به نام آنکه تن را نور جان داد

خرد را سوی دانایی عنان داد

دانشکده مهندسی برق

درس سیگنال و سیستم

گزارش پروژه نهایی

(تشخیص آهنگ با استفاده از اثر انگشت صوتی )

رادین خیام ( ۹۹۱۰۱۵۷۹ )

## خواسته های پروژه :

۱- تابع audio_import را کامل کنید .

```
function [downsampled_Fs, resampled_audio] = import_audio(path, song_num, format)
    % import the audio
    [audio, Fs] = audioread([path, 'music', num2str(song_num), format]);
    % getting mean over right and left channels
    %%% audioMono
    audio_avr=mean(audio,2);
    % downsample the audio to 8 KHz
    downsampled_Fs = 8000;
    resampled_audio = resample(audio_avr, downsampled_Fs, Fs);
    %%% resampled_audio
end
```

۲- تابع FFT را کامل کنید .

```
function single_sided_power_spectrum = FFT(X)
    %%% single_sided_power_spectrum
    L=length(X);
    Y=fft(X);
    p2=(abs(Y)/L).^2;
    p1= p2(1:floor(L/2)+1);
    p1(2:end-1) = 2*p1(2:end-1);
    single_sided_power_spectrum=p1;

end
```

۳- تابع STFT را کامل کنید .

```
function [time, freq, time_freq_mat] = STFT(audio, Fs, window_time)
    window_length = Fs*window_time;
    window_num = floor(length(audio)/(window_length/2));
    time_freq_mat = zeros(1+floor(window_length/2),window_num-1);
    % calculating fft using an overlapping sliding window
    for i = 1:window_num-1
        %%% time_freq_mat
        window_start=((floor((window_length/2))*(i-1))+1);
        window_end=floor(window_length*i/2+window_length/2);
        signal_part=audio(window_start:window_end);
        power_spectrum=FFT(signal_part);
        noise=0.001*rand(length(power_spectrum),1);
        time_freq_mat(:,i)=power_spectrum+noise;
    end
    % time and freq vectors
    time = (window_time/2)*(1:(window_num-1));
    freq = Fs*(0:floor(window_length/2))/window_length;
end
```

**۴ - فایل create_database.m را مطالعه کنید و نحوه عملکرد آن را به اختصار توضیح دهید.**

در این فایل میبینیم که در ابتدا، فانکشن های دیگر که از قبل نوشته شده است را به مسیر برنامه اضافه میکند .

در مرحله بعد با استفاده از دستور containers.map یک هش مپ خالی درست میکنیم تا اطلاعات دیتابیس که شامل hash-key ها و hash-value ها میشود را در آن قرار دهیم.

سپس در ادامه با استفاده از یک حلقه for و توابعی مانند fullfile و dir و deal و همچنین تابع import_audio که از قبل نوشته بودیم، تک تک موزیک های داخل پوشه musics را لود میکنیم تا بتوانیم اطلاعات مورد نظر را از آن ها استخراج کنیم .

در همین حلقه for سپس با استفاده از تابع STFT که از قبل کامل شده است و در آن از تابع FFT استفاده کردیم، ماتریس مورد نظر را بدست میاوریم و سپس spectrogram را بر اساس همین ماتریس رسم میکنیم .

در مرحله بعد با انخاب dt و df مناسب و تابع find_anchor_points که از قبل نوشته شده است، anchor point ها را پیدا میکنیم و با استفاده از تابع scatter نمودار آن ها را رسم میکنیم .

در ادامه برای اینکه hash_tag ها را بسازیم در ابتدا برای df_hash و dt_hash مقادیر مناسب را اختیار میکنیم و سپس با استفاده از تابع creat_hash_tags ، hash_key و hash_value ها را بدست میاوریم .

مرحله نهایی در این فایل این است که این hash_key ها و hash_value ها را به دیتابیسمون اضافه کنیم، بدین منظور یک حلقه دیگر for داخل حلقه اصلی ایجاد میکنیم که در هر مرحله برای هر آهنگ به ازای هر hash_key ، یک key_tag تعریف میشود که یک استرینگ است به فرمت f1*f2*t2-t1 و اطلاعات hash_key داخل آن وجود دارد، سپس چک میشود که این key_tag داخل دیتابیس وجود دارد یا خیر اگر وجود نداشت که مقادیر hash_value با همان فرمتی که با ستاره جدا میشدند محتوا داخل خانه متناظر با آن key_tag قرار میگیرد، اما در صورتی که این key_tag از قبل در دیتابیس موجود باشد، hash_value های جدید بعد از استفاده از یک کاراکتر + داخل خانه متناظر با آن key_tag دیتابیس قرار میگیرند.

در انتها این مپ که دیتابیس نام دارد را داخل فولدر database ذخیره میکنیم .

همانطور که صورت سوال خواسته است، ما دیتابیس را به ازای ۵۰ آهنگی که در پوشه musics قرار داشت کامل کردیم .

دقت شود که در بعضی قسمت های این فایل تغییراتی ایجاد کردم، زیرا فولدر ها را به درستی پیدا نمیکرد متلب و مجبور شدم آدرس فولد ها را به صورت کامل بدهم در برنامه، بنابراین ممکن است برای اجرا روی دیوایس های دیگه نیاز باشد که آدرس ها را تغییر بدهید .

### ۵ - فایل search_database.m را مطالعه کنید و نحوه دقیق ساخت متغیر list را شرح دهید.

در این فایل در ابتدا با استفاده از همان روشی که در تابع create_database توضیح داده شده بود، hash_key ها و hash_tag های این موزیکی تقطیع شده که به عنوان تست داده شده است را بدست میاوریم، سپس در مرحله بعد برای بدست آوردن متغیر list بدین صورت عمل میکنیم که یک حلقه for به اندازه تعداد hash_key ها تشکیل میدهیم و سپس برای hash_key طبق همان فرمولی که در قسمت قبل توضیح داده شد، key_tag مربوط به آن را میسازیم، سپس با استفاده از تابع isKey برررسی میکنیم که آیا این key_tag در دیتابیس وجود دارد یا خیر، چنانچه وجود نداشته باشد، به سراغ hash_key های بعدی میرویم، اما چنانچه وجود داشته باشد، در ابتدا با استفاده از تابع split، value متناظر با آن key_tag را در دیتابیس بر حسب وجود کاراکتر '+' جدا میکنیم ( دقت شود که در قسمت قبل دیدیم که ممکن است به ازای key_tag های مشابه چند value داشته باشیم و ما این value ها را در دیتابیس با استفاده از کاراکتر '+' از هم جدا کردیم ) سپس یک حلقه دیگر ایجاد میکنیم، که تعداد آن به اندازه تعداد رشته های موجود در temp1 بعد از اعمال تابع split میباشد.

به عنوان مثال ممکن است به ازای key_tag ، <166*99*99> ، مقدار ما در دیتابیس بدین صورت باشد : 1965*36+1142*22 حال در این صورت در متغیر temp1 ما دو رشته مجزا تحت عنوان 1142*22 و 1965*36 خواهیم داشت و تکرار حلقه دوم هم عدد ۲ خواهد بود، حال در این حلقه جدید این بار با استفاده از split رشته ها را از نقطه ای که کاراکتر * وجود دارد جدا میکنیم، به عنوان مثال در این مثال در تکرار اول حلقه خواهیم داشت، 1142 , 22 حال در نقطه پایانی متغیر لیست بدین صورت تشکیل میشود:

List یک ماتریس شامل سه ستون است، بدین صورت که در هر سطر آن ستون اول برابر با جز اول متغیر temp2 که میشود همان شماره آهنگی که شباهت با آن تشخیص داده شده است، ستون دوم زمان متناظر با این شباهت در آهنگ کامل هست و در نهایت ستون سوم هم زمان متناظر با این شباهت در موزیک نمونه (یا همان تیکه آهنگی که ما داریم) میباشد .

در مرحله آخر با استفاده از تابع scoring میزان احتمال اینکه این که تیکه آهنگ مربوط به هر یک از آهنگ های دیتابیس که حداقل با آن یک شباهت دارد باشد محاسبه میشود و لیست سورت شده چاپ میشود .

**۶ - در مورد داده ساختار hashmap و مرتبه زماني جستجو در آن، مطالعه کنید و بر این اساس، استفاده از این نوع داده ساختار در مسئله مذکور را توجیه نمایید.**

پس از کمی جست و جو متوجه میشویم که در صورت پیاده سازی صحیح در اکثر موارد اردر زمانی اضافه کردن، حذف کردن و جستجو در ساختار hashmap به صورت O(1) میباشد، بدین معنی که زمان جستجو فارغ از میزان آهنگ های داخل دیتا بیس است و عددی ثابت است، بنابراین میتوان گفت که حتی اگر چند میلیون آهنگ هم در دیتا بیس باشند، مشکلی از این بابت بر نخواهیم خورد .

**۷ - با اســتفاده از فایل search_database.m ، مشــخص کنید هر یک از تکه آهن گهای موجود در پوشــه test_musics مربوط به کدام آهنگ است و سپس با استفاده از متغیر list ، زمان تقریبی شروع این تکه آهنگ در آهنگ مذکور را بدست آورید. با گوش کردن به این قسمت از آهنگ شناسایی شده، از صحت عملکرد الگوریتم مطمئن شوید.**

| شماره نمونه | شماره آهنگ اصلی | ثانیه تقریبی | چک کردن آهنگ | چک کردن زمان |
|---|---|---|---|---|
| ۱ | ۵ | ۸۶ | ☑ | ۸۲ |
| ۲ | ۲ | ۱۹۱ | ☑ | ۱۹۳ |
| ۳ | ۱۳ | ۱۱۱ | ☑ | ۱۱۱ |
| ۴ | ۳۶ | ۱۲۲ | ☑ | ۱۱۰ |
| ۵ | ۴۳ | ۹۳ | ☑ | ۹۴ |
| ۶ | ۴۹ | ۱۴۱ | ☑ | ۱۴۸ |
| ۷ | ۱۷ | ۷۵ | ☑ | ۳۵ |
| ۸ | ۱۱ | ۱۲۱ | ☑ | ۱۱۸ |
| ۹ | ۳۲ | ۱ | ☑ | ۱ |
| ۱۰ | ۱ | ۱۳۳ | ☑ | ۱۳۶ |

**توضیح الگوریتم استفاده شده برای بدست آوردن زمان شروع تقریبی نمونه در آهنگ اصلی:**

ما این تیکه کد را در انتهای فایل search_database اضافه کرده ایم که میتواند زمان شروع تقریبی را برگرداند .

```
%% find the corresponding start time in main song
A=list(list(:,1)==score(1,1),:); % seprate rows that their first column is 5
M_A=median(A);
start_time=(M_A(1,2)-M_A(1,3))/20;  % finding start time in seconde
disp('start time = '+string(start_time));
```

در این کد به این صورت عمل کرده ایم که پس از تشخیص آهنگی که این نمونه از آن گرفته شده است، تمام سطر هایی از متغیر list که شماره آهنگ آن ها مطابق با این آهنگ تشخیص داده شده است را جدا میکنیم و در ماتریس A میریزیم، سپس از ستون دوم و سوم این ماتریس که به ترتیب نشان دهنده زمان در آهنگ اصلی و نمونه هستند ( البته برای بدست آوردن زمان در مقیاس ثانیه لازم است که یک تقسیم بر ۲۰ انجام بدهیم، دلیل آن هم مربوط به طول پنجره هایی است که در تابع STFT استفاده کرده ایم ) میانه میگیریم، و سپس با کم کردن این دو میانه از هم میتوانیم زمان شروع تقریبی را بدست آوریم . دقت شود که رویکرد استفاده از میانه برای این انتخاب شده است که ما وقتی ماتریس A را مشاهده میکنیم، میبینیم که تعداد زیادی از داده ها از الگوی خطی پیروی میکنند اما برخی از داده ها خیلی پرت هستند، برای همین رویکرد میانگین گرفتن میتواند باعث خطا های شدیدی شود اما میانه آن داده های پرت روش اثری ندارند.

همچنین لازم است گفته شود که این داده های پرت خیلی هاشون برای این هست که بعضا پیش میاید که در یک آهنگ یک تیکه چندین بار تکرار میشود مانند کورس در آهنگ های رپ، همین باعث میشود که ما در الگوریتم اصلی اشتباها آن جاهای دیگر را تطابق بدهیم و این باعث ایجاد داده های پرت میشود .

نتیجه آزمایش های انجام شده روی این ۱۰ تیکه آهنگ بدین صورت بود که هر ۱۰ تا را درست تشخیص داد، اما در بحث زمان تقریبی برای نمونه ۷ و نمونه ۴ بیش از ۱۰ ثانیه خطا داشتیم که نسبتا غیر قابل قبول میباشد، دلیل این هم احتمالا برای این هست که این آهنگ ها الگوی تکراری ای در طول آهنگ دارند.

به عنوان مثال در نمونه شماره ۷ که آهنگ امینم هست، میبینم که بخش های مختلف آهنگ شباهت نسبتا زیادی بهم دارند و همین باعث ایجاد این خطا شده است. احتمالا الگوریتم های دقیق تری هم برای بدست آوردن این زمان تقریبی وجود دارد، یکی از ایده هایی که به ذهن من رسید استفاده از رگرسیون خطی بود، بدین صورت که در ابتدا نموداری رسم کنیم از ماتریس A که در آن ستون دوم برحسب ستون سوم رسم شده است، یعنی زمان آهنگ اصلی بر حسب زمان نمونه، حال به یک تعداد نقطه میرسیم، سپس بگردیم ببینیم بهترین خطی که شیب یک دارد و از این نقاط عبور میکند به چه صورت است، عرض از مبدا این خط همان زمان شروع نمونه در آهنگ اصلی را به ما میدهد.

برای این کار هم میتوانیم از تابع fit در متلب استفاده کنیم .

**۸ - آهنگی از آهنگهای موجود در پوشـه musics را به دلخواه انتخاب کنید. به صـورت تصـادفی قسـمتی از این آهنگ به طول ۲۰ ثانیه انتخاب کنید. در ادامه با استفاده از دستور randn ، تکه آهنگ بدست آمده را به نویز سفید گوســی با میانگین صــفر و واریانس ۰۱ / ۰ آغشــته کنید . حال عملکرد فایل search_database.m را بر روی خروجی بررسی کنید. این کار را به ازای SNR های مختلف تکرار کنید و کمترین SNR ممکن برای تشخیص صحیح آهنگ را بدست آورید.**

ما نمونه شــماره ۸ را از پوشــه music_test به عنوان موزیکی که میخوایم روش نویز را اعمال کنیم انتخاب میکنیم، سپس با استفاده از دستور awgn به آن نویز سفید گوسی با snr های متفاوت اضافه میکنیم و نتایج را ثبت میکنیم.

```
% adding noise to the audio
snr = 10;
audioMono=awgn(audioMono,snr);
%%%
```

نتایج بدست آمده بدین صورت است :

به ازای snr =10 : هر بار آهنگ ۱۱ به درستی انتخاب شد و میزان احتمال حدودا ۰۷۵ است.

به ازای snr =7 : هر بار آهنگ ۱۱ به درستی انتخاب شد و میزان احتمال حدودا ۰٪۶ است.

به ازای snr=5 : هر بار آهنگ ۱۱ به درستی انتخاب شد و میزان احتمال حدودا ۰٪۴ است.

به ازای snr=3 : هر بار آهنگ ۱۱ به درستی انتخاب شد و میزان احتمال حدودا ۰٪۳ است.

به ازای snr=2 : هر بار آهنگ ۱۱ به درستی انتخاب شد و میزان احتمال حدودا ۰٪۲ است.

به ازای snr=1 : در ۱۰ بار یک بار پیش آمد که اشتباه تشخیص بدهد و میزان احتمال حدودا همان ۰٪۲ است .

**به ازای snr=0.1** : میزان بار هایی که اشتباه تشخیص میدهد تقریبا قابل توجه میشود در این عدد.

البته باید دقت داشت که حتی در snr=0.01 هم تعداد بار هایی که میتواند درست تخیص بدهد الگوریتم خیلی کم نیست.

فاکتور های زیادی در این بخش دخیل هستند مانند خود آهنگ و این که چه آهنگ هایی در دیتابیس هستند، هر چقدر آهنگ های بیشتری در دیتابیس شبیه آهنگ ما باشند میشود حدس زد که با افزایش نویز احتمال اینکه آهنگ اشتباهی تشخیص داده شود بالاتر میرود .

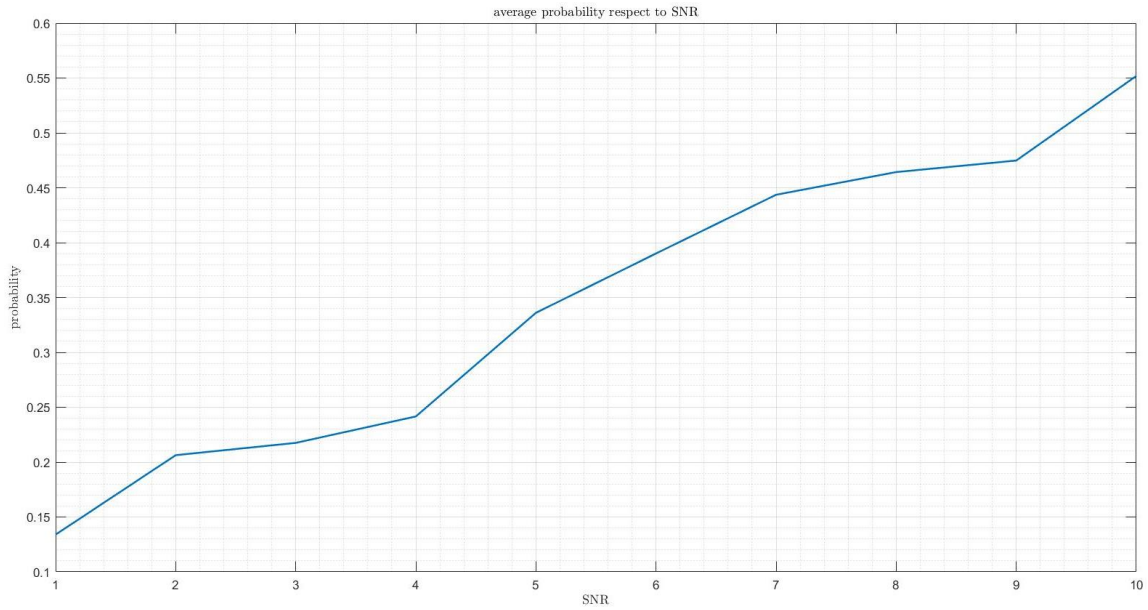
**۹ – آهنگی از آهنگهای موجود در پوشه musics را به دلخواه انتخاب کنید. در ادامه به ازای تغییر مقدار SNR از ۱۰ تا ۱ با دقت ۱ / ۰ با استفاده از دستور randn ، آهنگ مورد نظر را به نویز سفید گوسی با میانگین صفر و واریانس مناسب آغشته کنید. حال به ازای هر مقدار SNR ، به صورت تصادفی ۱۰۰ قسمت از این آهنگ به طول ۲۰ ثانیه انتخاب کنید و نمودار میانگین ا حتمال نسبت داده شده به آهنگ انتخاب شده را بر حسب SNR ، رسم کنید. این کار را برای چند آهنگ دیگر نیز تکرار کنید و نتایج را مقایسه کنید.**

کد مربوط به این سوال در فایلی تحت عنوان part9.m قرار گرفته است، بخش های مهم کد را همینجا هم قرار میدهم :

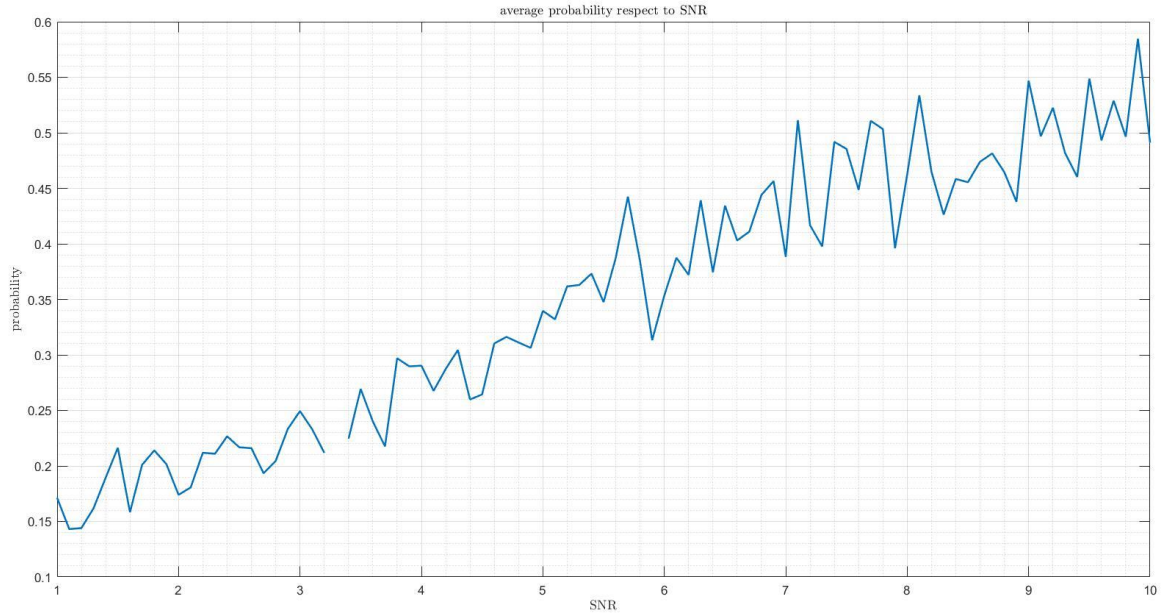
```
%% calculate the hash tags for the given song
snr=10;
prob_main=[];
while snr>=1
    %importing an audio
    path = 'D:\sharif\signals and system\project\musics\'; % test musics path
    song_num = 27; % music i
    format = '.mp3';
    [downsampled_Fs, audioMono] = import_audio(path, song_num, format);
    % adding noise to the audio
    audioMono=awgn(audioMono,snr);
    L=length(audioMono);

    prob_temp=[];
    p=1;
    while p<=100
        start_time=1+floor(rand()*(L-(20*downsampled_Fs)));
        end_time=start_time+(20*downsampled_Fs);
        audio_sample=audioMono(start_time:end_time);
```

```
        end
        % scoring
        clc; close all;

        score=scoring(list);

        prob_temp=[prob_temp;score(1,2)];
        p=p+1;
    end
    prob_main=[prob_main;mean(prob_temp),snr];
    snr=snr-0.1; % step  of SNR
end
% plot probability of audio1 and audio 2
plot(prob_main(:,2),prob_main(:,1),'Linewidth',1.5);
xlabel('SNR','interpreter','latex');
ylabel('probability','interpreter','latex');
title("average probability respect to SNR",'interpreter','latex');
grid on; grid minor;
```

**نمودار خروجی به ازای اینکه استپ های کاهش snr طول ۱ داشته باشند :**




**نمودار خروجی به ازای اینکه استپ های کاهش snr طول ۰/۱ داشته باشند :**




مطابق با انتظار میبینیم که نمودار صعودی میباشد .

**۱۰ - آهنگی از آهنگهای موجود در پوشه musics را به دلخواه انتخاب کنید. تکهای از این آهنگ را با لپتاپ پخش کنید و با گوشی خود صدای محیط را ضبط کنید. حال عملکرد فایل search_database.m را بر روی خروجی بررسی کنید (نمودار مربوط به STFT و درایه های anchor point را در گزارش قرار دهید). با تکرار این آزمایش و ایجاد سر و صدا در هنگام ضبط آهنگ، میزان دقت الگوریتم مورد استفاده را در کاربرده ای واقعی بسنجید.**

برای این قسمت من پوشه ای تحت عنوان part 10 ایجاد کردم و ۵ فایل موزیک با فرمت m4a. داخل آن قرار داده ام، در این فایل ها بخشی از آهنگ شماره ۲۶ را پخش کرده ام و به ترتیب از فایل ۱ تا ۵ سعی کرده ام نسبت میزان نویز به خود سیگنال را افزایش بدهم و عملکرد برنامه را تست کنم.

روش کار هم بدین صورت بوده که در فایل اول صرفا از لپتاپ موزیک را پخش کرده ام و بدون هیچ نویز اضافه کردنی و در فاصله نزدیک از لپتاپ صوت را با گوشیم ضبط کردم.

در فایل دوم پنکه اتاقم را روشن کردم و یک نویز ثابت به سیگنال اضافه شده است که کاملا مشهود است، در این بخش هم فاصله تا لپتاپ را همان مقدار نزدیک نگه داشته ام .

در فایل سوم سعی کردم که یک نویز غیر پیوسته ایجاد کنم، این کار رو با چند توپ پینگ پنگ انجام داده ام، اما همچنان فاصله تا لپتاپ را نزدیک نگه داشته ام .

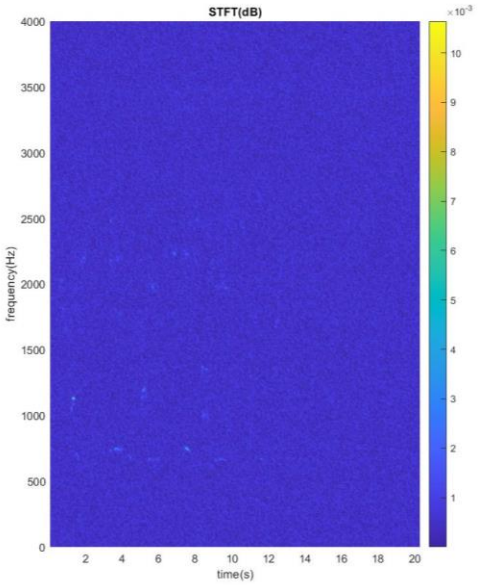
در فایل چهارم به گوشه اتاق رفته ام تا فاصله از منبع صوت بیشتر بشود، سپس کمی سر و صدا با فاصله نزدیک از گوشی ایجاد کرده ام .

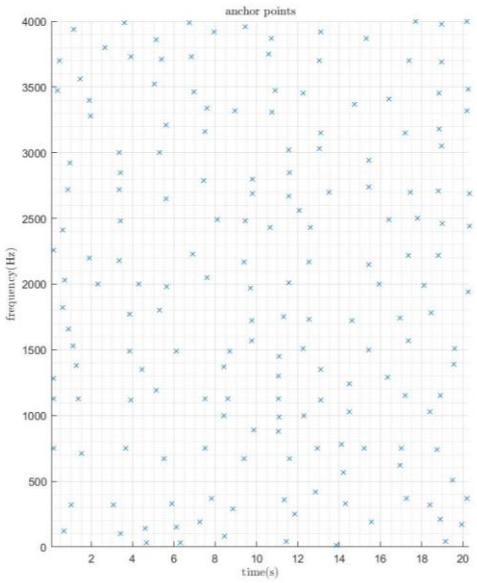
در فایل پنجم تقریبا از اتاق بیرون رفتم و  سر و صدای بخش های دیگه خونه هم داخل صوت هستند.

**نتیجه : در فایل های ۱ تا ۳ که ضبط کردم برنامه به درستی تشخیص میدهد آهنگ شماره ۲۶ را، ولی در فایل های ۴ و ۵ که فاصله از لپتاپ زیاد شده و یا به عبارتی SNR ما خیلی کم شده است، برنامه دچار تشخیص اشتباه میشود، اما در کل میتوان گفت که برنامه عملکرد قابل قبولی داشت، چون حتی خود shazam هم، در مواردی که شدت سیگنال صوت به اندازه کافی نیست تشخیصی نمیتوانست بدهد و ارور میداد. البته خوبی آن اینست که آهنگ اشتباه به ما تحویل نمیدهد و میفهمد که نسبت نویز به سیگنال زیاد است و صرفا ارور میدهد. اما در اینجا ما آهنگ اشتباهی به کاربر تحویل میدهیم.برای اینکار باید تعیین کنیم که اگر میزان احتمال آهنگی که در اولویت اول است از یک حدی کمتر بود، اصلا چیزی به کاربر نشان داده نشود.**

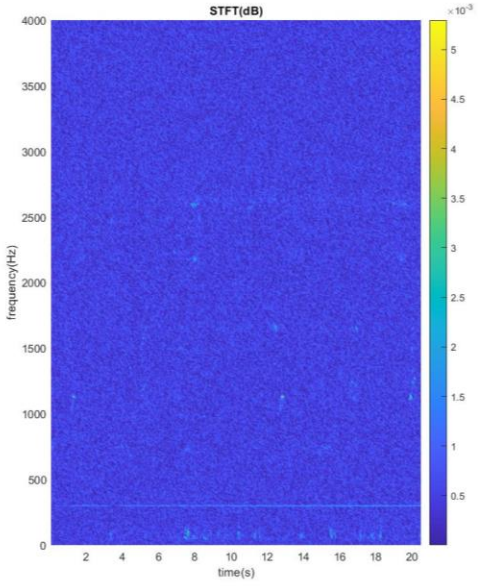
در ادامه نمودار های STFT و anchor points این ۵ فایل را قرار داده ام .

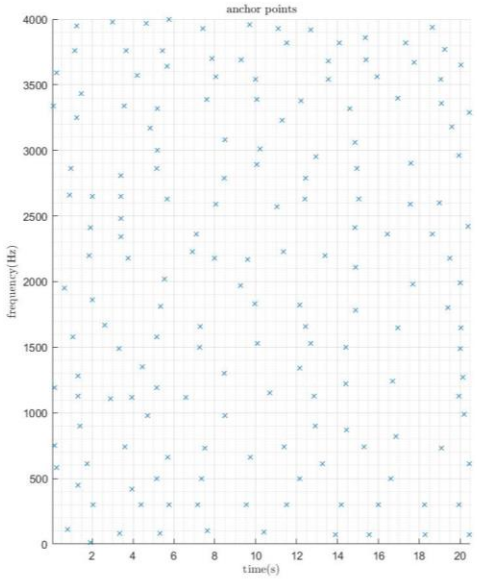
**فایل ۱ :**



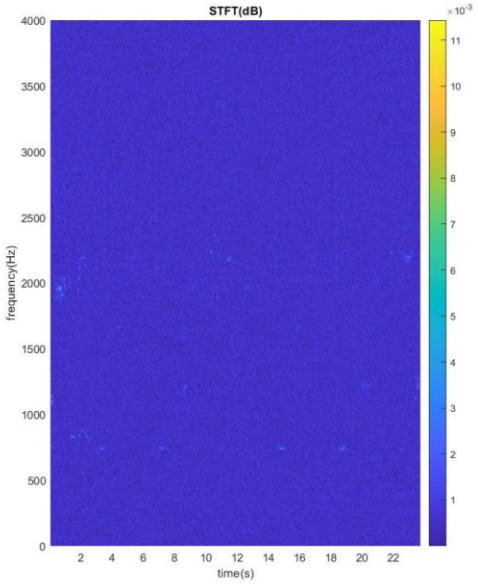

**فایل ۲ :**

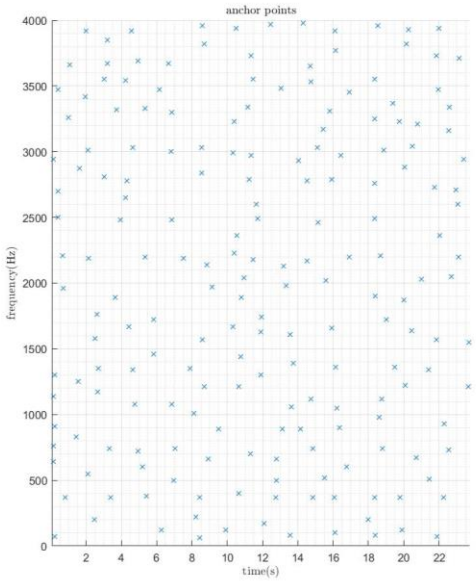


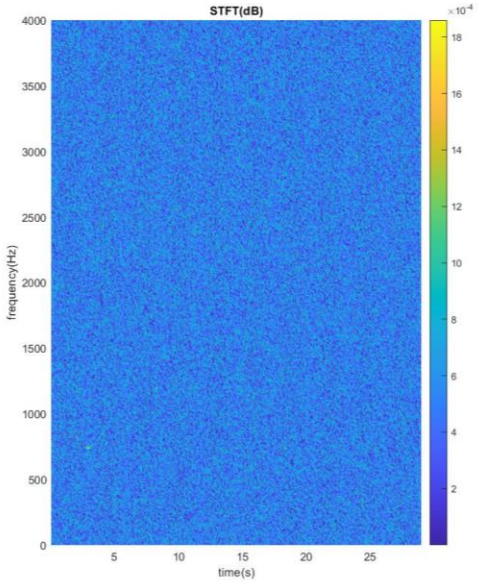

اثر نویز پنکه که اضافه شده است به صورت یک خط در فرکانس حدود ۳۵۰ هرتز در نمودار به خوبی دیده میشود . این به این دلیل است که این نویز تقریبا ثابت است در طول آهنگ.

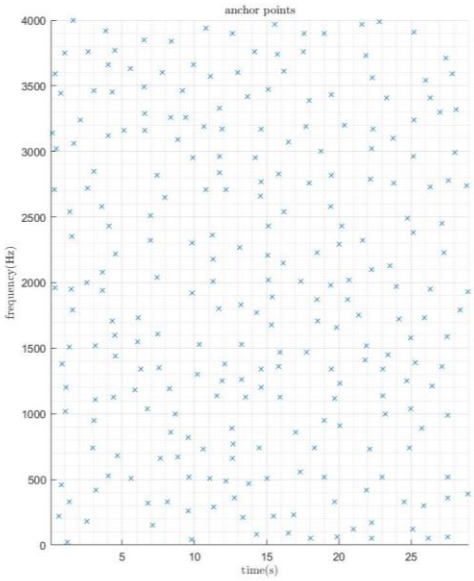
## فایل ۳ :


STFT(dB)
frequency(Hz)
time(s)

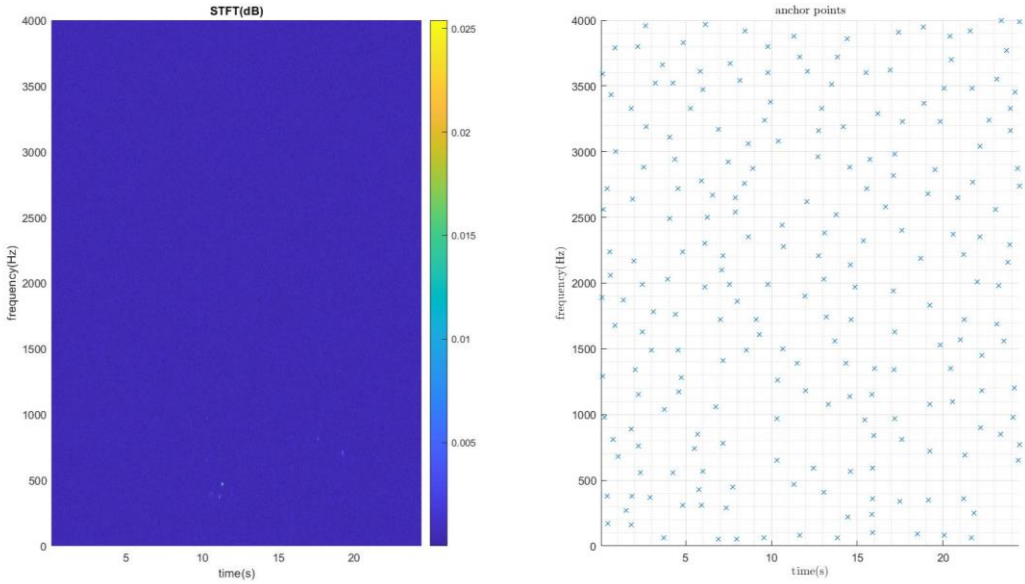

anchor points
frequency(Hz)
time(s)

## فایل ۴ :


STFT(dB)
frequency(Hz)
time(s)


anchor points
frequency(Hz)
time(s)

## فایل ۵ :




**۱۱ – دو آهنگ از آهنگ های موجود در پوشه musics را به دلخواه انتخاب کنید. به صورت تصادفی قسمتی از هر یک به طول ۲۰ ثانیه انتخاب کنید. د و تکه آهنگ بدست آمده را با ضرایب $\alpha$ و $1-\alpha$ به ازای $\alpha$=۰/۵ با یکدیگر جمع کنید. حال عملکرد فایل search_database.m را بر روی خروجی بررسی کنید. با تغییر ضریب $\alpha$ بین صفر تا یک، نمودار احتمال نسبت داده شده به هر یک از دو آهنگ بر حسب ضریب $\alpha$ را رسم کنید و تغییر تصمیم الگوریتم را مشاهده نمایید.**

در این بخش چون تغییرات نسبتا زیادی در کد میخواستم بدهم، فایل جدیدی ایجاد کردم تحت عنوان part 11.m که کد مربوط به این سوال در آن وجود دارد، اما بخش های مهم کد اضافه شده را همینجا هم قرار میدهم .

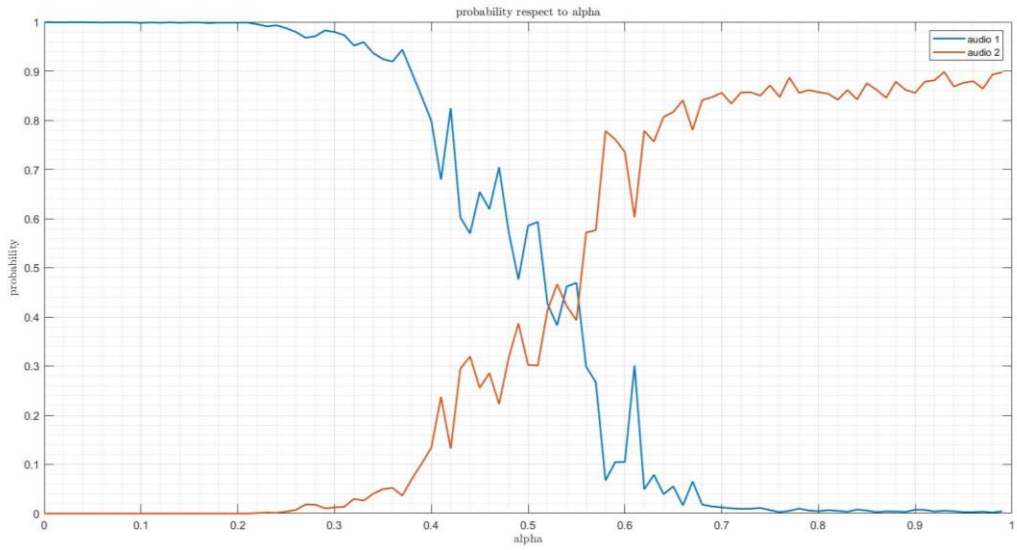
## کد ها :

```
path = 'D:\sharif\signals and system\project\musics\'; % test musics path
format = '.mp3';
song_num_1=27;
song_num_2=45;
[downsampled_Fs1, audio1] = import_audio(path, song_num_1, format); % music 27 - skyfall - adele
[downsampled_Fs2, audio2] = import_audio(path, song_num_2, format);% music 45 - callin U - outlandish
audio1=audio1(146*downsampled_Fs1:166*downsampled_Fs1);% split song noumber 27 from seconde 146 to 166
audio2=audio2(1*downsampled_Fs2:21*downsampled_Fs2);% split song noumber 45 from seconde 146 to 166
alpha=0;
prob_audio_1=[];
prob_audio_2=[];
while alpha<=1
```

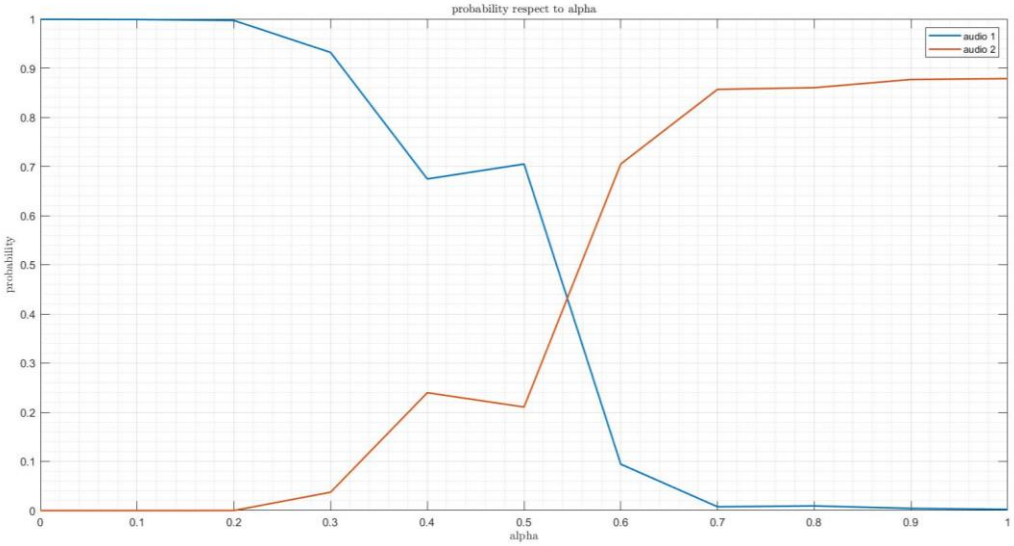
```
    score=scoring(list);
    prob_audio_1=[prob_audio_1;score(score(:,1)==song_num_1,2),alpha];
    prob_audio_2=[prob_audio_2;score(score(:,1)==song_num_2,2),alpha];

    alpha=alpha + 0.05; % step of the alpha
end

% plot probability of audio1 and audio 2
plot(prob_audio_1(:,2),prob_audio_1(:,1),'Linewidth',1.5);
hold on;
plot(prob_audio_2(:,2),prob_audio_2(:,1),'Linewidth',1.5);
xlabel('alpha','interpreter','latex');
ylabel('probability','interpreter','latex');
title("probability respect to alpha",'interpreter','latex');
grid on; grid minor;
legend('audio 1','audio 2');
```

**نتایج :**

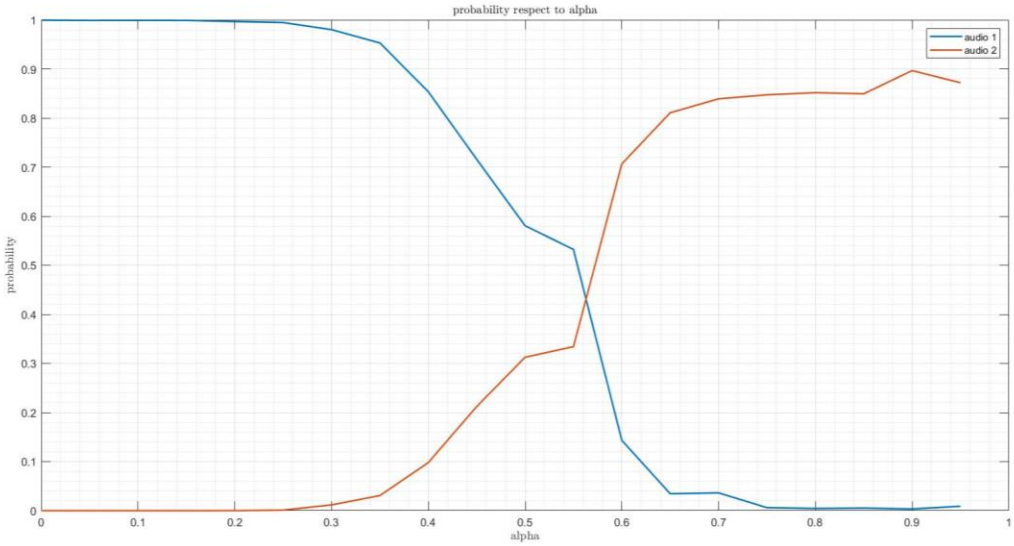
به ازای استپ های ۰٫۰۱ :



به ازای استپ های ۰/۱ :




به ازای استپ های ۰/۰۵ :




نتایج مطابق با انتظار بود، زیرا وقتی آلفا افزایش پیدا میکند درصد آهنگ اول در سیگنال مورد بررسی کاهش پیدا میکند و درصد آهنگ دوم افزایش پیدا میکند، بنابراین انتظار میرود روند نمودار احتمال آهنگ اول بر حسب آلفا نزولی باشد و روند آهنگ دوم بر حسب آلفا صعودی باشد، که میبینم همین طور است .

# پایان