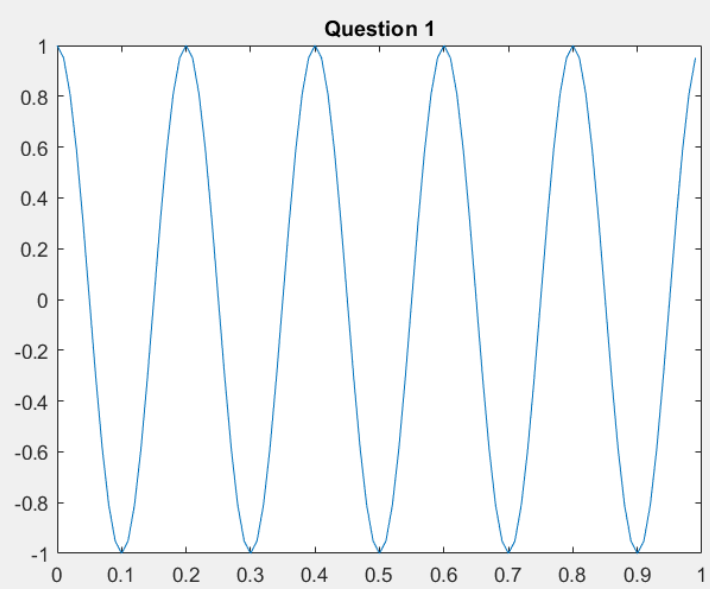
سیگنال و سیستم ها

سهیل حاجیان منش پروژه 6

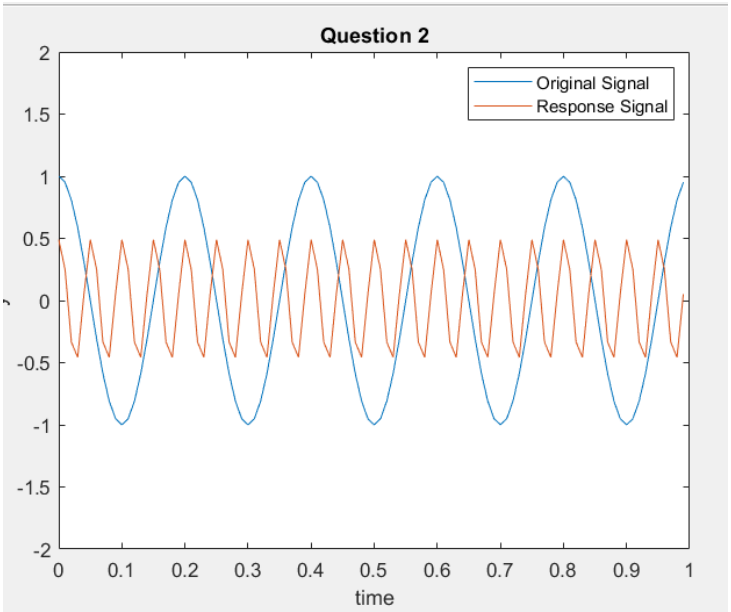
بخش اول) تمرین 1–1:

```
fc= 5;
tStart=0;
tEnd=1;
fs=100;
ts=1/fs;
t=tStart:ts:tEnd-ts;
y=cos(2*pi*fc*t);
plot(t,y)
xlabel="time";
ylabel= "y";
title("Question 1");
```



تمرین 2–1:

```
V=50;
R=250000;
beta=0.3;
alpha=0.5;
fd=beta*V;
C=3e8;
td=2*R/C;

y=cos(2*pi*fc*t);
responseSignal=alpha*cos(2*pi*(fc+fd)*(t-td));
plot(t, y);
ylim([-2 2]);
hold on    ;
plot(t, responseSignal);
hold off;
xlabel('time');
ylabel('y');
title('Question 2');
legend('Original Signal', 'Response Signal');
```



تمرین 3–1:

```
function [R,V]=calculateR_V(signal,fc)
    fs=100;
    C=3e8;
    beta=0.3;
    alpha=0.5;
    Y =fft(signal);
    N = length(Y);
    P2 = abs(Y/N);
    P1 = P2(1:N/2+1);
    P1(2:end-1) = 2*P1(2:end-1);

    phase = angle(Y);

    f = fs*(0:(N/2))/N;

    [magnitude_peak, peak_index] = max(P1);

    main_frequency = f(peak_index);
    main_phase = phase(peak_index);

    f_new = main_frequency;
    phi_new = main_phase;

    fd_ = f_new - fc;

    td_ = -phi_new / (2 * pi * (fc + fd_));

    R=td_*C/2;
    V=fd_/beta;
end
```

```
actual R vs calculated R: 250000 vs 2.500000e+05 m
actual V vs calculated V: 50 vs 50 m/s
```

تابع `***calculateR_V***` را برای محاسبه فاصله و سرعت از سیگنال رادار داپلر که دریافت شده است نوشته ان که در زیر توضیح می دهم:

سیگنال دریافتی با استفاده از fft به حوزه فوریه برده میشود تا اجزای فرکانسی آن تعیین شوند .پس از اجرای fft ، طیف دو طرفه `P2` بدست می‌آید و سپس طیف یک طرفه `P1`با اخذ نصف اول `P2` (برای نمایش فرکانس های مثبت)و تصحیح دامنه نقاط غیر منحصر به فرد تولید می‌شود.

محاسبه فاز برای کل طیف فرکانس‌ها از روی نتایج fft

یک بردار فرکانس `f` برای طیف یک طرفه محاسبه می‌شود که با تقسیم طول نیمه‌ی طیف `N/2` به تعداد نمونه‌ها و ضرب این تعداد در فرکانس نمونه برداری `fs` محاسبه می‌شود

بزرگ‌ترین پیک در طیف دامنه (`P1`) که متناظر با جز اصلی فرکانسی سیگنال است را پیدا میکنیم.

تعیین فرکانس اصلی (`main_frequency`) و فاز مربوط به آن (`main_phase`)بر اساس شاخص پیک و در نهایت، این تابع با استفاده از فرمول های مربوط به محاسبه سرعت و فاصله مقادیر تخمینی برای فاصله و سرعت شی را برمی‌گرداند.

تمرین 4–1:

پارامتر فاصله حساسیت خیلی زیادی به نویز دارد تا جایی که با قدرت نویز 0.000002 هم تشخیص فاصله بصورت دقیق انجام نمی شود:

```
Noise Rate : 0.000002
   R vs R_: 250000 vs 2.500004e+05 m
   V vs V_: 50 vs 50 m/s
```

اما سرعت به اندازه فاصله به نویز حساسیت ندارد و تا قدرت نویز 1.1 به درستی تشخیص می دهد. در ادامه عکس تعداد از تخمین های سرعت و فاصله را بر حسب قدرت نویز های مختلف گذاشته ام:

```
Noise Rate : 0.100000
   R vs R_: 250000 vs 2.351411e+05 m
   V vs V_: 50 vs 50 m/s
```

```
Noise Rate : 0.500000
   R vs R_: 250000 vs -1.826024e+03 m
   V vs V_: 50 vs 50 m/s
```

```
Noise Rate : 0.900000
   R vs R_: 250000 vs 9.064385e+04 m
   V vs V_: 50 vs 50 m/s
```

```
Noise Rate : 1.100000
   R vs R_: 250000 vs -9.392495e+03 m
   V vs V_: 50 vs 1.433333e+02 m/s
```

تمرین 5–1:

```
R1 = 250000;
V1 = 50;
alpha1 = 0.5;

R2 = 200000;
V2 = 60;
alpha2 = 0.6;

fd1 = beta * V1;
td1 = 2 * R1 / C;
fd2 = beta * V2;
td2 = 2 * R2 / C;

y1 = alpha1 * cos(2 * pi * (fc+fd1) * (t - td1));
y2 = alpha2 * cos(2 * pi * (fc+fd2) * (t - td2));

y = y1 + y2;

figure;
plot(t, y);
xlabel('Time (seconds)');
ylabel('Amplitude');
title('Received Signal from Two Objects');
grid on;
```

همانطور که در کد مشاهده میکنید هر سیگنال را جداگانه با فرمول

$$y(t) = \alpha \cos(2\pi (fc + fd)(t - td))$$

تشکیل میدهیم و سپس از جمع دو سیگنال، سیگنال برگشتی را محاسبه میکنیم و با دستور plot آن را رسم می کنیم.

تمرین 6–1:

```
function [R2, V2, R1, V1] = calculateR_V(signal, fc, fs, beta)
    C = 3e8;
    Y = fft(signal);
    N = length(Y);
    P2 = abs(Y/N);
    P1 = P2(1:N/2+1);
    P1(2:end-1) = 2*P1(2:end-1);
    f = fs*(0:(N/2))/N;
    phase = angle(Y(1:N/2+1));

    [mag_peaks, peak_indices] = maxk(P1, 2);

    f_peak1 = f(peak_indices(1));
    f_peak2 = f(peak_indices(2));

    phase_peak1 = phase(peak_indices(1));
    phase_peak2 = phase(peak_indices(2));

    fd1 = f_peak1 - fc;
    fd2 = f_peak2 - fc;
    td1 = -phase_peak1 / (2 * pi * (fc + fd1));
    td2 = -phase_peak2 / (2 * pi * (fc + fd2));
    if td1 < 0
        td1 = td1 + fs;
    end
    if td2 < 0
        td2 = td2 + fs;
    end
    V2 = fd1 / beta;
    V1 = fd2 / beta;
    R2 = td1 * C / 2;
    R1 = td2 * C / 2;
end
```

```
Estimated Values:
R1: 250000.00 m
V1: 50.00 m/s
R2: 200000.00 m
V2: 60.00 m/s
```

تابع calculateR_V2 برای محاسبه سرعت و فاصله رادار از سیگنال دریافتی از دوجسم نوشته ام که در زیر توضیح می دهم:

سیگنال دریافتی با استفاده از fft به حوزه فوریه برده میشود تا اجزای فرکانسی آن تعیین شوند .پس از اجرای fft ، طیف دو طرفه `P2` بدست می‌آید و سپس طیف یک طرفه `P1`با اخذ نصف اول `P2` و تصحیح دامنه نقاط غیر منحصر به فرد تولید می‌شود.

یک بردار فرکانس `f` برای طیف یک طرفه محاسبه می‌شود که با تقسیم طول نیمه‌ی طیف `N/2` به تعداد نمونه‌ها و ضرب این تعداد در فرکانس نمونه برداری `fs` محاسبه می‌شود .دو پیک بزرگ در طیف مغناطیسی با استفاده از دستور `maxk` یافت می‌شوند که این پیک‌ها بیانگر بردار فرکانس‌های مهم در سیگنال هستند.

فرکانس‌ها و فازهای مربوط به این دو پیک اصلاح شده و محاسبه می‌شوند.

شیفت داپلر (`fd1`, `fd2`) و تاخیر زمانی (`td1`, `td2`) بر اساس فرکانس‌های پیک و فازهای مرتبط با آن‌ها محاسبه می‌شوند.

اگر تاخیرهای زمانی منفی باشند، آنها با اضافه کردن فرکانس نمونه‌برداری اصلاح می‌شوند، زیرا تاخیر زمانی منفی از نظر فیزیکی معنی ندارد.

در نهایت، این تابع با استفاده از فرمول های مربوط به محاسبه سرعت و فاصله مقادیر تخمینی برای فاصله و سرعت دو شی را برمی‌گرداند. این الگوریتم مبنای کار سیستم‌های رادار داپلر است که از اختلاف فرکانس بین سیگنال‌های فرستاده شده و دریافتی برای تعیین سرعت و فاصله‌ی اشیاء استفاده می‌کنند.

همانطور که در عکس دوم مشخص است تابع برای ورودی سیگنال داده شده مقادیر سرعت و فاصله مربوط به دوجسم را درست تشخیص داده است.

تمرین 7–1:

در صورتی که دو جسم با سرعت معادلی حرکت کنند، امکان تفکیک آن‌ها در داده‌های دریافتی به دلیل همپوشانی فرکانس‌ها در تبدیل فوریه (FFT)وجود نخواهد داشت. در پردازش داده‌های راداری، توانایی در شناسایی و جداسازی اهداف مرتبط مستقیماً به رزولوشن فرکانسی برمی‌گردد. زمانی که سرعت اهداف مشابه است، فرکانس‌ها در نتایج FFTقابل تمیز دادن نیستند و برای برآورده کردن یک تخمین دقیق، نیاز به یک اختلاف حداقلی در سرعت هستیم که این با حد دقت تجزیه و تحلیل فرکانسی ما مطابقت داشته باشد

تمرین 8–1:

می‌توانیم تفاوت‌های میان دو شئ که از نظر فاصله (R) یکسان هستند را تشخیص دهیم. کد مربوطه از برجستگی‌ها در تبدیل فوریه سریع (FFT) برای گرفتن داده‌های مربوط به تغییرات فرکانسی (fd) و تأخیر زمانی (td) بهره می‌گیرد. این مشخصات عمدتاً توسط فرکانس‌ها مشخص می‌شوند، لذا تغییرات در فاصله‌ها (R) تأثیر چندانی بر توانایی ما در شناخت فاصله و سرعت اشیاء ندارند.

تمرین 9–1:

میتوانیم از استراتژی قسمت 5 استفاده کنیم و محدوده ای برای تعداد اجسام قایل نشویم.

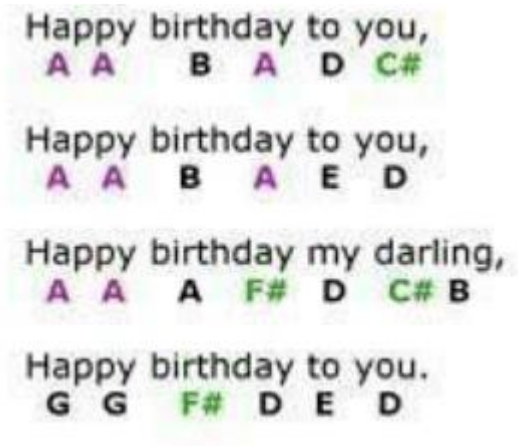
بخش دوم) تمرین 1–2:

```
function song = createSong(keys, pushTimes)
    noteMap = {'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, 830.61, 880, 932.33, 987.77];
    netMap = [noteMap; num2cell(freqs)];

    song = [];
    fs = 8000;
    T=0.5;
    tEnd = T;
    ts = 1/fs;
    tha = 0.025;

    for i = 1:length(keys)
        noteIndex = find(strcmp(netMap(1, :), keys(i)));
        if isempty(noteIndex)
            error('Invalid key: %s', keys{i});
        end
        fc = freqs(noteIndex);

        if pushTimes(i) == "T"
            t = 0:ts:tEnd-ts;
        else   pushTimes(i) == "T/2"
            t = 0:ts:(tEnd/2)-ts;
        end

        signal = sin(2*pi*fc*t);

        song = [song, signal, zeros(1, tha*fs)];
    end
end
```

با توجه به قطعه کد بالا در تابع ابتدا لیستی از کلیدهای فشرده شده به همراه لیستی از مدت زمان فشرده شدن هرکلید را بترتیب فشرده شدن دریافت می کنیم و سپس بترتیب به ازای هر کلید سیگنال مربوطه را به کمک جدول فرکانس ها میسازیم و به همراه وقفه مربوط به بعد از آن کلید در سیگنال آهنگ در حال ساخت قرار می دهیم.

تمرین 2–2:

در این بخش از این قطعه از این موسیقی نسبتا معروف استفاده کرده ام و از همان کد قبلی برای پخش موسیقی استفاده کرده ام.



تمرین 3–2:

```
audioSignal = createSong(melodyNotes, durationSymbols);
audiowrite("final_song.wav", audioSignal, sampleRate);

actualDurations = zeros(size(durationSymbols));
for idx = 1:length(durationSymbols)
    if strcmp(durationSymbols{idx}, 'T')
        actualDurations(idx) = noteDurationTime;
    elseif strcmp(durationSymbols{idx}, 'T/2')
        actualDurations(idx) = noteDurationTime/2;
    end
end

for noteIdx = 1:length(melodyNotes)
    startSample = round((sum(actualDurations(1:noteIdx-1)) + (noteIdx-1) * pauseDuration) * sampleRate) + 1;
    endSample = round((sum(actualDurations(1:noteIdx-1)) + actualDurations(noteIdx) + (noteIdx-1) * pauseDuration) * sampleRate);
    noteSegment = audioSignal(startSample:endSample);
    noteFourier = fftshift(fft(noteSegment));
    noteMagnitude = abs(noteFourier);
    freqVector = (-sampleRate/2):(sampleRate/length(noteSegment)):(sampleRate/2)-(sampleRate/length(noteSegment));
    [maxValue, peakIndex] = max(noteMagnitude);
    dominantFrequency = abs(freqVector(peakIndex));
    freqDiff = abs(pitchFrequencies - dominantFrequency);
    [minDifference, closestPitchIdx] = min(freqDiff);
    detectedNote = pitchNames{closestPitchIdx};

    fprintf("Detect Note: %s ,", detectedNote);
end
```

بخشی از خروجی:

```
Detect Note: D ,Detect Note: D ,Detect Note: G ,Detect Note: F# ,Detect Note: D ,Detect Note: D ,Detect Note: E
```

ایده ی پیاده سازی این است که صدا را با تقسیم کردن آن به قطعات بر اساس زمان هایی که مقدار 0 است پردازش می‌کنیم. سپس هر قطعه را به حوزه فوریه برده و فرکانس پیک را شناسایی می کنیم. حال این فرکانس را به نزدیک فرکانس در فرکانس های کلید ها نگاشت می کنیم . کد را برای هر تکه تکرار می کنیم که در نتیجه آن کلید موسیقی شناسایی‌شده را برای هر کدام نمایش می‌دهد.

نتیجه بدست آمده دقیقا برابر کلید های فشرده شده در بخش اول است.