به نام خدا

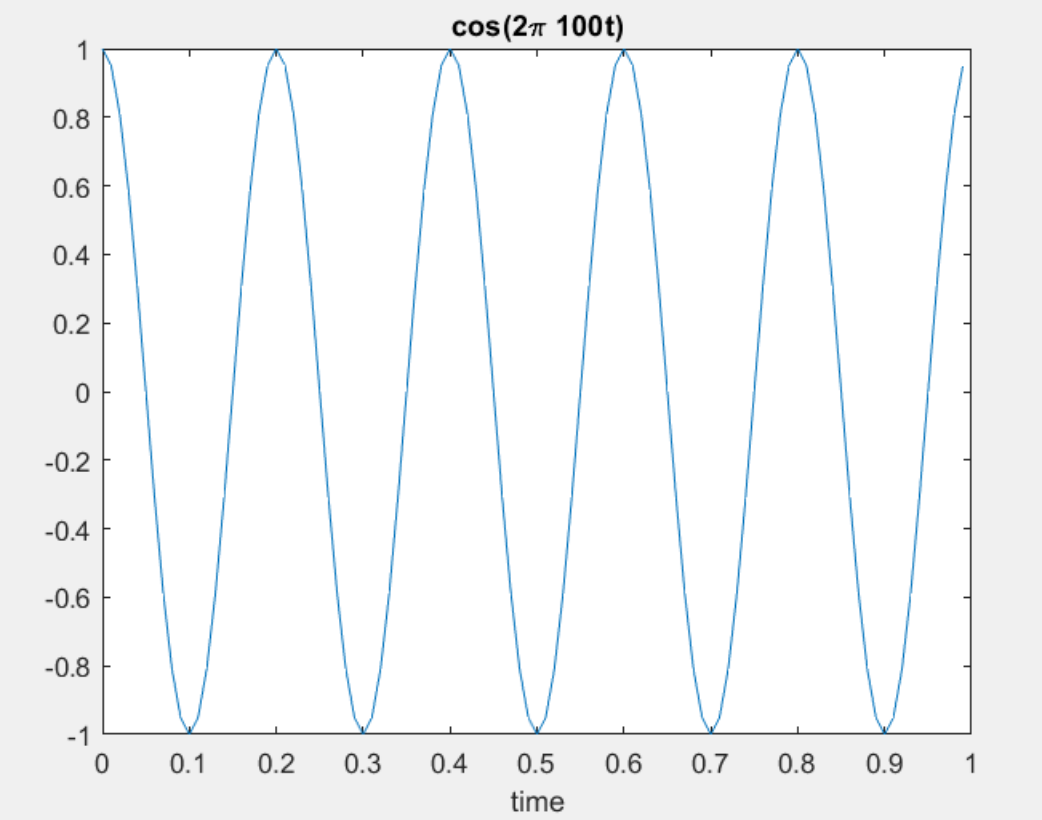
پروژه ششم درس سیگنال‌ها و سیستم‌ها

فاطمه‌زهرا برومندنیا-۸۱۰۱۰۰۰۹۴

بخش اول

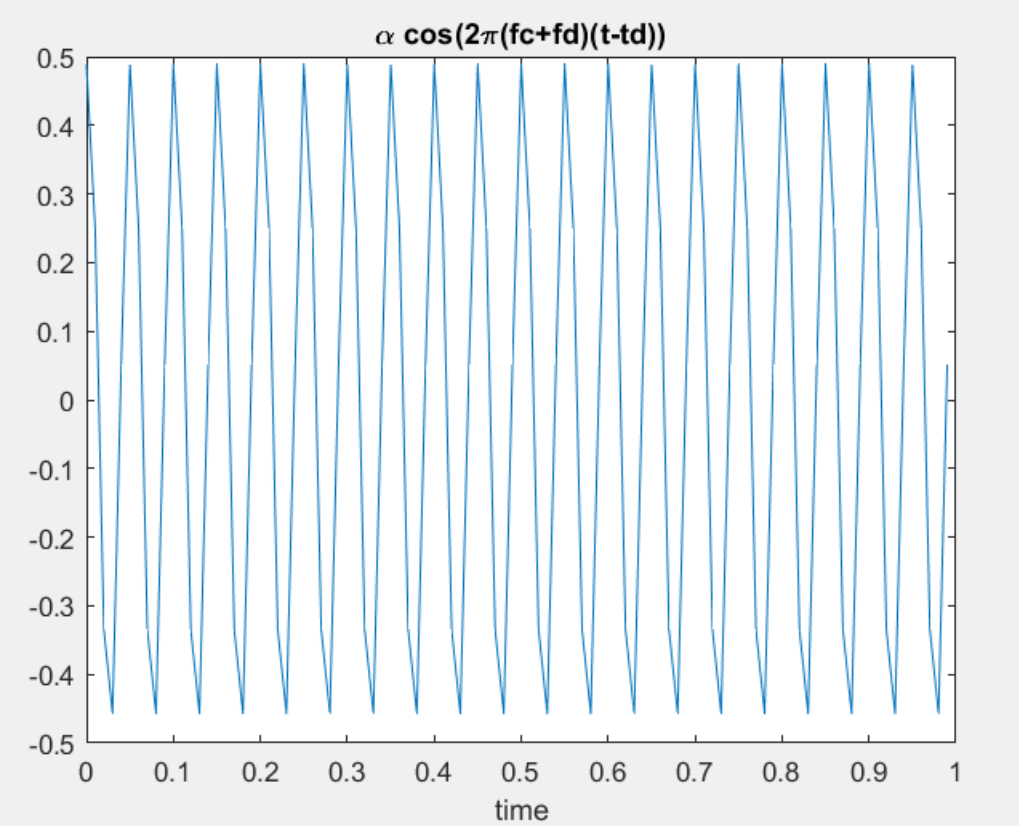
۱-۱)

```
clc, clear, close all
fs = 100;
Ts = 1/fs;
ts = 1 / fs;
T = 1;

fc=5;
t_start = 0;
t_end = T;
t = t_start:Ts:t_end-Ts;
N = length(t);
f = (-fs / 2):(fs / N):(fs / 2 - fs / N);
beta = 0.3;

%%  p1.1
sentSignal=cos(2*pi*fc*t);
plot(t, sentSignal);

```



(۲-۱

```
%% p1.2
V=50;
R=250000;
B=0.3;
A=0.5;
C=3e8 ;
p=2/C;
fd=B*V;
td=R*p;
recvSignal=A*cos(2*pi*(fc+fd)*(t-td));
plot(t,recvSignal);
title('\alpha cos(2\pi(fc+fd)(t-td))')
xlabel('time')
```



(۳-۱)

```
35    %% p1.3
36    N = length(t);
37 -  fm=100;
38 -  f = (-fm / 2):(fm / N):(fs / 2 - fm / N);
39 -  y=fftshift(fft(recvSignal));
40 -  F=y/max(abs(y));
41
42 -  subplot(1,2,1);
43 -  plot(f,abs(F));
44
45 -  tol=1e-6;
46 -  F(abs(F)<tol)=0;
47 -  theta = angle(F);
48 -  subplot(1,2,2);
49 -  plot(f,theta/pi);
50
51 -  maxVal = max(abs(F));
52 -  maxPos = find(abs(F) == maxVal, 1, 'last') - 51;
53 -  found_fd= maxPos - fc
54 -  [~, maxIndex] = max(F);
55 -  found_td = theta(maxIndex)/(2*pi*(found_fd+fc));
56 -  found_V = found_fd/B;
57 -  found_R = (found_td*C)/2;
58
59 -   disp(['P1.3: Estimated V is:',num2str(found_V*3.6),' and estimated R is:',num2str(found_R/1000)]);
60
```

ایده اصلی استفاده از تبدیل فوریه است. مشابه تمرین های قبل، کافی است با استفاده از تبدیل فوریه، چون سیگنال سینوسی داریم و می توان فرکانس را در نقاطی که پیک می‌زند (با کمک تابع find که ایندکسی که max در آن رخ داده را بیابیم) پیدا کنیم و همچنین فاز را، به فاز و فرکانس سیگنال دریافتی دست پیدا کنیم. پس از این باید از راهنمایی داده شده بهره جست؛ فرکانس به دست آمده را منهای fc می کنیم تا fd به دست بیاید، به همین ترتیب از روابط td را به دست می آوریم و سپس با کمک نسبت‌ها v و r را پیدا می‌کنیم.

(۴-۱)

```
61  %% p1.4
62  threshold = 6;
63  for noise_level = 0:0.5:threshold
64      x_noisy = recvSignal + noise_level * randn(size(recvSignal));
65      X_noisy = fftshift(fft(x_noisy));
66      X_noisy = X_noisy/max(abs(X_noisy));
67      figure;
68      subplot(2,1,1);
69      plot(t, x_noisy);
70      title(['Noisy Signal (Noise Level = ' num2str(noise_level) ')']);
71
72      subplot(2,1,2);
73      plot(f, abs(X_noisy));
74      title(['Magnitude Spectrum (Noise Level = ' num2str(noise_level) ')']);
75
76      maxVal = max(abs(X_noisy));
77      maxPos = find(abs(X_noisy) == maxVal, 1, 'last') - 51;
78      found_fd1= maxPos - fc;
79
80      [~, maxIndex] = max(X_noisy);
81      found_td1 = theta(maxIndex)/(2*pi*(found_fd1+fc));
82      found_V1 = found_fd1/B;
83      found_R1 = (found_td1*C)/2;
84      if found_V1 ~= found_V
85              disp(['change at V. Noise level at: ' num2str(noise_level)]);
86      end
87      if found_R1 ~= found_R
88          disp(['change at R. Noise level at: ' num2str(noise_level)]);
89      end
90      pause(1);
```

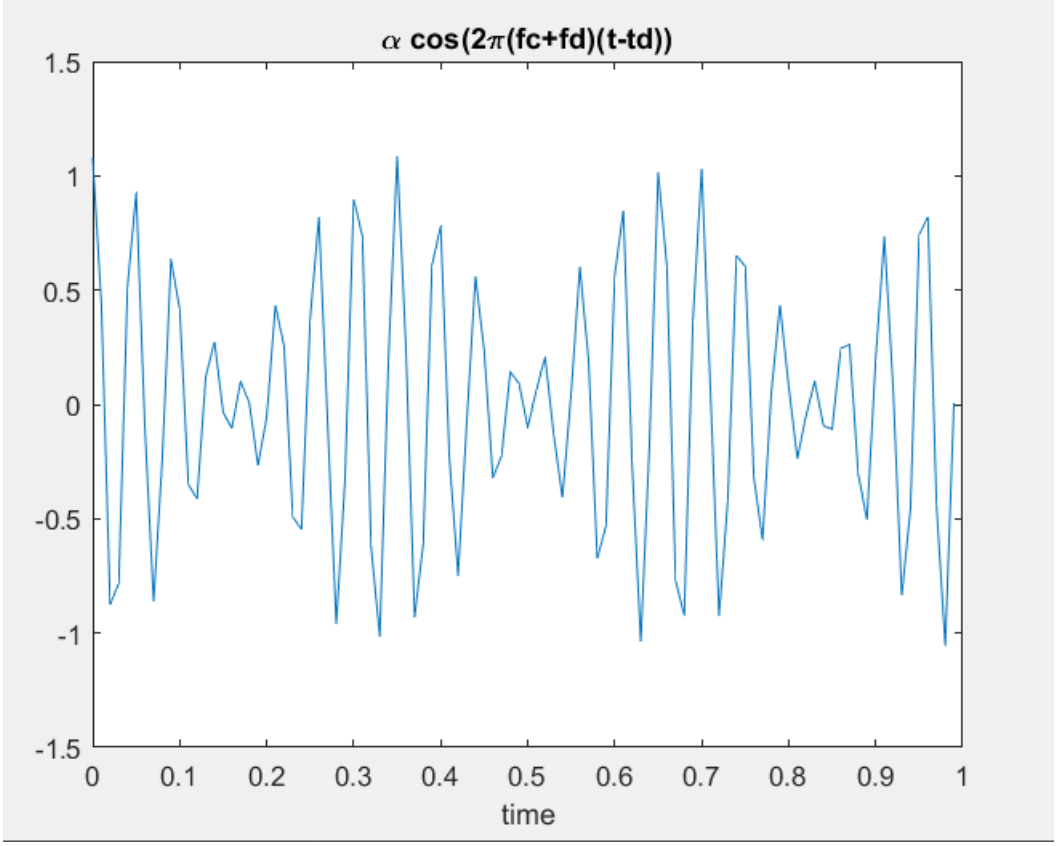
مشابه تمرین های قبل با کمک تابع randn و ضریب moise_level که حد آن را تعیین می‌کند، در این حلقه آرام آرام نویز را با step به اندازه ۰.۵ از ۰ تا ۶ می‌افزاییم. پیداکردن وی و آر مشابه بخش های قبل است، دو شرط آخر حلقه، چک می‌کنند که آیا مقدار درست تشخیص داده شده یا نه، و اگر اشتباه بود noise_levelی که به خطا خورده را ریپورت می‌کنند.

به نظر می رسد فاصله حساسیت بیشتری داشته باشد. برای چک کردن این مسئله، stepها را ریزتر کردم تا به پاسخ دقیق‌تری برسم (۰.۰۱) و پس از چندبار خروجی گرفتن، دیده شد که ابتدا R در میانگین ضریب ۰.۵۵ به خطا میرود.

```
change at R. Noise level at: 0.57
K>>
```

(۵-۱

```
%% p1.5

V1 = 50;
R1 = 250000;
alpha1 = 0.5;
td1 = (2*R1)/C;
fd1 = B*V1;
x_1 = alpha1*cos(2*pi*(fc+fd1)*(t-td1));

V2 = 60;
R2 = 200000;
alpha2 = 0.6;
td2 = (2*R2)/C;
fd2 = B*V2;
x_2 = alpha2 * cos(2 * pi * (fc+fd2) * (t-td2));

recived_sig = x_1+x_2;
figure
plot(t, recived_sig)
title('\alpha cos(2\pi(fc+fd)(t-td))')
xlabel('time')
```



مشابه بخش‌های اول، دو سیگنال را با فرکانس ها و اطلاعات داده شده تولید می‌کنیم، سپس جمع این ها را به عنوان سیگنال دریافتی اعلام می‌کنیم.

۱-۶)

```
%% p1.6

y6 = fftshift(fft(recived_sig));
sig_fft2 = abs(y6) / max(abs(y6));
[peak,x_peaks]= findpeaks(abs(sig_fft2));
[peak,index]=sort(peak,'descend');
x_peaks = x_peaks(index);
sig_phase = angle(y6);

found_fd_1=abs(x_peaks(2*1)-fs/2-1)-fc;
found_td_1 = abs((sig_phase(x_peaks(2*1)))/(2*pi*(found_fd_1+fc)));
Vrecv1 = found_fd_1/B;
Rrecv1 = (found_td_1/(2/C));
 disp(['P1.6: Estimated V1 is:',num2str(Vrecv1*3.6),' and estimated R1 is:',num2str(Rrecv1/1000)]);

found_fd_2=abs(x_peaks(2*2)-fs/2-1)-fc;
found_td_2 = abs((sig_phase(x_peaks(2*2)))/(2*pi*(found_fd_2+fc)));
Vrecv2 = found_fd_2/B;
Rrecv2 = (found_td_2/(2/C));
 disp(['P1.6: Estimated V2 is:',num2str(Vrecv2*3.6),' and estimated R2 is:',num2str(Rrecv2/1000)]);
```

برای این بخش، می دانیم چون جمع خطی دو سیگنال کسینوسی را داریم، در ۴ نقطه پیک خواهیم داشت. با استفاده از تابع findpeaks و سپس مرتب کردن آن ها، و محاسبه نسبت های متناظر مشابه بخش ۳، میتوانیم سرعت ها را به دست بیاوریم. برای به دست آوردن فاصله ها نیز به فاز هرکدام از پیک‌ها نیاز داریم که کافی است فاز هرکدام از نقاط پیک را در ماتریسی که از پاسخ فاز تبدیل فوریه به دست آمده بگذاریم و محاسبه کنیم.

در نهایت برای نمایش، چون هنگام محاسبه به متر بر ثانیه تبدیل کرده بودیم، به کیلومتر بر ساعت برمیگردانیم.

۱-۷ و ۱-۸ و ۱-۹)

اگر سرعت ها یکسان باشد، نمیتوانیم پارامترها را استخراج کنیم چون فرکانس هایی که در آن ها پیک رخ میدهد روی هم می افتند و قابل تمییز برای به دست آوردن پارامترها و استخراج اطلاعات نیستند. اما اگر فاصله ها یکسان باشند و سرعت ها متفاوت، چون خیالمان راحت است فرکانسی که در آن پیک می زند متفاوت است، برای به دست آوردن فاز متناظر نیز چون برای پیک متفاوتی است به مشکل نمی خوریم و

این مسئله را با امتحان کردن در کدهای این بخش نیز میتوان دید.

برای پاسخ به سوال بخش ۹، تفاوتی نمیکند، روشی که در بخش ۶ استفاده شده را میتوان به صورت تعمیم یافته برای چند جسم نیز استفاده کرد؛ پیداکردن پیک ها، مرتب کردن نزولی آن ها، برای هرکدام فاز متناظر پیدا کرد و پارامترها را به دست آورد.

بخش دوم

۲-۱)

```
fs = 8000;
Ts = 1/fs;
T = 0.50;
tau = 0.025;

keyFrequencies = [523.25, 554.37, 587.33, 622.25, 659.25, 698.46, 739.99, 783.99, 830.61, 880, 932.33, 987.77];

keyNames = {'C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B'};

userKeys = {'D' 'D' 'G' 'F#' 'D' 'D' 'E' 'E' 'D' 'F#' 'D' 'E' 'D' 'E' 'F#' 'E' 'D' 'E' 'E' 'D' 'F#' 'D' 'E' 'D' 'E' 'D' 'F#' 'E
noteDurations = [1 1 2 2 1 1 1 1 1 1 1 2 2 2 2 2 1 1 1 1 1 1 2 2 1 1 2 2 2 1 1 2 2 1 1 2 1 1 2 1 1 2 2 2 1 1 2 2 2];

t_start = 0;
t_end1 = T;
t_end2 = T/2;
t1 = t_start:Ts:t_end1-Ts;
t2 = t_start:Ts:t_end2-Ts;
pauseTime=t_start:Ts:tau-Ts;
pauseTime=zeros(1,length(pauseTime));

music = [];
for i = 1:length(userKeys)
    keyName = userKeys{i};

    keyIndex = find(strcmp(keyNames, keyName));
    if ~isempty(keyIndex)
        frequency = keyFrequencies(keyIndex);
        duration = noteDurations(i);
```

```
        duration = noteDurations(i);

        if duration == 1
            signal = sin(2*pi*frequency*t2);
        else
            signal = sin(2*pi*frequency*t1);
        end

        music = [music signal pauseTime];
    else
        disp(['Invalid key name: ' keyName]);
    end
end
sound(music, fs);
audiowrite('loveStory.wav',music,fs);
```

سیگنال متناظر با هر نت ایجاد می‌شود. به این صورت که در هر مرحله از حلقه `for`، نام کلید از `userKeys` استخراج شده و سپس فرکانس متناظر با آن کلید از `keyFrequencies` گرفته می‌شود. مدت زمان هر نت از `noteDurations` خوانده می‌شود.

سپس، بسته به مدت زمان هر نت، سیگنال با استفاده از توابع `sin` با فرکانس و دامنه متناسب ایجاد می‌شود. اگر مدت زمان یک نت برابر با T باشد، از دامنه `t2` استفاده می‌شود؛ در غیر این صورت از دامنه `t1` استفاده می‌شود.

سپس سیگنال هر نت به آخرین موسیقی افزوده شده و pauseها (سکوت) به صورت دستی به آن کانکت می‌شود (در اینجا با صفرها که نمایانگر سکوت هستند).

در نهایت، موسیقی با استفاده از تابع `sound` پخش می شود.

۲-۲)

```
clc, clear, close all
fs = 8000;
Ts = 1/fs;
T = 0.50;
tau = 0.025;

keyFrequencies = [523.25, 554.37, 587.33, 622.25, 659.25, 698.46, 739.99, 783.99, 830.61, 880, 932.33, 987.77];

keyNames = {'C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B'};

userKeys = {'E', 'E', 'E', 'E', 'E', 'E', 'E', 'G', 'C', 'D', 'E', 'F', 'F', 'F', 'F', 'F', 'E', 'E', 'E', 'E', 'D', 'D', 'E', '

noteDurations = ones(1,length(userKeys));
noteDurations(2)=2;

t_start = 0;
t_end1 = T;
t_end2 = T/2;
t1 = t_start:Ts:t_end1-Ts;
t2 = t_start:Ts:t_end2-Ts;
pauseTime=t_start:Ts:tau-Ts;
pauseTime=zeros(1,length(pauseTime));

music = [];
for i = 1:length(userKeys)
    keyName = userKeys{i};

    keyIndex = find(strcmp(keyNames, keyName));
    if ~isempty(keyIndex)
```

```
            frequency = keyFrequencies(keyIndex);
            duration = noteDurations(i);

            if duration == 1
                signal = sin(2*pi*frequency*t1);
            else
                signal = sin(2*pi*frequency*t2);
            end

            music = [music signal pauseTime];
        else
            disp(['Invalid key name: ' keyName]);
        end
    end
    plot(length(music),music);
    sound(music, fs);
    audiowrite('mysong.wav',music,fs);

    t = (0:length(signal)-1) * Ts;
    figure;
    plot(t(1:2000), signal(1:2000), 'b');
    info = audioinfo('mysong.wav');
    bitDepth = info.BitsPerSample;
    disp(['my song uses ' num2str(bitDepth) ' bits per sample.']);

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

        y=fftshift(fft(music));
```

در این قسمت هم مشابه قبل، با عوض کردن نوت ها آهنگ موردنظر تولید شده. برای اینکه بفهمیم هر سمپل از داده ای که ساخته شده با چند بیت ذخیره شده، از دستور audioinfo و اتریبیوت BitsPerSample استفاده شده.

۲-۳)

```
clc, clear, close all
[music, fs] = audioread('loveStory.wav');

keyFrequencies = [523.25, 554.37, 587.33, 622.25, 659.25, 698.46, 739.99, 783.99, 830.61, 880, 932.33, 987.77];
keyNames = {'C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B'};

fs = 8000;
Ts = 1/fs;
T = 0.50;
tau = 0.025;
t_start = 0;
t_end1 = T;
t_end2 = T/2;
t1 = t_start:Ts:t_end1-Ts;
t2 = t_start:Ts:t_end2-Ts;
pauseTime = t_start:Ts:tau-Ts;
pauseTime=zeros(1,length(pauseTime)+1);

index = strfind(music', pauseTime);

threshold = 0.02;

detectedNotes = [];
detectedDurations = [];

stepSize1 = length(t1) + length(pauseTime);
stepSize2 = length(t1) + length(pauseTime);

chunkStart = 1;
m=1;
```

```
  m=1;
while chunkStart <= length(music)
    if(m>length(index))
        break;
    elseif(m==length(index))
        chunkEnd = length(music);
    else
        chunkEnd = index(m);
    end
    chunk = music(chunkStart:chunkEnd, 1);

    N = length(chunk);
    frequencies = (0:N-1) * fs / N;
    fft_result = fft(chunk, N);
    magnitude_spectrum = abs(fft_result);

    [~, idx] = max(magnitude_spectrum);
    dominant_frequency = frequencies(idx);

    [~, keyIndex] = min(abs(keyFrequencies - dominant_frequency));
    detectedNote = keyNames{keyIndex};
    difference_t1 = abs(chunkEnd - chunkStart - length(t1));
    difference_t2 = abs(chunkEnd - chunkStart - length(t2));

    if difference_t1 <= difference_t2
        detectedDuration = 1;
    else
        detectedDuration = 2;
    end

```

```
58 -        end
59
60 -        detectedNotes = [detectedNotes, detectedNote];
61 -        detectedDurations = [detectedDurations,detectedDuration];
62
63
64 -        chunkStart = index(m) +length(pauseTime)+1;
65 -        m=m+1;
66 -    end
67
68
69 -    disp('Detected Notes:');
70 -    disp(detectedNotes);
71
72 -    disp('Detected Durations:');
73 -    disp(detectedDurations);
74
```

در این کد برای نت‌ها و مدت‌زمان هر یک:

۱. **خواندن فایل صوتی:**

– با استفاده از `audioread`، فایل صوتی با نام "loveStory.wav" خوانده می‌شود و اطلاعات اصلی شامل `music` (سیگنال صوتی) و `fs` (فرکانس نمونه‌برداری) استخراج می‌شود.

– فرکانس‌ها و نام‌های متناظر با کلیدهای پیانو (keyFrequencies و keyNames) تعیین شده‌اند.

– از تابع `strfind` برای یافتن بازه های سکوت در موسیقی استفاده شده است. این تابع ایندکس های قسمت هایی که بازه سکوت شروع می شود را در index می ریزد. بعد در حلقه while، با استفاده از همین نقاط مشخص chunkها برای جداشدن مشخص می شوند، شروع هرchunk از محل شروع هر pause به اضافه طول آن و پایان هر chunk در نقطه شروع pause بعدی است.

– یک حلقه `while` برای پردازش و جداکردن chunkهای مختلف از موسیقی و تشخیص نوت اجرا می‌شود. برای هر chunk با استفاده از تبدیل فوریه، فرکانس peak و در نتیجه نت متناظر آن قطعه تعیین می‌شود.

– با مقایسه فاصله میان طول‌های `t1` و `t2` با طول قطعه، مدت‌زمان هر نت که T1 است یا T2تشخیص داده می‌شود.

– نت و مدت‌زمان هر نت در آرایه‌های `detectedNotes` و `detectedDurations` ذخیره می‌شوند.