Music Information Retrieval HW3 Report

102062209 邱政凱

這次的作業要求我們用 Non-negative Matrix Factorization 實作 Source Separation 的演算法，並且做各種分析和比較。

B1、

第一個問題是要我們用 bss_eval 這個 toolbox 來測試 Source Separation 中經常用來當作度量標準的 SDR (Source Distortion Ratio)在各種不同音檔以及設定下的結果。第一列是 TARGET 音檔，第二列是 true source 音檔，第三列則是算出來的 SDR 值。Where a = 01_vio.wav　b = 01_cla.wav　c= 01_mix.wav　n = randn

[c';c']/2	[a';b']	[b';a']	[2*a';2*b']	(a+0.01*n)'	(a+0.1*n)'	(a+n)'	(a+0.01*b)'	(a+0.1*b)'	(a+b)'
[a';b']	[a';b']	[a';b']	[a';b']	a'	a'	a'	a'	a'	a'
0.2725 -0.2924	256.7621 246.9101	256.7621 246.9101	256.7621 246.9101	20.1376	0.1666	-18.840	40.2861	20.2848	0.2725

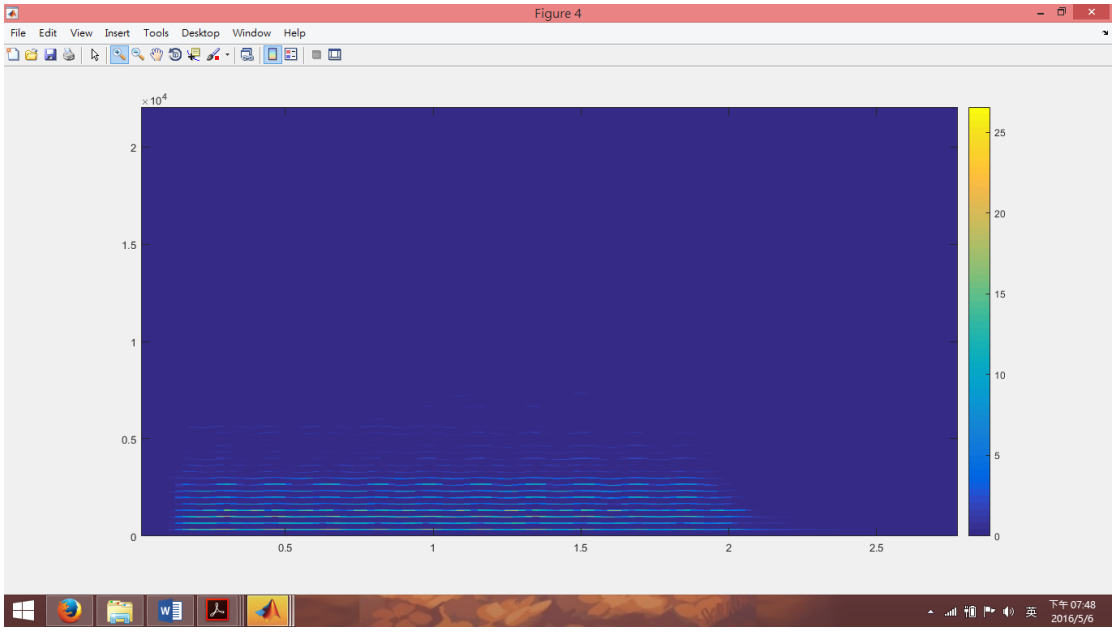
從表中可以看出，混合音檔跟單獨小提琴的音檔是比較像的(正值)，而跟單獨單簧管的音檔則是偏向於不像的(負值)。然後從第二行跟第三行可以看出，bss_eval 在計算 SDR 時會自動找出跟每個 source 最相近的 target 來輸出 SDR 值，所以 TARGET 音檔的順序就不是重點了。第四行我們可以看出 bss_eval 在計算 SDR 時應該是有經過 Normalization 的程序，因為就算把 target 音檔直接乘上兩倍算出來的 SDR 值還是跟沒有強化過的是一樣的。第五到第七行可以看出，音檔裡面其實只需一些些微的噪音就可以讓 SDR 值大幅下降。第八行到第十行則可以看到，相較於 random 的 noise，儘管 a 和 b 兩個音檔是不同樂器和不同的音高，但也許是因為都是 Harmonics 的關係，把 b 當成雜訊加進 a 裡面造成的 SDR 下降不像 random noise 那麼明顯。

B2、

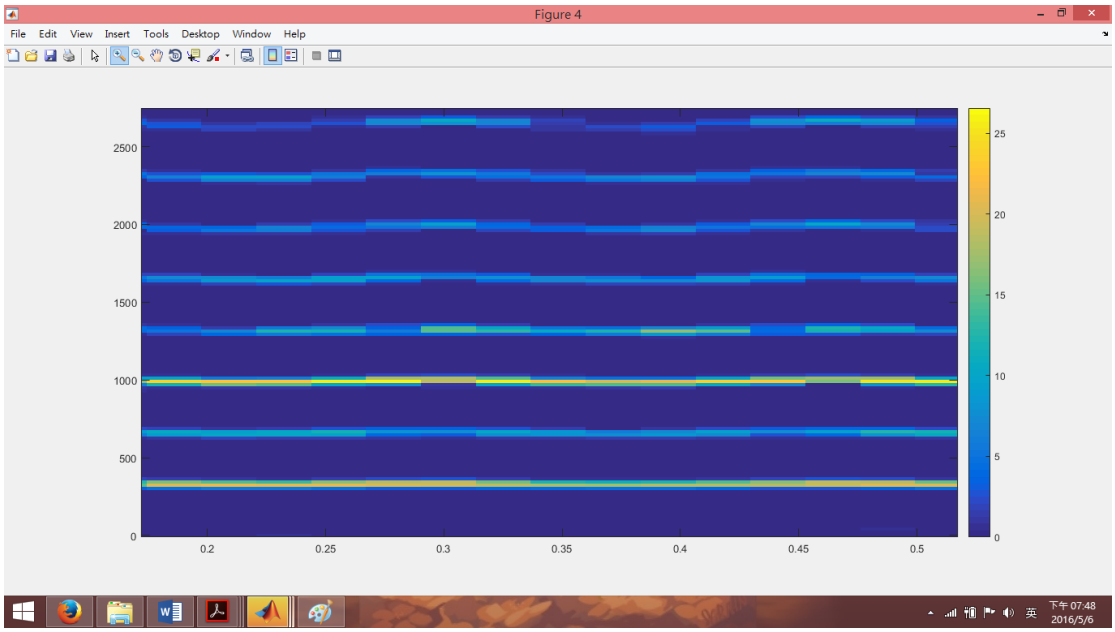
這題是要我們試試看 Inverse Short Time Fourier Transform 的效果。把 vio_64.wav 音檔轉換成 spectrogram 的部分，我使用的是 matlab 內建的 spectrogram 函式，並且設定 Window size 為 2048，Hop size 為 1024，number of fft point 設為跟 window size 一樣。實作的部分，把 spectrogram 中對應於取出的 frequency vector 中裡面大於或小於 1200 的部分直接設成 0 然後再做 istft。

```
[spect f t] = spectrogram(vio,w,h,nfft,fs);
spect = abs(spect(1:nfft/2+1,:));
%f = f * fs / 57.2958;
lspect = spect;
hspect = spect;
lspect(f>1200,:) = 0;
hspect(f<1200,:) = 0;
lvio = istft(lspect,h,nfft,fs);
hvio = istft(hspect,h,nfft,fs);
```

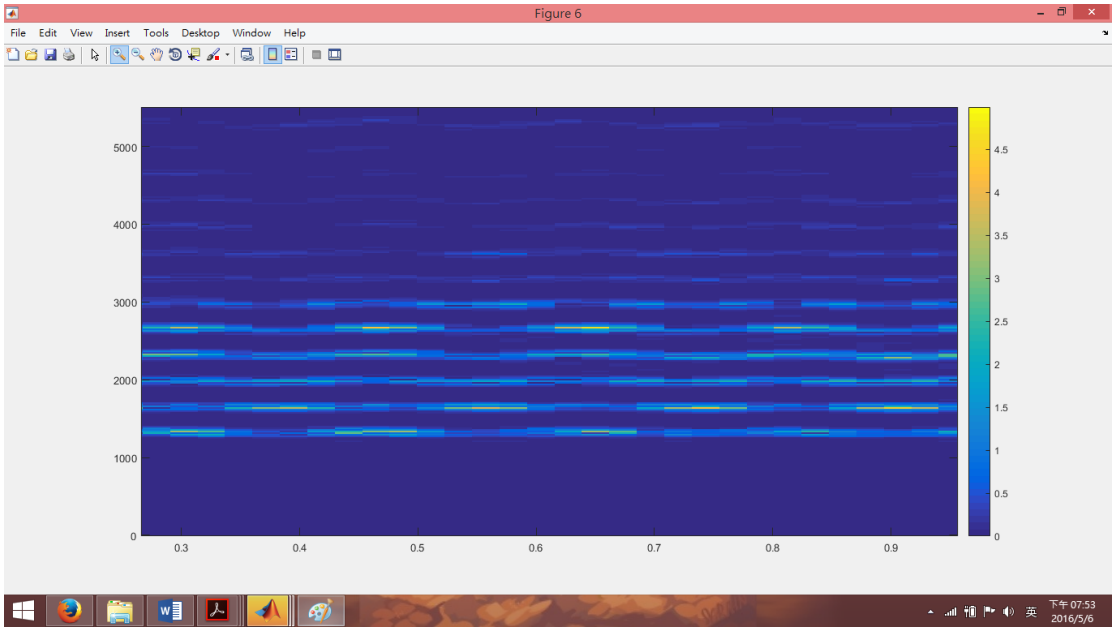
存成音檔之後，再次使用 spectrogram 函數，把 lvio 和 hvio 兩個低高頻的音檔轉換成頻譜圖，並且用 imagesc 的函數來跟原本的音檔比較。



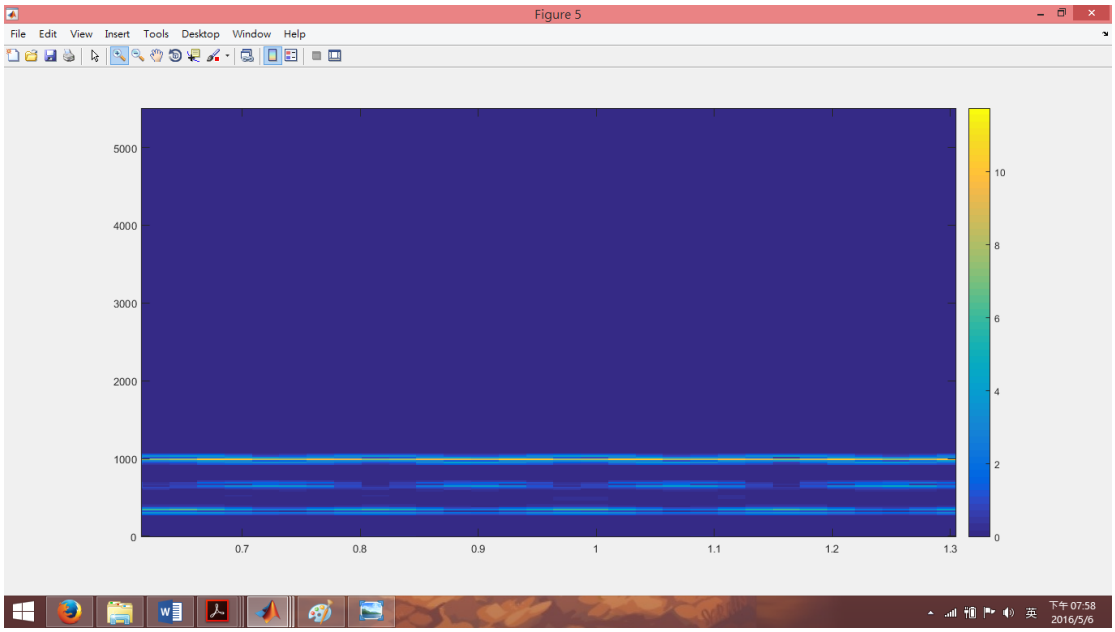
這是原本音檔的 spectrogram。



上圖放大的結果。Pitch id 64 對應到的基頻是 330，可以看到跟此圖吻合

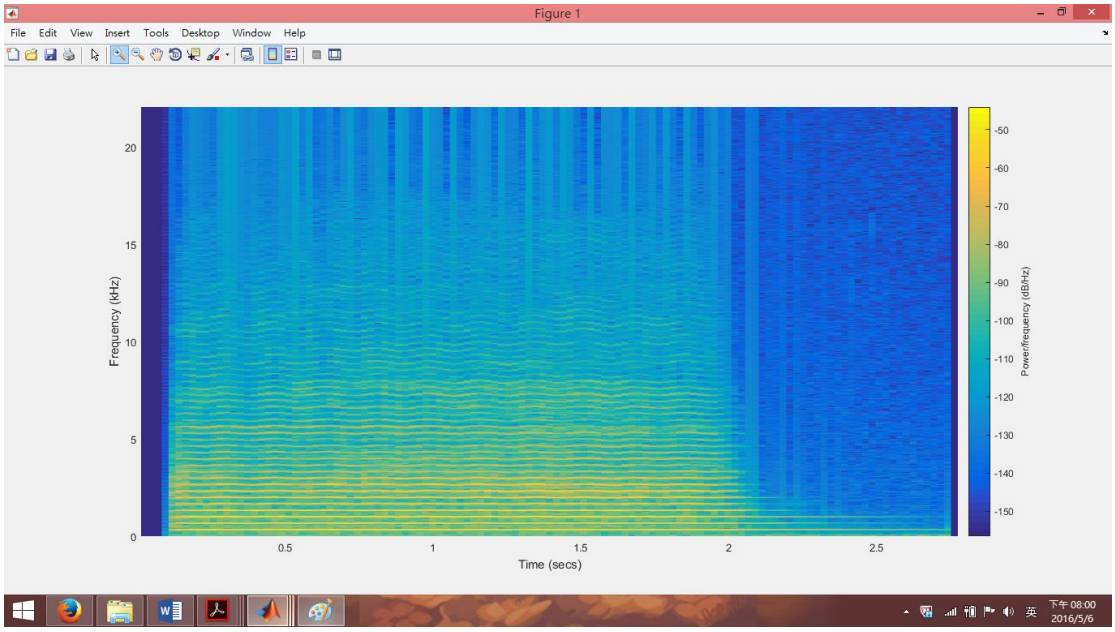


這是把小於 1200 頻率部分取掉之後的頻譜圖



這則是把大於 1200 頻率取掉之後的頻譜圖(注意刻度差異)

從上面我們大致可以看到，Pitch id = 64 對應的 330 為基頻以及其泛音所產生的頻譜圖。另外值得一提的是，我試著把 imagesc 函數中的尺度改成 log scale 之後再次把原音檔的 spectrogram 畫了一次，得到下圖的結果。可以看到在很多時間點的地方都有出現橫跨所有頻率的低能量雜訊。



至於三個音檔之間，直接聽起來的話可以感覺到 vio_64_hp 只有細微、高頻而且還有明顯快速震動的 vibrato 的聲音。vio_64_lp 聽起來則只剩下較低頻率的嗡嗡聲。然而兩者其實都已經聽不太出來是小提琴的聲音了(高頻的還多少比較可以聽出來)，可以猜測樂器的音色其實跟整個泛音的結構都有關係，刪除基

頻和附近的泛音或者是只保留基頻和附近的泛音都會讓音色出現很大的變化。

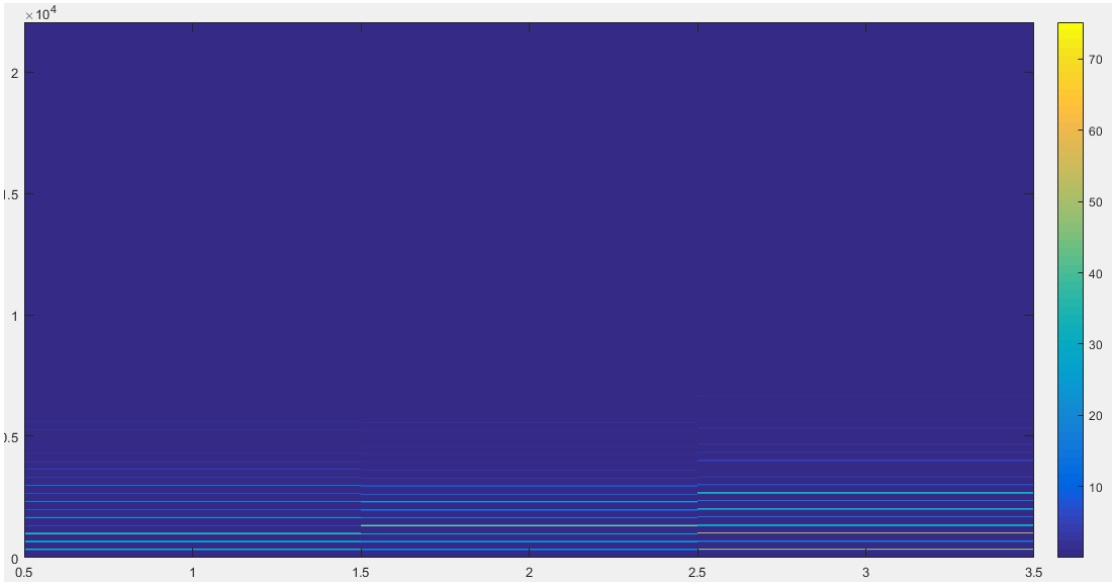
vio_64_hp 和 vio_64_lp 兩者和原本音檔之間的 SDR 分別為 30.6156、34.0651，可以得知兩者和原本音檔的相似程度差不多，然而我還測試了直接把原本音檔做 stft -> istft 後再跟原本音檔做比較，發現 SDR 僅有 73.9281，我認為這應該是在轉換過程中的誤差，導致得出的 SDR 不如第一題中我們直接把兩個相同音檔拿去算 SDR 所得到的值。

B3、

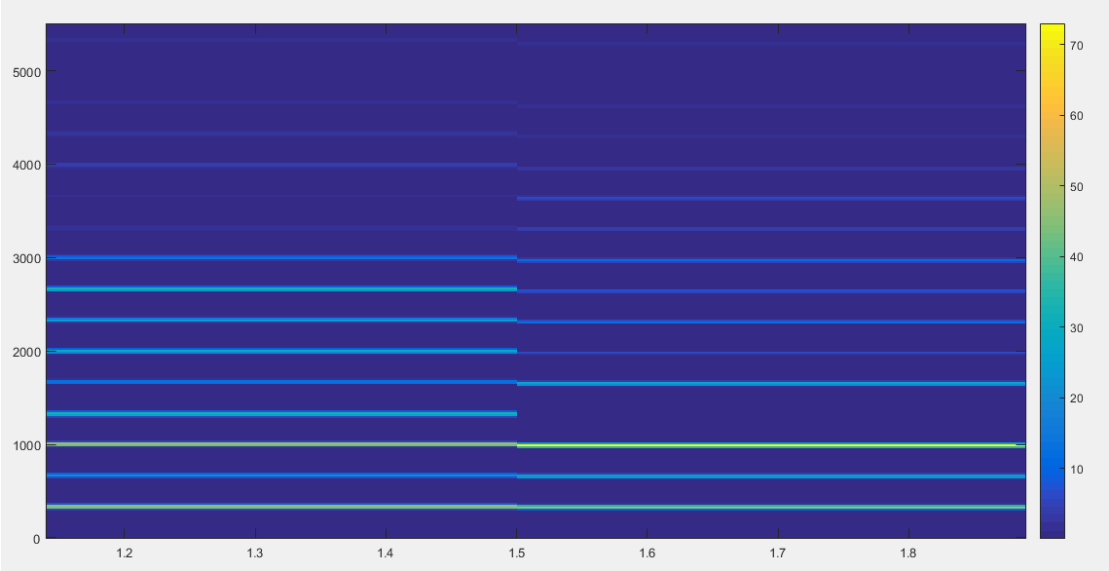
這題則是要我們實作 NMF 演算法然後試著去分解音檔 vio_64.wav。我的實作方法基本上就跟講義上寫到的 pseudo code 一樣。

```
function [ W , H ] = NMF( V , R , epsilon )
%NMF Summary of this function goes here
%   Detailed explanation goes here
 W = rand(size(V,1),R);H = rand(R,size(V,2)); WW = W; HH = H;
 deltaW = 99999999; deltaH = 99999999;delta = 99999999;a = 0;
 while deltaH>epsilon || deltaW>epsilon,
    W = WW;
    H = HH;
    HH = H.*(((W)'*V)./ ((W)'*(W*H+eps)));
    WW = W.*((V*(HH)')./((W*HH+eps)*(HH)'));
    deltaW = norm(WW - W);
    deltaH = norm(HH - H);
 end
    W = WW;
    H = HH;

end
```

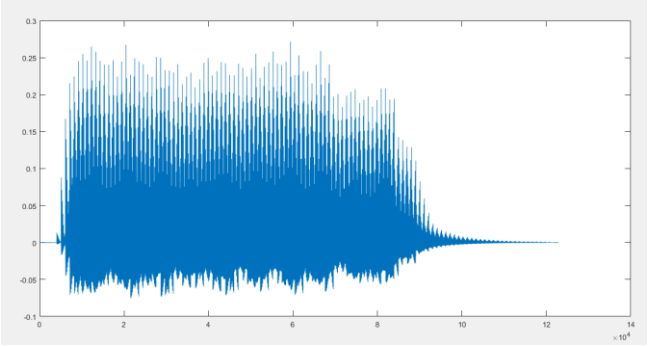


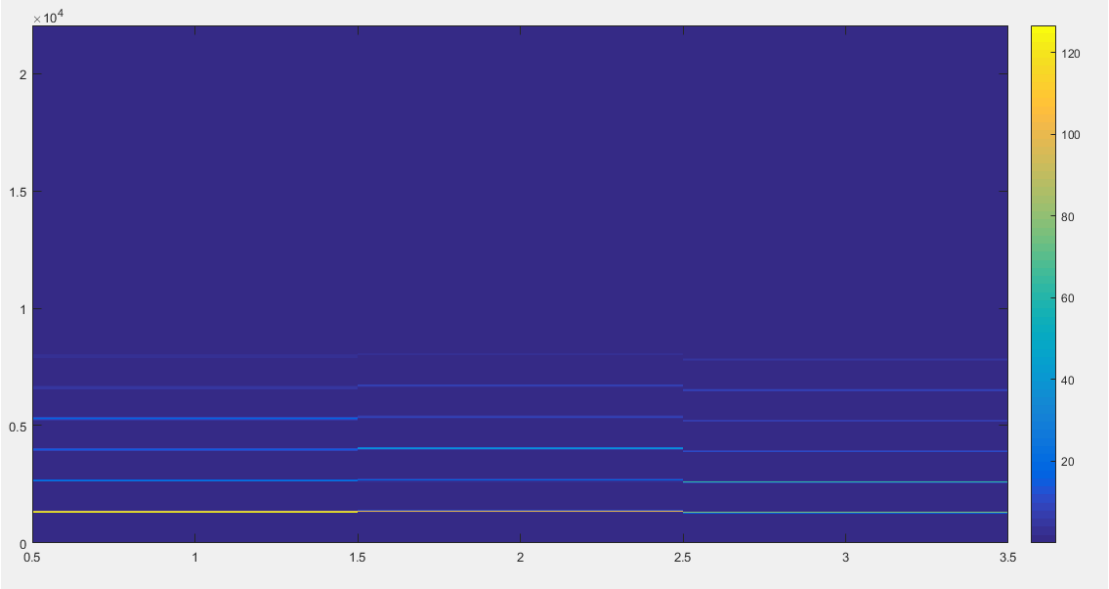
vio_64.wav 用 NMF 算出來的 3 個 template

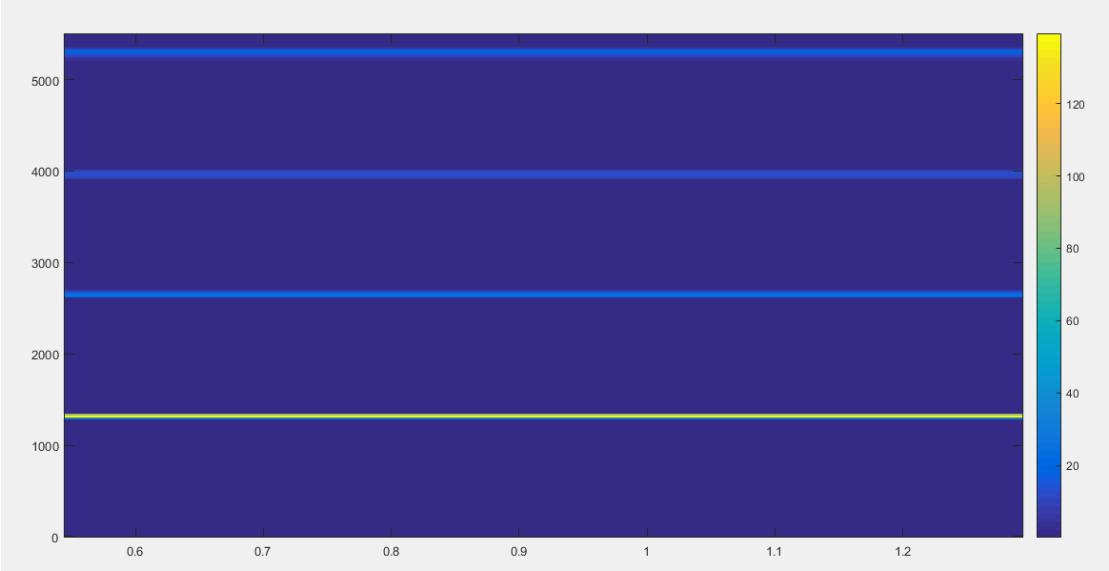


放大觀看

從上圖可以看出，其實算出來的 template 其實大致是有和原音檔吻合的(Pitch id = 64，基頻為 330 的音色)。不過三個 template 之間大致上雖然幾乎一樣但是還是有一些些微的差異。如果把重構後的音檔(W*H)用 ISTFT 轉回時域之後畫出來，大致上會像下圖。

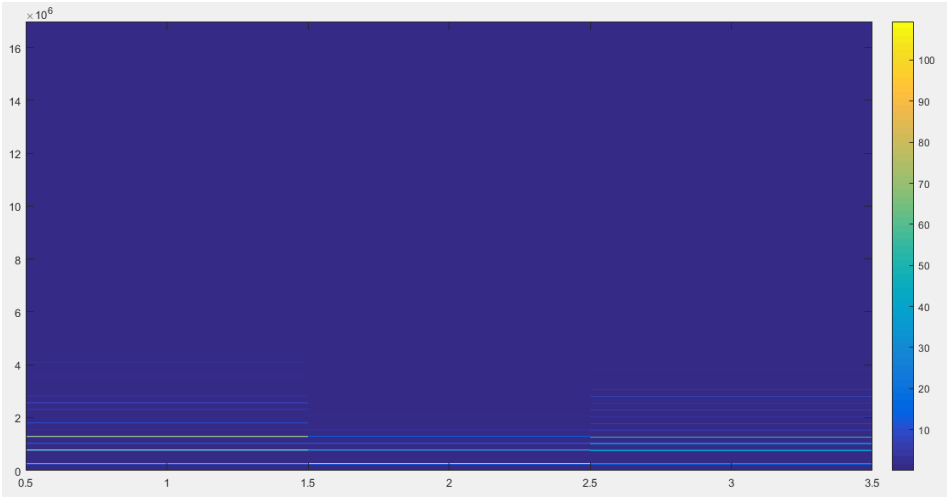


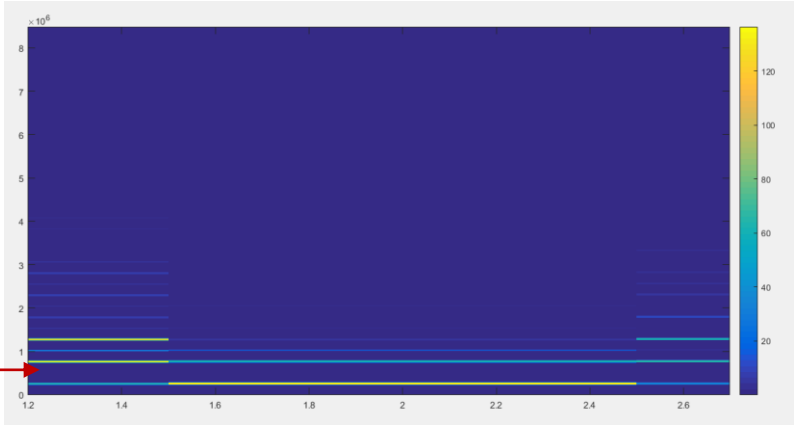




Vio_88.wav 做 NMF 分解的三個 template

不難看出，因為 pitch id =88 比 64 還要高上不少，因此基頻的位置自然比 vio_64.wav 的 template 還要高。





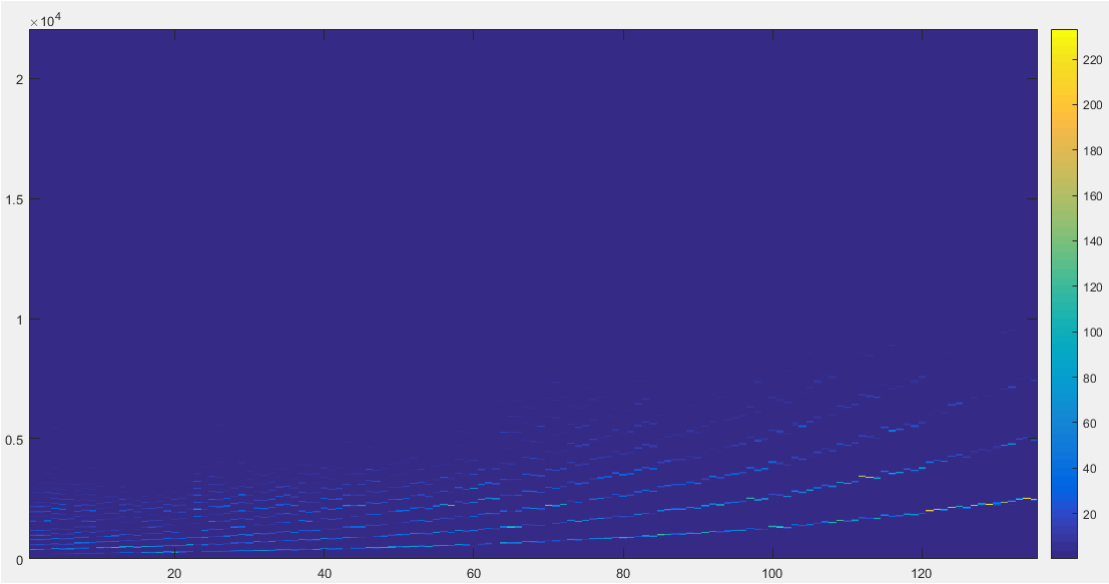
cla_64.wav 的 template

可以看出儘管同樣是 pitch id =64 的音檔，但是單簧管學出來的 template 卻跟小提琴學出來的頻譜不太一樣。很明顯，因為單簧管是 Open Tube 的樂器，

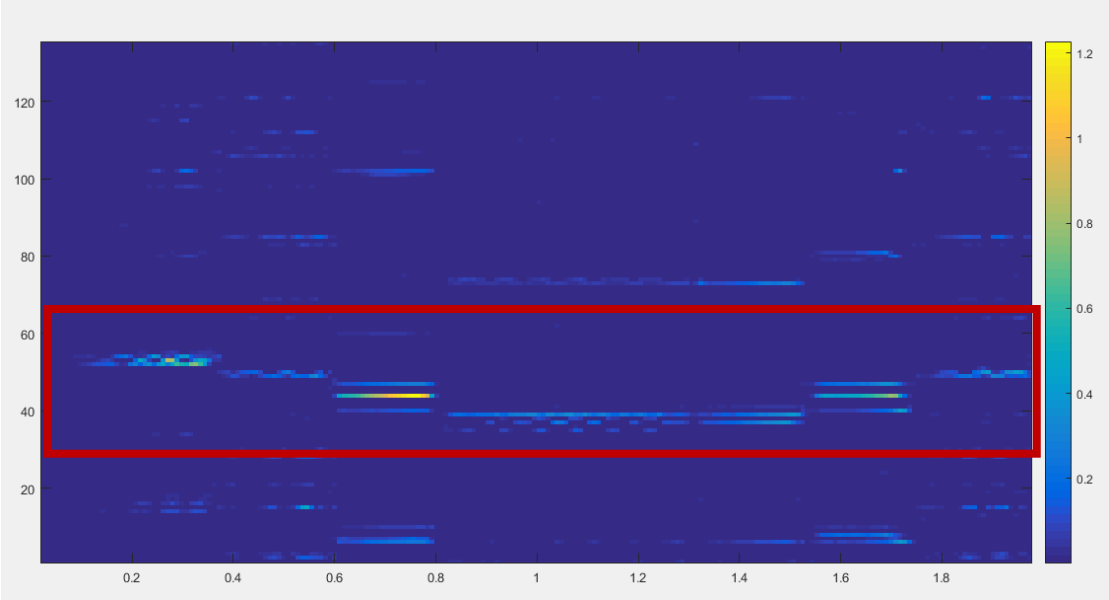
第 1、3、5…倍的泛音基本上是會消失(或者是很明顯較弱)在學出來的 template 之中的(如上圖箭頭所示)。

R1、

這題是教我們更改剛剛寫出來的 NMF 演算法去從小提琴所有 Note 的音檔中個別學出三個 template，總共 135 個 template 的 dictionary 出來，並且用這個 dictionary 去拆解 validation set 中的 01_vio.wav 這個檔案。看看固定 template matrix 時去算出的來的 activation matrix 是否可以讓兩者相乘有效的還原出音檔。因為這題必須要固定 W 只更改 H，所以稍微把 NMF 的演算法做了些修改。



小提琴的 Dictionary



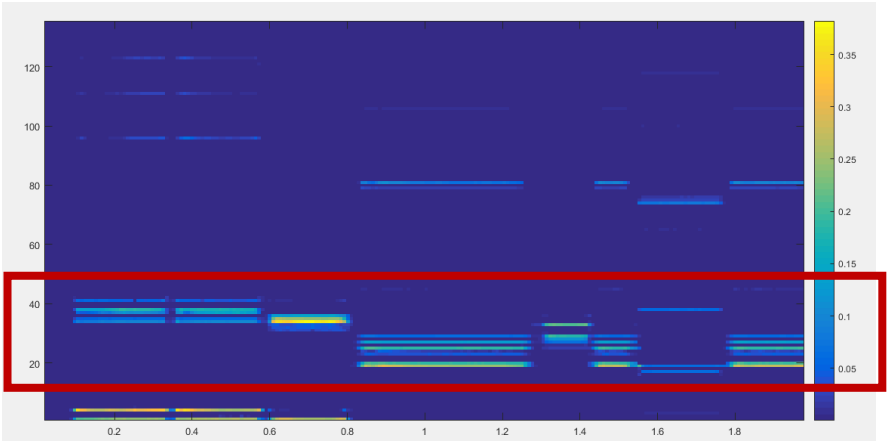
用上圖的 dictionary 去拆解 01_vio.wav 所算出來的 activation matrix

從 Dictionary 中可以看到，從左到右整體的 note 的頻譜逐漸升高，而且上升的方式是非線性的 Pitch Scale。而 Activation matrix 裡，紅線框起來的部分可以很明確的跟音檔中的旋律吻合。把兩個矩陣相乘，接著再把從原曲的頻譜鐘用 angle()函數提取出來的 phase 資訊加回去，接著做 ISTFT 即可得到用 template 和 activation matrix 重建出來的音檔。

重建出來的音檔，音色可以很明確地聽出來是小提琴的音色，旋律也可以聽得出來跟原本的音檔 01_vio.wav 是一樣的，只是會有雜音，而且是在每次有 Vibrato(抖音)的技巧的時候雜音會特別明顯。SDR 則是 9.7051。如果在做 ISTFT 之前不把 Phase 加回去的話效果會明顯地變差很多，雜訊量大幅增加，SDR 也變成-12.8497。

R2、

這一題則是讓我們試著使用上一題學出來的小提琴 Dictionary 去分解單簧管的音檔 01_cla.wav。以下是用小提琴的 Dictionary 當 W 用 NMF 算出來的 H。



重建出來的 activation matrix 如上。可以看到紅線框起來的部分其實是跟原音檔一樣的旋律，只是很奇妙的是全部的 note 的上下都多出了很多其他的 note 同時出現，導致原本只有一個單音的地方全部都變成了數個 note 同時出聲。

用 W*H 算出來 V 之後加回原本的 Phase 之後用 ISTFT 重建，可以得到用小提琴的 Template 去模擬出來的單簧管聲音。整個音檔從頭到尾都很不穩定，有很大的雜訊干擾(如同 activation matrix 所示，每個音出現時都是數個相近的 note 同時發聲)，不過仔細聽的話會覺得，其實背後好像有類似單簧管的聲音藏在裡面，只是有很強大的干擾。所以雖然這個音檔是用小提琴的 Dictionary 去重建出來的，但是聽起來音色卻是偏向單簧管的。

R3 & R4、

這題是要我們真正的去做 Source separation 的演算法了。首先照著之前學出小提琴 dictionary 的方式同樣去學出單簧管的 dictionary。接著個別用兩個 dictionary 去對 01_mix.wav 這個音檔做 NMF(也就是用兩個 Dictionary 去當作固定的 W 去算出 H)。接著把 W 和 H 相乘，接著用 Wiener Filter 去做 Masking，算出個別音色的 Spectrogram 之後把 Phase 加回去再做 ISTFT 就可以得到分離後的兩種樂器的音檔了。我用以下的方式控制 Wiener Filter 的 Soft/Hard 以及參數 c，以方便找出最適合的分離方法。

```
if isHard==1,
   mask = rVio > rCla ;
   A = absV .* mask;
   mask = rCla > rVio;
   B = absV .* mask;
else
    mask = (rVio.^c) ./ ( rVio.^c + rCla.^c);
    A = absV .* mask;
    mask = (rCla.^c) ./ ( rVio.^c + rCla.^c);
    B = absV .* mask;
end
```

經過試驗後，發現 Hard Mask 的效果比較好(分離出來的音檔跟原音檔的平均 SDR 值較高)，所以最後的 Wiener Mask 也是用 Hard Mask。C=1 或 C=2 則幾乎沒什麼差，於是我都把 C 設成 1。

至於 NMF 的部分，我發現學習 Dictionary 時使用 KL Divergence 當作 Cost function 的效果會稍微好於用 Euclid Distance 當作 Cost function。所以把 NMF 的 iteration 改成以下：

```
for i=1:MAXITER
% update activations
H = H .* (W'*( V./(W*H+eps))) ./ (W'*ONES);
% update dictionaries
W = W .* ((V./(W*H+eps))*H') ./(ONES*H');
end
```

NMF 分解時，每個 Note 的 template 數量的部分，我發現使用四個以上的 template 去解 NMF 的話會解不出來所以無法使用，使用兩個話效果差別不大，使用一個的話效果則會明顯變差，所以最後我還是保留每個 Note 學出來的 Template 數都是 3 個。

所以在 R3 跟 R4 中，我最後用的演算法都是如上所示。

R3 中，Validation set 中的音檔分離出來的兩個樂器分別對應的 SDR、SIR、SAR 如下所示(第一列是小提琴，第二列是單簧管)：

	1	2	3	4	5
1	9.7866	10.0541	6.3042	0.7886	8.1714
2	2.3696	8.5716	5.3582	-2.5808	8.0844

SDRs

	1	2	3	4	5
1	18.1661	14.8952	13.8234	9.7419	13.4990
2	2.7611	9.8695	6.0340	-1.9329	9.0051

SIRs

2x5 double

	1	2	3	4	5
1	10.5339	11.9189	7.3268	1.8177	9.8686
2	14.8599	14.8758	14.7374	10.0854	15.7879

SARs

可以看到，在 SDR(分離出來的跟原音檔有多像)的部分很明顯小提琴的表現優於單簧管的。也可以注意到第四首是最難分離的(聽了一下音檔我覺得應該是因為音檔的前半兩種音色的 Pitch 幾乎重疊了)。SIR(分離音檔跟其他音源有多像)的部分也是小提琴比較高。SAR(跟不是任何待分離音源的相似度)則是單簧管比較高。綜合以上觀察，可以很明顯地得出：小提琴的分離效果是優於單簧管的這樣的結論。

Test Set 的部分，因為沒有分離好的音檔的關係當然就沒有辦法計算 SDR 值，就留待老師幫忙計算囉。

A1、

我還額外做了進階題的第一題。這題是要我們在有樂譜資訊的情況下去初始化 Activation matrix，看看最後用這樣初始值去 iterate 出來的 H 的效果如何。

```
onset= [];
offset = [];
kind = [];
pitch = [];
fid = fopen(strcat('score-info/0',num2str(x),'.txt'));
while ~feof(fid),
     d = fscanf(fid,'%d',1);
     onset = [onset d];

     d = fscanf(fid,'%d',1);
     if d>5000,
        d = 5000;
     end
     offset = [offset d];

     d = fscanf(fid,'%d',1);
     pitch = [pitch d];
     d = fscanf(fid,'%d',1);
     kind = [kind d];
end
```

首先用以上的 code 把 txt 檔中的樂譜資訊讀入。

```
l = size(absV,2);
initVioH = zeros(R * 45,1);
initClaH = zeros(R * 40,1);
for y = 1:length(onset),
     if kind(y)==1,
      for xx = 1:R,
        initVioH(R*(pitch(y)-55)+xx, floor(onset(y)/5000 * l) : floor(offset(y)/5000 * l)) = 1;
      end
     else
       for xx=1:R,
        initClaH(R*(pitch(y)-50)+xx, floor(onset(y)/5000 * l) : floor(offset(y)/5000 * l)) = 1;
       end
   end
end
```

接著依照原本頻譜時間軸的長度把樂譜的時間長度和頻譜的 x 軸長度統一，並把介於 Onset 跟 Offset 時間內的 initial H 設成 1。

剩下的部分就跟前面 R3 的部分一樣了。Score informed NMF 的結果(SDR、SIR、SAR)如下：

2x5 double

	1	2	3	4	5
1	14.6089	14.4395	12.0687	11.3739	17.6409
2	11.3870	13.8570	12.0797	5.6719	16.9880

SDR

2x5 double

	1	2	3	4	5
1	24.9634	20.9009	19.2236	16.9531	29.7422
2	14.1628	16.1881	14.3054	7.9691	18.6099

SIR

2x5 double

	1	2	3	4	5
1	15.0426	15.5865	13.0491	12.8678	17.9218
2	14.8086	17.7760	16.2064	10.1789	22.1107

SAR

可以看到，整體的表現明顯都變好了許多。聽了一下音檔也可以發現，分離的效果真的得到了大幅的提升，除了少許的雜訊外幾乎是分離的很乾淨，可見做 NMF 時矩陣的初始值對整體效果的影響之大。