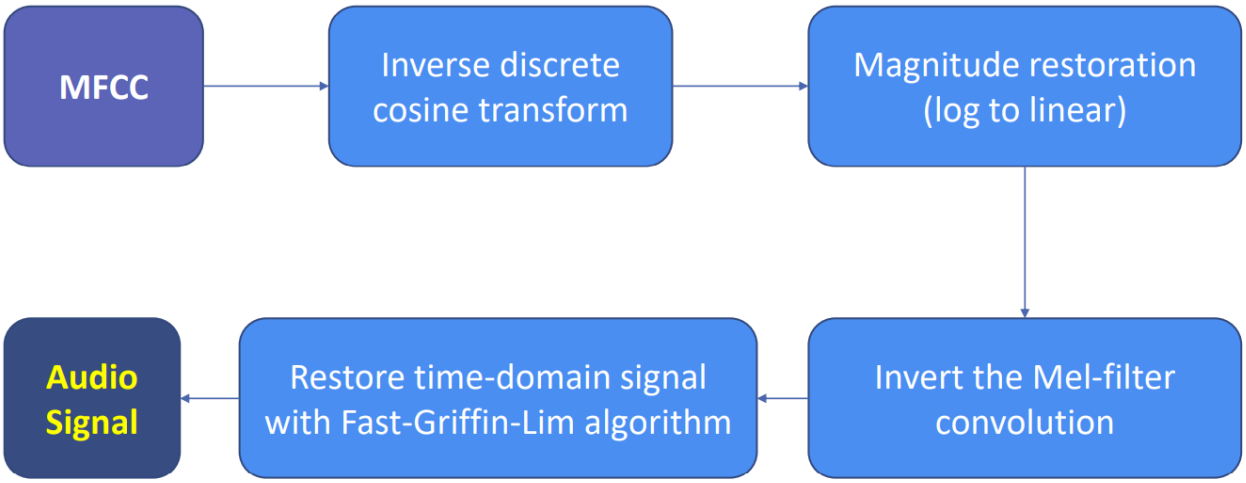
Lab2 109061217 林峻霆

● Design Specification

由前一個 Lab 提取出的 MFCC 來時做一個 inverse system，還原回語音訊號，並與原先的語音訊號進行比較。

● Design Implementation

這個部份我會著重於講解各個部分是如何實作以及他們背後各自的原理。頻譜圖以及問題討論的部分會放到下個部分(Discussion)：



取自助教提供的講義

要將訊號還原的基本想法就是將上一個 Lab 的步驟全部反做回去，但我們會發現某些步驟是不可逆的，例如：乘上 mel-filter bank(降維)、對頻譜取 absolute value。在這些步驟時，我們就得去尋找一個最佳近似的解，來當作我們還原的結果。

1. Inverse DCT and Magnitude restoration

這部分都是可逆的，基本上就呼叫 python 的 function 就能達成。

2. Invert the Mel-filter convolution (Least-Square Problem)

Mel-Filter 是不可逆的，因為它是一個降維的步驟。我們要解決的問題如下：

$$Ax = b, \quad A = melfilter, x = spectrogram, b = feature$$

我們希望能透過 feature 和 mel-filter 找回 spectrogram。要注意的是，A 不是一個方陣，所以我們不能直接計算 A 的反矩陣。這成為一個 least-square problem，我們希望能找到一個 $\hat{x}$，使得 $J(x) = (A\hat{x} - b)^T(A\hat{x} - b)$ 有最小值。透過微分，我們得到 $\hat{x} = (A^TA)^{-1}A^Tb = A^{\dagger}b = A^T(AA^T)^{-1}b$，我們就稱 $A^{\dagger}$ 為 A 的 pseudo inverse。透過

pseudo inverse，我們就能找回最佳近似的$\hat{x}$。

3. **Restore time-signal (Phase Recovery)**

這部分要解決的問題在於 phase。在找 MFCC 時，我們對 spectrogram 取 absolute value，也就是他的 magnitude，把 phase 給捨去了。這事實上是一個凸優化的問題(前面的 least-square problem 也是)，我們這邊採用 Griffin-Lim Algorithm(GLA)，其步驟如下：

Algorithm 1 Griffin-Lim algorithm (GLA)

Fix the initial phase $\angle c_0$
Initialize $c_0 = s \cdot e^{i\angle c_0}$
Iterate for $n = 1, 2, \ldots$
$c_n = P_{\mathcal{C}_1}\left(P_{\mathcal{C}_2}(c_{n-1})\right)$
Until convergence
$x^* = \mathbf{G}^{\dagger} c_n$

取自＂ The Fast Griffin-Lim Algorithm"

其中的 Pc1 以及 Pc2 分別代表：

$$P_{c1}(c) = GG^{\dagger}c \quad where\ G^{\dagger}\ is\ ISTFT\ and\ G\ is\ STFT$$
$$P_{c2}(c) = Se^{i\angle c_0} \quad where\ S\ is\ the\ spectrogram$$

Griffin-Lim Algorithm 透過多次迭代來來逐步縮小與原先訊號的 phase 差異，來達成還原的效果。而 Fast Griffin-Lim 的想法就在於透過調整 step size 來加快 convergence。

Algorithm 2 Fast Griffin-Lim algorithm (FGLA)

Fix the initial phase $\angle c_0$
Initialize $c_0 = s \cdot e^{i\angle c_0}$, $t_0 = P_{\mathcal{C}_2}\left(P_{\mathcal{C}_1}(c_0)\right)$
Iterate for $n = 1, 2, \ldots$
$t_n = P_{\mathcal{C}_1}\left(P_{\mathcal{C}_2}(c_{n-1})\right)$
$c_n = t_n + \alpha_n(t_n - t_{n-1})$
Update α_n
Until convergence
$x^* = \mathbf{G}^{\dagger} c_n$

α的值介於 0 和 1 之間，通常是視 convergence 的程度作 adaptive 調整，但是在這次

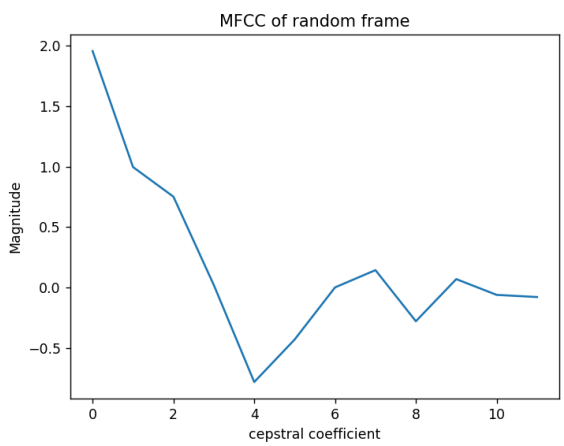
的 Lab 中我們都固定為 0.99。

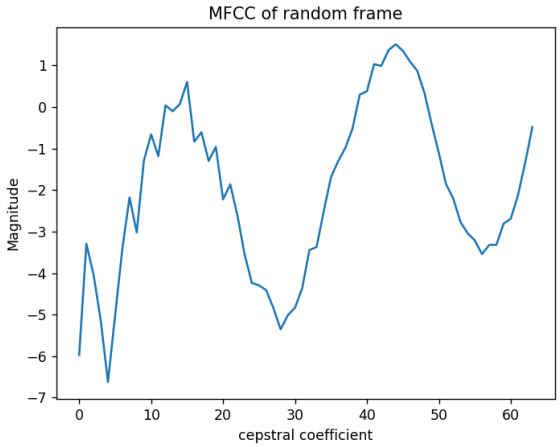
- **Discussion**

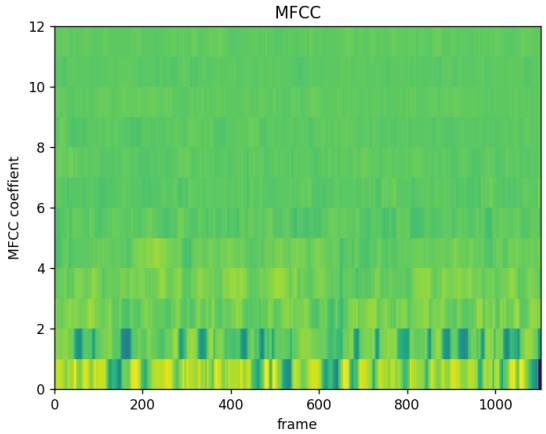
這部分我們就來比較我們得到的一些結果(頻譜圖、音檔)。

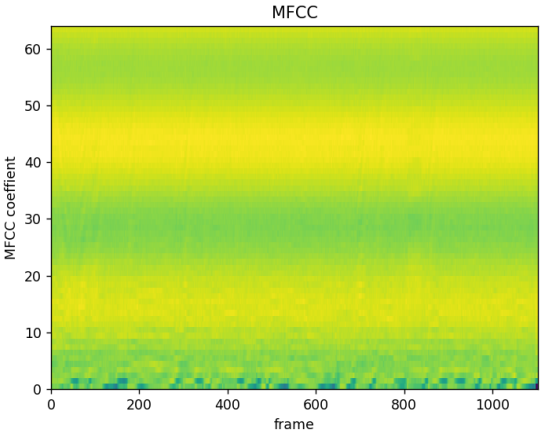
1. **Different Number of Mel-filter bank**

以下我們比較 12 個 filter bank 以及 64 個 filter bank 所產生的差異。左邊是 12 個 filter bank，右邊是 64 個 filter bank。





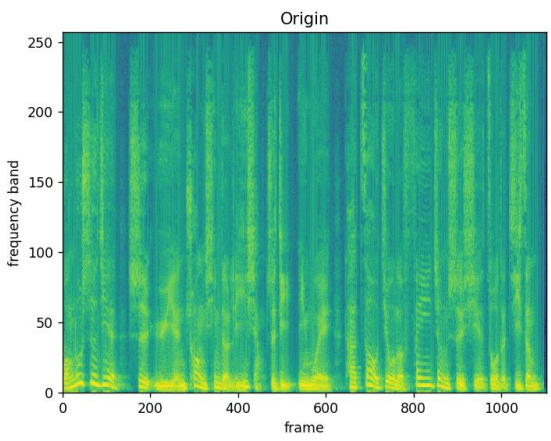


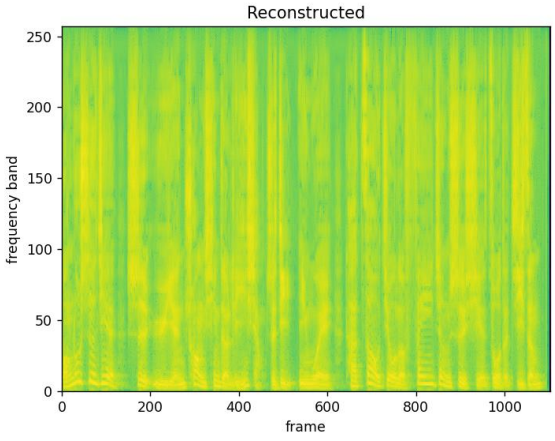


可以發現當 filter bank 的數量提高時，quefrency resolution 會變高，原先的 MFCC 在第 6 個 coefficient 後(較高頻率)幾乎沒有變化，但改成 64 個 filter bank 後我們就能看出高頻部分的變化。原因在於我們將(freq_min , freq_high)切得更細了，高頻處的 filter 更多，更容易區分高頻處的特性(因為 mel-scale nonlinear 的特性，低頻出 filter 多、高頻處 filter 少，當 filter bank 數量少時，高頻的 filter bank 容易橫跨大範圍頻率，導致這部分的 quefrency resolution 不夠高)

2. Original v.s Reconstruct

以下是原始語音訊號與經過 reconstruct 的語音訊號的 spectrogram 比較：





可以看出兩者的差異在於：reconstructed 之後的語音訊號的高頻振幅較高，從音檔中也可以聽得出聲音比較尖。若我們進一步去比較其他的 case：1. 12 or 64 filter bank 2. 有無 pre-emphasis，我們可以發現，當 filter bank 較少時 reconstruct 的語音會聽起來比較模糊，而且聲音比較低沉，另一方面，有經過 pre-emphasis 的聽起來音調較高，這是因為 pre-emphasis 是一個 high pass 的原因。

接著我們討論一些實作上的問題：

1. **Effect of each step in Inverse MFCC**

   在 MFCC 中，convolve with mel-filter(降維)以及做 STFT 時取 absolute value(magnitude，捨去 phase)這兩個步驟是不可逆的。因此，我們在 inverse MFCC 中的 pseudo inverse 以及 GLA 就是透過迭代、最佳化近似的方式去人造出一個最符合原先訊號的語音訊號。然而，這樣的方式注定沒辦法完全恢復回原先訊號(從音檔、頻譜就能看出)，原因在於：1. pseudo inverse 生出的 magnitude 只是 least-square 的解，跟原先的 magnitude 不同 2. Phase 透過 GLA 迭代產生，但我們不可能做無限次迭代，再加上迭代時用的 magnitude 是來自 pseudo inverse 得到的 magnitude，本來就不適原先的 magnitude。

2. **Effect of frame_length, frame_step and # of filter bank**

   我們先說各自影響什麼：frame_length → # of FFT point, time period of a frame
   frame_step → # of frame, difference between two frame
   # of filter bank → # of MFCC, frequency resolution

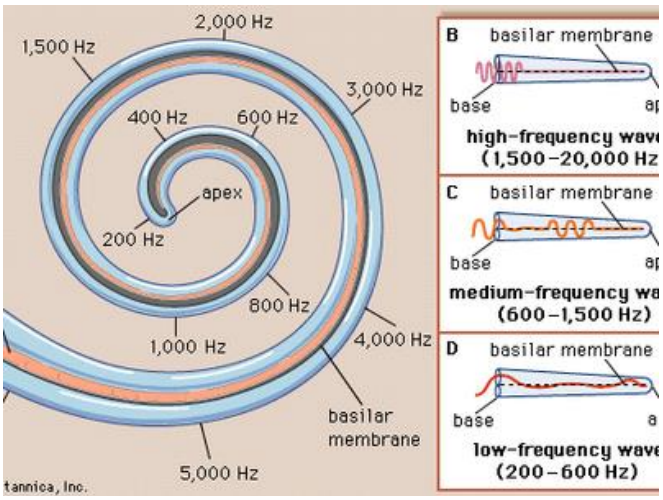
   首先是 frame_length，他影響兩個部分：首先是# of FFT quantization point，frame length 越大，FFT 取的點越多，就越接近理想的 DTFT，就能到越完整的頻譜，但另一方面，我們在做 inverse MFCC 時，也會造成維度差異更大，導致 pseudo inverse 的結果與原始值差異變大，因此我認為 frame length 與# of filter bank 是需要一起做調整的。另一個影響的點是一個 frame 的 time period，做 STFT 就是為了看出頻譜對時間的變化，而 frame length 拉大就表示我們拿來做 STFT 的音訊片段變長，訊號的瞬間變化就會被稀釋，導致 spectrogram 變化程度下降。

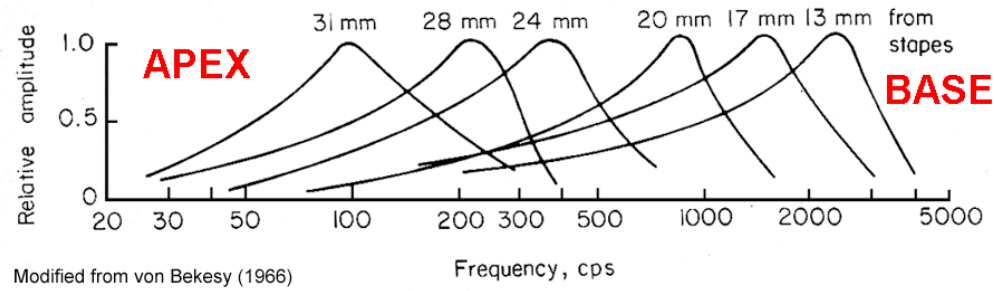
   接著是 frame_step，frame_step 決定前後 frame 的差異有多大，frame step 越大，前後 frame 差異越大，這會使各個 frame 經還原後比較不具有連續性，容易出現爆裂音，尤其是這個 Lab 當我們把 frame_step 調至 512 時，爆裂音幾乎掩蓋我們的原先語音。但 frame step 也不是越小越好，除了人耳辨識度沒有這麼高，運算量也會大幅提升。

最後是#of filter bank，這個從我們前面的 12 v.s 64 MFCC 圖就可以看出，filter bank 的數量提升能提高 quefrency resolution，當# of filter bank 越高，還原的訊號會越接近原先的訊號(因為維度更高，MFCC 保存的資訊越多)，除了 Lab 中的兩種，我還嘗試 32、128、256、512 這四組。從 spectrogram 來看，32 以上的差別不大；從音檔來聽，256 以及 512 兩者幾乎跟原訊號一模一樣，128 以下越小雜音越多。我認為這些差異就在於 MFCC 維度不同，維度越低我們能保存的資訊就越少，在解 least-square problem 時就與實際值差異越大。

3. **Why not use rectangular filter for energy calculation ?**

   方形的 filter 代表對所有頻率的 frequency response 相同，但人耳對各種 pure tone 的 frequency response 並非相同，而是如下圖：





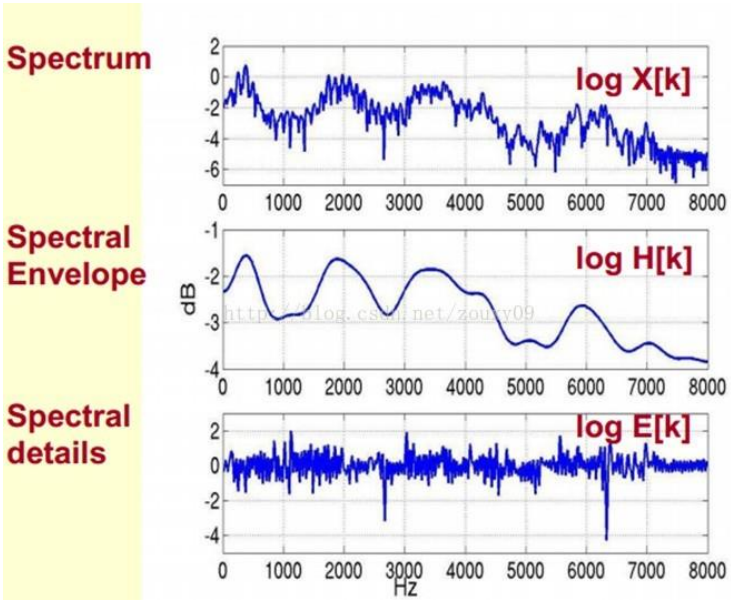
   因此這種 triangular filter 的設計，模仿人耳的基底膜各部位的 spectrum 會比較符合人耳聽到的實際狀況。

4. **Why should Mel-filter be overlapping ?**

   這個問題從第三題的圖就可以看出，每個基底膜各部分的 spectrum 本身就有重疊，因此這也是一個仿生的設計，讓提取 MFCC 的系統更貼近人耳

5. **Why are high-quefrency MFCCs usually abandoned when doing speech recognition ?**

當我們在做 speech recognition 時，我們希望獲取的語音資訊藏在訊號的 envelope(共振峰所連成的曲線)。這部分的資訊主要集中在低頻的部分，因此我們通常只取前 13 個 MFCC 來做語音辨識，後半部的 MFCC 所帶有的資訊是頻譜的細節(音調音色變化)，捨去不太會影響語音辨識的結果。



6. **Is MFCC good for speaker identification ?**

   雖然尚未進行 Lab 3 和 Lab 4，但從網路上的資訊來看，MFCC 應該是一個相當不錯的語音辨識資訊。而從我個人的觀點，MFCC 能用少量的參數來表示一個語音訊號的 envelope，而語音資訊大多藏在 envelope 中，因此應該是一個有效且能降低辨識時運算複雜度的好工具。

7. **If two sounds have similar MFCC, does that means they sound similar to our ears ?**

   我認為是相似的，但並不一定會完全一樣。以這次的 Lab 為例，我嘗試加一些程式碼，我在所有步驟前先做一次 STFT，但這次不取 absolute value，然後對每個 frequency component 加上 random phase，然後做 Inverse STFT 並將這個 modified signal 寫成一個音檔。依據 MFCC 的步驟，這個 signal 跟原本 signal 的 MFCC 會完全一樣(因為我們會捨去 phase information)，但實際聽起來兩者會有些許的差異 modified signal 聽起來較混濁、有雜音但大致上還是聽得懂內容是什麼。另一方面，從 MFCC 降維的角度來思考，或許會存在另一組音訊他的 magnitude 跟我們用的音訊的 magnitude 有差異，但在降維後兩者 MFCC 差異不大，不過這部分我尚未實作或想出一組這樣的訊號。

8. **Are there any other things we can implement to make the reconstruction better ?**

   我目前想到的只有一個：如果我們在一開始有經過 pre_emphasis，那我們可以在最後

多一個 filter，讓他跟一開始的 pre_emphasis 在頻域相乘為 1。而他的實作如下：

```
def de_emphasis(signal, coefficient = 0.95):
    length = len(signal)
    result = np.zeros(length)
    result[0] = signal[0]
    for i in range(1, length):
        result[i] = coefficient * result[i - 1] + signal[i]

    return result
```

基本上就是把 pre_emphasis 反著做，這樣就能抵消 pre_emphasis 的 high pass 特性，讓 reconstruct 的語音訊號更接近原先的語音訊號。

9. **Dimension reduction/reconstruction in the DCT/inv-DCT**

通常我們只取前 12~14 個 MFCC，捨棄後面的 MFCC。所以我們在 reconstruct 時也只能使用這些 MFCC。因此我們要修改程式碼的以下部分：

```
num_MFCC = 20  # number of MFCC we take to do speech recognition
MFCC = dct(features, norm='ortho')[:, :num_MFCC]    # DCT with data compression
inv_DCT = idct(MFCC, n = num_bands, norm = 'ortho')   # iDCT with data decompression
```

我們一樣拿經過 triangular filter 的所有點(= number of filter bank)來做 DCT，但我們只取前面某幾個(ex. 12~14 個)，接著我們可能拿這些 MFCC 去做 speech recognition，並將得到的成果拿回來做 inverse MFCC，其中的 inverse DCT 的輸出點數量需要等於 number of filter bank，這時我們就需要做 zero padding 再做 inverse DCT。經過實驗後，只取 12~14 個 MFCC 做 reconstruct 的訊號仍然聽得出來講者說話的內容，但音調音色等特徵幾乎都被消除了。

◆ Conclusion

這個 Lab 將前一個 Lab 所提取的 MFCC 嘗試還原回原先的語音訊號。過程中我們需要克服一些不可逆的步驟，使用一些迭代(GLA)或最佳化近似(Pseudo Inverse)的演算法，最後人造出一個接近原先語音訊號的訊號。實作過程中我學會如何用 python 實現上述的演算法，並比較調整實驗參數各結果之間的異同。此外，這個 Lab 也讓我感受到線性代數的威力，不論是 least-square problem 或是後面的 phase recovery 中的數學都讓我大開眼界

◆ References

- 謝宗翰的隨筆：[矩陣分析] 擬反矩陣(Pseudo Inverse Matrix)
  https://ch-hsieh.blogspot.com/2015/07/pseudo-inverse-matrix.html
  理解什麼是一個矩陣的pseudo inverse，以及其特性

- 維基百科：Griffin-Lim Algorithm
- Nathanaël Perraudin , Peter Balazs and Peter L. Søndergaard, “A FAST GRIFFIN-LIM ALGORITHM”, IEEE Workshop on Applications of Signal Processing to Audio and Acoustics 2013

  理解fast griffin-lim algorithm背後的數學原理以及如何執行演算法。
- DSP Stack Exchange

  https://dsp.stackexchange.com/
- 梅爾頻率倒譜系數（MFCC）學習筆記

  https://www.796t.com/content/1548626964.html
- 教授與助教的講義