گزارش پروژه ششم درس سیگنال و سیستم‌ها

کوثر شیری جعفرزاده 810101456 پریا پاسه‌ورز 810101393

بخش اول

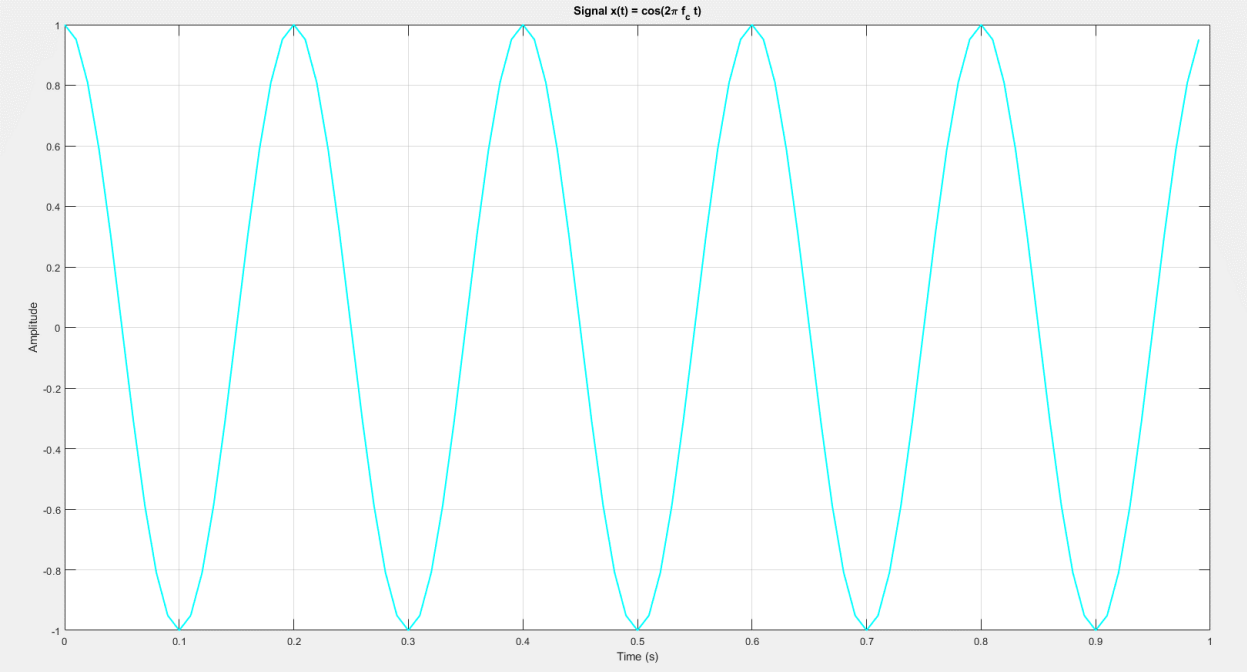
تمرین 1_1)

در این بخش سیگنال ارسالی را که از رابطه $x(t) = cos(2\pi f_c t)$ پیروی می‌کند را رسم کرده‌ایم.

کد:

```
%% PLOT SENT SIGNAL

f_s = 100;
f_c = 5;
t_start = 0;
t_end = 1;

t = linspace(t_start , t_end, f_s+1);
t(end) = [];

x_t = cos(2*pi*f_c*t);

figure;
plot(t, x_t, 'c', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Amplitude');
title('Signal x(t) = cos(2\pi f_c t)');
grid on;
```

خروجی:



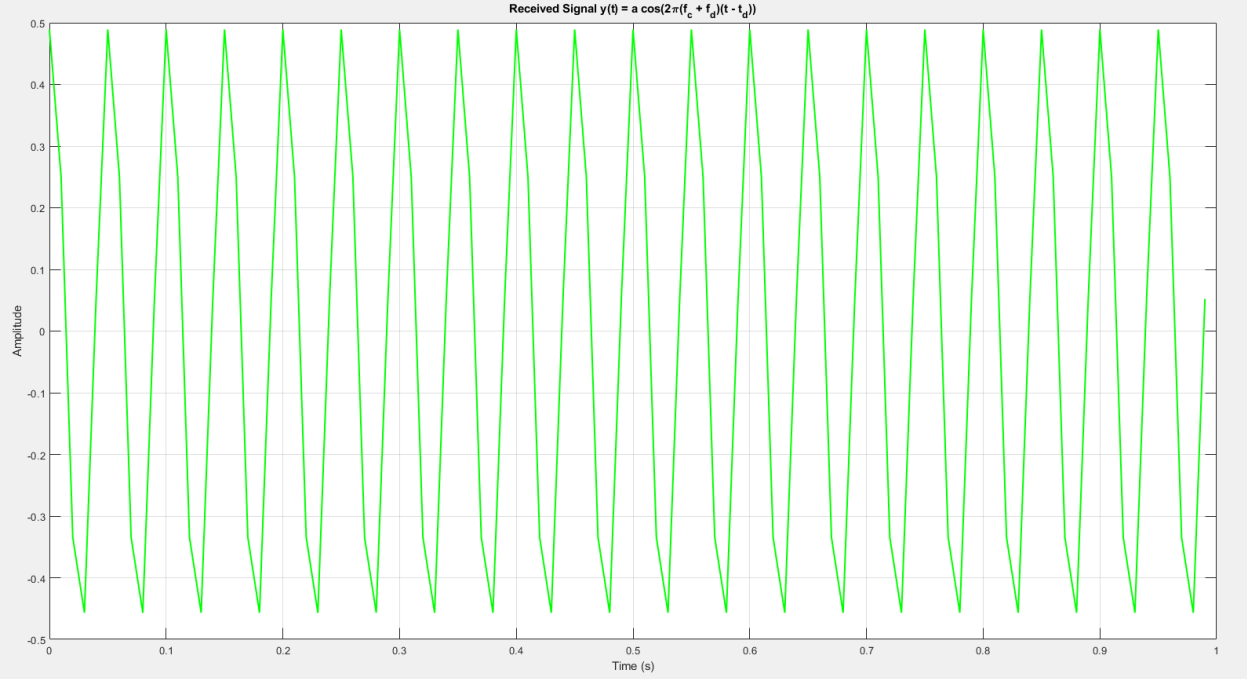
تمرین 1_2)

در این بخش سیگنال دریافتی را که از رابطه $y(t) = \alpha cos\big(2\pi(f_c + f_d)(t - t_c)\big)$ پیروی می‌کند را رسم کرده‌ایم.

کد:

```
%% PLOT RECEIVED SIGNAL

alpha = 0.5;
betha = 0.3;
c = 3e08;
R = 250000;
V = 180 * 1000 / 3600;
t_d = 2/c*R;
f_d = betha*V;
y_t = alpha * cos(2*pi*(f_c+f_d)*(t-t_d));

figure;
plot(t, y_t, 'g', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Amplitude');
title('Received Signal y(t) = a cos(2\pi(f_c + f_d)(t - t_d))');
grid on;
```

خروجی:



تمرین 1_3)

در این بخش می‌خواهیم از روی سیگنال دریافتی f_d و t_d را تخمین زده و سپس از طریق آنها سرعت فاصله را پیدا کنیم. به همین منظور سیگنال دریافتی را به شکل زیر بازنویسی می‌کنیم:

$$y(t) = a\cos[2\pi(f_c + f_d)(t - t_d)]$$

$$y(t) = a\cos[2\pi(f_c + f_d)t - 2\pi(f_c + f_d)t_d]$$

$$y(t) = a\cos(2\pi f_{new}t + \phi_{new})$$

سیگنال دریافتی، سیگنالی با فرکانس f_{new} و فاز ϕ_{new} است.

کد:

```
%% ESTIMATE VELOCITY AND DISTANCE USE RECEIVED SIGNAL

[estimated_V, estimated_R] = estimate_speed_and_distance(y_t, f_s, f_c, c , betha);
disp(['Estimated Velocity: ', num2str(estimated_V)])
disp(['Estimated Distance: ', num2str(estimated_R)])
fprintf('\n');
```

کد تابع estimate_speed_and_distance:

```
function [estimated_V, estimated_R] = estimate_speed_and_distance(y_t, f_s, f_c, c, betha)
    [t_d , f_d, ~, ~] = find_t_and_f_dopler(y_t, f_s, f_c);
    estimated_R = t_d/2*c;
    estimated_V = f_d/betha;
end
```

این تابع با استفاده از تابع find_t_and_f_dopler زمان تاخیر و فرکانس داپلر را محاسبه می‌کند، سپس با توجه به روابط

$$t_d = \frac{2R}{c}, f_d = \beta v$$

مسافت و سرعت را محاسبه می‌کند.

کد تابع find_t_and_f_dople:

```
function [td ,fd, fourier, peek_freq_index] = find_t_and_f_dopler(echo_signal, fs, fc)
    fourier = fftshift(fft(echo_signal));
    [td, fd, peek_freq_index] = extract_data_from_fourier(fourier, fs, fc);
end
```

این تابع از سیگنال داده شده تبدیل فوریه می‌گیرد، سپس به کمک تبدیل به دست آمده، زمان تاخیر و فرکانس داپلر و نقطه‌ای که فرکانس فالب تشخیص داده شده است را محاسبه می‌کند.

کد تابع extract_data_from_fourier:

```
function [t_d, f_d, peek_freq_index] = extract_data_from_fourier(fourier, f_s, f_c)
    fr_range = -f_s/2:1:f_s/2-1;
    positive_frtransform = fourier(:, f_s/2+1:end);
    positive_freq = fr_range(:, f_s/2+1:end);
    [~, peek_freq_index] = max(abs(positive_frtransform));
    f_d_plus_f_c_found = positive_freq(peek_freq_index);
    f_d = f_d_plus_f_c_found - f_c;
    new_phase = angle(positive_frtransform(peek_freq_index));
    t_d = (new_phase) / (-pi*2*f_d_plus_f_c_found);
    peek_freq_index = peek_freq_index + f_s/2;
end
```

در این تابع، ابتدا بازه فرکانس متناظر با تبدیل فوریه را مشخص می‌کنیم. تبدیل فوریه طبق خاصیت Hermitian symmetry نسبت به مبدا متقارن است، به همین دلیل فقط روی یک بخش از آن (در اینجا بخش مثبت) تمرکز کرده‌ایم و تبدیل فوریه و فرکانس آن را جدا کرده‌ایم. سپس نقطه پیک تبدیل فوریه را پیدا کرده‌ایم. مقدار موجود در این نقطه همان f_{new} است که از جمع f_d و f_c به دست می‌آید. پس از اینجا فرکانس داپلر محاسبه می‌شود.

با کمک تابع angle، فاز را در نقطه پیک محاسبه کرده‌ایم. این فاز همان ϕ_{new} است که از رابطه

$$\phi_{\text{new}} = -2\pi(f_c + f_d)t_d$$

محاسبه می‌شود. پس زمان تاخیر نیز الان قابل محاسبه است. در نهایت از آنجایی که که range فرکانسی در ابتدا به صورت fr_range = -f_s/2:1:f_s/2-1 در نظر گفته شده است، لازم است ایندکس پیک نهایی را به اندازه f_s/2 شیفت دهیم.

خروجی:

```
Estimated Velocity: 50
Estimated Distance: 250000
```

همانطور که می‌بینیم، سرعت و فاصله به درستی تخمین زده شده‌اند.

تمرین 1_4)

در این بخش می‌خواهیم به سیگنال دریافتی کمی نویز اضافه کنیم و بسنجیم تا چه قدرتی از نویز، سرعت و فاصله به درستی تخمین زده می‌شوند و کدام یک به نویز حساس‌تر است.

```
noise_levels = 0:0.01:2;
v_detected = true(size(noise_levels));
R_detected = true(size(noise_levels));
```

مقادیر نویز در بازه 0 تا 2 هستند و با گامهای 0.01 افزایش پیدا می‌کنند. دو آرایه نیز به اندازه نویزها تعریف کرده‌ایم تا درست یا غلط تخمین زده شدن را در آنها ذخیره کنیم.

```
for i = 1:length(noise_levels)
    noise = noise_levels(i) * randn(1, f_s);
    noisy_signal = y_t + noise;

    [found_V, found_R] = estimate_speed_and_distance(noisy_signal, f_s, f_c, c, betha);

    % disp(['Noise Level: ', num2str(noise_levels(i))]);
    % disp(['Estimated Velocity: ', num2str(found_V)]);
    % disp(['Estimated Distance: ', num2str(found_R)]);
    % disp('**************************************');

    if V ~= found_V
        v_detected(i) = false;
    end
    if R ~= found_R
        R_detected(i) = false;
    end
end
```

حال در یک حلقه، با کمک تابع randn نویزهایی با قدرت مشخص شده می‌سازیم و به سیگنالمان اضافه می‌کنیم. سپس سرعت و فاصله را با استفاده از تابع estimate speed and distance تخمین می‌زنیم.

در نهایت اگر سرعت و فاصله تخمین زده شده با سرعت و فاصله حقیقی یکسان نبود، ایندکس متناظر با آن قدرت نویز را در آرایه false می‌کنیم.

```
% Determine sensitivity
v_noise_limit = noise_levels(find(~v_detected, 1, 'first'));
R_noise_limit = noise_levels(find(~R_detected, 1, 'first'));

disp(['Velocity can be detected up to noise level: ', num2str(v_noise_limit)]);
disp(['Distance can be detected up to noise level: ', num2str(R_noise_limit)]);

if isempty(v_noise_limit)
    disp('Velocity detection is robust to the tested noise levels.');
end
if isempty(R_noise_limit)
    disp('Distance detection is robust to the tested noise levels.');
end

if ~isempty(v_noise_limit) && ~isempty(R_noise_limit)
    if v_noise_limit < R_noise_limit
        disp('Velocity is more sensitive to noise.');
    elseif R_noise_limit < v_noise_limit
        disp('Distance is more sensitive to noise.');
    else
        disp('Velocity and Distance are equally sensitive to noise.');
    end
end

fprintf('\n');
```

اولین قدرت نویزی که از آن به بعد خروجی ناصحیح بوده است را برای سرعت و فاصله پیدا می‌کنیم و گزارش می‌کنیم. سپس با مقایسه شماره ایندکس مشخص می‌کنیم که کدام یک نسبت به نویز حساس‌تر بوده است.

خروجی:

```
Velocity can be detected up to noise level: 0.82
Distance can be detected up to noise level: 0
Distance is more sensitive to noise.
```

همانطور که مشاهده می‌کنیم، سرعت تا قدرت نویز 0.82 به نویز مقاوم بوده است، ولی فاصله اصلا به افزودن نویز به سیگنال مقاوم نیست.

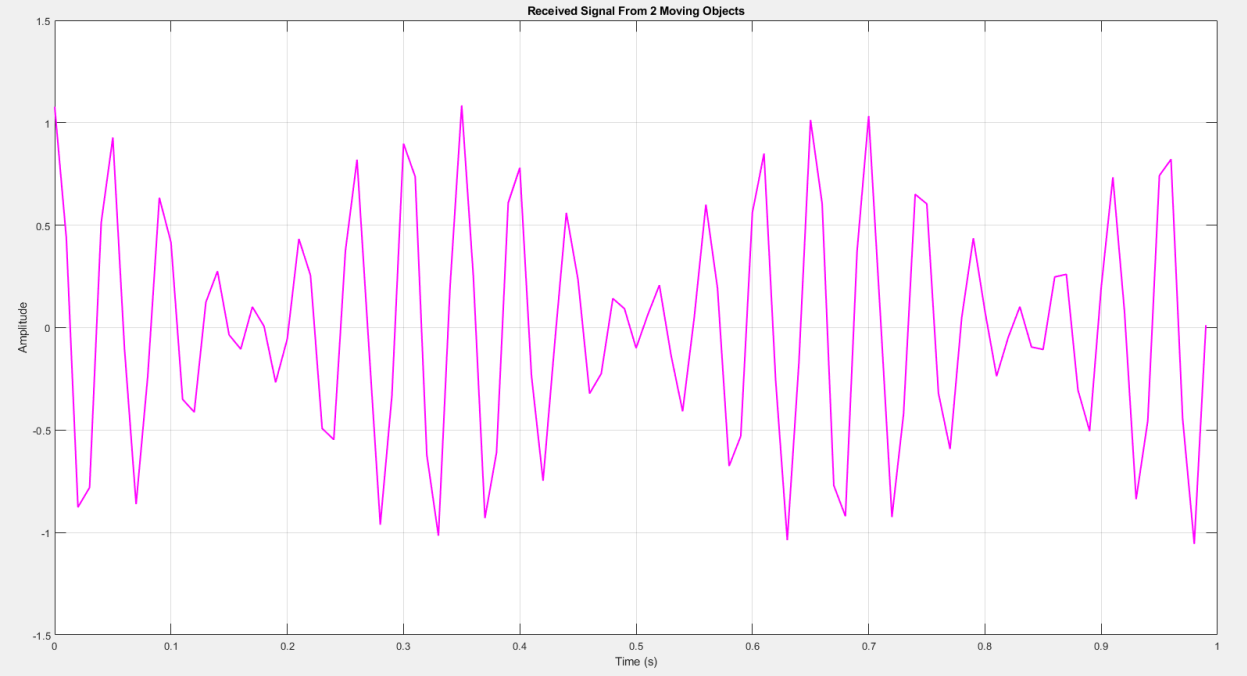
تمرین 1_5)

در این بخش دو جسم داریم که سیگنال دریافتی رادار، جمع اکوهای برگشتی از این دو جسم است. هر یک را محاسبه و جمع کرده و در نهایت رسم می‌کنیم:

کد:

```
%% PLOT RECEIVED SIGNAL FROM TWO MOVING OBJECTS

alpha_1 = 0.5;
alpha_2 = 0.6;

betha = 0.3;
c = 3e08;

R_1 = 250000;
R_2 = 200000;

V_1 = 180 * 1000 / 3600;
V_2 = 216 * 1000 / 3600;

t_d_1 = 2/c*R_1;
t_d_2 = 2/c*R_2;

f_d_1 = betha*V_1;
f_d_2 = betha*V_2;

y_t_1 = alpha_1 * cos(2*pi*(f_c+f_d_1)*(t-t_d_1));
y_t_2 = alpha_2 * cos(2*pi*(f_c+f_d_2)*(t-t_d_2));
y_t = y_t_1 + y_t_2;

figure;
plot(t, y_t, 'm', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Amplitude');
title('Received Signal From 2 Moving Objects');
grid on;
```

خروجی:



تمرین 1_6)

در این بخش می‌خواهیم از روی سیگنال دریافتی، سرعت و مسافت هر یک از دو جسم را پیدا کنیم:

کد:

```
%% ESTIMATE VELOCITY AND DISTANCE USE RECEIVED SIGNAL WITH 2 MOVING OBJECTS

[estimated_V_1, estimated_R_1, estimated_V_2, estimated_R_2] = estimate_speed_and_distance_for_two_objects(y_t, f_s, f_c, c , betha);
disp(['Estimated Velocity For Moving Object 1: ', num2str(estimated_V_1)])
disp(['Estimated Distance For Moving Object 1: ', num2str(estimated_R_1)])
fprintf('\n');
disp(['Estimated Velocity For Moving Object 2: ', num2str(estimated_V_2)])
disp(['Estimated Distance For Moving Object 2: ', num2str(estimated_R_2)])
```

کد تابع estimate_speed_and_distence_for_two_objects:

```
function [found_V1, found_R1, found_V2, found_R2] = estimate_speed_and_distance_for_two_objects(echo_signal_comb, fs, fc, c, betha)
    [found_V1, found_R1, filterd_fourier] = find_and_filter_highest_frq(echo_signal_comb, fs, fc, c, betha);
    [td2, fd2, ~] = extract_data_from_fourier(filterd_fourier, fs, fc);
    found_V2 = fd2/betha;
    found_R2 = td2/2*c;
end
```

سیگنال دریافتی در اینجا چون دو تا جسم داریم، پس دو تا پیک دارد. اول با کمک تابع find_and_filter_highest_frq، سرعت و مسافت جسم اول را پیدا می‌کنیم و پیک آن را از سیگنال اولیه حذف می‌کنیم، سپس سیگنال جدید را به تابع extract_data_from_fourier که قبلا تعریف کرده بودیم می‌دهیم تا زمان تاخیر و فرکانس داپلر جسم دیگر را نیز محاسبه کند و بتوانیم سرعت و مسافت جسم دیگر را از آن استخراج کنیم.

کد تابع find_and_filter_highest_freq:

```
function [found_V1, found_R1, filterd_fourier] = find_and_filter_highest_frq(echo_signal_comb, fs, fc, c , betha)
    [td1, fd1, foureier, peek_indx] = find_t_and_f_dopler(echo_signal_comb, fs, fc);
    foureier(1, peek_indx) = 0;
    filterd_fourier = foureier;
    found_V1 = fd1/betha;
    found_R1 = td1/2*c;
end
```

در این تابع، اول با کمک تابع find_t_and_f_dopler زمان تاخیر، فرکانس داپلر، تبدیل فوریه و نقطه پیک اول را پیدا می‌کنیم. سپس سرعت و مسافت را از روی این اطلاعات به دست آورده و نقطه پیک را از روی سیگنالمان فیلتر کرده و از بین می‌بریم تا بتوانیم پیک دوم را تشخیص دهیم. نهایتا سرغت و مسافت و سیگنال فیلتر شده را برمی‌داریم.

خروجی:

```
Estimated Velocity For Moving Object 1: 60
Estimated Distance For Moving Object 1: 200000

Estimated Velocity For Moving Object 2: 50
Estimated Distance For Moving Object 2: 250000
```

همانطور که مشاهده می‌کنیم، سرعت و مسافت برای هر دو جسم به درستی تخمین زده شده‌اند.

تمرین 1-7)

خیر غیرقابل تشخیص خواهند بود دلیل این امر این است که اگر سرعت دو جسم برابر باشد (یا نزدیک به هم باشد)در حوزه فوریه، فرکانس هاي این دوجسم روی هم خواهد افتاد. با توجه به اینکه از فضای فوریه استفاده می کنیم، درصورتی که اختلاف این دو از رزولوشن فرکانسی کمتر باشد امکان تفکیک فرکانس ها فراهم نیست لذا نمی توان سرعت و فاصله ها را استخراج کرد.

از بین چهار پارامتر داده شده(v1,v2,v3,v4) تنها می توان یکی از سرعت هارا درست تشخیص داد و سه پارامتر دیگر قابل تشخیص نیستند .

$$y(t) = \alpha \cos\big((2\pi(f_c + f_d)(t - t_d)\big)$$

با توجه به اینکه در اینجا رزولوشن فرکانسی برابر 1 است پس نیازی داریم f_d با هم یک واحد فاصله داشته باشند داریم:

$$f_d = \beta v = 0.3 * v$$

$$(f_{d1} - f_{d2}) > 1$$

$$0.3 * (v_1 - v_2) > 1$$

$$(v_1 - v_2) > 3.33 \frac{m}{s}$$

$$(v1 - v_2) > 11.88 \frac{km}{h}$$

همانطور که محاسبات بالا نشان می دهد باید سرعت ها حداقل 12 کیلومتر بر ساعت اختلاف داشته باشند تا تشخیص تمامی پارامتر ها ممکن باشد

تمرین 1-8)
با برابر بودن فاصله، صرفا مقدار فاز سیگنال تغییر خواهد کرد و تاثیری روی فرکانس و تحلیل فضای فوریه ندارد(دو سیگنال می توانند دارای فاز برابر اما فرکانس های متفاوتی باشند) پس می توان هردو مقدار سرعت و فاصله هردو سیگنال را به شکل خوبی به دست آورد.

تمرین 1-9)
بله این امکان وجود دارد

اگر تعداد اجسام را ندانیم سیگنال دریافتی را گرفته و از ان تبدیل فوریه میگیریم سپس باید تعداد پیک ها را در یک سمت حساب کنیم (در شرایط قبلی تنها سیگنال غالب را انتخاب می کردیم اما در اینجا به دلیل امکان وجود چند جسم باید تمامی پیک هارا شناسایی کنیم، دلیل اینکه تنها یک سمت را در نظر میگیریم این است که با توجه به سیگنال دریافتی شکلی متقارن حول 0 ایجاد می شود که تنها یک سمت آن مدنظر ماست).

برای محاسبه سرعت و فاصله نیز باید در هر فرکانس f_{new} را محاسبه کنیم بدین ترتیب سرعت و فاز استخراج می شوند که می توان با استفاده از فاز، فاصله را نیز حساب کرد (این فرآیند برای تمامی پیک ها که نمایانگر اجسام هستند تکرار می شود).

بخش دوم

تمرین 2-1)
در ابتدا نیاز است تا مقادیر اولیه مورد نیاز را تولید کنیم:

```
fs = 8000;
ts = 1/fs;
tstart = 0;
tend = 0.5;
tstop = 0.025;
ton = tstart:ts:tend-ts;
rest = tstart:ts:tstop-ts;
stop = zeros(size(rest));
```

همانطور که مشخص است فرکانس برابر 8 کیلو هرتز قرار گرفته است. زمان شروع و پایان به ترتیب برابر 0 و 0.5 گذاشتیم حال در قدم بعدی نیاز است تا بازه ی زمانی بین 0 تا 0.5 را با step size برابر

$$\frac{1}{fs} = \frac{1}{8000} = ts$$

تقسیم بندی کنیم.

از طرف دیگر می دانیم که بین نت های نواخته شده باید به اندازه ی 0.025 ثانیه سیگنال صفر قرار دهیم تا نت ها از هم قابل تشخیص باشند (stop در اینجا نمایانگر همین فاصله بین نت هاست).

در قدم بعدی باید نت ها و فرکانس های متناظر را به صورت یک سلول ایجاد کنیم تا بتوانیم در ادامه با استفاده از نت به فرکانس متناظر آن دست پیدا کنیم.

```
Notes = ["C","C#","D","D#","E","F","F#","G","G#","A","A#","B"];
Frequencies = [523.25,554.37,587.33,622.25,659.25,698.46,739.99,783.99,830.61,880,932.33,987.77];
```

حال باید نت هایی که به منظور ایجاد کردن در داخل پی دی اف آمده است را وارد کنیم:

```
input_song = {'D' ,'D' ,'G' ,'F#', 'D'  ,    'D' , 'E' , 'E' , 'D' , 'F#', 'D' , 'E'    ,   'D' , 'E' , 'F#', 'E'
              0.25,0.25, 0.5, 0.5, 0.5  ,    0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5    ,   0.5 , 0.5 , 0.5 , 0.5
```

در اینجا همانگونه که در دستور ذکر شده است باید به ازای فشردن کامل کلید عدد 0.5 و درصورت نیمه فشردن آن 0.25 را در نظر بگیریم.

اما نکته ای در اینجا حائز اهمیت است می دانیم که سیگنال سینوسی ما در بین 0 تا 0.5 است و فشردن کامل یک کلید برابر خود این سیگنال است پس در ادامه باید در 0.5 عدد 2 را ضرب کنیم تا مطمئن شویم که کل سیگنال در نظر گرفته می شود از طرف دیگر اکر کلید نیمه فشرده شود پس باید به اندازه 0.25 باشد این در حالس است که اگر در دامنه 0.5 عدد 0.25 را ضرب کنیم به عدد 0.125 می رسیم که اشتباه است پس نیاز است در اینجا نیز 2*0.25 را در دامنه اصلی ضرب کنیم.

پس به طور کلی میتوان گفت درصورت فشردن کامل باید ضریب 1 در سیگنال سینوسی ضرب شود و با نیمه فشردن این کلید باید 0.5 در سیگنال ضرب شود

به منظور تطابق شکل دستور پروژه با دیتاست ایجاد شده این تغییر بعدا و در هنگام تعیین سیگنال اعمال می شود.

```
out_voice = [];
for i = 1:length(input_song)

    n = find(Notes == input_song{1,i});
    note = sin(2*pi*Frequencies(n)*ton);
    duration =  2* input_song{2,i} * size(ton,2);
    out_voice = [out_voice note(1:duration) stop];
end
```

در اینجا یک آرایه خالی تعریف می کنیم که خروجی را در خود نگه دارد، برروی نوت های ورودی شروع به حرکت می کنیم به ازای هر نوت، فرکانس متناظر آن را بدست می آوریم.

در قدم بعدی باید برحسب فرکانس به دست آمده سیگنال سینوسی را تشکیل دهیم این کار با بازه ی زمانی که در ابتدا تعیین کردیم بین 0 تا 0.5 اتفاق می افتد. وقتی سیگنال به دست آمد حال نوبت آن است که مشخص کنیم آیا کلید کامل فشرده شده است یا نه !

همانطور که قبل هم اشاره کردیم در اینجا تغییری به وجود می آوریم بدین شکل که اگر کلید کامل فشرده شده باشد به اندازه کل سیگنال باشد پس در 2 ضرب می شود و در عین حال اگر قرار است به اندازه نصف سیگنال باشد باید 2*0.25 ضرب شود تا نتیجه درست باشد.

در قدم آخر این سیگنال را به مجموعه سیگنال های خروجی اضافه کنیم و باتوجه به اینکه بعد از هر نوت یک استراحت 0.025 باید داشته باشیم بعد اضافه شدن سیگنال جدید stop را که نمایانگر همان استراحت است به انتهای آن اضافه می کنیم.

```
sound(out_voice)
audiowrite('part1_song.wav', out_voice, fs);
```

در نهایت نیز صدای ایجاد شده را پخش می کنیم.

باتوجه به اینکه در قسمت سوم نیاز داریم تا برروی این صدا کار انجام دهیم این صدا را با فرکانس تعیین شده ذخیره می کنیم.

تمرین 2-2)

برای این بخش از اهنگ زیر استفاده می کنیم:

C – C – G – G – A – A – G – F – F – E – E – D – D – C – G – G – F – F – E – E – D – G – G – F – F – E – E – D – C – G – G – A – G – F – F – E – E – D – C.

This sequence of notes correspond to the entire Twinkle Twinkle Little Star piano song, played with the right hand only. If you press all these notes one after the other, you will easily recognize the melody of the famous song.

نحوه ی عملکرد ما به صورت گذشته است :

untitled | p2_1.m | p2_2.m

```
fs = 8000;
ts = 1/fs;
tstart = 0;
tend = 0.5;
tstop = 0.025;
ton = tstart:ts:tend-ts;
rest = tstart:ts:tstop-ts;
stop = zeros(size(rest));

Notes = ["C","C#","D","D#","E","F","F#","G","G#","A","A#","B"];
Frequencies = [523.25,554.37,587.33,622.25,659.25,698.46,739.99,783.99,830.61,880,932.33,987.77];

song = { 'C','C','G','G','A','A','G','F','F','E','E','D','D','C','G','G','F','F','E','E','D','G','G','F','F','E','E','D','C','G'
         0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5 0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5

mysong = [];
for i = 1:length(song)
    n = find(Notes == song{1,i});
    note = sin(2*pi*Frequencies(n)*ton);
    duration = 2*song{2,i} * size(ton,2);
    mysong = [mysong note(1:duration) stop];

end

sound(mysong)
audiowrite('mysong.wav',mysong,fs)
```

در ابتدا مقادیر ثابت را تعریف کرده ایم و سپس دیتاست مربوط به نت ها را ایجاد کرده ایم، نت های نوشته شده برای آهنگ مدنظر را وارد می کنیم و میزان نگه داشتن هریک از کلید هارا تعیین می کنیم.

در ادامه مانند گذشته یک حلقه داریم که برحسب فرکانس مدنظر مقدار سینوس را محاسبه کرده و درون متغییر my song ذخیره می کند و پس از اتمام هر نوت به آن استراحت 0.025 ثانیه ای می دهد.

در نهایت صدای تولید شده را پخش کرده و آن را با نام تعیین شده در صورت پروژه و فرکانس 8000 ذخیره می کنیم.

برای فهمیدن اینکه هر سمپل داده ای با چند بیت ذخیره شده است به قسمت workspace می رویم:

Value	Name	Size	Bytes
1x85800 double	mysong	1x85800	686400

همانطور که مشخص است متغییر mysong دارای 85800 سمپل با 686400 بایت می باشد برای اینکه بفهمیم هر سمپل برای ذخیره سازی از چند بایت استفاده کرده است داریم:

$$\frac{686400}{85800} = 8\ byte$$

با توجه به اینکه هر بایت از 8 بیت تشکیل شده است پس برای ذخیره سازی هر سمپل به 64 بیت نیاز داریم.

تمرین 2-3)

نحوه انجام این فرآیند بدین شکل است که ما می دانیم چیزی که هر یک از نت هارا از هم جدا کرده است همان سکوت یا وقفه 0.025 ثانیه ای بین نت های ماست پس کافیست به دنبال این سکوت بگردیم و از پایان سکوت قبلی تا ابتدای سکوت فعلی را یک نت در نظر بگیریم.

برای اینکه مطمئن شویم این روش کار می کند میزان جلو رفتن ما در هرگام به اندازه طول سکوت است چون اگر بیشتر باشد ممکن است که ابتدا و انتهای سکوت را تشخیص ندهیم و اگر کمتر از 0.025 باشد امکان دارد دو بار سکوت در نظر بگیرد.

حال که توانستیم بازه های مربوط به هر نوت را در نظر بگیریم این نکته پر اهمیت می شود که طول هر نت را چقدر بگیریم؟

با توجه به اینکه نحوه تشخیص شروع و پایان یک نوت براساس سکوت بود پس با کم کردن این دو ایندکس می توانیم میزان فشرده شدن هر کلید را به دست بیاوریم (با توجه به اینکه میزان فشرده شدن هر نت برابر 0.5 یا 0.25 بود در صورتی که این فاصله را ثابت در نظر می گرفتیم این موضوع در تشخیص صحیح نت ها اشکال وارد می کرد)

برای تشخیص نوع نوت باید از تبدیل فوریه استفاده کنیم:

باتوجه به اینکه سیگنال تولید شده یک سیگنال سینوسی است اگر از آن تبدیل فوریه بگیریم به فرم زیر می شود

$$\sin(w_0 t) \xrightarrow{F} \pi(\delta(w + w_0) - \delta(w - w_0))$$

$$2f_c = w_0$$

همانطور که مشخص است تبدیل فوریه این تابع در صورت رسم اندازه آن دارای دوتا ضربه در فرکانس های قرینه هم هست پس کافیست یک سمت اندازه این تبدیل فوریه را در نظر گرفته و فرکانس غالب را پیدا کنیم و نت متناظر با آن فرکانس را استخراج کنیم (حتی اگز در این بخش نویز وارد شود چون ما نزدیک ترین فرکانس به عدد را انتخاب م یکنیم تا حدی به تصمیم گیری بهتر کمک می کند)

بدین ترتیب ما همزمان توانستیم هم میزان فشرده شدن یک نت و هم نت مربوطه را پیدا کنیم.

توضیح مثال این بخش:

در ابتدا فایل صوتی مدنظر را می خوانیم و دیتاست مربوطه برای استخراج نوت ها برحسب فرکانس را ایجاد می کنیم:

```
[audio, fs] = audioread('part1_song.wav');

notes = ["C","C#","D","D#","E","F","F#","G","G#","A","A#","B"];
frequencies = [523.25,554.37,587.33,622.25,659.25,698.46,739.99,783.99,830.61,880,932.33,987.77];

t_step = 0.025;
samples_per_step = round(t_step * fs);
silence_threshold = 0.01;
```

همانطور که می دانیم در این روش به دنبال سکوت بین نوت ها می گردیم تا بتوانیم نت مدنظر را تشخیص دهیم پس نیاز است تا این نوت سکوت را به ازای فرکانس داده شده ایجاد کنیم تا بتوان از آن بهره برد، از طرف دیگر ممکن است در این زمان صفر مطلق نبوده و اندکی از آن بیشتر باشیم پس یک مقدار به عنوان threshold معرفی می کنیم تا بتوانیم حالات را از هم تشخیص دهیم.

```
detected_notes = [];
detected_duration=[];
i = 1;
while i <= length(audio)
    segment = audio(i:min(i+samples_per_step-1, length(audio)));
    if max(abs(segment)) < silence_threshold
        i = i + samples_per_step;
        continue;
    end
    start_idx = i;
    while i <= length(audio)
        segment = audio(i:min(i+samples_per_step-1, length(audio)));
        if max(abs(segment)) < silence_threshold
            break;
        end
        i = i + samples_per_step;
    end
    end_idx = i - 1;
    duration = (end_idx - start_idx + 1) / fs;
    active_segment = audio(start_idx:end_idx);
    fft_result = fft(active_segment);
    S_mag = abs(fft_result(1:floor(length(fft_result)/2)));
    f_axis = linspace(0, fs/2, length(S_mag));
    [~, max_idx] = max(S_mag);
    dominant_freq = f_axis(max_idx);
    [~, note_idx] = min(abs(frequencies - dominant_freq));
    detected_notes = [detected_notes, notes(note_idx)];
    detected_duration=[detected_duration duration];
end
```

در اینجا ابتدا یک آرایه خالی برای ذخیره نوت های استخراج شده و همچنین مدت زمان فشردن هر یک از این کلید ها تعیین می کنیم سپس در قدم بعدی حلقه ای را برروی سمپل های موجود اغاز می کنیم:

ابتدا یک سگمنت به اندازه ی مقدار سکوت بین نوت ها جدا می کنیم اگر این بخش جدا شه مقدار صفر داست پس همان سکوت بین نوت هاست و ما باید بدون انجام عملیاتی از آن گذر کنیم اما اگر این مقدار برابر 0 نبود پس باید به سراغ استخراج نوت مربوطه برویم.

برای استخراج کردن کل نوت مربوطه با توجه به اینکه ممکن است کلید برای 0.25 یا 0.5 فشرده شده باشد پس شروع می کنیم از ایندکس فعلی به اندازه 0.025 گام برمیداریم تا زمانی که به سکوت بین دو نوت برسیم در آن صورت می فهمیم که نوت مورد نظر را یافته ایم.

حال برای یافتن مدت رمان این نوت کافیست ایندکس شروع را از ایندکس پایان کم کنیم (نکته حائز اهمیت این است که چون ما به محض دیدن شروع سکوت ایندکس را به روزرسانی نکرده ایم پس می دانیم که از ابتدای نت فعلی تا سر آغاز سکوت را داریم)

حال که توانستیم ایندکس شروع و پایان را بدست بیاوریم آن بخش موسیقی را استخراج می کنیم و از آن تبدیل فوریه می گیریم همانطور که در بخش تئوری گفته شده است این موضوع باعث می شود که ما دو ضربه به عنوان پاسخ در فرکانس های قرینه هم دریافت کنیم که نمایانگر فرکانس غالب و مربوط به نت مدنظر ماست.

همانطور که ذکر شد فرکانس ها قرینه یکدیگر خواهند بود پس نیاز است تا دامنه را نصف کرده و در بخش مثبت و البته اندازه ی این تبدیل فوریه به دنبال فرکانس غالب بگردیم پس از یافتن این فرکانس غالب آن را از فرکانس های مربوط به نت های موسیقی کم می کنیم با این کار نزدیک ترین عدد به صفر نوت مربوطه را به ما نشان می دهد که یعنی توانستیم نوت متناظر با فرکانس را پیدا کنیم.

در نهایت این نوت و میزان فعال بودن آن را به خروجی اضافه می کنیم و به استخراج سایر نت ها می پردازیم.

لازم به ذکر است که اپدیت شدن اندیس i در طی فرآیند تشخیص انجام می شود.

```
disp('Detected Data:');
for j=1:5:length(detected_notes)
    fprintf('%s ',detected_notes{j:min(j+4,length(detected_notes))});
        fprintf('\n');
    for i=j:min(j+4,length(detected_duration))
        fprintf("%.3f ",detected_duration(i));
    end
    fprintf('\n');
end
```

درنهایت پس از اتمام فایل صوتی کافیست مقادیر به دست آمده را پرینت کنیم :

```
>> p2_3
Detected Data:
D D G F# D
0.250 0.250 0.500 0.500 0.500
D E E D F#
0.250 0.250 0.250 0.250 0.250
D E D E F#
0.250 0.500 0.500 0.500 0.500
E D E E D
0.500 0.250 0.250 0.250 0.250
F# D E D E
0.250 0.250 0.500 0.500 0.250
D F# E D E
0.250 0.500 0.500 0.500 0.250
D F# E D D
0.250 0.500 0.500 0.250 0.250
E F# E F# F#
0.500 0.250 0.250 0.500 0.250
E F# F# D
0.250 0.500 0.500 0.500
>>
```

همانطور که مشاهده می شود دیتای استخراج شده از فایل صوتی با دیتای اصلی تطابق دارد که نشان می دهد می توان از این روش برای استخراج نت ها استفاده کرد.