بسمه تعالی

دانشگاه صنعتی شریف

دانشکده مهندسی برق

گزارش پروژه‌ی درس سیگنال و سیستم

استاد درس : دکتر آرش امینی

سپهر کاظمی رنجبر

۹۹۱۰۶۵۹۹

کد بخش‌های مربوطه در فایل my_answers.m موجود است.

۱

در این قسمت کد تابع import_audio را به این صورت کامل می‌کنیم که ابتدا چک می‌کنیم سیگنال دوکاناله هست یا نه اگر بود میانگین کانال چپ و راست را به دست می‌آوریم در غیر این صورت همان تک کانال را استفاده می‌کنیم. سپس با دستور resample، سیگنال به دست‌آمده از مرحله قبل را down sample می‌کنیم، این تابع یک Time Series و یک بردار زمانی به عنوان ورودی گرفته و یک Time Series به عنوان خروجی برمیگرداند که مقادیر سیگنال را در زمان‌های داده‌شده در ورودی با تقریب خطی حساب می‌کند. در نهایت سیگنال با نرخ $44100Hz$ به سیگنال با نرح $8000Hz$ تبدیل می‌شود.

۲

با تابع fft ابتدا تبدیل فوریه سیگنال را به دست می‌آوریم و بر طول سیگنال تقسیم کرده و به توان دو می‌رسانیم. سپس از آنجا که فقط فرکانس‌های مثبت را می‌خواهیم و فرکانس‌های $[\pi, 2\pi]$ جزو فرکانس های منفی محسوب می‌شوند و همچنین چون تابع fft مقدار تبدیل فوریه را در فرکانس‌های $\left\{\frac{2k\pi}{N}\right\}_{k=0}^{N-1}$ حساب می‌کند، به همین جهت باید صرفاً فرکانس‌های $\left\{\frac{2k\pi}{N}\right\}_{k=0}^{\lfloor\frac{N}{2}\rfloor}$ را درنظربگیریم. از آنجایی که اندیس‌ها در متلب از یک شروع می‌شود در نتیجه $k = 0$ معادل با اندیس شماره‌ی 1 است. پس باید تا اندیس $1 + \lfloor\frac{N}{2}\rfloor$ از خروجی تابع fft را نگه داریم. درنهایت با دو برابر کردن همه فرکانس‌ها به جز صفر تبدیل فوریه یک‌طرفه‌ی سیگنال را به دست می‌آوریم.

۳

ابتدا با ضرب نرخ نمونه‌برداری در طول زمانی پنجره، تعداد نمونه‌ای موجود در یک پنجره به دست می‌آید. حال از آنجایی که پنجره‌ها، 50 درضد باهم همپوشانی دارند در نتیجه می‌توان با تقسیم کل نمونه‌ها بر تعداد نمونه‌ی موجود در یک پنجره تقسیم بر دو، تعداد کل پنجره‌ها را به دست آورد. همچنین طبق قسمت قبل می‌دانیم که طول تبدیل فوریه‌ی پنجره‌ای به طول L، $1 + \lfloor\frac{L}{2}\rfloor$ می‌باشد. به همین ترتیب ماتریس time _freq _mat که هر ستون آن نشان دهنده تبدیل فوریه یک پنجره است را تعریف می‌کنیم و هر ستون i اُم آن، تبدیل فوریه یک طرفه‌ی پنجره‌ی i اُم می‌باشد. همچنین زمان شروع پنجره‌ها را هم می‌توان با ضرب کردن نصف طول پنجره در شمارنده ایجاد کرد. فرکانس‌های تبدیل فوریه را نیز می‌توان از رابطه‌ی $\left\{\frac{2k\pi}{L}\right\}_{k=0}^{\lfloor\frac{L}{2}\rfloor}$ به دست می‌آید که این فرکانس مربط به تعداد نمونه‌ها می‌باشد که باید به فرکانس زمانی تبدیل شود که در نهایت داریم:

$$f = \left\{\frac{F_s}{L}\right\}_{k=0}^{\lfloor\frac{L}{2}\rfloor}$$

۴

در فایل create_database ابتدا با استفاده از دستور Container.Map متغیر database را درست می‌کنیم که key value و value type آن از نوع char می‌باشد. سپس با استفاده از fullfile دایرکتوری را درست می‌کنیم و با دستور dir تمام فایل‌های موجود در آن را می‌گیریم و آن‌ها را در متغیر filenames قرار می‌دهیم سپس روی تمام موزیک‌های موجود در دایرکتوری یا همان filenames فور می‌زنیم. در هر حلقه، تابع import_audio را روی موزیک k اُم صدا می‌زنیم سپس سیگنال به دست آمده را به تابع STFT می‌دهیم تا ماتریس مربوطه را به دست آوریم. در نهایت ماتریس زمان–فرکانس را به تابع find_anchor_points می‌دهیم، این تابع به ازای تمام درایه‌های ماتریس time _freq _mat پنجره‌ای به ابعاد $2df \times 2dt$ در نظر گرفته و ماکس این پنجره را به دست می‌آورد، نکته مهم این است اگر $t + dt$ از طول زمان ماتریس بیشتر شد باید همان طول ماتریس را بزاریم در مورد فرکانس هم همین مسئله برقرار است در نتیجه برای این‌ها باید ماکس را حساب کنیم همچنین اگر مقدار $t - dt$ از میمنموم زمان ماتریس کمتر شد باید مینیموم زمان ماتریس را بگذاریم در مورد فرکانس هم این مسئله برقرار است به همین جهت باید باید مینیموم بگیریم.در نهایت اگر ماکسیمووم درایه پنجره درایه مدنظر بود آن را در anchor point قرار می‌دهیم در غیر این صورت آن را قرار نمی‌دهیم. حال با استفاده از این تابع ماتریس anchor point را به دست می‌آوریم، در ادامه ماتریس time _mat _freq را با دستور pcolor به صورت رنگی رسم می‌کنیم همچنین ماتریس anchor point را نیز به صورت نقطه‌ای با دستور scatter رسم می‌کنیم. در ادامه با استفاده از ماتریس anchor point، hash tag ها را می‌سازیم به این صورت که ابتدا ابعاد $df_{hash}, dt_{\text{hash}}$ را به ترتیب یک‌دهم طول فرکانسی anchor point و $\frac{20}{\text{windows time}}$ می‌کذاریم سپس تابع create hashtag را فراخوانی می‌کنیم. در این تابع ابتدا روی تمام anchor point ها فور می‌زنیم و به ازای هر anchor point روی تمام anchor point های دیگر فور می‌زنیم، چون می‌دانیم پنجره جلوی anchor point زده می‌شود درنتیجه نیازی به فور زدن روی همه‌ی آنها نیست و فقط کافی است روی anchor point ه بعد از آن فور بزنیم چون میدانیم anchor point ها به ترتیب زمان مرتب شده‌اند. حال به ازای هرکدام از anchor point ها دیگر چک می‌کنیم که داخل پنجره هستند یا نه(ابعاد پنجره نسب به نقطه‌ی مرجع یک df بالاتر و یک df پایین‌تر است. همچنین به اندازه‌ی یک dt جلوتر است.) هر کدام از نقاط که داخل پنجره افتاد مقدار فرکانس آن نقطه و فرکانس نقطه مرجع و اختلاف زمانی این دو را به صورت یک سه‌تایی در بردار hashkey نگه‌داری می‌کنیم همچنین شماره‌اهنگ و زمان نقطه‌ی مرجع را نیز به صورت یک دوتایی در بردار hashvalue نگه‌داری می‌کنیم. نکته مهم این است که قبل تخصیص hashvalue چک کنیم که آیا hashkey در هش‌مپ وجود دارد یا نه در صورتی که وجود نداشت تخصیص را راحت انجام می‌دهیم ولی در صورتی که وجود داشت باید با کاراکتر +، مقدار hashvalue را به مقدار قبلی اضافه می‌کنیم، همچنین خود المان‌های hashkey,hashvalue را با کاراکتر * از هم جدا می‌کنیم.در نهایت هش‌مپ را با دستور save، ذخیره می‌کنیم.

۵

در search_database ابتدا دیتابیس را لود می‌کنیم سپس مانند create database، has tag های مربوط به تکه آهنگ بیست ثانیه‌ای را می‌یابیم. سپس بردار list را تعریف می‌کنیم. هرالمان این بردار حاوی شماره‌ی آهنگ و زمان anchor point در آهنگ اصلی و زمان anchor point در تکه‌ی بیست ثانیه‌ای است. این سه‌تایی‌ها به این شکل به دست می‌آیند که روی تمام hash key ها مربوط به تکه‌ی بیست ثانیه‌ای فور می‌زنیم، حال به ازای هر hash key اگر توانستیم مثل آن را در کلیدهای دیتابیس پیدا کنیم آن‌گاه با گرفتن تمام value های مربوط به آن کلید (این کار با دستور split که ابتدا کاراکتر + را اسپلیت می‌کند و سپس روی تمام hash value ها فور می‌زنیم و با کاراکتر * آن hash value را اسپلیت می‌کنیم تا به شماره آهنگ و زمان anchor point دسترسی داشته باشیم.) بردار list را کامل می‌کنیم یعنی سه‌تایی که در بالا ذکر شد را به لیست اضافه می‌کنیم، المان سوم سه‌تایی همان زمان anchor point تکه‌آهنگ است که نیازی به دیتابیس ندارد و مستقسیما به hash value مربوط به آن کلید پر می‌شود. عملا با این کار تمام تشابهات را در متغیر list ذخیر می‌کنیم و در نهایت با تابع scoring با توجه به تعداد تکرار یک آهنگ در متغیر list و اختلاف زمان شروع در دیتابیس و تکه‌اهنگ (اگر همه چی ایده‌آل باشد این اختلاف زمانی ثابت است در نتیجه منطقی است که در تابع scoring واریانس این اختلاف ظاهر شود چون اگر صفر باشد چون هرچه کمتر باشد انتخاب دقیق‌تر است چون ممکن است چند آهنگ hash key های یکسان داشته‌باشند ولی در اختلاف زمانی‌های متفاوت) در تابع scoring نیز ابتدا تکرار هر آهنگ و اختلاف زمان شروع در دیتابیس و تکه‌ی بیست ثانیه را به دست می‌آوریم و سپس با استفاده از فورمول داده شده احتمال آن را حساب می‌کنیم. و در خروجی به ترتیب بیشترین به کمترین چاپ می‌کنیم.

۶

ساختار داده‌ی هش‌مپ به این صورت کار می‌کند که به هر key یک hash code نسبت می‌دهد. اگر hash funcion یی که این نسبت را می‌دهد خوب عمل کند به هر key یک hash code جدا نسبت می‌دهد که باعث میشود در $O(1)$ هش‌مپ به مقدار کلید دسترسی داشت. ابته در مواقعی ممکن است هش‌کدهای یکسانی به کلیدها نسبت داده شود که در این صورت باید روی همه آن iterate کند و تا بتواند کلید مدنظر را بیابد. نکته مهم این ساتکه در بدترین حالت مرتبه جست‌وجو از $O(n)$ است ولی در حالت میانگین به $O(1)$ کاهش می‌یابد. پس برای ساختار دیتابیس‌های بزرگ ساختار داده مناسبی است. مثلا اگر می‌خواستیم از array استفاده کنیم در حالت میانگین هم اردر جست‌وجوی آن از $O(n)$ بود. به همین جهت هش‌مپ انتخاب مناسبی است.

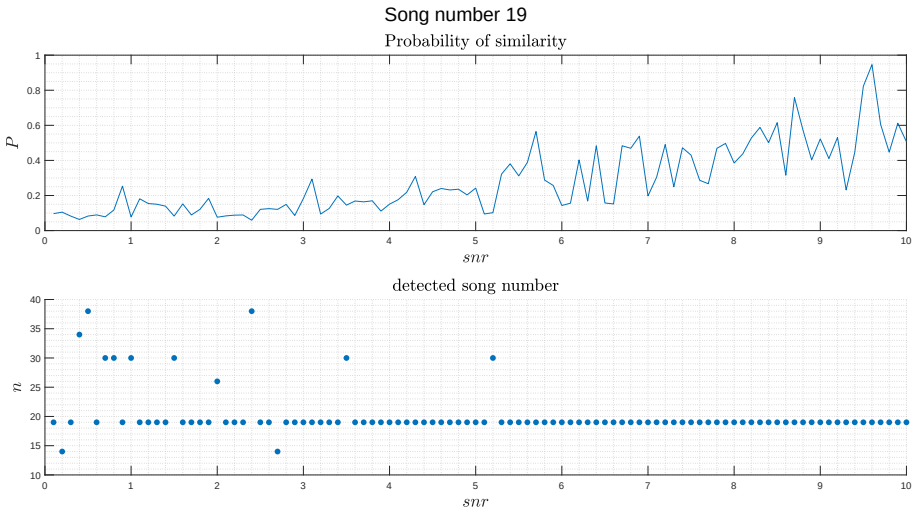
۷

کد این بخش در سکشن Part 7 فایل my_answers قرار دارد.
برای یافتن زمان تقریبی شروع تکه موسیقی، یکی از کارهایی که می‌شود کرد این است که ابتدا با scoring پیدا می‌کنیم که تکه بیست ثانیه‌ای مروبط به چه آهنگی بوده‌است سپس روی تمام list فور می‌زنیم و المانی از list که شماره آهنگ آن یکی با شماره آهنگ کشف شده توسط الگوریتم است را در نظر می‌گیریم و زمان شروع آن المان را در یک بردار دیگر ذخیره می‌کنیم در نهایت بین تمام این زمان‌های شروع میانه را انتخاب می‌کنیم انتخاب میانه به این دلیل است که تمامی آن hash key در سرتاسر موزیک پخش شده‌اند ولی تراکم آن طبعا در بیست ثانیه انتخاب شده بیشتر است. از طرفی قبل بیست ثانیه و بعد بیست ثانیه انتخابی به طور احتمالاتی چگالی برباری دارند در نتیجه با انتخاب میانه نقطه‌ای مناسب در بازه بیست ثانیه به ما به عنوان تقریب زمان شروع می‌دهد. کد را ران کرده و مشاهده می‌کنیم به ازای همه آهنگ‌ها تشخیص آهنگ و زمان شروع تقریبی درست است.

```
detected song is : 5
begin time of test music 1 (s) : 98.800000
----------------------

detected song is : 2
begin time of test music 2 (s) : 196.100000
----------------------

detected song is : 13
begin time of test music 3 (s) : 123.550000
----------------------

detected song is : 36
begin time of test music 4 (s) : 127.950000
----------------------

detected song is : 43
begin time of test music 5 (s) : 100.300000
----------------------

detected song is : 49
begin time of test music 6 (s) : 150.300000
----------------------

detected song is : 17
begin time of test music 7 (s) : 86.050000
----------------------

detected song is : 11
begin time of test music 8 (s) : 128.950000
----------------------

detected song is : 32
begin time of test music 9 (s) : 6.200000
----------------------

detected song is : 1
begin time of test music 10 (s) : 139.350000
```

۸

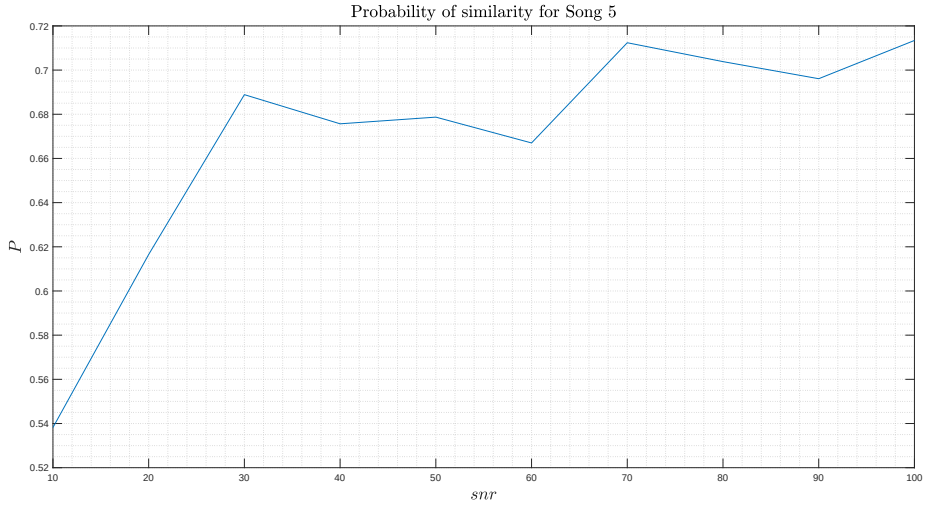
در این قسمت برای آهنگ شماره ۱۹، snr را از 10 تا 0.1 با دقت 0.1 پایین می‌آییم و مشاهده می‌کنیم اولین جا درکجا تشخیص آهنگ به مشکل می‌خورد. برای اینکار نمودار احتمال و تشخیص آهنگ را در یک figure برای آهنگ رسم می‌کنیم و از روی آن مشخص می‌کنیم. همچنین برای اینکه به صورت تصادفی تکه‌ای از آهنگ را انتخاب کنیم، با دستور randi نقطه شروعی به صورت تصادفی در بازه مناسب انتخاب می‌کنیم،همچنین برای ساخت نویز گوسی سفید از agwn استفاده می‌کنیم. مابقی کد مشابه search_database است که از توضیح دوباره‌ی آن خودداری می‌کنیم. در نهایت داریم:

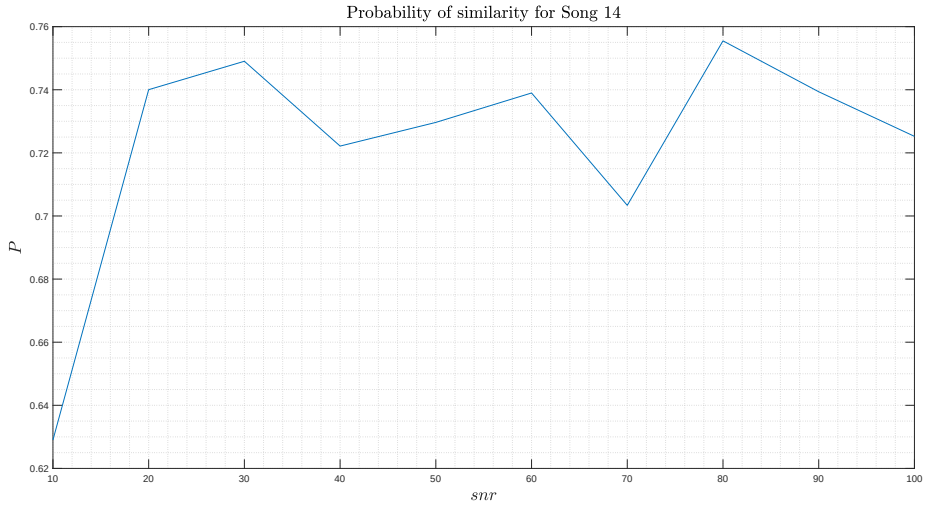


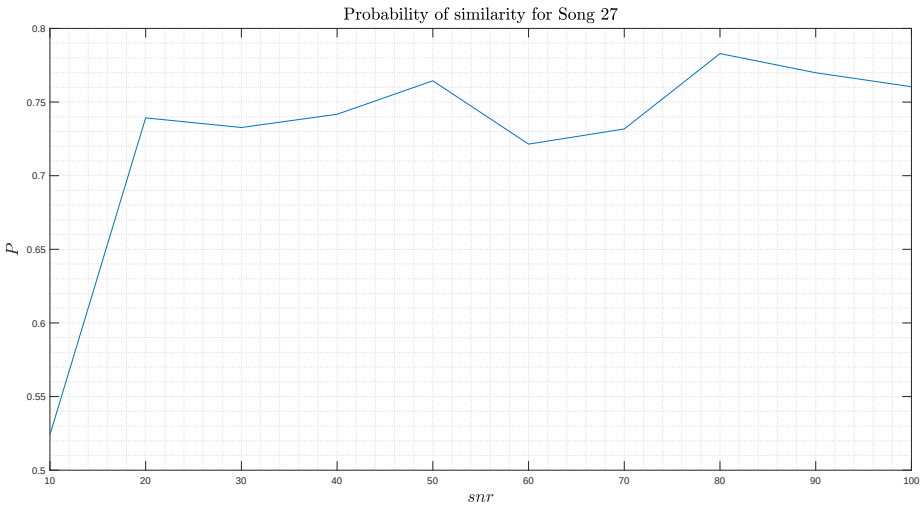
مشاهده می‌کنیم که برای آهنگ ۱۹، $snr_{\min} = 5.2$، می‌باشد.

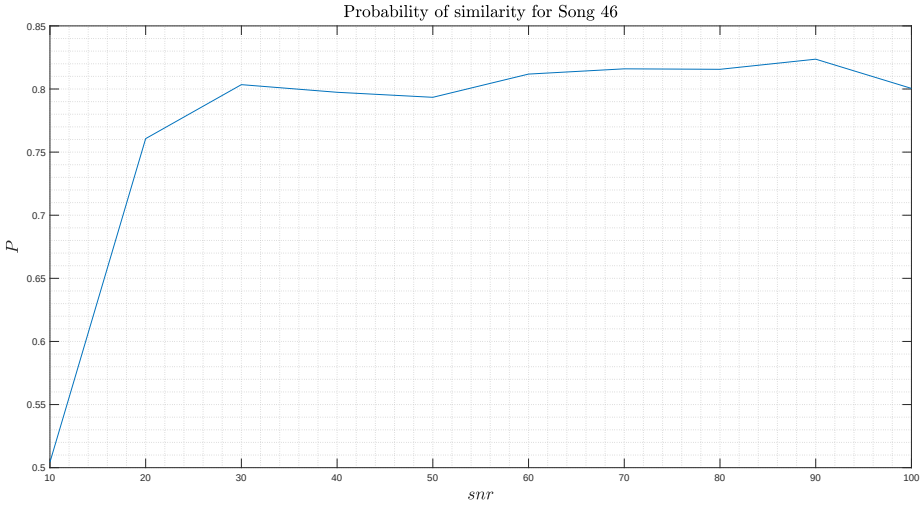
۹

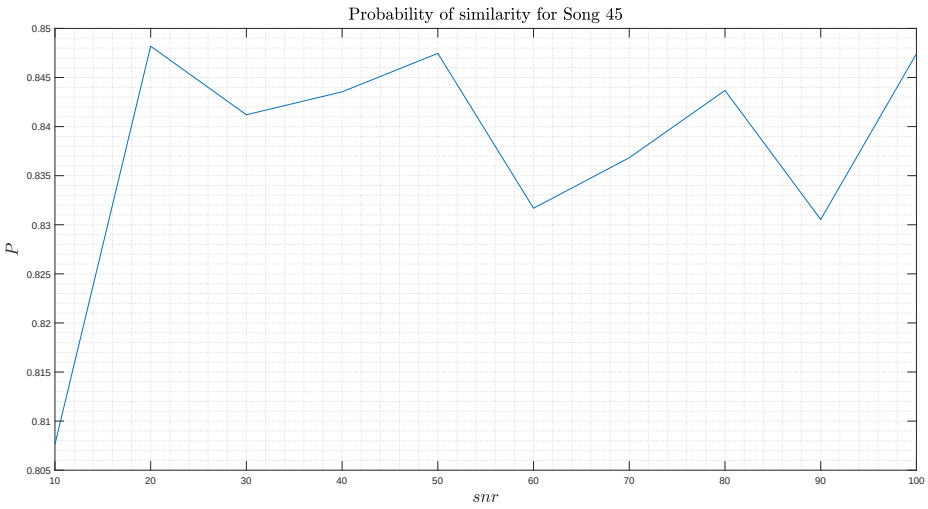
برای این قسمت مانند قسمت قبل عمل می‌کنیم با این تفاوت که برای چند آهنگ دل یک فور می‌زنیم(در اینجا آهنگ‌ها را رندوم انتخاب کردیم و تعداد آن‌ها ۵ تاست) سپس به ازای هر snr از ده تا یک با گام یک‌دهم، ۱۰۰ مرتبه احتمال آن را حساب می‌کنیم و میانگین آن را درنظر می‌گیریم. در نهایت نمودار آن را رسم می‌کنیم. فرایند تصادفی انتخاب کردن هم مانند قسمت قبل است.






Probability of similarity for Song 27
P
snr
0.8
0.75
0.7
0.65
0.6
0.55
0.5
10
20
30
40
50
60
70
80
90
100


Probability of similarity for Song 46
P
snr
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
10
20
30
40
50
60
70
80
90
100



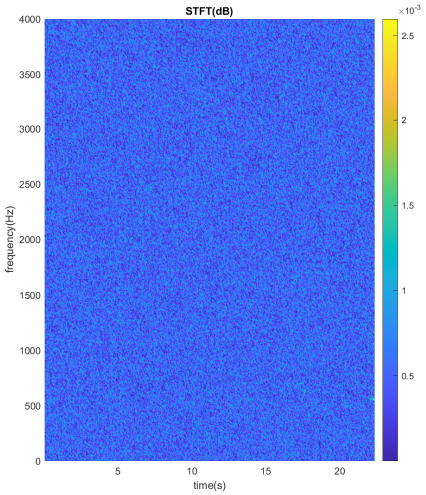
مشاهده می‌کنیم احتمال تشابه با افزایش snr بیشتر می‌شود یعنی هرچه نویز کمتر باشد تشابه بیشتر است.

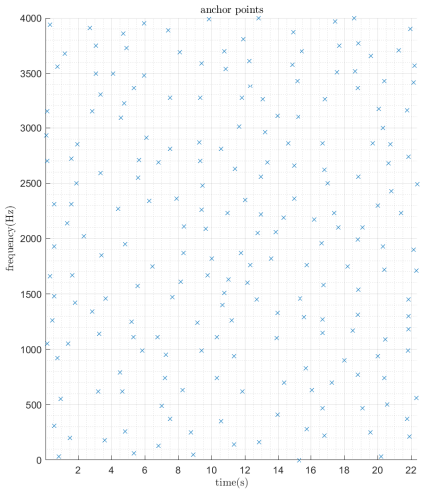
۱۰

در این قسمت مطابق گفته صورت سوال عمل می‌کنیم و دو آهنگ به پوشه test_music اضافه می‌کنیم. آهنگ music11 ضبط‌شده با کیفیت و آهنگ music12 ضبط شده‌ی بی‌کیفیت است. مشاهده می‌کنیم که :

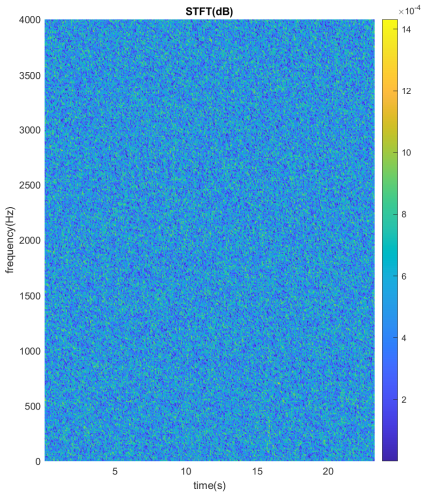
```
detected song is : 6
begin time of test music 11 (s) : 100.200000
----------------------

detected song is : 6
begin time of test music 12 (s) : 119.100000
----------------------
```

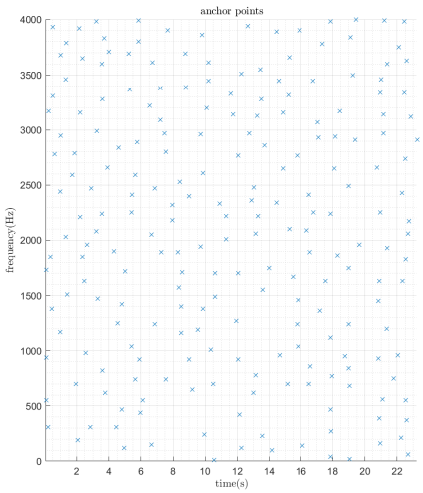
مشاهده می‌کنیم که تشخیص آن برای هر دو حالت درست است. حال نمودار را برای آهنگ ۱۱ رسم می‌کنیم:





حال نمودار را برای آهنگ ۱۲ رسم می‌کنیم:

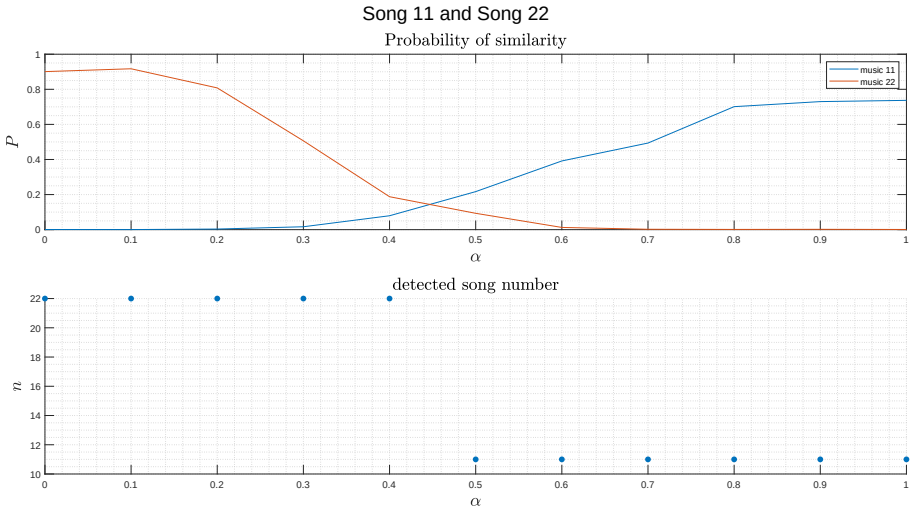




مشاهده می‌کنیم که دقت الگوریتم بالاست و حتی در شرایط بد نیز خوب عمل می‌کند.

۱۱

مطابق گفته صورت سوال عمل کرده و داریم:



مشاهده می‌کنیم تا $\alpha = 0.4$ الگورتیم آهنگ ۲۲ را انتخاب می‌کند ولی از $\alpha = 0.5$ به بعد الگوریتم آهنگ ۱۱ را انتخاب می‌کند. این نتیجه منطقی است چون هنگامی که درصد بیشتری از آهنگ مربوط به هرکدام است

الگوریتم آن آهنگ را انتخاب می‌کند فقط در $\alpha = 0.5$ با توجه به اینکه چه آهنگ‌هایی انتخاب شده‌اند نتیجه فرق می‌کند.