));
idx=0;
for noise_std=std
    idx=idx+1;
    for j=1:n_test
        noise=noise_std*randn(size(x_no_noise));
        x=x_no_noise+noise;
        X=fftshift(fft(x));
        X=X/max(abs(X));
        tol = 1e-2;
        X(abs(X) < tol) = 0;
        [M,new_freq]=max(X(length(X)/2+2+fc:length(X)));
        new_freq=new_freq+fc;
        new_phase=abs(angle(X(new_freq+length(X)/2+1)));
        estimated_fd=new_freq-fc;
        estimated_td=new_phase/(2*pi*(fc+estimated_fd));
        estimated_V=estimated_fd/beta;
        estimated_R=estimated_td/Ro;
        error_R(idx)=error_R(idx)+(abs(estimated_R-R))/(n_test*R);
        error_V(idx)=error_V(idx)+(abs(estimated_V-V))/(n_test*V);
    end
end
```

تصویر نمودار خروجی خطا برای سرعت و فاصله:



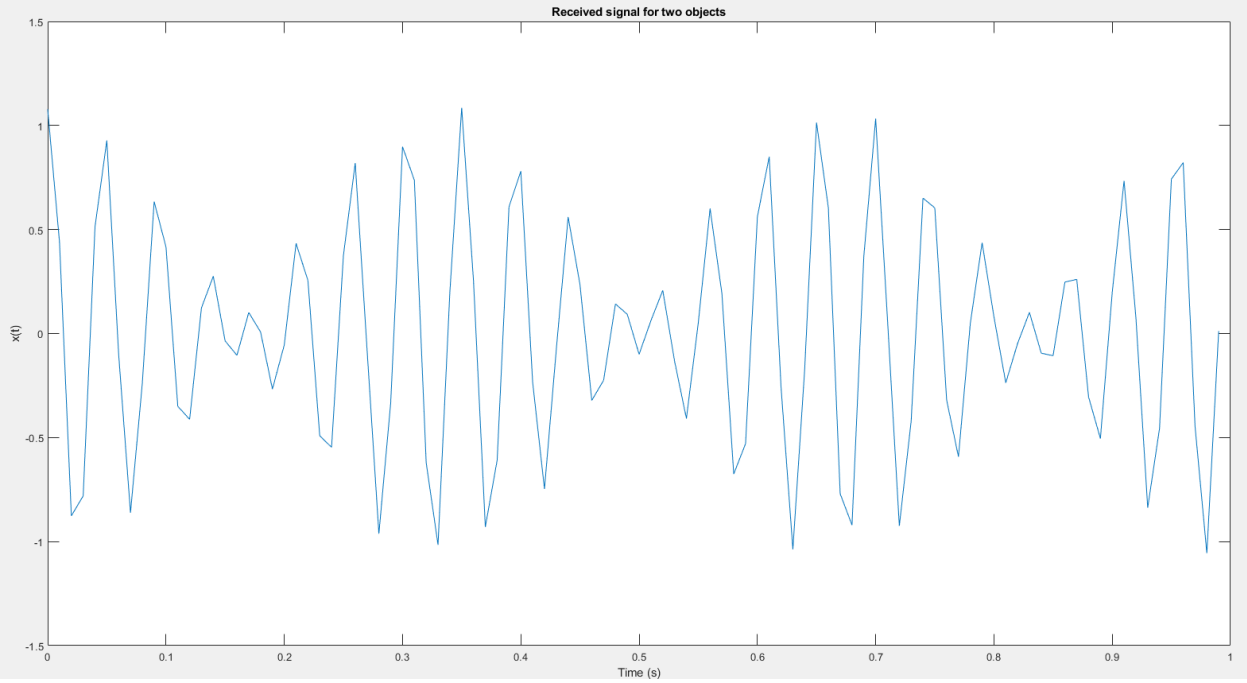
می‌توان دید که حساسیت فاصله به نویز بسیار بیشتر از حساسیت سرعت به نویز است و به نظر ما دلیل آن است که در رابطه حساب کردن فاصله نیاز به td تخمین زده شده داریم و برای تخمین زدن td هم به فرکانس و هم فاز سیگنال نیاز داریم که هر دو تخمینی هستند پس خطای تخمین این دو باعث خطای بیشتر در td و در نتیجه در فاصله می‌شود در صورتی که برای حساب کردن سرعت تنها تخمین فرکانس کافی است. دلیل دیگری که شاید بتوان ذکر کرد این است که چون برای فرکانس argmax می‌گیریم ولی به فاز مستقیم نویز اضافه می‌شود پس فاز همواره خطا دارد در صورتی که اگر فرکانس کم شود ولی همچنان بیشینه باشد باز هم به درستی تشخیص می‌دهیم.

همچنین از نمودار مشخص است که برای فاصله تا نویز با انحراف معیار ۰.۵ خطای کمتر از ۵۰ درصد داریم در صورتی که برای سرعت با نویز با مقدار ۱.۳ خطا کمتر از ۵۰ درصد است.

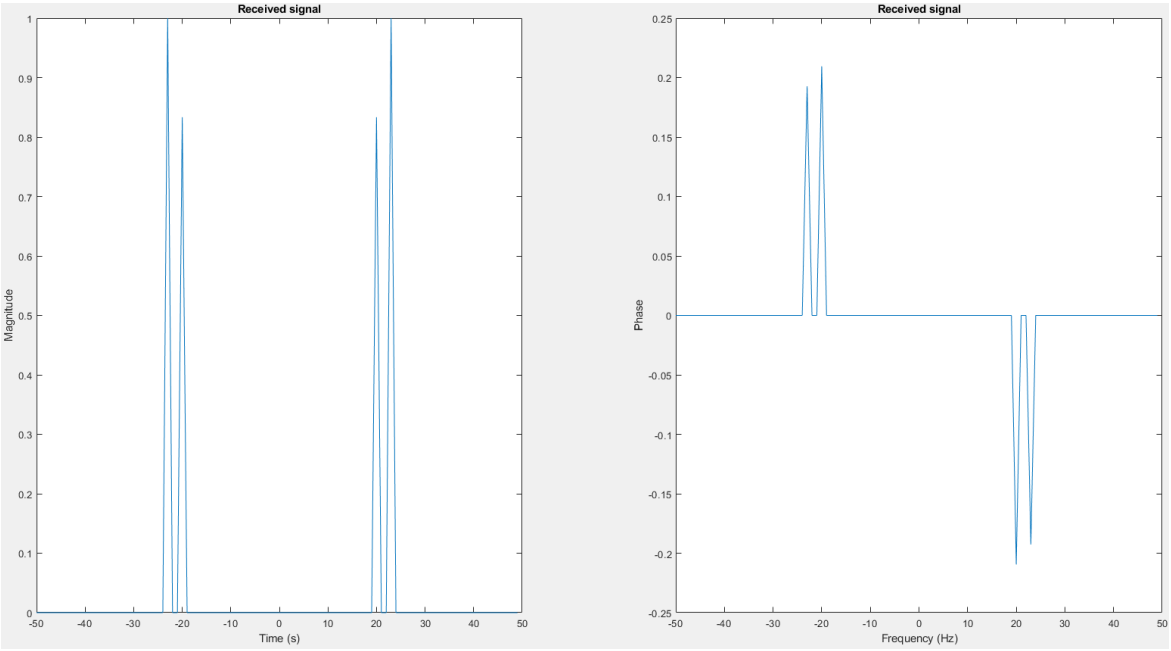
۵-۱) دو سیگنال دریافت شده را ایجاد و جمع آن‌ها را رسم می‌کنیم. تصویر اسکریپت:

```
fs=100;
ts=1/fs;
t_start=0;
t_end=1;
T=t_end-t_start;
N = fs * T;
t = t_start:ts:t_end-ts;
f = -fs/2:fs/N:fs/2-fs/N;
fc=5;
beta=0.3;
alpha1=0.5;
R1=250000;
V1=180/3.6;
fd1=beta*V1;
td1=Ro*R1;
alpha2=0.6;
R2=200000;
V2=216/3.6;
fd2=beta*V2;
td2=Ro*R2;
C=3e8;
Ro=2/C;
x1=alpha1*cos(2*pi*(fc+fd1)*(t-td1));
x2=alpha2*cos(2*pi*(fc+fd2)*(t-td2));
plot(t,x1+x2);
title('Received signal for two objects');
xlabel('Time (s)');
ylabel 'x(t)';
```

تصویر خروجی:



۶-۱) ابتدا سیگنال را در حوزه فرکانس رسم می‌کنیم:



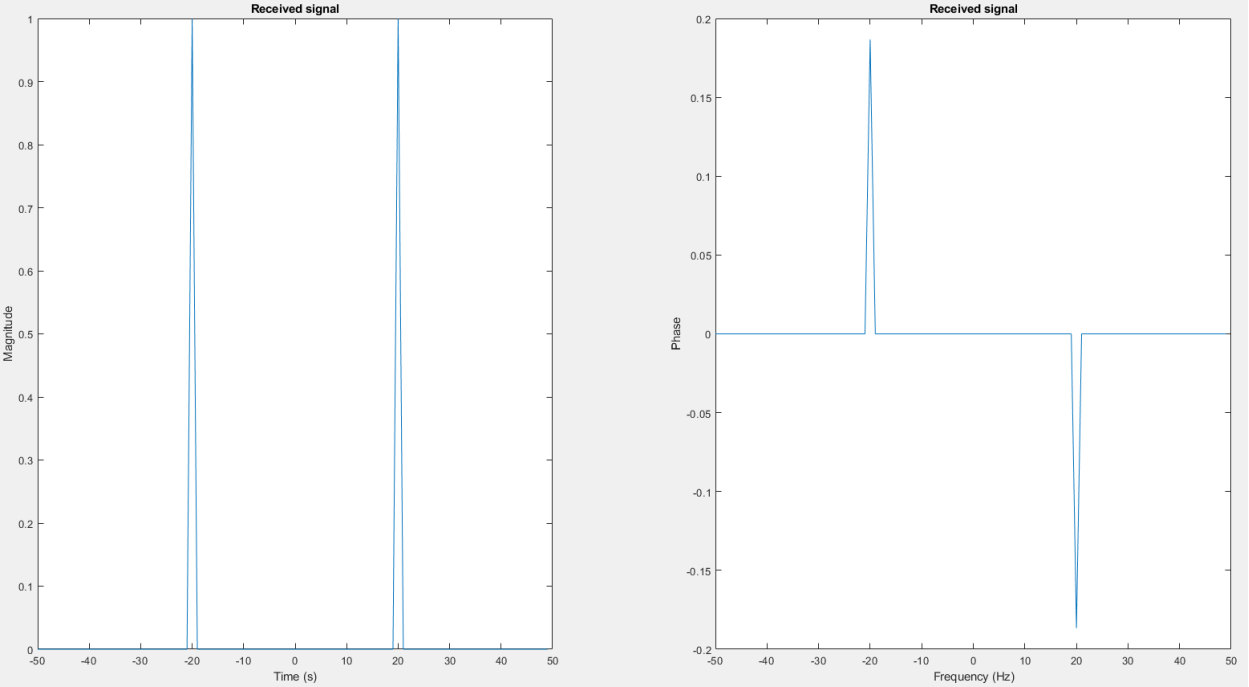
می‌توان مشاهده کرد که حال چهار قله در حوزه فرکانس برای این سیگنال وجود دارد که لزوما قله‌ای که اندازه بزرگتری دارد فاز بزرگتری ندارد. برای پیدا کردن دو فاز و دو سیگنال مانند قبل عمل می‌کنیم و پس از پیدا کردن یک فرکانس و ذخیره فاز متناظر آن، سیگنال را در آن خانه را صفر کرده و دنبال نقطه بیشنیه بعدی می‌گردیم و بدین ترتیب می‌توان فاز و اندازه هر دو سیگنال را پیدا کرد. تصویر اسکریپت:

```
theta = angle(X);
subplot(1,2,2);
plot(f,theta);
title('Received signal');
xlabel 'Frequency (Hz)';
ylabel 'Phase';
[M,new_freq1]=max(X(length(X)/2+2+fc:length(X)));
new_freq1=new_freq1+fc;
arg_max=new_freq1+length(X)/2+1;
new_phase1=abs(angle(X(arg_max)));
estimated_fd1=new_freq1-fc;
estimated_td1=new_phase1/(2*pi*(fc+estimated_fd1));
estimated_V1=estimated_fd1/beta
estimated_R1=estimated_td1/Ro
X(arg_max)=0;
[M,new_freq2]=max(X(length(X)/2+2+fc:length(X)));
new_freq2=new_freq2+fc;
new_phase2=abs(angle(X(new_freq2+length(X)/2+1)));
estimated_fd2=new_freq2-fc;
estimated_td2=new_phase2/(2*pi*(fc+estimated_fd2));
estimated_V2=estimated_fd2/beta
estimated_R2=estimated_td2/Ro
```

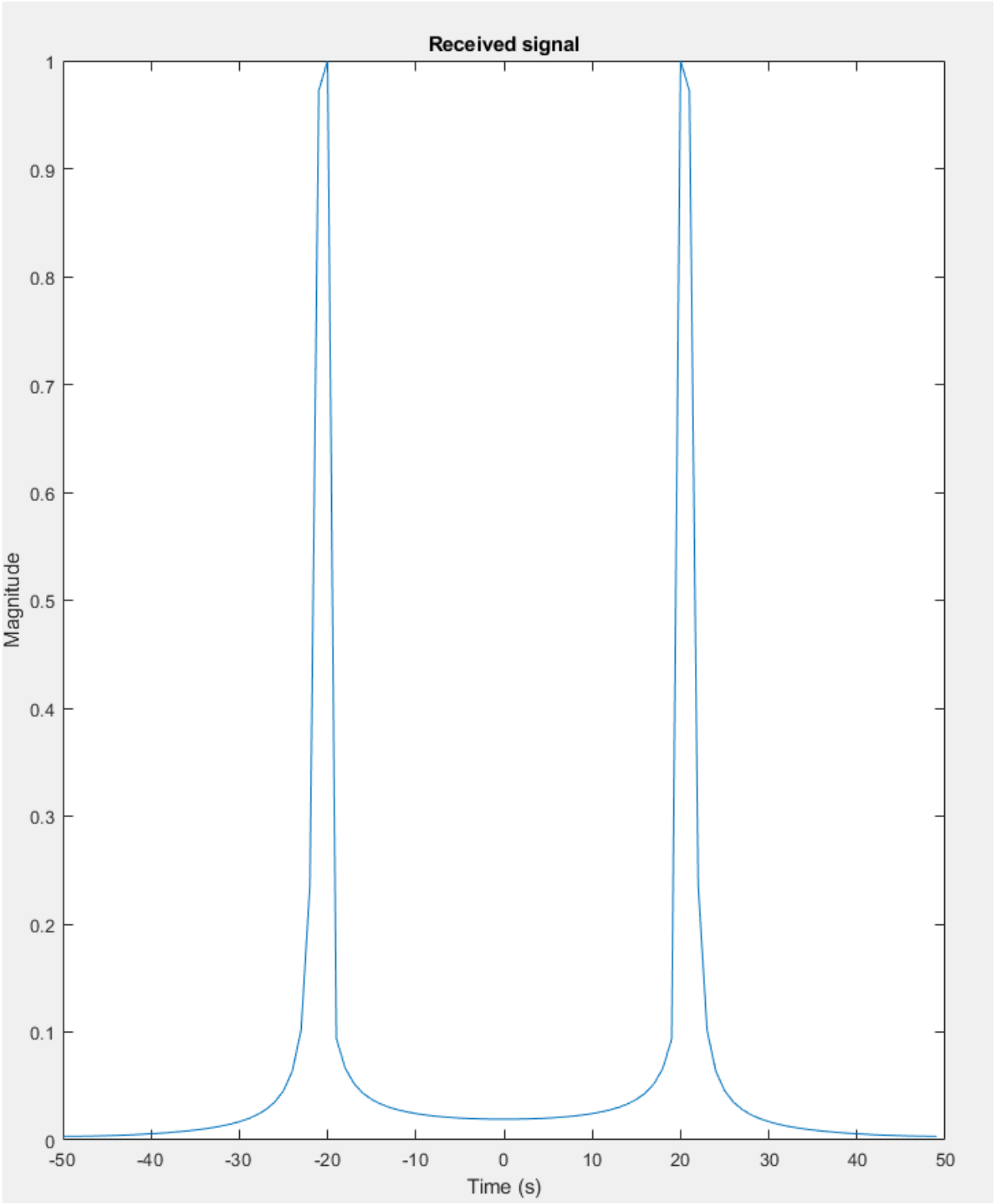
ommand Window

```
estimated_V1 =

    60


estimated_R1 =

   2.0000e+05


estimated_V2 =

    50


estimated_R2 =

   2.5000e+05
```

۷-۱) می‌دانیم فرکانس سیگنال با سرعت رابطه مستقیم دارد پس اگر اختلاف دو سرعت خیلی کم باشد، در حوزه فرکانس، دو سیگنال قابل تفکیک نخواهند بود. برای مثال اگر هر دو سرعت را برابر ۱۸۰ قرار دهیم شکل زیر نشان‌دهنده سیگنال در حوزه فرکانس است که مشخص است دو سیگنال روی هم افتاده‌اند:



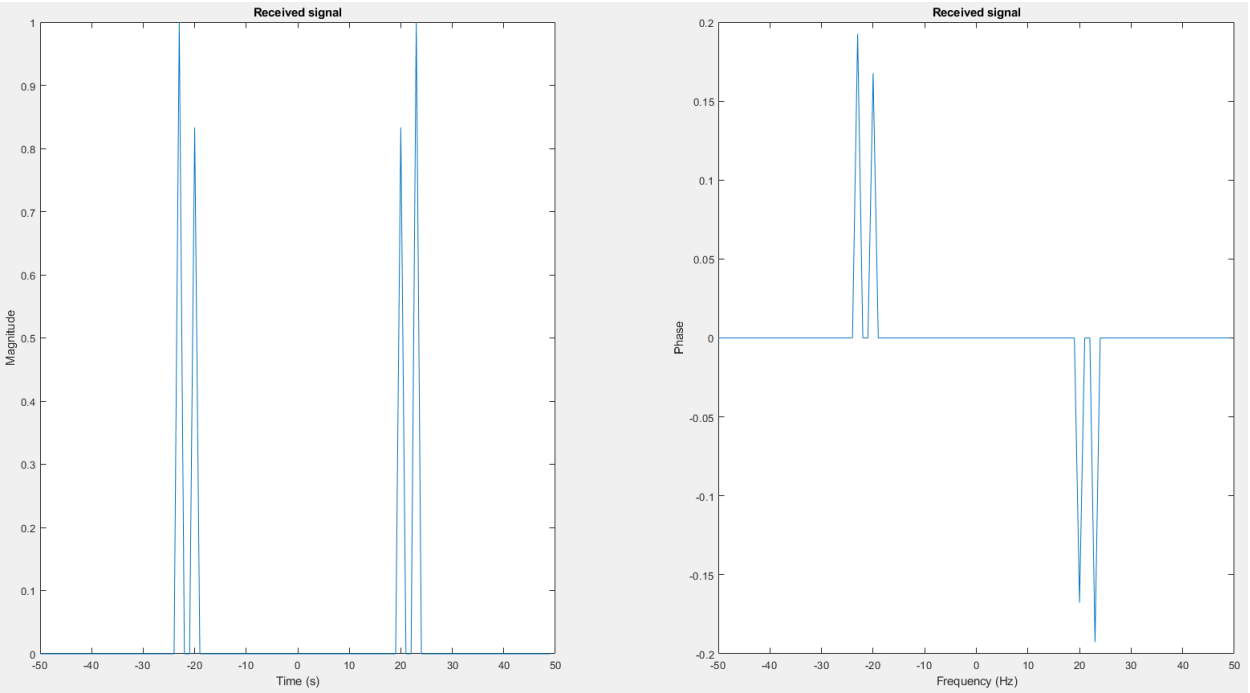
دلیل این موضوع همان طور که در مقدمه توضیح داده شد، همان رزولوشن فرکانسی است. پس می‌توان حداقل اختلاف فرکانس برای قابل تفکیک بودن را بدست آورد که از رابطه موجود در مقدمه می‌شود ۱ هرتز. حال می‌توانیم با تقسیم این فرکانس حداقل بر بتا، به اختلاف سرعت حداقل برسیم که می‌شود حدود ۳.۳۳ متر بر ثانیه. پس سرعت جسم دوم را ۴ متر بر ثانیه بیشتر از جسم دیگر قرار می‌دهیم و می‌توان در اندازه سیگنال دوباره چهار قله مشاهده کرد:


Received signal
Magnitude
Time (s)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-50
-40
-30
-20
-10
0
10
20
30
40
50

۸-۱) چون فاصله تنها روی مقدار فاز تاثیر دارد پس اگر فاز دو جسم برابر باشد باز هم قادر به تشخیص آن‌ها خواهیم بود پس با برابر قرار دادن فاصله، دوباره برنامه‌مان را تست می‌کنیم.

```
estimated_V1 =

    60


estimated_R1 =

   2.0000e+05


estimated_V2 =

    50


estimated_R2 =

   2.0000e+05
```

می‌توان دید تخمین‌ها به درستی صورت گرفته و در تصویر زیر هم مشخص است که سیگنال‌ها در حوزه فرکانس قابل تشخیص هستند:



۹-۱) اگر تعداد اجسام را ندادیم، باید روشی ارائه دهیم که تعداد قله‌های موجود در حوزه فرکانس را بتوان پیدا کرد. مثلا می‌توان یک threshold مشخص کرد و تعداد نقاطی که در حوزه فرکانس اندازه بیشتر از آن دارند نشان‌دهنده دو برابر تعداد اجسام است.

بخش دوم:

۲-۱) ابتدا تابع add_note را تشکیل می‌دهیم. این تابع سه ورودی دریافت می‌کند: یک آرایه که نشان‌دهنده یک موسیقی است. یک نت که به صورت حرف دریافت می‌کند و یک متغیر که مشخص می‌کند آیا نت مورد نظر نیمی از زمان استاندارد فعال است یا به اندازه یک زمان استاندارد فعال است.

این تابع نت ورودی را به مدت خواسته شده، به آرایه موسیقی داده شده اضافه می‌کند و آرایه جدید را برمی‌گرداند. تصویر اسکریپت تابع:

```
function music_out=add_note(is_half,note,music_in)
    Notes = ["C" "C#" "D" "D#" "E" "F" "F#" "G" "G#" "A" "A#" "B"];
    Freqs = [523.25 554.37 587.33 622.25 659.25 698.46 739.99 783.99
    d = dictionary(Notes,Freqs);
    t_start=0;
    t_end=0.5;
    T=t_end-t_start;
    fs=8000;
    ts=1/fs;
    t_normal=t_start:ts:t_end-ts;
    t_half=t_start:ts:t_end/2-ts;
    tau=0.025;
    if(is_half)
        music_out=[music_in sin(2*pi*d(note)*t_half)];
    else
        music_out=[music_in sin(2*pi*d(note)*t_normal)];
    end
    silence=zeros([1,tau*fs]);
    music_out=[music_out silence];
end
```

حال با تولید موسیقی خواسته شده این تابع را درستی آزمایی می‌کنیم. تصویر اسکریپت:

```
music=add_note(false,'D',music);
music=add_note(false,'E',music);
music=add_note(false,'F#',music);
music=add_note(false,'E',music);

music=add_note(true,'D',music);
music=add_note(true,'E',music);
music=add_note(true,'E',music);
music=add_note(true,'D',music);
music=add_note(true,'F#',music);
music=add_note(true,'D',music);
music=add_note(false,'E',music);

music=add_note(false,'D',music);
music=add_note(true,'E',music);
music=add_note(true,'D',music);
music=add_note(false,'F#',music);
music=add_note(false,'E',music);

music=add_note(false,'D',music);
music=add_note(true,'E',music);
music=add_note(true,'D',music);
music=add_note(false,'F#',music);
music=add_note(false,'E',music);

music=add_note(true,'D',music);
music=add_note(true,'D',music);
music=add_note(false,'E',music);
music=add_note(true,'F#',music);
music=add_note(true,'E',music);
music=add_note(false,'F#',music);

music=add_note(true,'F#',music);
music=add_note(true,'E',music);
music=add_note(false,'F#',music);
music=add_note(false,'F#',music);
music=add_note(false,'D',music);

fs=8000;
sound(music,fs);
audiowrite('music1.wav',music,fs);
```

می‌توان شباهت بین صدا با آهنگ Happy Birthday را تشخیص داد که البته تفاوت‌هایی به دلیل استفاده تنها از یک نت حس می‌شود.

۲-۲) ما نت Twinkle Twinkle Little Star را نتخاب و به طور مشابه در اسکریپتی نوشتیم و حاصل به شکلی که انتظار داشتیم شد. با بررسی properties فایل حاصل، مشاهده کردیم که نرخ بیت آن ۱۲۸ کیلوبیت بر ثانیه است. که قابل پیشبینی بود زیرا فرکانس نمونه‌برداری را ۸ کیلوهرتز انتخاب کردیم و هر خانه از موسیقی تولیدی ۱۶ بیت حافظه اشغال می‌کند که حاصل ضرب این دو، ما را به ۱۲۸ کیلوبیت بر ثانیه می‌رساند.

۲-۳) برای پیاده‌سازی عملیات معکوس، یعنی تبدیل صوت به نت‌ها و اینکه هر نت در نیمی از زمان استاندارد یا یک زمان استاندارد اجرا شود یا خیر، تابع extract_notes را می‌نویسیم. این تابع، یک آرایه که نشان دهنده موسیقی است را می‌گیرد و بررسی می‌کند در ابتدای موسیقی چه نتی است و اینکه نصفه هست یا خیر و این دو را برمی‌گرداند و همچنین باقی موسیقی بدون نت اول را نیز برمی‌گرداند. ابتدا این تابع با بررسی سکوتی که باید پس از هر نت باشد متوجه می‌شود که نت ابتدا، نصفه بوده یا خیر. سپس از روی تبدیل فوریه نت، فرکانس بیشینه را یافته و به ازای هر فرکانس، اختلاف آن را با تمام نت‌ها مقایسه می‌کند و نتی که کمترین اختلاف را داشته به عنوان نت اول موسیقی برمی‌گرداند.

```
function [is_half,note,music_out]=extract_note(music_in)
    Notes = ["C" "C#" "D" "D#" "E" "F" "F#" "G" "G#" "A" "A#" "B"];
    Freqs = [523.25 554.37 587.33 622.25 659.25 698.46 739.99 783.99
    t_start=0;
    t_end=0.5;
    T=t_end-t_start;
    fs=8000;
    tau=0.025;
    len=length(music_in);
    idx_normal=(T)*fs;
    idx_half=(T/2)*fs;
    if(idx_normal+tau*fs>len || music_in(idx_half+1)==0)
        is_half=true;
        music_out=music_in(idx_half+tau*fs+1:len);
        X=fftshift(fft(music_in(1:idx_half)));
        X=X/max(abs(X));
        [~,new_freq]=max(X(length(X)/2+2:length(X)));
        new_freq=new_freq*2/T;
    else
        is_half=false;
        music_out=music_in(idx_normal+tau*fs+1:len);
        X=fftshift(fft(music_in(1:idx_normal)));
        X=X/max(abs(X));
        [~,new_freq]=max(abs(X(length(X)/2+2:length(X))));
        new_freq=new_freq/T;
    end
    min_dif=1e5;
    for i=1:length(Freqs)
        if(abs(new_freq-Freqs(i))<min_dif)
            min_dif=abs(new_freq-Freqs(i));
            note=Notes(i);
        end
    end
end
```

قطعه‌ای که در قسمت قبل تولید کردیم را به این تابع می‌دهیم و مشاهده می‌کنیم که به درستی تمام نت‌ها را تشخیص می‌دهد.

```
1   [music, ~]=audioread('mysong.wav');
2   i=1;
3   while ~isempty(music)
4       [is_half,note,music]=extract_note(music);
5       fprintf('%d: Note: %c is_half: %d\n', i,note,is_half);
6       i=i+1;
7   end
```

ommand Window

```
>> p2_3
1: Note: C is_half: 1
2: Note: C is_half: 1
3: Note: G is_half: 1
4: Note: G is_half: 1
5: Note: A is_half: 1
6: Note: A is_half: 1
7: Note: G is_half: 0
8: Note: F is_half: 1
9: Note: F is_half: 1
10: Note: E is_half: 1
11: Note: E is_half: 1
12: Note: D is_half: 1
13: Note: D is_half: 1
14: Note: C is_half: 0
15: Note: C is_half: 1
16: Note: C is_half: 1
17: Note: G is_half: 1
18: Note: G is_half: 1
19: Note: A is_half: 1
20: Note: A is_half: 1
21: Note: G is_half: 0
22: Note: F is_half: 1
23: Note: F is_half: 1
24: Note: E is_half: 1
25: Note: E is_half: 1
26: Note: D is_half: 1
27: Note: D is_half: 1
28: Note: C is_half: 0
```