1-1：About This Book (有關本書)

本書內容的重點如下：

- 介紹音訊處理與辨識（Audio Signal Processing and Recognition）的基本原理。
- 說明如何以 MATLAB 進行音訊處理與辨識的程式碼實作。
- 以實際生活中的資料來說明音訊處理與辨識的各種相關應用。

筆者本著寫書著作的一貫宗旨，期望本書能達到下列終極目標：

- 範例式的教學：以簡單的範例來說明基本概念，然後再輔以正式的數學分析與推導。
- 理論與實作並重：所有的演算法都附有 MATLAB 的程式碼，讓使用者能夠穩紮穩打、Learning by Doing。
- 應用導向：所有的範例、理論與程式碼，最後都會用在現實世界中的應用，以讓讀者親自感受到各種演算法的長處和短處，以及程式碼實作方面可能遇到的困難。

當然，本書無法包山包海，所介紹的方法，都是比較偏向音訊處理與辨識的基本面，對一般碩士班研究生而言，應該已經夠用，但對於博士班研究生而言，可能尚嫌不足。不過無論讀者的背景為何，本書都可以做為音訊處理與辨識的入門書籍，並適合用於大學和研究所的相關課程。

當然，本書還是需要一些數學背景，才比較容易吸收，因此若做為課程教科書，修課同學應該先修過下列課程：基本微積分、線性代數、機率。

如果你在研究論文內要引用此書，可採用下列兩種寫法：

- *Jyh-Shing Roger Jang, "Audio Signal Processing and Recognition," (in Chinese) available at the links for on-line courses at the author's homepage at http://www.cs.nthu.edu.tw/~jang.*
- *張智星，"音訊處理與辨識"，網路線上課程，可由作者之網頁 http://www.cs.nthu.edu.tw/~jang 連結到此線上課程。*

1-2：Example Programs (如何取得程式碼)

若要親自體驗本書之程式碼，需下載數個由筆者撰寫之相關工具箱，如下：

1. Utility Toolbox
2. DCPR Toolbox
3. Audio Procesing Toolbox
4. ASR Toolbox
5. Melody Recognition Toolbox
6. 本書的範例程式碼

下載這些檔案後，請展開目錄並將加入 MATLAB 的搜尋路徑中，就可以開始執行本書的範例程式碼。（你可以直接修改範例程式集中的 addMyPath.m，加入上述兩個 Toolbox 的目錄，以後直接執行 addMyPath.m 即可。）

由於本書的程式碼會持續發展與修改，因此若發現程式碼有無法執行之現象，可採取下列步驟：

1. 重新下載所有的程式碼，再試一次。
2. 若仍然不行，請告知筆者，筆者將進行查核及修正。

若發覺程式碼有任何錯誤，也歡迎隨時來信告知。

本書之程式碼僅供學習，並不保證完全正確，若因使用此程式碼所導致的任何營利損失或個人傷害，本人無法負責，請見諒！

最後，希望這些程式碼能為各位讀者帶來學習的樂趣！

1-3：Web Resources (網路資源)

在網際網路上面，有關於「音訊處理與辨識」的資源相當多，也可以找到很多相關的程式碼。我們在這裡介紹幾個常用到的重要網址，以便讀者能夠隨時上來查看最新的資訊。

- http://www.phys.unsw.edu.au/~jw/dB.html
  對於音量單位「分貝」（Decibels）的詳細介紹。

- http://www.phys.unsw.edu.au/~jw/hearing.html
  對於「等主觀音量曲線」（Curves of Equal Loudness）的詳細介紹。
- http://www.phys.unsw.edu.au/music/
  「音樂音響學」（Music Acoustics）的首頁。
- http://www.phys.unsw.edu.au/~jw/musFAQ.html
  「音樂音響學」（Music Acoustics）的 FAQ。
- http://www.wotsit.org
  各種檔案格式的介紹，包含音訊和音樂。
- http://www.speech.cs.cmu.edu/comp.speech/index.html
  Comp.Speech 新聞群組的的 FAQ。
- http://www.bdti.com/faq/dsp_faq.htm
  Comp.DSP 新聞群組的的 FAQ。
- http://www.harmony-central.com/Effects/effects-explained.html
  解釋如何產生各種音效，例如迴音、歌劇院的感覺等。
- http://neural.cs.nthu.edu.tw/jang/research/web/speech
  以前我整理的資料，不過已經有點舊了。

第 2 章作業

- (*) **When does n! explodes**： Write a MATLAB script findN01.m to find the minimal value of n such that n!>realmax. What is the value of n? What is the value of (n-1)! ?
- (*) **Basic function for both scalar and vector inputs**： Write a MATLAB function myFun01.m to evaluate the following expression:

  0.5*exp(x/3)-x*x*sin(x)

  where x is the input and y is the output of the function. You function should be able to handle both the situations when x is a scalar or a vector. Please use the following code segment to plot the expression:

  x=0:0.1:10; plot(x, myFun01(x));
- (**) **Function for computing the Fibonacci number**：
  - Write a recursive function fibo.m to compute the Fibonacci sequence, which is defined as follows:

    fibo(n+2)=fibo(n+1)+fibo(n)

    The initial conditions are fibo(1)=0, fibo(2)=1.
  - What is the value returned by fibo(25)?
  - Please use the commands tic and toc to measure the computation time of fibo(25). If you don't know how to use tic and toc, please try "help tic" and "help toc" in MATLAB command window. Please state the specs of your computer and the computation time of fibo(25).
  - If you have taken Discrete Mathematics, you should already known that the n-th term of a Fibonacci sequence can be expressed as

    fibo(n)={[(1+a)/2]^(n-1)-[(1-a)/2]^(n-1)}/a

    where a is the square root of 5. Write a non-recursive function fibo2.m to compute Fibonacci sequence according to the above expression.
  - What is the value of fibo2(25)? Is it the same as fibo(25)?
  - Compare the computation time of fibo2(25) and fibo(25).
  - Please list the advantages and disadvantages of fibo.m and fibo2.m.
- (**) **Find the minimum of a matrix**：
  - Write a MATLAB function minxy.m which can find the minimal element in a 2-dimensional matrix:

[minValue, minIndex] = minxy(matrix)

where matrix is a 2-d matrix, minValue is the minimal element of the input matrix, and minIndex is an integer vector of length 2 indicating the indices of the minimal element. (In other words, matrix(minIndex(1), minIndex(2)) is equal to minValue.)

- Test you program by typing the following statement in MATLAB command window:

  [minValue, minIndex]=minxy(magic(20))

  What are the returned values of minValue and minIndex?

- (***) **Function for ranking**： We can use a vector to store the scores of midterm exam. Please write a function ranking01.m that takes the score vector and return the ranking. For instance, if x = [92, 95, 58, 75, 69, 82], the vector returned by ranking01(x) should be [2, 1, 6, 4, 5, 3], indicating 92 is ranked second, 95 is ranked first, etc. (If possible, please try vectorized code instead of for/while loops.)

Chapter 3: Introduction to Audio Signals (音訊的簡介)

Audio signals are generally referred to as signals that are audible to humans. Audio signals usually come from a sound source which vibrates in the audible frequency range. The vibrations push the air to form pressure waves that travels at about 340 meters per second. Our inner ears can receive these pressure signals and send them to our brain for further recognition.

所謂「聲音訊號」（Audio Signals）簡稱「音訊」，泛指由人耳聽到的各種聲音的訊號。一般來說，發音體會產生震動，此震動會對空氣產生壓縮與伸張的效果，形成聲波，以每秒大約 340 公尺的速度在空氣中傳播，當此聲波傳遞到人耳，耳膜會感覺到一伸一壓的壓力訊號，內耳神經再將此訊號傳遞到大腦，並由大腦解析與判讀，來分辨此訊號的意義。

There are numerous ways to classify audio signals. If we consider the source of audio signals, we can classify them into two categories:

- Sounds from animals: Such as human voices, dog's barking, cat's mewing, frog's croaking. (In particular, Bioacoustics is a cross-disciplinary science, which investigates sound production and reception in animals.)
- Sounds from non-animals: Sounds from car engines, thunder, door slamming, music instruments, etc.

音訊可以有很多不同的分類方式，例如，若以發音的來源，可以大概分類如下：

- 生物音：人聲、狗聲、貓聲等。
- 非生物音：引擎聲、關門聲、打雷聲、樂器聲等。

If we consider repeated patterns within audio signals, we can classify them into another two categories:

- Quasi-periodic sound: The waveforms consist of similar repeated patterns such that we can perceive the pitch. Examples of such sounds include monophonical playback of most music instruments (such as pianos, violins, guitars, etc) and human's singing/speaking.
- Aperiodic sound: The waveforms do not consists of obvious repeated patterns so we cannot perceive a stable pitch. Examples of such sounds include thunder pounding, hand clapping, unvoiced part in a human's utterance, and so on.

若以訊號的規律性，又可以分類如下：

- 準週期音：波形具有規律性，可以看出週期的重複性，人耳可以感覺其穩定音高的存在，例如單音絃樂器、人聲清唱等。
- 非週期音：波形不具規律性，看不出明顯的週期，人耳無法感覺出穩定音高的存在，例如打雷聲、拍手聲、敲鑼打鼓聲、人聲中的氣音等。

In principle, we can divide each short segment (also known as frame, with a length of about 20 ms) of human's voices into two types:

- Voiced sound: These are produced by the vibration of vocal cords. Since they are produced by the regular vibration of the vocal cords, you can observe the fundamental periods in a frame. Moreover, due to the existence of the fundamental period, you can also perceive a stable pitch.
- Unvoiced sound: These are not produced by the vibration of vocal cords. Instead, they are produced by the rapid flow of air through the mouse, the nose, or the teeth. Since these sounds are produced by the noise-like rapid air flow, we can not observed the fundamenta period and no stable pitch can be perceived.

It is very easy to distinguish between these two types of sound. When you pronunce an utterance, just put your hand on your throat to see if you feel the vibration of your vocal cords. If yes, it is voiced; otherwise it is unvoiced. You can also observe the waveforms to see if you can identify the fundamental periods. If yes, it is voiced; otherwise, it is unoviced.

以人聲而言，我們可以根據其是否具有音高而分為兩類，如下：

- Voiced sound: 由聲帶振動所發出的聲音，例如一般的母音等。由於聲帶振動，造成規律性的變化，所以我們可以感覺到音高的存在。
- Unvoiced soung: 由嘴唇所發出的氣音，並不牽涉聲帶的震動。由於波形沒有規律性，所以我們通常無法感受到穩定音高的存在。

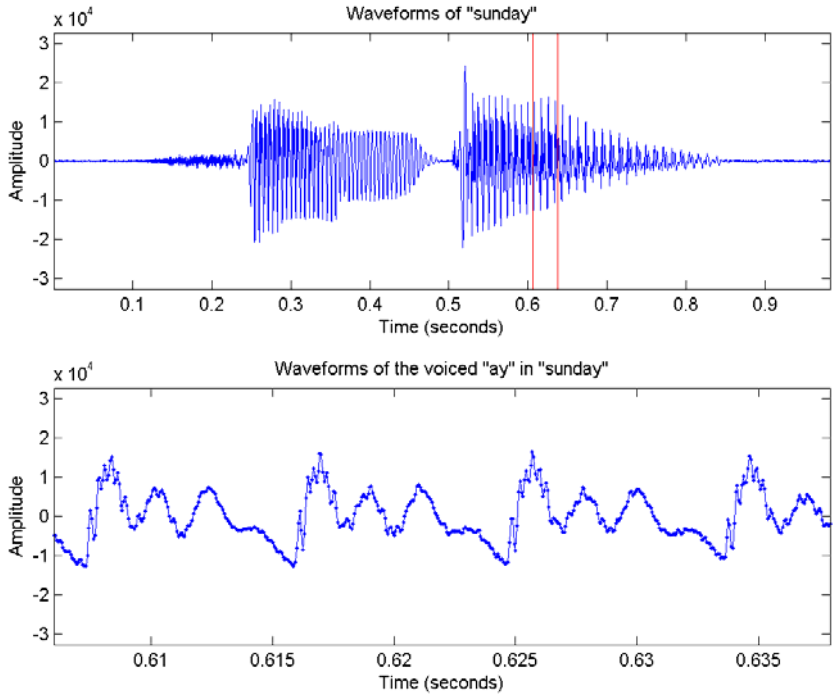
要分辨這兩種聲音，其實很簡單，你只要在發音時，將手按在喉嚨上，若有感到震動，就是 voiced sound，如果沒有感到震動，那就是 unvoiced sound。

The following figure shows the voiced sound of "ay" in the utterance "sunday".

Example 1**Input file** audioIntro/voicedShow01.m

```
figure;
[y, fs, nbits]=wavReadInt('sunday.wav');
subplot(2,1,1)
time=(1:length(y))/fs;
plot(time, y); axis([min(time), max(time), -2^nbits/2, 2^nbits/2]);
xlabel('Time (seconds)'); ylabel('Amplitude'); title('Waveforms of "sunday"');

frameSize=512;
index1=0.606*fs;
index2=index1+frameSize-1;
line(time(index1)*[1, 1], 2^nbits/2*[-1 1], 'color', 'r');
line(time(index2)*[1, 1], 2^nbits/2*[-1 1], 'color', 'r');
subplot(2,1,2);
time2=time(index1:index2);
y2=y(index1:index2);
plot(time2, y2, '.-'); axis([min(time2), max(time2), -2^nbits/2, 2^nbits/2]);
xlabel('Time (seconds)'); ylabel('Amplitude'); title('Waveforms of the voiced "ay" in "sunday"');
```

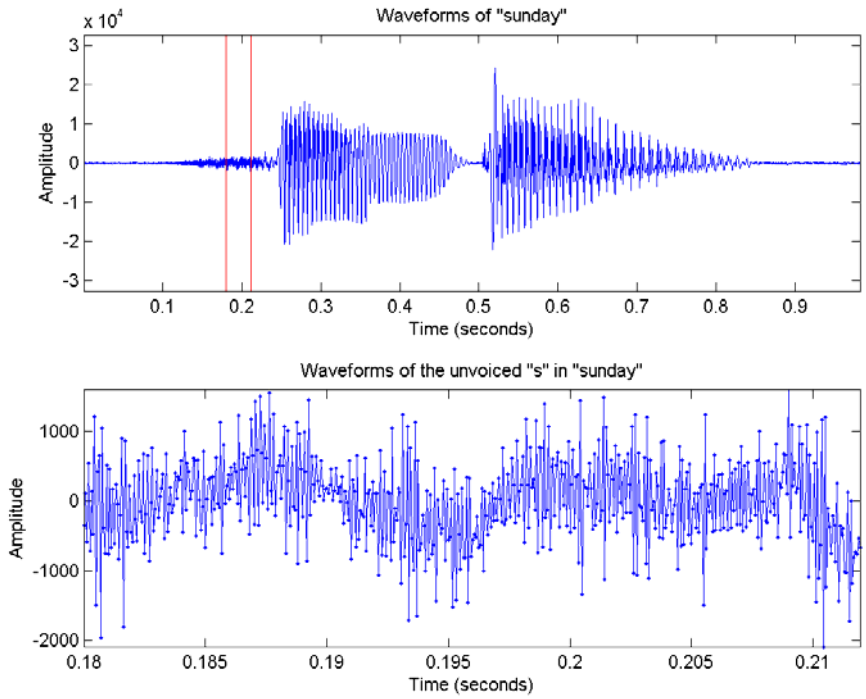
Output figure



You can easiy identify the fundamental period in the closeup plot.
On the other hand, we can also observe the unvoiced sound of "s" in the utterance "sunday", as shown in the following example:
Example 2**Input file** audioIntro/unvoicedShow01.m

```
[y, fs, nbits]=wavReadInt('sunday.wav');
subplot(2,1,1)
time=(1:length(y))/fs;
plot(time, y); axis([min(time), max(time), -2^nbits/2, 2^nbits/2]);
xlabel('Time (seconds)'); ylabel('Amplitude'); title('Waveforms of "sunday"');

frameSize=512;
index1=0.18*fs;
index2=index1+frameSize-1;
line(time(index1)*[1, 1], 2^nbits/2*[-1 1], 'color', 'r');
line(time(index2)*[1, 1], 2^nbits/2*[-1 1], 'color', 'r');
subplot(2,1,2);
time2=time(index1:index2);
y2=y(index1:index2);
plot(time2, y2, '.-'); axis([min(time2), max(time2), -inf inf]);
xlabel('Time (seconds)'); ylabel('Amplitude'); title('Waveforms of the unvoiced "s" in "sunday"');
```

Output figure



In contract, there is no fundamental periods and the waveform is noise-like.

Hint

You can also use CoolEdit for simple recording, replay and observation of audio signals.

若要對聲音進行簡單的錄音、播放、觀察及處理，可以使用 CoolEdit 軟體。

Audio signals actually represent the air pressure as a function of time, which is a continuous in both time and signal amplitude. When we want to digitize the signals for storage in a computer, there are several parameter to consider.

- Sample rate: This is the number of sample points per second, in the unit of Hertz (abbreviated as Hz). A higher sample rate indicate better sound quality, but the storage space is also bigger. Commonly used sample rates are listed next:
    1. 8 kHz: Voice quality for phones and toys
    2. 16 KHz: Commonly used for speech recognition
    3. 44.1 KHz: CD quality
- Bit resolution: The number of bits used to represent each sample point of audio signals. Commonly used bit resolutions are listed next:
    1. 8-bit: The corresponding range is 0~255 or -128~127.
    2. 16-bit: The corresponding range is -32768~32767.

  In other words, each sample point is represented by an integer of 8 or 16 bits. However, in MATLAB, all audio signals are normalized to floating-point number within the range [-1, 1] for easy manipulation. If you want to revert to the original integer values, you need to multiply the float-point values by 2^nbits/2, where nbits is the bit resolution.
- Channels: We have mono for single channel and stereo for double channels.

聲音代表了空氣的密度隨時間的變化，基本上是一個連續的函數，但是若要將此訊號儲存在電腦裡，就必須先將此訊號數位化。一般而言，當我們將聲音儲存到電腦時，有下列幾個參數需要考慮：

- 取樣頻率（sample Rate）：每秒鐘所取得的聲音資料點數，以 Hertz（簡寫 Hz）為單位。點數越高，聲音品質越好，但是資料量越大，常用的取樣頻率如下：
    1. 8 kHz：電話的音質、一般玩具內語音 IC 的音質

2. 16 KHz：一般語音辨識所採用
3. 44.1 KHz：CD 音質

- 取樣解析度（Bit Resolution）：每個聲音資料點所用的位元數，常用的數值如下：
  1. 8-bit：可表示的數值範圍為 0~255 或 -128~127
  2. 16-bit：可表示的數值範圍為 -32768~32767

  換句話說，每個取樣點的數值都是整數，以方便儲存。但是在 MATLAB 的表示法，通常把音訊的值正規化到 [-1, 1] 範圍內的浮點數，因此若要轉回原先的整數值，就必須再乘上 2^nbits/2，其中 nbits 是取樣解析度。
- 聲道：一般只分單聲道（Mono）或立體聲（Stereo），立體音即是雙聲道。

Let take my utterance of sunday for example. It is a mono recording with a sample rate of 16000 (16 KHz) and a bit resolution of 16 bits (2 bytes). It also contains 15716 sample points, corresponding to a time duration of 15716/16000 = 0.98 seconds. Therefore the file size is about 15716*2 = 31432 bytes = 31.4 KB. In fact, the file size for storing audio signals is usually quite big without compression. For instance:

- If we used the same parameters for a one-minute recording, the file size will be 60 sec x 16 KHz x 2 Byte = 1920 KB, close to 2 MB.
- For audio music in a CD, we have stereo recordings with a sample rate of 44.1 KHz, a bit resolution of 16 Bits. Therefore for a 3-minute audio music, the file size is 180 sec x 44.1 KHz x 2 Byte x 2 = 31752 KB = 32 MB. (From here you will also know the MP3 compression ratio is about 10.)

以我所錄的「sunday」來說，這是單聲道的聲音，取樣頻率是 16000（16 KHz），解析度是 16 Bits（2 Byte），總共包含了 15716 點（等於 15716/16000 = 0.98 秒），所以檔案大小就是 15716*2 = 31432 bytes = 31.4 KB 左右。由此可以看出聲音資料的龐大，例如：

- 如果我以相同的參數來進行錄音一分鐘，所得到的檔案大小大約就是 60 秒 x 16 KHz x 2 Byte = 1920 KB 或將近 2 MB。
- 以一般音樂 CD 來說，大部分是立體聲，取樣頻率是 44.1 KHz，解析度是 16 Bits，所以一首三分鐘的音樂，資料量的大小就是 180 秒 x 44.1 KHz x 2 Byte x 2 = 31752 KB = 32 MB。（由此可知，MP3 的壓縮率大概是 10 倍左右。）

3-2 Basic Acoustic Features (基本聲學特徵)

When we analyze audio signals, we usually adopt the method of short-term analysis since most audio signals are more or less stable within a short period of time, say 20 ms or so. When we do frame blocking, there may be some soverlap between neighboring frames to capture subtle change in the audio signals. Note that each frame is the basic unit for our analysis. Within each frame, we can observe the three most distinct acoustic features, as follows.

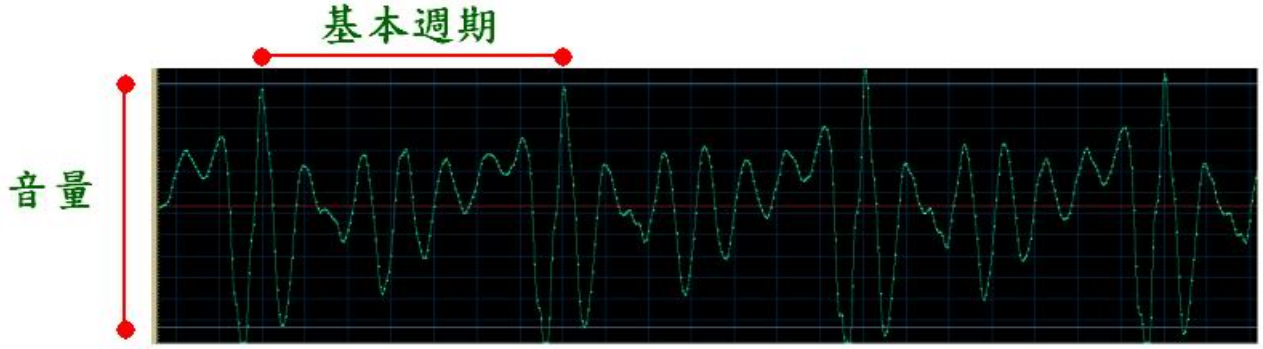
- Volume: This feature represents the loudness of the audio signal, which is correlated to the amplitude of the signals. Sometimes it is also referred to as energy or intensity of audio signals.
- Pitch: This feature represents the vibration rate of audio signals, which can be represented by the fundamental frequency, or equivalently, the reciprocal of the fundamental period of voiced audio signals.
- Timbre: This feature represents the meaningful content (such as a vowel in English) of audio signals, which is characterized by the waveform within a fundamental period of voice signals.

These three acoustic features can be related to the waveform of audio signals, as follows:

當我們在分析聲音時，通常以「短時距分析」（Short-term Analysis）為主，因為音訊在短時間內是相對穩定的。我們通常將聲音先切成音框（Frame），每個音框長度大約在 20 ms 左右，再根據音框內的訊號來進行分析。在一個特定音框內，我們可以觀察到的三個主要聲音特徵可說明如下：

- 音量（Volume）：代表聲音的大小，可由聲音訊號的震幅來類比，又稱為能量（Energy）或強度（Intensity）等。
- 音高（Pitch）：代表聲音的高低，可由基本頻率（Fundamental Frequency）來類比，這是基本週期（Fundamental Period）的倒數。
- 音色（Timbre）：代表聲音的內容（例如英文的母音），可由每一個波形在一個基本週期的變化來類比。

這些特徵可用圖形說明如下：



Take human voices as an example, then the above three acoustic features will correlates to some physical quantities:

- Volume: It correlates to the compression of your lungs. A large volume of audio signals corresponds to a large compression.
- Pitch: It correlates to the vibration frequency of your vocal cord. A high pitch corresponds to a high vibration frequency.
- Timbre: It correlates to the positions and shapes of your lips and tongue. Different timbres corresponds to different positions and shapes of your lips and tongue.

如果是用人聲來說明，這些語音特徵的物理意義如下：

- 音量：代表肺部壓縮力量的大小，力量越大，音量越大。
- 音高：代表聲帶震動的快慢，震動越快，音高會越高。
- 音色：代表嘴唇和舌頭的位置和形狀，不同的位置和形狀，就會產生不同的語音內容。

We shall explain methods to extract these acoustic features in the other chapters of this book. It should be noted that these acoustic features mostly corresponds to human's "perception" and therefore cannot be represented exactly by mathematical formula or quantities. However, we still try to "quantitify" these features for further computer-based analysis in the hope that the used formula or quantities can emulate human's perception as closely as possible.

有關這些語音特徵的抓取和分析，會在後續章節有詳細說明。特別要注意的是，這些特徵都是代表「人耳的感覺」，並沒有一定的數學公式可尋，所以當我們試著在「量化」這些特徵時，只是根據一些數據和經驗來量化，來盡量逼近人耳的感覺，但並不代表這些「量化」後的數據或公式就可以完全代表聲音的特徵。

The basic approach to the extraction of audio acoustic features can be summarized as follows:

1. Perform frame blocking such that a strem of audio signals is converted to a set of frames. The time duration of each frame is about 20~30 ms. If the frame duration is too big, we cannot catch the time-varying characteristics of the audio signals. On the other hand, if the frame duration is too small, then we cannot extract valid acoustic features. In general, a frame should be contains several fundamental periods of the given audio signals. Usually the frame size (in terms of sample

points) is equal to the powers of 2 (such as 256, 512, 1024 ,etc) such that it is suitable for fast fourier transform.
2. If we want to reduce the difference between neighboring frames, we can allow overlap between them. Usually the overlap is 1/2 to 2/3 of the original frame. The more overlap, the more computation is needed.
3. Assuming the audio signals within a frame is stationary, we can extract acoustic features such as zero crossing rates, volume, pitch, MFCC, LPC, etc.
4. We can perform endpoint detection based on zero crossing rate and volume, and keep non-silence frames for further analysis.

音訊特徵抽取的基本方式如下：

1. 將音訊切成一個個音框，音框長度大約是 20~30 ms。音框若太大，就無法抓出音訊隨時間變化的特性；反之，音框若太小，就無法抓出音訊的特性。一般而言，音框必須能夠包含數個音訊的基本週期。（另，音框長度通常是 2 的整數次方，若不是，則在進行「傅立葉轉換」時，需補零至 2 的整數次方，以便使用「快速傅立葉轉換」。）
2. 若是希望相鄰音框之間的變化不是太大，可以允許音框之間有重疊，重疊部分可以是音框長度的 1/2 到 2/3 不等。（重疊部分越多，對應的計算量也就越大。）
3. 假設在一個音框內的音訊是穩定的，對此音框求取特徵，如過零率、音量、音高、MFCC 參數、LPC 參數等。
4. 根據過零率、音量及音高等，進行端點偵測（Endpoint Detection），並保留端點內的特徵資訊，以便進行分析或辨識。

When we are performing the above procedures, there are several terminologies that are used often:

- Frame size: The sampling points within each frame
- Frame overlap: The sampling points of the overlap between consecutive frames
- Frame step (or hop size): This is equal to the frame size minus the overlap.
- Frame rate: The number of frames per second, which is equal to the sample frequency divided by the frame step.

Hint

Note that these terminologies are not unified. Some papers use frame step to indicate hop size or frame rate instead. You should be cautious when reading papers with these terms.

For instance, if we have a stream of audio signals with sample frequency fs=16000, and a frame duration of 25 ms, overlap of 15 ms, then

- Frame size = fs*25/1000 = 400 (sample points)。
- Frame overlap = fs*15/1000 = 240 (sample points)。
- Frame step (or hop size) = 400-240 = 160 (sample points)。
- Frame rate = fs/160 = 100 frames/sec。

在進行上述分析時，有幾個名詞常用到，說明如下：

- 音框點數（Frame Size）：每一個音框所含有的點數。
- 音框重疊量（Frame Overlap）：音框之間重疊的點數。
- 音框跳距（Frame Step or Hop Size）：此音框起點和下一個音框起點的距離點數，等於音框點數減去音框重疊。
- 音框率（Frame Rate）：每秒出現的音框數目，等於取樣頻率除以音框跳距。

舉例而言，如果取樣頻率 fs=16000 且每一個音框所對應的時間是 25 ms，重疊 15 ms，那麼

- Frame size = fs*25/1000 = 400 點。
- Frame overlap = fs*15/1000 = 240 點。
- Frame step (or hop size) = 400-240 = 160 點。
- Frame rate = fs/160 = 100 frames/sec。

3-3 Human Voice Production (人聲的產生)

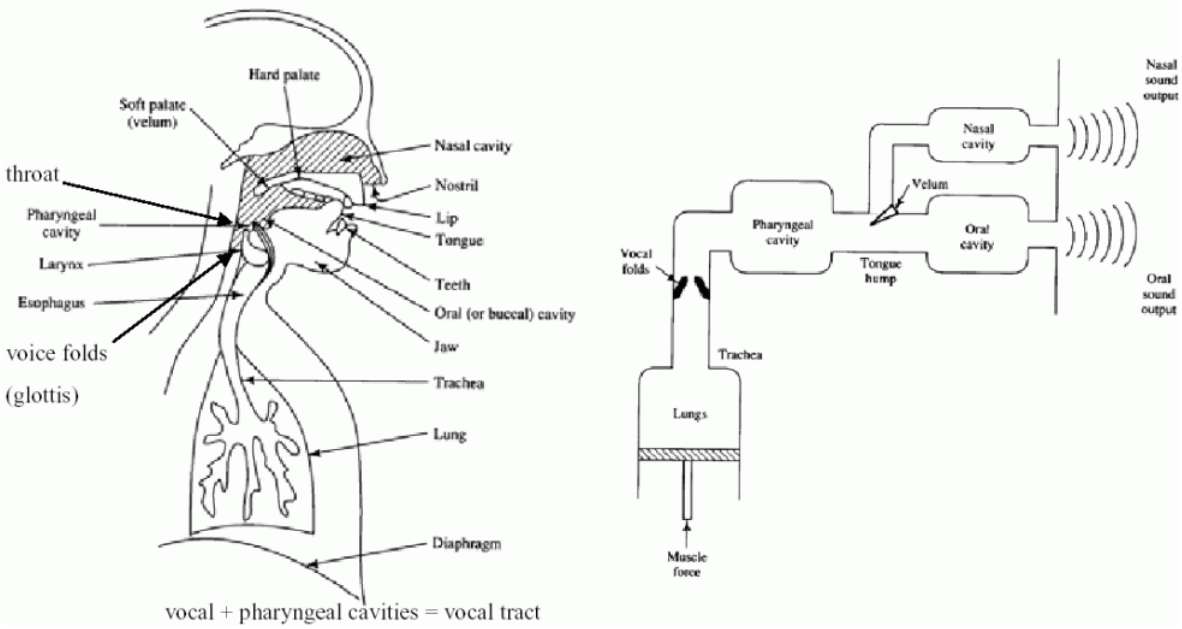
The procedure from human voice production to voice recognition involves the following steps:

1. Rapid open and close of your vocal cords (or glottis) to generate the vibration in air flow.
2. Resonance of the pharyngeal cavity, nasal cavity, and oral cavity.
3. The vibration of air.
4. The vibration of the ear drum (or tympanum).
5. The reception of the inner ear.
6. The recognition by the brain.

The following diagram demonstrate the production mechanism for human voices.
人聲的發音與接收流程，可以列出如下：

1. 聲門的快速打開與關閉
2. 聲道、口腔、鼻腔的共振
3. 空氣的波動
4. 接收者耳膜的振動
5. 內耳神經的接收
6. 大腦的辨識

下列圖形說明人聲的發音機制：



The production mechanism of human voices.
人聲的發音機制

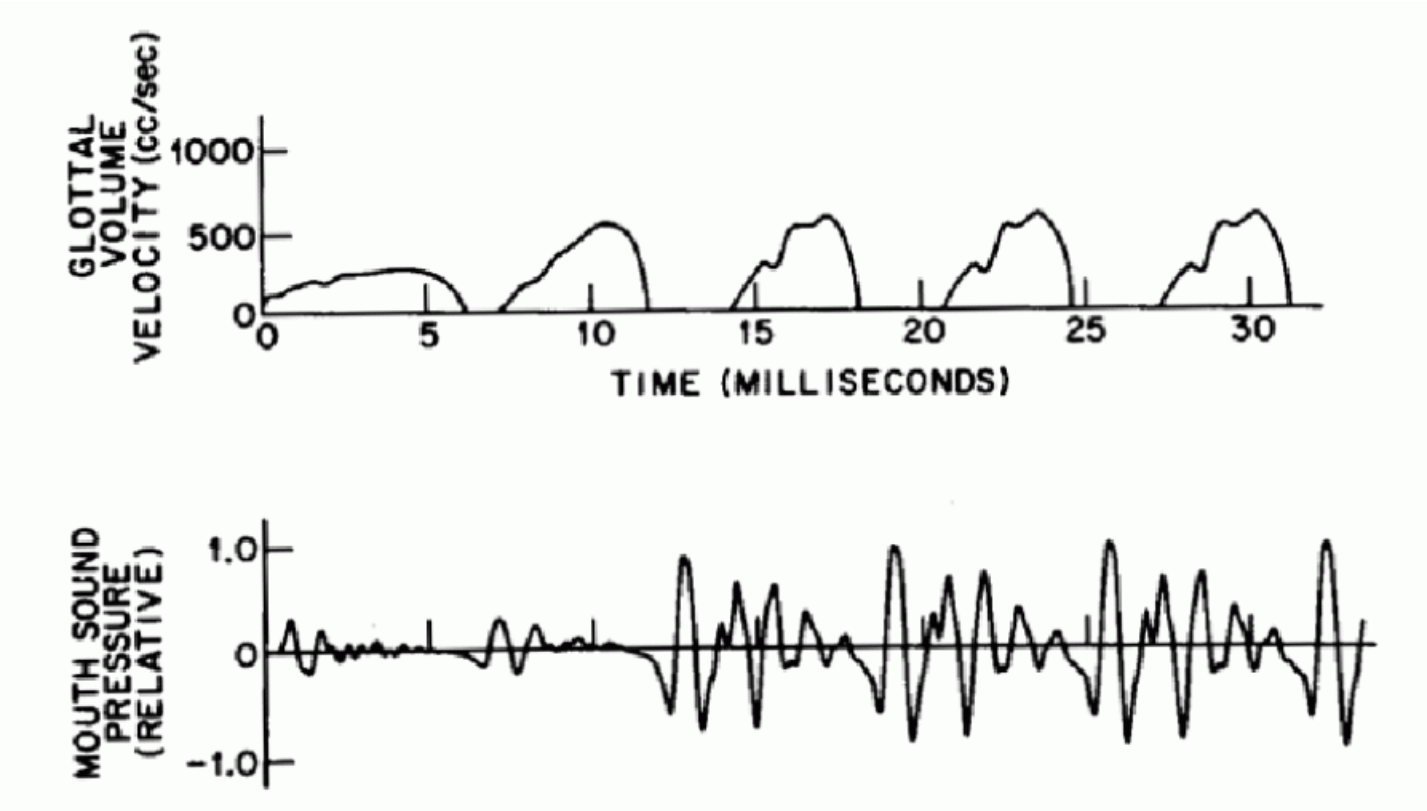
Due to the pressure of the glottis and the air pushed from the lungs, the vocal cords can open and close very quickly, which generates vibrations in the air. The vibration is modulated by the resonances of pharyngeal/nasal/oral cavities, forming different timbre of your voices. In other words:

- The vibration frequency of the vocal cords determines the pitch of the voices.
- The positions/shapes of your lips, tongue, and nose determine the timbre.
- The compression from your lungs determine the loudness of the voices.

由於聲門（Glottis）的肌肉張力，加上由肺部壓迫出來的空氣，就會造成聲門的快速打開與關閉，這個一疏一密的空氣壓力，就是人聲的源頭，在經由聲道、口腔、鼻腔的共振，就會產生不同的聲音（音色）。換句話說：

- 聲門震動的快，決定聲音的基本頻率（即音高）。
- 口腔、鼻腔、舌頭的位置、嘴型等，決定聲音的內容（即音色）。
- 肺部壓縮空氣的力量大小，決定音量大小。

The following figure demonstrates the airflow velocity around the glottis and the voice signals measured around the mouth.
下面這一張圖，顯示聲門附近的空氣流速，以及最後在嘴巴附近所量測到的聲波：

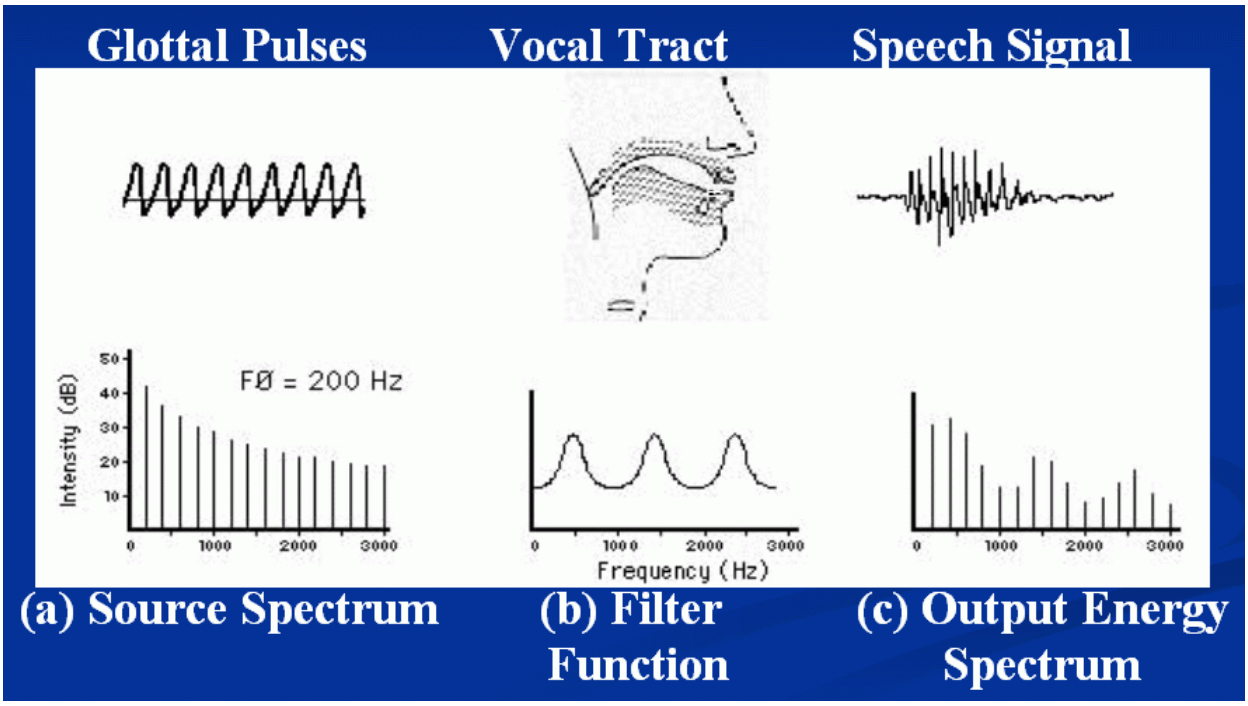


Airflow velocity around the glottis and the resultant voices signals

You can observe the movement of the vocal cords from the following link:
經由下面這個連結，可以看到聲門運動的現象：
http://www.humnet.ucla.edu/humnet/linguistics/faciliti/demos/vocalfolds/vocalfolds.htm （local copy）

In fact, it is not easy to capture the movements of vocal cords due to its high frequency in movement. So we need to have high-speed cameras for such purpose, for instance:
要拍到聲門運動，是相當不容易，必須使用高速的攝影機，例如
http://www.kayelemetrics.com/Product%20Info/9700/9700.htm （local copy）

We can conceive the production of human voices as a source-filter model where the source is the airflow caused by the vocal cords, and the filter includes the pharyngeal/nasal/oral cavities. The following figure shows the representative spectrum for each stage:
所以人發音的過程，是由訊號源（聲門），經過濾波器（口腔、鼻腔、嘴型等），才得到最後的聲音，這個過程可以和頻譜訊號一一對應如下：

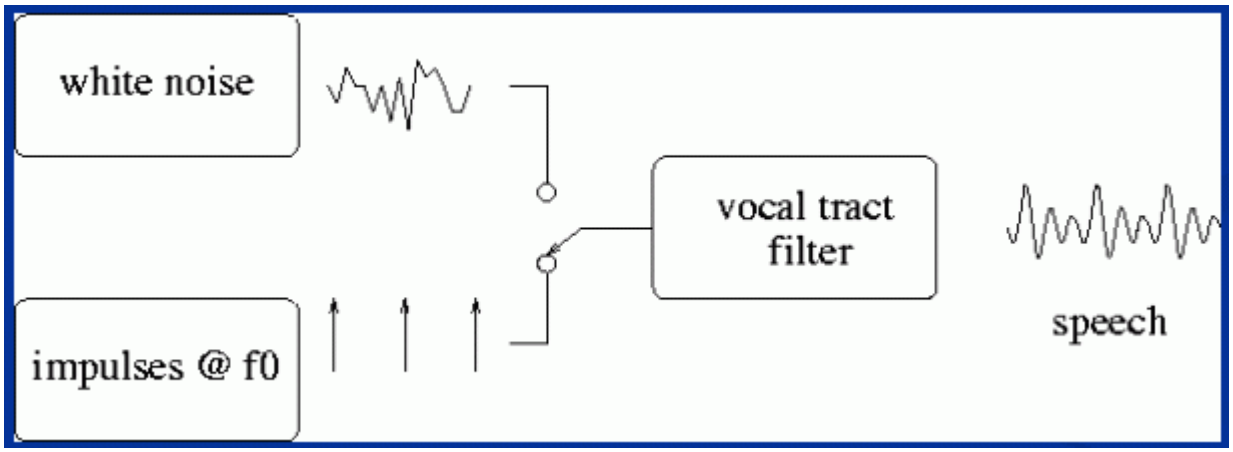


Source-filter model and the corresponding spectra

人聲發音過程與與頻譜的對應

We can also use the following block diagram to represent the source-filter model of human voice production:

若用數學模型表示，可用下列方塊圖：

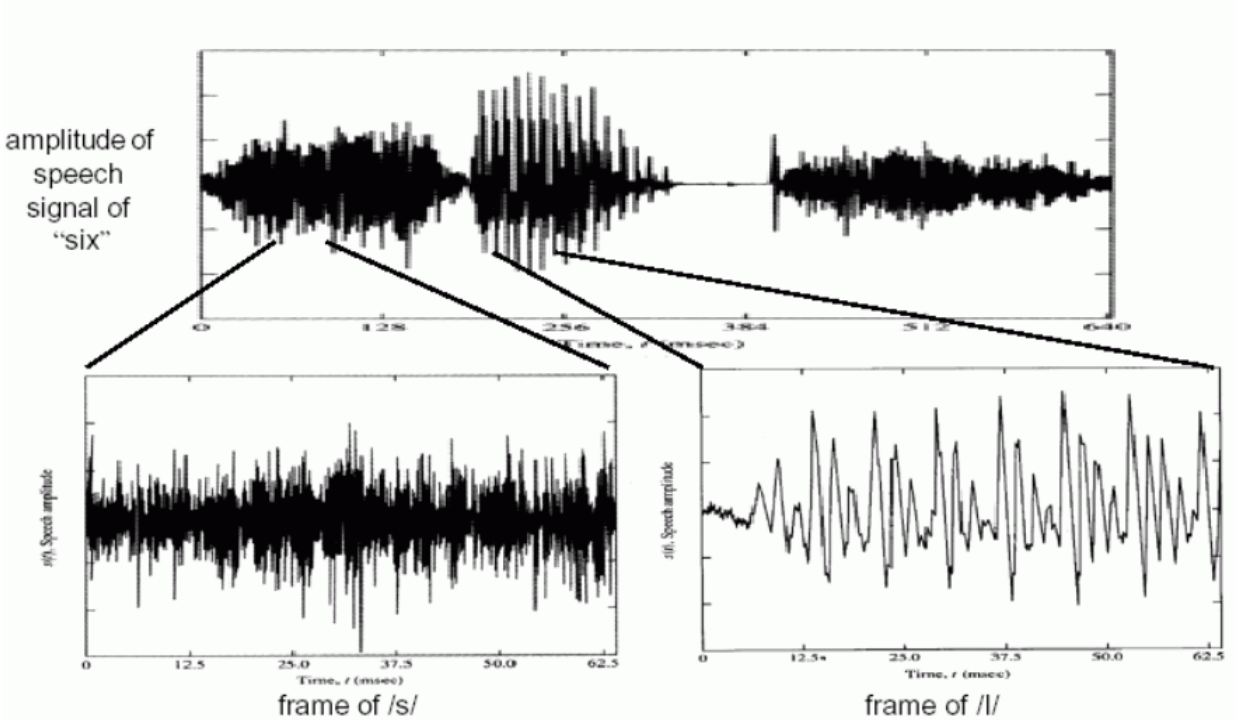


Block diagram representation of source-filter model

人聲發音過程的數學模型

In general, a regular vibration of the glottis will generate quasi-periodic voiced sounds. On the other hand, if the source is irregular airflow, then we will have unvoiced sounds. Take the utterance of "six" for example:

一般來說，當訊號源是間隔規律的波形時，通常代表有聲音，如果訊號源是雜亂的訊號，則得到氣音，以下列的發音「six」為例：



Unvoiced and voiced sounds

氣音和有聲音

We can clearly observe that "s" and "k" are unvoiced sounds, while "i" is a voiced sound.

其中「s」和「k」都是無聲的氣音，只有「i」是有聲音。

For Mandarin, almsot all unvoiced sounds happen at the beginning of a syllable. Take the utterance of "清" as in "清華大學" for example:

1. No vibration from the glottis. Close your teech and push forward your tongue tip against the lower teeth to generate the unvoiced sound "ㄑ" by a jet of airflow.
2. Keep almost the sampe position but start glottis vibration to pronunce the voiced "ㄧ".
3. Keep glottis vibrate but retract your tongue to pronuced the final voiced "ㄥ".

一般而言，中文的氣音只發生在字頭，不會是在字尾。以「清華大學」的「清」為例：

1. 聲門不震動，上下顎咬合，舌頭前伸，完全是氣音，發出「ㄑ」
2. 姿勢類似，聲門震動，發出「ㄧ」。
3. 聲門維持同樣的震動，但是舌頭後縮，發出「ㄥ」。

Hint

Just put your hand on your throat, you can feel the vibration of the glottis.

若要判斷你的聲門是否有震動，只要將手放在你的喉嚨位置，就可以感覺到聲門是否有震動。

Here are some terminologies in both English and Chinese for your reference:

以下是一些名詞的中英對照表：

1. Cochlea：耳蝸
2. Phoneme：音素、音位
3. Phonics：聲學；聲音基礎教學法（以聲音為基礎進而教拼字的教學法）
4. Phonetics：語音學
5. Phonology：音系學、語音體系
6. Prosody：韻律學；作詩法
7. Syllable：音節
8. Tone：音調
9. Alveolar：齒槽音
10. Silence：靜音

11. Noise：雜訊
12. Glottis：聲門
13. larynx：喉頭
14. Pharynx：咽頭
15. Pharyngeal：咽部的，喉音的
16. Velum：軟顎
17. Vocal chords：聲帶
18. Glottis: 聲門
19. Esophagus：食管
20. Diaphragm：橫隔膜
21. Trachea：氣管

Chapter 4: MATLAB for Audio Signal Processing

4-1 Introduction

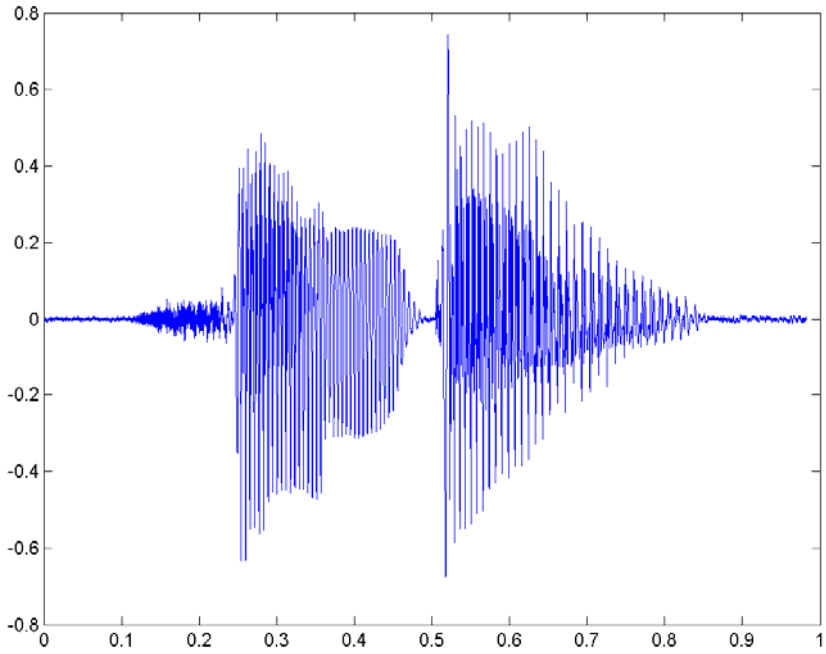
The chapter introduces the functions within MATLAB that can be used for audio signal processing. In particular, we shall cover the topics of how to read .wav files, how to play audio signals, how to record from microphone, and how to save .wav files.
This chapter is a partial translation of the original tutorial in Chinese: 20-音訊讀寫、錄製與播放.pdf」.

4-2 Reading Wave Files

On the Windows platform, the most common file extension for audio signals is "wav". MATLAB can read such wave files via the command "wavread". The following example reads the wave file "sunday.wav" and display its waveform directly.
Example 1**Input file** matlab4asp/readWave01.m

```
[y, fs]=wavread('sunday.wav');
sound(y, fs);                % Playback of the sound data (播放此音訊)
time=(1:length(y))/fs;       % Time vector on x-axis (時間軸的向量)
plot(time, y);               % Plot the waveform w.r.t. time (畫出時間軸上的波形)
```

Output figure



In the above example, "fs" is the sample rate which is 11025 in this case. This indicates that there are 11025 sample points per second when the clip was recorded. The vector "y" is a column vector containing the sample points of the speech signals. We can use "sound(y, fs)" to play the audio signals read from the wave file. "time" is a time vector in which each element corresponds to the time of each sample point. Therefore we can plot "y" against "t" to show the waveform directly.

Most audio signals are digitized to have a bit resolution of 8 or 16 bits. If we want to know the bit resolution of the file "welcome.wav", we can use more output arguments to "wavread" to get the information, such as

```
[y, fs, nbits]=wavread('sunday.wav');
```

Moreover, if we want to know the time duration of a stream of audio signals, we can use "length(y)/fs" directly. The following example can obtain most of the important information of the wave file "welcome.wav".

Example 2**Input file** matlab4asp/readWave02.m

```
fileName='welcome.wav';
[y, fs, nbits]=wavread(fileName);
fprintf('Information of the sound file "%s":\n', fileName);
fprintf('Duration = %g seconds\n', length(y)/fs);
fprintf('Sampling rate = %g samples/second\n', fs);
fprintf('Bit resolution = %g bits/sample\n', nbits);
```

Output message

```
Information of the sound file "welcome.wav":
Duration = 1.45134 seconds
Sampling rate = 11025 samples/second
Bit resolution = 8 bits/sample
```

From the above example, it can be observed that all the audio signals are between -1 and 1.

However, each sample point is represented by an 8-bit interger. How are they related? First of all, we need to know that

1. If a wave file has a bit resolution of 8 bits, then each sample point is stored as an unsigned integer between 0 and 255 (= 2^8-1).
2. If a wave file has a bit resolution of 16 bits, then each sample point is stored as an unsigned integer between -32768 (= 2^16/2) and 32767 (= 2^16/2-1).

Since almost all variables in MATLAB have the data type of "double", therefore all sample points are converted into a floating-point number between -1 and 1 for easy manipulation. Therefore to retrieve the original integer values of the audio signals, we can proceed as follows.

1. For 8-bit resolution, we can multiply "y" (the value obtained by wavread) by 128 and then plus 128.
2. For 16-bit resolution, we can multiply "y" (the value obtained by wavread) by 32768.

Here is an example.

Example 3**Input file** matlab4asp/readWave03.m

```
fileName='welcome.wav';
[y, fs, nbits]=wavread(fileName);
y0=y*(2^nbits/2)+(2^nbits/2);                    % y0 is the original values stored in the wav file (y0 是原先儲存在音訊檔案中的值)
difference=sum(abs(y0-round(y0)))
```

Output message

```
difference =

     0
```

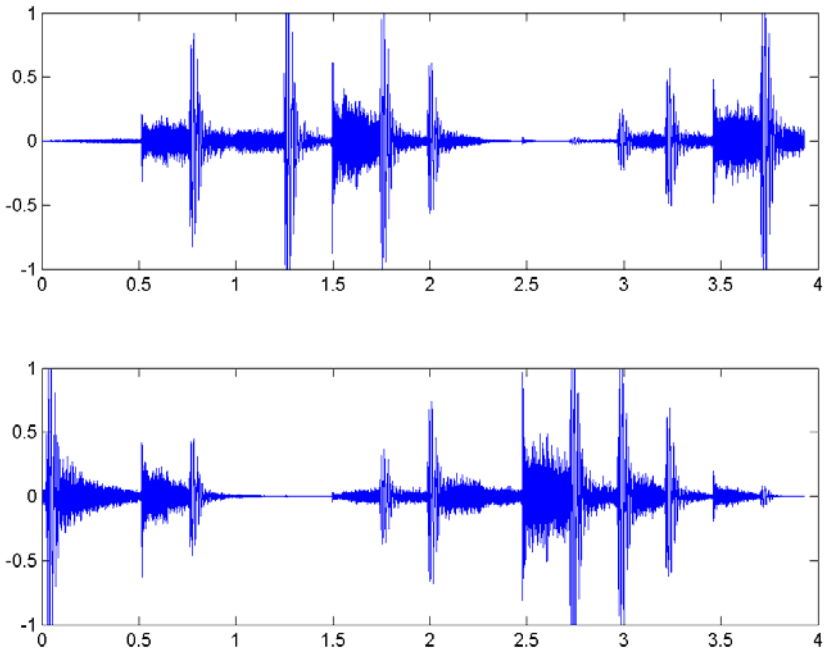
In the above example, the difference is zero, indicating the retrived y0 contains no fractional parts. Moreover, to increase the generality, we use 2^nbits/2 directly instead of 128.

We can also use the command "wavread" to read a stereo wave file. The returned variable will be a matrix of 2 columns, each containing the audio signals from a single channel. Example follows.

Example 4**Input file** matlab4asp/readWave04.m

```
fileName='flanger.wav';
[y, fs]=wavread(fileName);  % Read wave file (讀取音訊檔)
sound(y, fs);                           % Playback (播放音訊)
left=y(:,1);                            % Left channel (左聲道音訊)
right=y(:,2);                           % Right channel (右聲道音訊)
subplot(2,1,1), plot((1:length(left))/fs, left);
subplot(2,1,2), plot((1:length(right))/fs, right);
```

Output figure



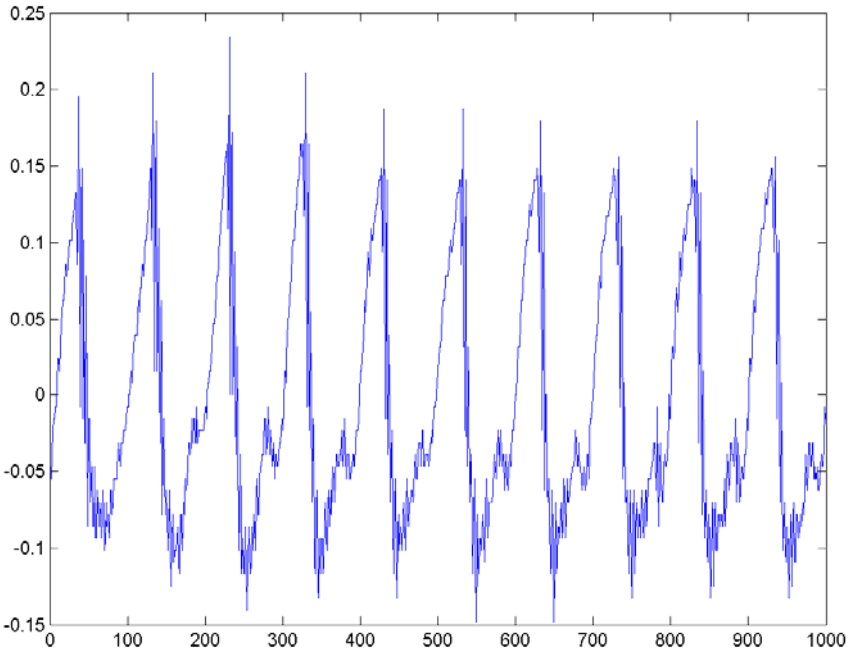
In the above example, MATLAB will read the wave file "flanger.wav", play the stereo sound, and plot two streams of audio signals in two subplots. Because the intnesities of these two channels are more or less complemntary to each other, which let us have an illusion that the sound source is moving back and forth between two speakers. (A quiz: how do you create such effect given a stream of audio signals?)

If the wave file is too large to be read into memory directly, we can also use "wavread" to read a part of the audio signals directly. See the following example.

Example 5**Input file** matlab4asp/readWave05.m

```
[y,fs]=wavread('welcome.wav', [4001 5000]);   % Read 4001~5000 sample points (讀取音訊檔第 4001 至 5000 點)
figure; plot(y)
```

Output figure



The waveform in the above example represent the vowel of the second spoken Chinese character "迎" in the original "歡迎光臨". It is obvious that the waveform contain a fundamental period of about 100 sample points, corresponding to a time duration of 100/fs = 0.0091 seconds = 9.1 ms. This corresponds to a pitch frequency of 11025/100 = 110.25 Hz. This pitch is pretty close to two octave down the central la, or the 5th white key counting from the left.

The perception of pitch of human ear is proportional to the logrithm of the fundamental frequency. The central la of a piano has a fundamental frequency of 440 Hz. One octave above it is 880 Hz, while one octave below it is 220 Hz. Each octave in the piano keyboard contains 12 keys, including 7 white keys and 5 black ones, corresponding to 12 semitones within a octave. If we adopt the standard of MIDI files, the semitone of the central la is 69 with a fundamental frequency of 440. Therefore we can have a formula that converts a frequency into a semitone:

$$\text{semitone} = 69 + 12*\log_2(\text{freq}/440)$$

The process of computing the pitch contour of audio signals is usually called "pitch tracking". Pitch tracking is an important operation for applications such as text-to-speech synthesis, tone recognition and melody recognition. We shall explain more sophisticated methods for pitch tracking in the following chapters.

If we want to obtain more information about a wave file, we can retrieve it from the 4th output arguments of the command "wavread", as follows.

Example 6**Input file** matlab4asp/readWave06.m

```
[y, fs, nbits, opts]=wavread('flanger.wav');
opts.fmt
```

Output message

```
ans =
```

```
      wFormatTag: 1
       nChannels: 2
  nSamplesPerSec: 22050
nAvgBytesPerSec: 88200
     nBlockAlign: 4
  nBitsPerSample: 16
```

In the above example, some quantities are explained next.

1. wFormatTag is the format tag of the wave file.
2. nChannels is the number of channels.
3. nSamplePerSec is the number of samples per second, which is equal to the samping rate 22050.
4. nAveBytesPerSec is the number of bytes per second. In this case, since we have two channels and the bit resolution is 2 bytes, therefore we have 22050*4 = 88200.
5. nBlockAlign is equal to the rato between nAveBytesPerSec and nSamplePerSec.
6. nBitsPerSample is the bit resolution.

Besides ".wav" files, MATLAB can also use the command "auread" to read the audio files with extension ".au". You can obtain related online help by typing "help auread" within the MATLAB command window.

4-3 Playback of Audio Signals

Once we can read the wave file into MATLAB, we can start process the audio signals by modifying their intensities, or changing their sample rates, and so on. Once the audio signals are processed, we need to play them for aural inspection. This section will introduce the MATLAB commands for play audio signals.

The basic command for playing audio signals is "wavplay". The following example can load a stream of audio signals from the file "handel.mat" and play the signal immediately.

Example 1**Input file** matlab4asp/wavPlay01.m

```
load handel.mat          % Load the signals stored in handel.mat (載入儲存於 handel.mat 的音訊)
wavplay(y, Fs);          % Playback of the signals (播放此音訊)
```

Since the volume of playback is determined by the amplitude of the audio signals, we can change the amplitude to change the volume, as follows.

Example 2**Input file** matlab4asp/playVolume01.m

```
[y, fs]=wavread('welcome.wav');
wavplay(1*y, fs, 'sync');     % Playback with original amplitude (播放 1 倍震幅的音訊)
wavplay(3*y, fs, 'sync');     % Playback with 3 times the original amplitude (播放 3 倍震幅的音訊)
wavplay(5*y, fs, 'sync');     % Playback with 5 times the original amplitude (播放 5 倍震幅的音訊)
```

In the above example, we increase the amplitude gradually, so we can perceive the increasing volume during playbacks. In particular, "wavplay" assume the input signals are between -1 and 1. The the input signals are too large, we can hear "broken sound". To try out this for yourself, you can try "wavplay(100*y, fs)" to hear the result. Moreover, we put an extra input argument 'sync' in

the above example. This will make "wavplay" to play the signals synchronously. That is, the playback will not start until the previous playback is finished. We shall have more details later on.

Hint

In the above example, though we have increase the amplitude by a factor of 5, the intensity perceived by our ears is not by the factor of 5. This serve to exemplify that the perception of volume is not proportional linearly to the amplitude. In fact, it is proportional to the logrithm of the amplitude.

If we change the sample rate during playback, it will affect the time duration as well as the perceived pitch. In the following example, we shall increase the sample rates gradually. So you will hear a shorter sound with high-pitch, just like the sound from the Disney cartoon character Donald Fauntleroy Duck.

Example 3**Input file** matlab4asp/playFs01.m

```
[y, fs]=wavread('welcome.wav');
wavplay(y, 1.0*fs, 'sync');   % Playback at the original speed (播放 1.0 倍速度的音訊)
wavplay(y, 1.2*fs, 'sync');   % Playback at 1.2 times the original speed (播放 1.2 倍速度的音訊)
wavplay(y, 1.5*fs, 'sync');   % Playback at 1.5 times the original speed (播放 1.5 倍速度的音訊)
wavplay(y, 2.0*fs, 'sync');   % Playback at 2.0 times the original speed (播放 2.0 倍速度的音訊)
```

On the other hand, if we lower the sample rate gradually, we shall get longer and low-pitched sounds. Eventually it will sound like a cow's sound.

Example 4**Input file** matlab4asp/playFs02.m

```
[y, fs]=wavread('welcome.wav');
wavplay(y, 1.0*fs, 'sync');   % Playback at the original speed (播放 1.0 倍速度的音訊)
wavplay(y, 0.9*fs, 'sync');   % Playback at 0.9 times the original speed (播放 0.9 倍速度的音訊)
wavplay(y, 0.8*fs, 'sync');   % Playback at 0.8 times the original speed (播放 0.8 倍速度的音訊)
wavplay(y, 0.6*fs, 'sync');   % Playback at 0.6 times the original speed (播放 0.6 倍速度的音訊)
```

If we want to keep the same time duration but increase or decreasing the pitch of audio signals, then we need to perform pitch shift or pitch scaling. It is beyond the scope of this chapter and will be explained in later chapters.

If we reverse the audio signals by multiplying by -1, the perception will be exactly the same as the original. (This also serve to demonstrate that human's perception of audio is not affect by its phase.) However, if the reverse the audio signals in time axis, then it will sound like a foreign language. Pleae try the following example.

Example 5**Input file** matlab4asp/playReverse01.m

```
[y, fs]=wavread('welcome.wav');
wavplay(y, fs, 'sync');                    % Playback of the original signal (播放正常的音訊波形)
wavplay(-y, fs, 'sync');                   % Playback of the up-down flipped signal (播放上下顛倒的音訊波
形)
wavplay(flipud(y), fs, 'sync');            % Playback of the left-right flipped signal (播放前後顛倒
的音訊波形)
```

When playing a strem of audio signals, MATLAB has two modes when playing a stream of audio signals, as follows.

1. Synchronous: MATLAB will stop all the other execution of commands until the playback is finished.
2. Asynchronous: MATLAB will continue the execution of other commands while the playback is proceeded.

The following example can be used to demonstrate these two modes of playback.

Example 6**Input file** matlab4asp/playSync01.m

```
[y, fs]=wavread('welcome.wav');
wavplay(y, 1.0*fs, 'sync');   % Synchronous playback (同步播放 1.0 倍速度的音訊)
wavplay(y, 0.8*fs, 'async');  % Asynchronous playback at 0.8 of the original speed (非同步播放 0.8 倍速度的音訊)
wavplay(y, 0.6*fs, 'async');  % Asynchronous playback at 0.6 of the original speed (非同步播放 0.6 倍速度的音訊)
```

After executing the above example, you will hear a synchronous playback with two asynchronous playbacks.
When we are using "wavplay(y, fs)", the data type of the variable "y" can assume one of the following types: "double", "single", "int16", "uint8". If the type of "double" is assumed, the range of "y" has to be within -1 and 1. Any out-of-range elements of "y" will be clipped.
MATLAB has another similar command for playback: sound. Please try the following example.
Example 7**Input file** matlab4asp/playSync02.m

```
load handel.mat
sound(y, Fs);                 % Playback at the right sampling rate
sound(y, 1.2*Fs);  % Playback at a faster sampling rate
```

In the above example, we will two playbacks with slow and fast paces. This is simply the default mode of "sound" is asynchronous. Another similar command is "soundsc" which can scale up the audio signals for better playback result. Example follows.
Example 8**Input file** matlab4asp/soundsc01.m

```
[y, fs]=wavread('welcome.wav');
sound(y, fs);                 % Playback of the original sound
fprintf('Press any key to continue...\n'); pause
soundsc(y, fs);               % Playback of the sound after scaling
```

Output message

```
Press any key to continue...
```

The volume of the original "welcome.wav" is too small. After using "soundsc", the playback result is much better.

4-4 Recording from Microphone

You can also use the MATLAB command "wavrecord" to read the audio signals from the microphone directly. The command format is

y = wavrecord(n, fs);

where "n" is number of samples to be recorded, and "fs" is the sample rate. The following example will record 2 seconds from your microphone.
Example 1**Input file** matlab4asp/wavRecord01.m

```
fs=16000;                     % Sampling rate (取樣頻率)
duration=2;                   % Recording duration (錄音時間)
fprintf('Press any key to start %g seconds of recording...', duration); pause
fprintf('Recording...');
y=wavrecord(duration*fs, fs);          % duration*fs is the total number of sample points
fprintf('Finished recording.\n');
```

```
fprintf('Press any key to play the recording...'); pause
wavplay(y,fs);
```

(You need to execute the above program to see the recording procedure clearly.) In the above example, "duration*fs" is the number of sample points to be recorded. The recorded sample points are stored in the variable "y", which is a vector of size 32000x1. The data type of "y" is double and the memory space taken by "y" is 256,000 bytes.

Hint

You can use the command whos to show the memory usage by all variables in the work space

Hint

The commands wayplay and wavrecord are only supported in Microsoft Windows platform.

In the previous example, the number of channels is 1 and the data type for sample points is double. If we want to change these two default settings, we can introduce extra input arguments to the command "wavrecord". A detailed format of "wavrecord" is:

y = wavrecord(n, fs, channel, dataType);

where "channel" (usually 1 or 2) is the number of recording channels, and "dataType" (such as 'double', 'single', 'int16', 'uint8') is the data type of the recorded sample points. Different data types will require different amount of memory space. Example follows.

Example 2**Input file** matlab4asp/wavRecord02.m

```
fs=16000;                   % Sampling rate (取樣頻率)
duration=2;                 % Recording duration (錄音時間)
channel=1;                  % Mono (單聲道)
fprintf('Press any key to start %g seconds of recording...', duration); pause
fprintf('Recording...');
y=wavrecord(duration*fs, fs, channel, 'uint8');          % duration*fs is the number of total sample
points
fprintf('Finished recording.\n');
fprintf('Pressy any key to hear the recording...'); pause
wavplay(y,fs);
```

This example is almost the same as the previous one, except that the data type is 'uint8'. The sample points are still kept in the variable "y" with the same size 32000x1. But the elements within "y" are integers between 0 and 255. The memory space of "y" is now only 32000 bytes, which is only 1/8 of that in the previous example.

Hint

You can use class(y) to display the data type of variable "y".

The following table shows the data types supported by the command wavrecord.

Data types	Space requirement per sample	Range of the sample data
double	8 bytes/sample	Real number within [-1, 1]
single	4 bytes/sample	Real number within [-1, 1]
int16	2 bytes/sample	Integer within [-32768, 32767] or [-2^(nbits-1), 2^(nbits-1)-1]
uint8	1 byte/sample	Integer within [0, 255] or [0, 2^nbits-1]

1. (*) **Obtain info from a mono audio file**： Write a MATLAB script that can read the wave file "welcome.wav" and display the following information within this script.
    a. Number of sample points.
    b. Samping rate.
    c. Bit resolution
    d. Number of channels.
    e. Time duration of the recording (in terms of seconds)
2. (*) **Obtain info from a stereo audio file**： Repeat the previous exercise with a MATLAB program to obtain the same information from the wave file "flanger.wav".
3. (*) **Wave recording**： Write a MATLAB script to record 10 seconds of your utterance such as "My name is Roger Jang and I am a senior student at the CS department of National Tsing Hua University". Save your recording as myVoice.wav. Other recording parameters are: sample rate = 16 KHz, bit resolution = 16 bits. Please use the script print out answers to the following questions within the MATLAB window.
    a. How much space is taken by the audio data in the MATLAB workspace?
    b. What the data type of the audio data?
    c. How do you compute the amount of the required memory from the recording parameters?
    d. What is the size of myVoice.wav?
    e. How many bytes is used in myVoice.wav to record overheads other than the audio data itself?
4. (*) **Reverse playback**： Write a MATLAB script to accomplish the following tasks:
    a. Record your utterance of "we" and play it backwards. Does it sound like "you"? (Please save the result to a wave file and demo its playback to the TA.)
    b. Record your utterance of "you" and play it backwards. Does it sound like "we"? (Please save the result to a wave file and demo its playback to the TA.)
    c. Record your utterance of "上海自來水來自海上" (for Chinese students) or "We are you" (for other students) and play it backwords. What does it sound like? (Please save the result to a wave file and demo its playback to the TA.)
    d. Can you think of any other utterances that sound meaningful when played backwords?
5. (*) **Audio signal manipulation**： Write a MATLAB script to record your utterance of "today is my birthday". Try to explain the playback effect you observe after you try the following operations on the audio signals.
    a. Multiply the audio signals by -1.
    b. Reverse the audio signals in time axis.
    c. Multiply the audio signals by 10.
    d. Replace each sample by its square root.
    e. Replace each sample by its square.
    f. Clip the waveform such that sample data out of the range [-0.5, 0.5] are set to zero.
    g. Modify the waveform such that samples in the range [-0.5, 0.5] are set to zero; samples out of the range [-0.5, 0.5] are moved toward zero by the amount 0.5.
6. (*) **Audio signal grafting**： Write a MATLAB script to accomplish the following tasks. (You need to find the boundaries by trials and errors, and put the related boundary indices into your MATLAB program for creating the required audio segments.)
    - (For Mandarin-speaking student) Record your utterance of "清華大學資訊系" and save it to a file first.

    a. If you connect the consonant part of "大" to the vowel part of "系", can you get the sound of "地"? (Please save the result to a wave file and demo its playback to the TA.)
    b. If you connect "系" to the vowel part of "大", can you get the sound of "下"? (Please save the result to a wave file and demo its playback to the TA.)
  - (For other students) Record your utterance of "keep it simple" and save it to a file.
    a. Can you get the sound of "pimple" by connecting some portions of "Keep" and "simple"? (Please save the result to a wave file and demo its playback to the TA.)
    b. Can you get the sound of "simplest" by connect some portions of your recording? (Please save the result to a wave file and demo its playback to the TA.)

7. (**) **Experiments on the sample rate**： Write a MATLAB script to record your utterance of "my name is ***" with a sample rate of 32 KHz and 8-bit resolution. Try to resample the audio signals at decreasing sample rates of 16 KHz, 8 KHz, 4 KHz, 2 KHz, 1 KHz, and so on. At which sample rate you start to have difficulty in understanding the contents of the utterance?
8. (**) **Experiments on adding noise**：Write a MATLAB script to record your utterance of "my name is ***" with a sample rate of 8 KHz and 8-bit resolution. We can add noise to the audio signals by using the following program snippet:

```
k = 0.1;
    y2 = y + k*randn(length(y), 1);          % Add noise
    sound(y2, 8000);                         % Playback
    plot(y2);
```

   Increase the value of k by 0.1 each time and answer the following questions.
   . At what value of K you start to have difficulty in understanding the content of the playback?
   a. Plot the waveforms at different values of k. At what value of k you start to have difficulty in identifying the fundamental period of the waveform?
13. (**) **Create the illusion of a moving sound source**： In the previous exercise, you have create myVoice.wav which is a mono audio file. Write a MATLAB script that can read the audio data from myVoice.wav, duplicate the audio data to create a stereo audio, and then modify the volume of each channels such that the playback can create an illusion that the sound source is moving between your two speakers. (Hint: You can observe the waveforms of the two channels in "flanger.wav".)
14. (**) **Resample audio signals**： Write a MATLAB script to resample the audio signals in "sunday.wav" such that new waveform has a new sample rate of 11025. Plot these two waveform in the suplot(2, 1, 1). Plot their absolute difference in subplot(2, 1, 2).
15. (*) **基本錄音**： 請用 MATLAB 寫一小段程式，進行錄音三秒，錄音的內容是「清華大學資訊系」，其中取樣頻率是 16 KHz，解析度是 8 位元，請將音訊儲存成 myVoice.wav 檔案。
    . 請問檔案大小為何？
    a. 若將音訊左右顛倒來播放，會有什麼效果？
    b. 若將音訊上下顛倒來播放，或有什麼效果？
    c. 若將音訊乘以 10 倍，或有什麼播放效果？
    d. 若將音訊的大小進行開平方，會有什麼播放效果？
    e. 若將音訊的大小進行平方，會有什麼播放效果？
    f. 若將音訊超過[-0.5, 0.5]的部分均設定為零，會有什麼播放效果？

g. 若將音訊介於[-0.5, 0.5]的部分均砍掉，會有什麼播放效果？

（提示：會用到的指令有 wavrecord, wavwrite, flipud, sign, sound 等。）

16. (*) **讀入整數的音訊資料：** 請寫一個函數 wavRead2.m，其用法和 MATLAB 內建的函數 wavread.m 相同，用法如下：

```
[y, fs, nbits] = wavread2('file.wav');
```

唯一不同點，是所傳回來的音訊變數 y 是整數值，如果 nbits 是 8，則 y 的範圍必須介於 -128 到 127 之間；如果 nbits 是 16，那麼 y 的範圍就會介於 -32768 至 32767 之間。（提示：你必須先瞭解 wavread() 的用法。）

17. (**) **如何建立音框：** 請寫一個函數 buffer2.m，用法如下

```
framedY = buffer2(y, frameSize, overlap);
```

其中 y 是音訊訊號，frameSize 是音框的點數，overlap 則是相鄰音框重疊的點數，framedY 則是一個矩陣，其列數等於音框的點數，行數則等於音框的數目。（若最後幾點的音訊不足以塞滿一個音框，則捨棄這幾點資料。）使用範例如下：

```
>> y=[1 2 3 4 5 6 7 8 9 10 11 12 13 14];
>> buffer2(y, 4, 1)

ans =
     1     4     7    10
     2     5     8    11
     3     6     9    12
     4     7    10    13
```

另，請問這個函數 buffer2.m 和 Signal Processing Toolbox 中的 buffer 函數有何不同？

18. (*) **取樣頻率的影響：** 請用 MATLAB 寫一小段程式，進行錄音兩秒，錄音的內容是「我是某某某」，其中取樣頻率是 32 KHz，解析度是 8 位元，請將音訊儲存成 myVoice2.wav 檔案。請對訊號進行重新取樣（Resample），讓取樣頻率變成 16 KHz, 8 KHz, 4 KHz, 2 KHz ...等等，請問當取樣頻率掉到多低時，你已經聽不出來原先的聲音？

19. (*) **雜訊的影響：** 請用 MATLAB 寫一小段程式，進行錄音兩秒，錄音的內容是「我是某某某」，其中取樣頻率是 8 KHz，解析度是 8 位元。假設音訊訊號是存在一個行向量 y，我們可以以下列方式加入雜訊：

```
20.         k = 0.1;
21.         y2 = y + k*randn(length(y), 1);                % 加入雜訊
22.         sound(y2, 8000);                               % 放音
23.         plot(y2);
```

當 k 值由 0.1、0.2、0.3 等慢慢增大時，慢慢你的聲音會越來越模糊。

. 請問到 k 值是多少時，你會聽不出來原先講話的內容？

a. 請問到 k 值是多少時，你會看不出來 y2 包含一段聲音的訊號？（換言之，在放大 y2 的圖形後，你已經看不出來有基本週期的存在。）

24. (**) **重新取樣：** 請用 interp1 指令，對 myVoice2.wav 的音訊進行重新取樣，讓取樣頻率變成 11025 Hz，並將結果儲存成 myVoice3.wav。

25. (*) **時間反轉播放：** 在下列的錄音及播放過程，請自行選定錄音參數，錄音時間 3 秒即可。

. 請錄製聲音「上海自來水來自海上」，請先正向播放一次，然後再翻轉時間軸來反向播放一次，聽看看有時麼差別。你可以從反向播放的聲音，預測原先正向播放的聲音嗎？

a. 請錄製聲音「we are you」，盡量放平聲調。請先正向播放一次，然後再翻轉時間軸來反向播放一次，聽看看有時麼差別。為何麼反向播放和正向播放聽到類似的聲音？中文是否有類似的範例？

（提醒：假設錄音所得的音訊向量是 y，可以使用 flipud(y) 來進行上下翻轉或 fliplr(y) 來進行左右翻轉。）

26. (*) **音訊剪接 1：** 請用 MATLAB 完成下列事項：
    . 先進行錄音，取樣頻率為 16 KHz，解析度為 16 Bits，錄音時間三秒，錄音內容是「清華大學資訊系」，請問檔案大小為何？如何直接由錄音設定來計算此大小？
    a. 請觀察波形，將所有的氣音設定為靜音（訊號值為零），存檔後播放聽看看。是否能聽出來原先的內容？
    b. 請觀察波形，將所有的母音設定為靜音（訊號值為零），存檔後播放聽看看。是否能聽出來原先的內容？
    c. 若將「大」的前半部（ㄉ）接到「系」的後半部（ㄧ），會不會得到「低」的聲音？剪接後存成 wav 檔，再播放出來聽看看。
    d. 若將「系」的前半部（ㄒ）接到「清」的後半部（ㄧㄥ），會不會得到「星」的聲音？剪接後存成 wav 檔，再播放出來聽看看。
27. (*) **音訊剪接 2：** 請改用 CoolEdit 來完成上一題。
28. (***) **單聲道變雙聲道：** 本題將一個單聲道音訊檔案經過處理後，變成一個左右游移的雙聲道音訊檔案。
    . 請將此檔案 flanger.wav 的左右聲道波形畫出來。由於左右聲道的音量漸進互補，因此在播放時，會產生音源在左右聲道游移的效果。
    a. 請使用 load handel.mat 來載入一個音訊 y 及取樣頻率 Fs，請仿照 flanger.wav 的方式，產生一個雙聲道的音訊檔案 handel.wav，使其聲音游移的特性和 flanger.wav 類似。

Chapter 5: Basic Features of Audio Signals (音訊的基本特徵)

5-1 Volume (音量)

The loudness of audio signals is the most prominant features for human aural perception. In general, there are several terms to describe the loudness of audio signals, including volume, intensity, and energy. Here we use the term "volume" for further discussion. Volume is a basic acoustic feature that is correlated to the sample amplitudes within each frame. Basically, there are two methods to compute the volume of each frame:

1. The sum of absolute samples within each frame:
   $$\text{volume} = \Sigma_{i=1}^{n} |s_i|$$
   where s_i is the i-th sample within a frame, and n is the frame size. This method requires only integer operations and it is suitable for low-end platform such as micro-controllers.
2. 10 times the 10-based logorithm of the sum of sample squares:
   $$\text{volume} = 10*\log(\Sigma_{i=1}^{n} s_i^2)$$
   This method requires more floating-point computations, but it is (more or less) linearly correlated to our perception of loudness of audio signals. The quantity computed is also referred as the "log energy" in the unit of decibels. More explanations about decibel can be found in the page:
   http://www.phys.unsw.edu.au/~jw/dB.html（local copy）

「音量」代表聲音的強度，又稱為「力度」、「強度」（Intensity）或「能量」（Energy），可由一個音框內的訊號震幅大小來類比，基本上兩種方式來計算：

1. 每個音框的絕對值的總和：這種方法的計算較簡單，只需要整數運算，適合用於低階平台（如微電腦等）。
2. 每個音框的平方值的總和，再取以 10 為底對數值，再乘以 10：這種方法得到的值是以分貝（Decibels）為單位，是一個相對強度的值，比較符合人耳對於大小聲音的感覺。以下網頁有對分貝的詳細說明：
   http://www.phys.unsw.edu.au/~jw/dB.html（近端備份）

Some characteristics of volume are summarized next.

- For recording in a common office using a uni-directional microphone, the volume of voiced sounds is larger than that of unvoiced sounds, and the volume of unvoiced sounds is also larger than that of environmental noise.
- Volume is greatly influenced by microphone setups, mostly the microphone gain.
- Volume is usually used for endpoint detection which tries to find the region of meanful voice activity.
- Before computing the volume, it is advised that the frame should be zero-justified (the average value of the frame should be subtracted from each samples) to avoid the common DC-bias due to hardware defects.

音量具有下列特性：

- 一般而言，有聲音的音量大於氣音的音量，而氣音的音量又大於雜訊的音量。
- 是一個相對性的指標，受到麥克風設定的影響很大。
- 通常用在端點偵測，估測有聲之音母或韻母的開始位置及結束位置。
- 在計算前最好先減去音訊訊號的平均值，以避免訊號的直流偏移（DC Bias）所導致的誤差。

The following example demonstrate how to use these two mehtods for volume computation:
以下顯示如何以兩種方法來計算音量：

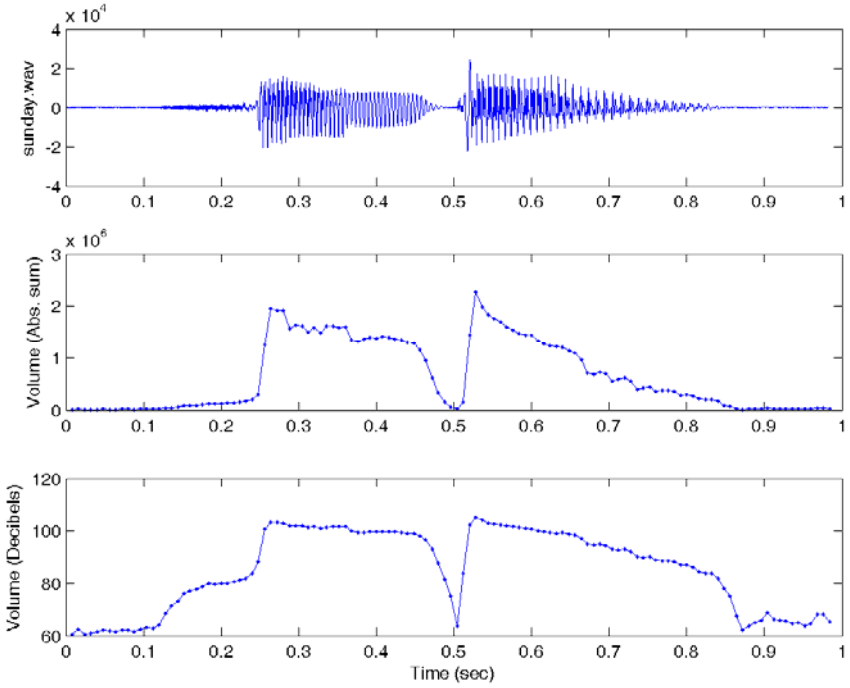
Example 1**Input file** basicFeature/volume01.m

```
waveFile='sunday.wav';
frameSize=256;
overlap=128;

[y, fs, nbits]=wavReadInt(waveFile);
fprintf('Length of %s is %g sec.\n', waveFile, length(y)/fs);
frameMat=buffer(y, frameSize, overlap);
frameNum=size(frameMat, 2);
volume1=zeros(frameNum, 1);
volume2=zeros(frameNum, 1);
for i=1:frameNum
        frame=frameMat(:,i);
        frame=frame-mean(frame);          % zero-justified
        volume1(i)=sum(abs(frame));               % method 1
        volume2(i)=10*log10(sum(frame.^2));       % method 2
end

time=(1:length(y))/fs;
frameTime=((0:frameNum-1)*(frameSize-overlap)+0.5*frameSize)/fs;
subplot(3,1,1); plot(time, y); ylabel(waveFile);
subplot(3,1,2); plot(frameTime, volume1, '.-'); ylabel('Volume (Abs. sum)');
subplot(3,1,3); plot(frameTime, volume2, '.-'); ylabel('Volume (Decibels)'); xlabel('Time (sec)');
```

Output message

```
Length of sunday.wav is 0.98225 sec.
```

Output figure

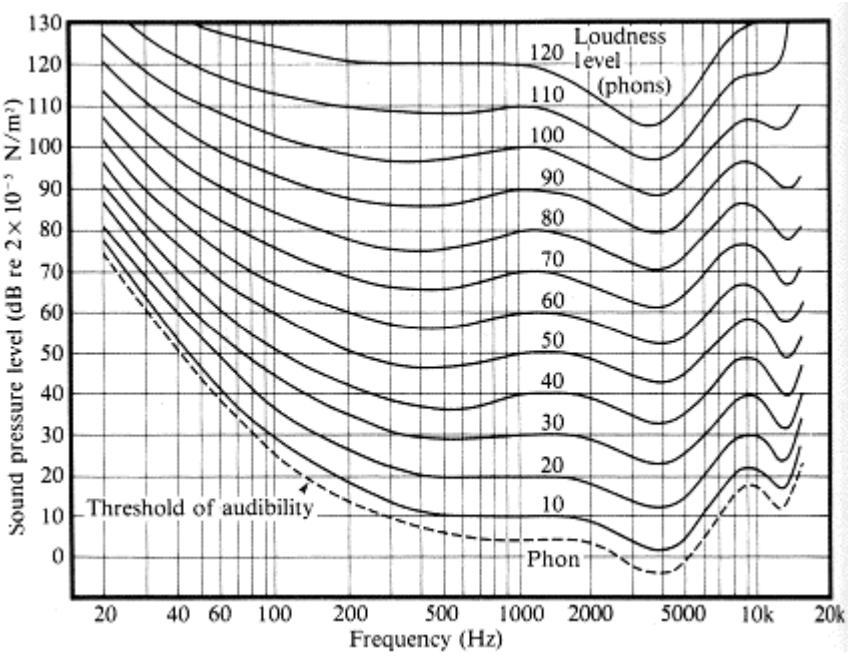


Hint

Note that in the above example, we have use a function wavReadInt() which converts all the samples into integer values. This function is available in the Audio Processing Toolbox.

The above two methods of computing volume are only an approximation to our perception of loudness. However, the loudness is based on our perception and there could be significant differences between the "computed loudness" and the "perceived loudness". In fact, the perceived loudness is greatly affect by the frequcney as well as the timber of the audio signals. If we plot the equal percieved loudness against sinusoidal signals of various frequencies, we will have the following curves of equal loudness:

基本上我們使用音量來表示聲音的強弱，但是前述兩種計算音量的方法，只是用數學的公式來逼近人耳的感覺，和人耳的感覺有時候會有相當大的落差，為了區分，我們使用「主觀音量」來表示人耳所聽到的音量大小。例如，人耳對於同樣振福但不同頻率的聲音，所產生的主觀音量就會非常不一樣。若把以人耳為測試主體的「等主觀音量曲線」（Curves of Equal Loudness）畫出來，就可以得到下面這一張圖：



Curves of equal loudness determined experimentally by Fletcher, H. and Munson, W.A. (1933) J.Acoust.Soc.Am. 6:59.

The above figure also shows the sensitivity of the human ear with respect to frequency, which is simply the frequency response of the human ear. If you want to obtain the frequency response of your ears, you can jumpt the the "Equal Loudness Tester" pages:

上面這一張圖，也代表人耳對於不同頻率的聲音的靈敏程度，這也就是人耳的頻率響應（Frequency Response）。如果你要測試你自己的耳朵的頻率響應，可以到這個網頁「Equal Loudness Tester」試試看：

http://www.phys.unsw.edu.au/~jw/hearing.html（近端備份）

Besides frequencies, the perceived loudness is also greatly influenced by the timbre. For instance, we can try to pronounce several vowels using the same loudness level, and then plot the volume curves to see how they are related to the timbre or shapes/positions of lips/tougue, as shown in the following example.

主觀音量除了和頻率有關外，也和音訊的內容（音色或是基本週期的波形）有關，例如，我們可以盡量使用相同的主觀音量來錄下幾個發音比較單純的母音（ㄚ、ㄧ、ㄨ、ㄝ、ㄛ、ㄜ、ㄩ），然後再用音量公式來算它們的音量，就應該可以看出來音量公式和發音嘴型的關係。

Example 2**Input file** basicFeature/volume02.m

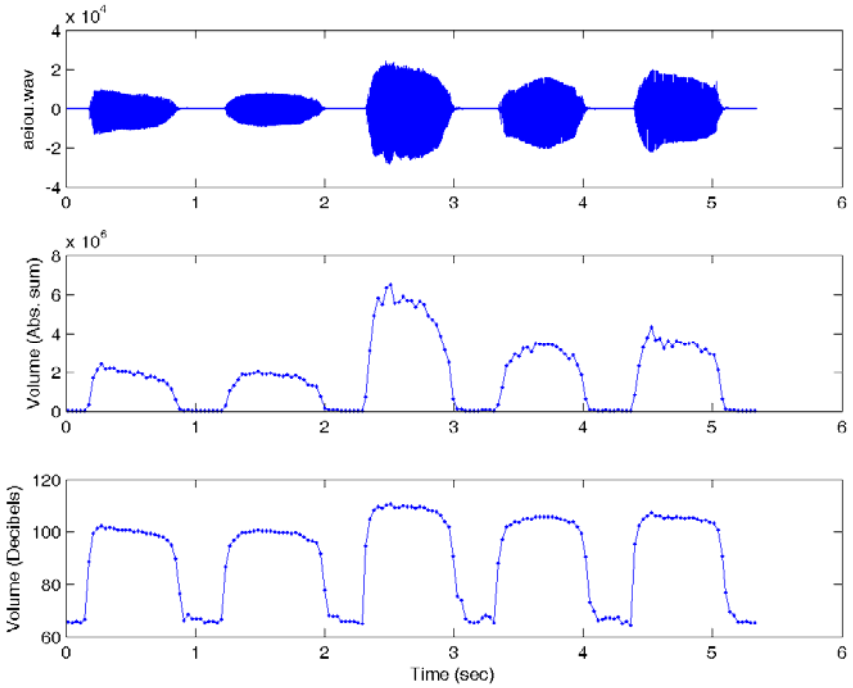
```
waveFile='aeiou.wav';
frameSize=512;
overlap=0;

[y, fs, nbits]=wavReadInt(waveFile);
fprintf('Length of %s is %g sec.\n', waveFile, length(y)/fs);
frameMat=buffer(y, frameSize, overlap);
frameNum=size(frameMat, 2);
volume1=frame2volume(frameMat, 1);          % method 1
volume2=frame2volume(frameMat, 2);          % method 2

time=(1:length(y))/fs;
frameTime=((0:frameNum-1)*(frameSize-overlap)+0.5*frameSize)/fs;
subplot(3,1,1); plot(time, y); ylabel(waveFile);
subplot(3,1,2); plot(frameTime, volume1, '.-'); ylabel('Volume (Abs. sum)');
```

```
subplot(3,1,3); plot(frameTime, volume2, '.-'); ylabel('Volume (Decibels)'); xlabel('Time (sec)');
```

Output message

```
Length of aeiou.wav is 5.337 sec.
```

Output figure



Hint

In the above example, we have use a function frame2volume() which computes the volume using two methods. This function is available in the Audio Processing Toolbox.

From the above example, you can observed that though the perceived loudness is the same, the computed volumes depend a lot on the timbers. In fact, we can perform another experiment to pronounce the same vowel but with different pitch to see how the perceived loudness depends on the fundamental frequency. This is left as an exercise.

Since the perceived loudness is easily affected by the fundamental frequency as well as the timber, we need to adjust the amplitudes accordingly when we are performing text-to-speech synthesis or singing voice synthesis.

主觀音量容易受到頻率和音色的影響，因此我們在進行語音或歌聲合成時，常常根據聲音的頻率和內容來對音訊的振福進行校正，以免造成主觀音量忽大忽小的情況。

5-2 Zero Crossing Rate (過零率)

Zero-crossing rate (ZCR) is another basic acoustic features that can be computed easily. It is equal to the number of zero-crossing of the waveform within a given frame. ZCR has the following characteristics:

- In general, ZCR of both unvoiced sounds and environment noise are larger than voiced sounds (which has observable fundamental periods).
- It is hard to distinguish unvoiced sounds from environment noise by using ZCR alone since they have similar ZCR values.

- ZCR is often used in conjunction with the volume for end-point detection. In particular, ZCR is used for detecting the start and end positings of unvoiced sounds.
- Some people use ZCR for fundamental frequency estimation, but it is highly unreliable unless further refine procedure is taken into consideration.

「過零率」（Zero Crossing Rate，簡稱 ZCR）是在每個音框中，音訊通過零點的次數，具有下列特性：

- 一般而言，雜訊及氣音的過零率均大於有聲音（具有清晰可辨之音高，例如母音）。
- 是雜訊和氣音兩者較難從過零率來分辨，會依照錄音情況及環境雜訊而互有高低。但通常氣音的音量會大於雜訊。
- 通常用在端點偵測，特別是用在估測氣音的啟始位置及結束位置。
- 可用來預估訊號的基頻，但很容易出錯，所以必須先進行前處理。

The following facts should be well understood when we try to implement ZCR:

1. If a sample is exactly located at zero, should we count it as zero crossing? Depending on the answer to this question, we have two method for ZCR computation.
2. Most ZCR computation is based on the integer values of audio signals. If we want to do mean substraction, the mean value should be rounded to the nearest integer too.

一般而言，在計算過零率時，需注意下列事項：

1. 由於有些訊號若恰好位於零點，此時過零率的計算就有兩種，出現的效果也會不同。因此必須多加觀察，才能選用最好的作法。
2. 大部分都是使用音訊的原始整數值來進行，才不會因為使用浮點數訊號，在減去直流偏移（DC Bias）時，造成過零率的增加。

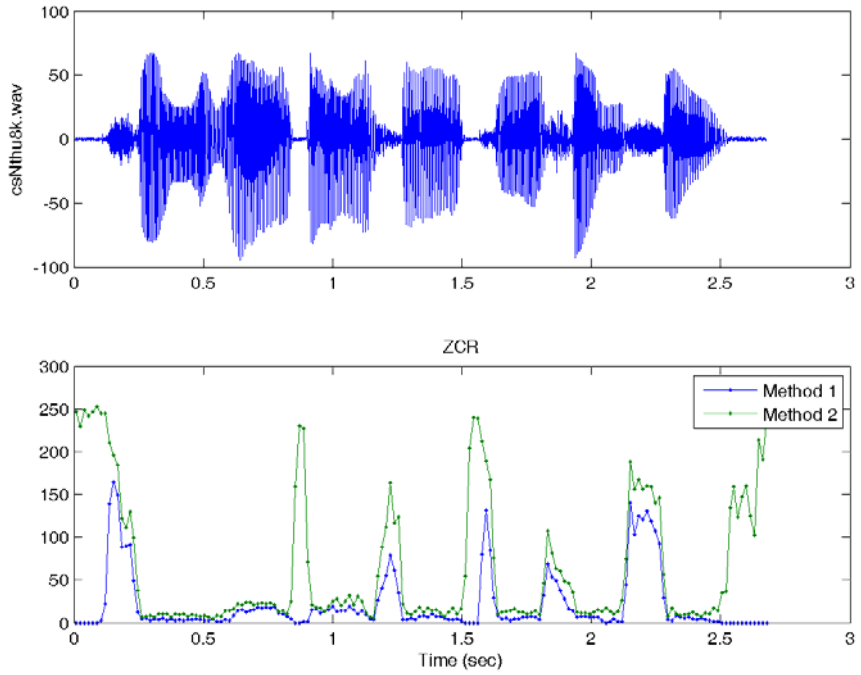
In the following, we use the above-mentioned two methods for ZCR computation of the wave file csNthu8b.wav:

在以下範例中，我們使用兩種不同的方法來計算過零率：

Example 1**Input file** basicFeature/zcr01.m

```
waveFile='csNthu8b.wav';
frameSize=256;
overlap=0;

[y, fs, nbits]=wavReadInt(waveFile);
frameMat=buffer(y, frameSize, overlap);
zcr1=sum(frameMat(1:end-1, :).*frameMat(2:end, :)<=0);                    % Method 2
time=(1:length(y))/fs;
frameNum=size(frameMat, 2);
frameTime=((0:frameNum-1)*(frameSize-overlap)+0.5*frameSize)/fs;

subplot(2,1,1); plot(time, y); ylabel(waveFile);
subplot(2,1,2); plot(frameTime, zcr1, '.-', frameTime, zcr2, '.-');
title('ZCR'); xlabel('Time (sec)');
legend('Method 1', 'Method 2');
```

Output figure



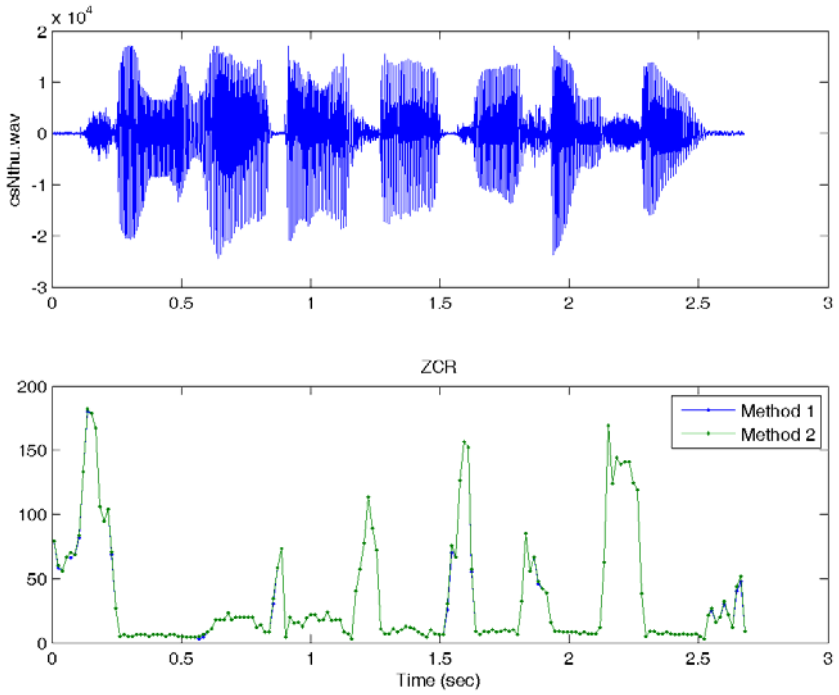
From the above example, it it obvious that these two methods generate different ZCR curves. The first method does not count "zero positioning" as "zero crossing", there the corresponding ZCR values are smaller. Moreover, silence is likely to have low ZCR of method 1 and high ZCR for method 2 since there are likely to have many "zero positioning" in the silence region. However, this observation is only true for low sample rate (8 KHz in this case). For the same wave file with 16 KHz (csNthu.wav), the result is shown next:

在上述的範例中，我們使用了兩種方式來計算過零率，得到的效果雖然不同，但趨勢是一致的。（另外有一種情況，當錄音環境很安靜時，靜音的訊號值都在零點或零點附近附近跳動時，此時是否計算位於零點的過零率，就會造成很大的差別。）如果取樣頻率提高，得到的結果也會不同：

Example 2**Input file** basicFeature/zcr02.m

```
waveFile='csNthu.wav';
frameSize=256;
overlap=0;

[y, fs, nbits]=wavReadInt(waveFile);
frameMat=buffer(y, frameSize, overlap);
zcr1=frame2zcr(frameMat, 1);                    % Method 1
zcr2=frame2zcr(frameMat, 2);                    % Method 2
time=(1:length(y))/fs;
frameNum=size(frameMat, 2);
frameTime=((0:frameNum-1)*(frameSize-overlap)+0.5*frameSize)/fs;

subplot(2,1,1); plot(time, y); ylabel(waveFile);
subplot(2,1,2); plot(frameTime, zcr1, '.-', frameTime, zcr2, '.-');
title('ZCR'); xlabel('Time (sec)');
legend('Method 1', 'Method 2');
```

Output figure

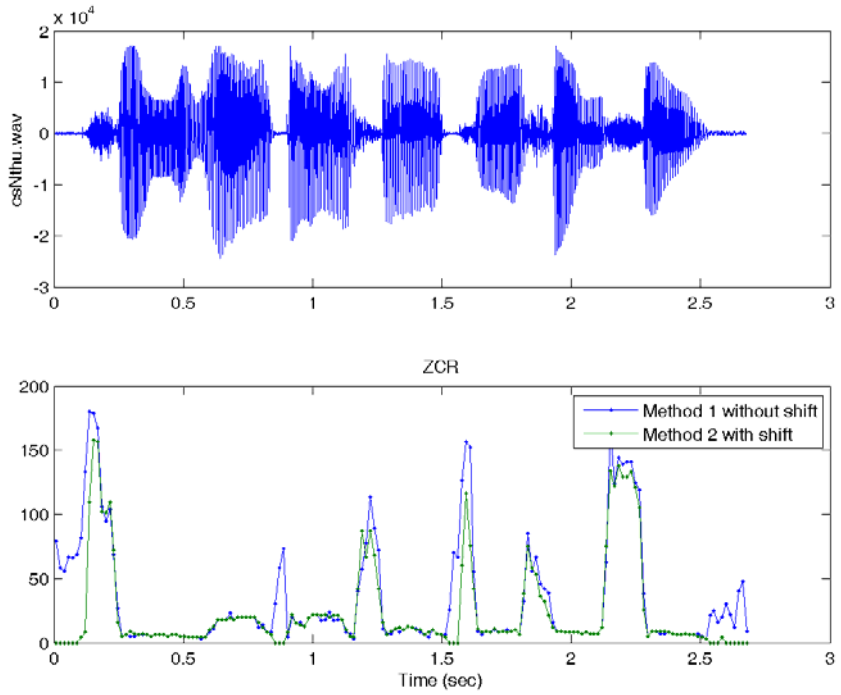


In the above example, methods 1 and 2 return similar ZCR curves. In order to used ZCR to distinguish unvoiced sounds from environment noise, we can shift the waveform before computing ZCR. This is particular useful is the noise is not too big. Example follows:

Example 3**Input file** basicFeature/zcr03.m

```
waveFile='csNthu.wav';
frameSize=256;
overlap=0;

[y, fs, nbits]=wavReadInt(waveFile);
frameMat=buffer(y, frameSize, overlap);
volume=frame2volume(frameMat);
[minVolume, index]=min(volume);
shiftAmount=2*max(abs(frameMat(:,index)));  % shiftAmount is equal to the max. abs. sample within the frame of min. volume
zcr1=frame2zcr(frameMat, 1);
zcr2=frame2zcr(frameMat, 1, shiftAmount);

subplot(2,1,1); plot(time, y); ylabel(waveFile);
subplot(2,1,2); plot(frameTime, zcr1, '.-', frameTime, zcr2, '.-');
title('ZCR'); xlabel('Time (sec)');
legend('Method 1 without shift', 'Method 2 with shift');
```

Output figure



In this example, the shift amount is equal to the maximal absolute sampe values within the frame with the minimum volume. Therefore the ZCR of the silence is reduced dratically, making it easier to tell unvoiced sounds from silence using ZCR.
If we want to detect the meaningful voice activity of a stream of audio signals, we need to perform end-point detection or speech detection. The most straight method for end-point detection is based on volume and ZCR. Please refer to the next chapter for more information.
若要偵測聲音的開始和結束，通常稱為「端點偵測」（Endpoint Detection）或「語音偵測」（Speech Detection），最簡單的方法就是使用音量和過零率來判別，相關細節會在後續章節說明。

5-3 Pitch (音高)

Old Chinese version

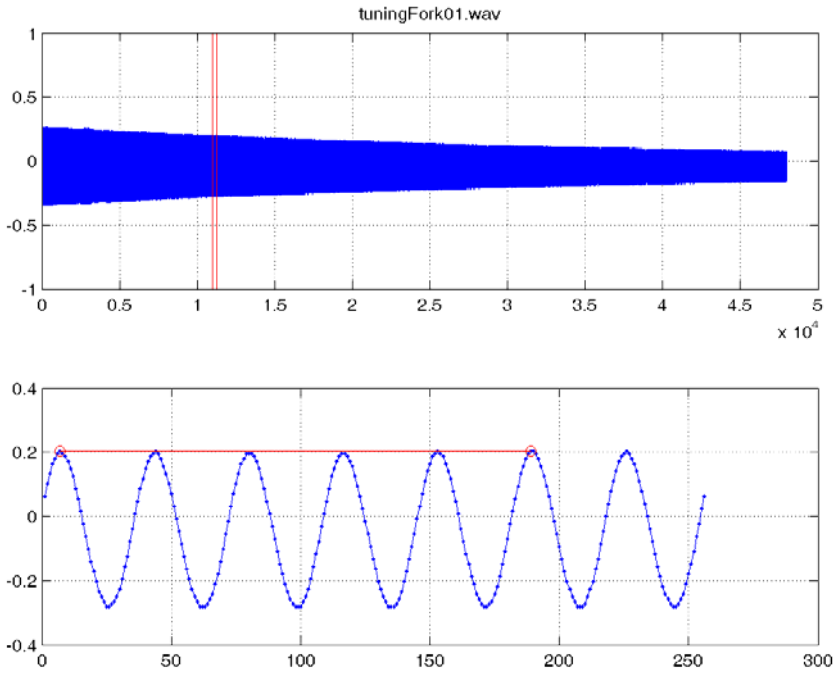
Pitch is an important feature of audio signals, especially for quasi-periodic signals such as voiced sounds from human speech/singing and monophonic music from most music instruments. Intuitively speaking, pitch represent the vibration frequency of the sound source of audio signals. In other words, pitch is the fundamental frequency of audio signals, which is equal to the reciprocal of the fundamental period.
「音高」（Pitch）是另一個音訊裡面很重要的特徵，直覺地說，音高代表聲音頻率的高低，而此頻率指的是「基本頻率」（Fundamental Frequency），也就是「基本週期」（Fundamental Period）的倒數。
Generally speaking, it is not too difficult to observe the fundamental period within a quasi-periodic audio signals. Take a 3-second clip of a tuning fork tuningFork01.wav for example. We can first plot a frame of 256 sample points and identify the fundamental period easily, as shown in the following example.
若直接觀察音訊的波形，只要聲音穩定，我們並不難直接看到基本週期的存在，以一個 3 秒的音叉聲音來說，我們可以取一個 256 點的音框，將此音框畫出來後，就可以很明顯地看到基本週期，請見下列範例：
Example 1**Input file** basicFeature/pitchTuningFork01.m

```
waveFile='tuningFork01.wav';
[y, fs, nbits]=wavread(waveFile);
index1=11000;
frameSize=256;
index2=index1+frameSize-1;
frame=y(index1:index2);

subplot(2,1,1); plot(y); grid on
title(waveFile);
line(index1*[1 1], [-1 1], 'color', 'r');
line(index2*[1 1], [-1 1], 'color', 'r');
subplot(2,1,2); plot(frame, '.-'); grid on
point=[7, 189];
line(point, frame(point), 'marker', 'o', 'color', 'red');

periodCount=5;
fp=((point(2)-point(1))/periodCount)/fs;          % fundamental period (in sec)
ff=1/fp;                                          % fundamental frequency (in Hz)
pitch=69+12*log2(ff/440);                         % pitch (in semitone)
fprintf('Fundamental period = %g second\n', fp);
fprintf('Fundamental frequency = %g Hertz\n', ff);
fprintf('Pitch = %g semitone\n', pitch);
```

Output message

```
Fundamental period = 0.002275 second
Fundamental frequency = 439.56 Hertz
Pitch = 68.9827 semitone
```

Output figure



In the above example, the two red lines in the first plot define the start and end of the frame for our analysis. The second plot shows the waveform of the frame as well as two points (identified visually) which cover 5 fundamental periods. Since the distance between these two points is 182 units, the fundamental frequency is fs/(182/5) = 16000/(182/5) = 439.56 Hz, which is equal to 68.9827 semitones. The formula for the conversion from pitch frequency to semitone is shown next.

在上述範例中，上圖紅線的位置代表音框的位置，下圖即是 256 點的音框，其中紅線部分包含了 5 個基本週期，總共佔掉了 182 單位點，因此對應的基本頻率是 fs/(182/5) = 16000/(182/5) = 439.56 Hz，相當於 68.9827 半音（Semitone），其中由基本頻率至半音的轉換公式如下：

$$\text{semitone} = 69 + 12*\log_2(\text{frequency}/440)$$

In other words, when the fundamental frequency is 440 Hz, we have a pitch of 69 semitones, which corresponds to "central la" or A4 in the following piano roll.

換句話說，當基本頻率是 440 Hz 時，對應到的半音差是 69，這就是鋼琴的「中央 La」或是「A4」，請見下圖。

Hint

The fundamental frequency of the tuning fork is designed to be 440 Hz. Hence the tuning fork are usually used to fine tune the pitch of a piano.

一般音叉的震動頻率非常接近 440 Hz，因此我們常用音叉來校正鋼琴的音準。

In fact, semitone is also used as unit for specify pitch in MIDI files. From the conversion formula, we can also notice the following facts:

- Each octave contains 12 semitones, including 7 white keys and 5 black ones.
- Each transition to go up one octave corresponds to twice the frequency. For instance, the A4 (central la) is 440 Hz (69 semitones) while A5 is 880 Hz (81 semitones).

- Pitch in terms of semitones (more of less) correlates linearly to human's "perceived pitch".

上述公式所轉換出來的半音差，也是 MIDI 音樂檔案所用的標準。從上述公式也可以看出：

- 每個全音階包含 12 個半音（七個白鍵和五個黑鍵）。
- 每向上相隔一個全音階，頻率會變成兩倍。例如，中央 la 是 440 Hz（69 Semitones），向上平移一個全音階之後，頻率就變成 880 Hz（81 Semitones）。
- 人耳對音高的「線性感覺」是隨著基本頻率的對數值成正比。

The waveform of the tuning fork is very "clean" since it is very close to a sinusoidal signal and the fundamental period is very obvious. In the following example, we shall use human's speech as an examle of visual determination of pitch. The clip is my voice of "清華大學資訊系" (csNthu.wav). If we take a frame around the character "華", we can visually identify the fundamental period easily, as shown in the following example.

音叉的聲音非常乾淨，整個波形非常接近弦波，所以基本週期顯而易見。若以我的聲音「清華大學資訊系」來說，我們可以將「華」的部分放大，也可以明顯地看到基本週期，請見下列範例：

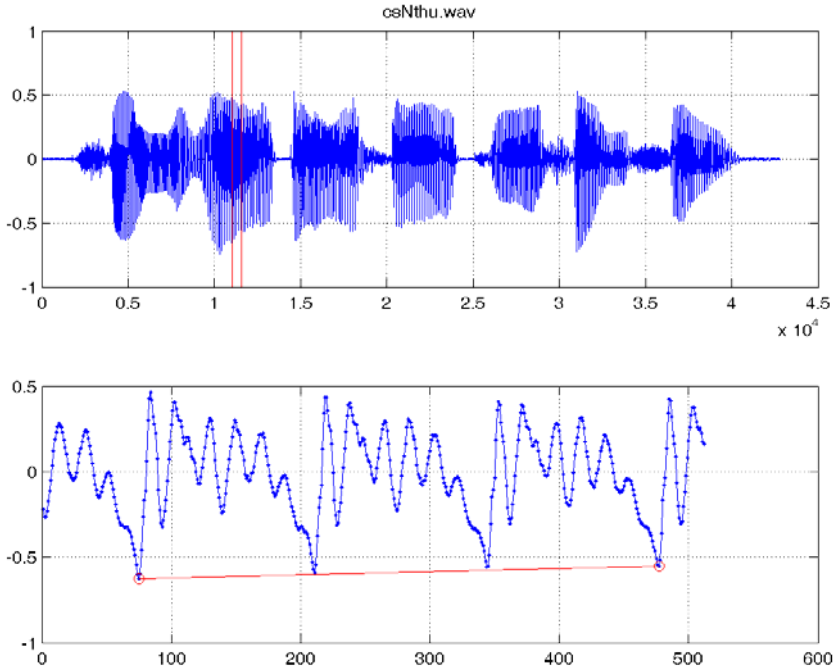
Example 2**Input file** basicFeature/pitchVoice01.m

```
waveFile='csNthu.wav';
[y, fs, nbits]=wavread(waveFile);
index1=11050;
frameSize=512;
index2=index1+frameSize-1;
frame=y(index1:index2);

subplot(2,1,1); plot(y); grid on
title(waveFile);
line(index1*[1 1], [-1 1], 'color', 'r');
line(index2*[1 1], [-1 1], 'color', 'r');
subplot(2,1,2); plot(frame, '.-'); grid on
point=[75, 477];
line(point, frame(point), 'marker', 'o', 'color', 'red');

periodCount=3;
fp=((point(2)-point(1))/periodCount)/fs;          % fundamental period
ff=fs/((point(2)-point(1))/periodCount);          % fundamental frequency
pitch=69+12*log2(ff/440);
fprintf('Fundamental period = %g second\n', fp);
fprintf('Fundamental frequency = %g Hertz\n', ff);
fprintf('Pitch = %g semitone\n', pitch);
```

Output message

```
Fundamental period = 0.008375 second
Fundamental frequency = 119.403 Hertz
Pitch = 46.42 semitone
```

Output figure



In the above example, we select a 512-point frame around the vowel of the character "華". In particular, we chose two points (with indices 75 and 477) that covers 3 complete fundamental periods. Since the distance between these two points is 402, the fundamental frequency is fs/(402/3) = 16000/(402/3) = 119.403 Hz and the pitch is 46.420 semitones.

上列範例的下圖，是從「華」的韻母附近抓出來的 512 點的音框，其中紅線部分包含了 3 個基本週期，總共佔掉了 402 單位點，因此對應的基本頻率是 fs/(402/3) = 16000/(402/3) = 119.403 Hz，相當於 46.420 半音，與「中央 La」差了 22.58 個半音，接近但還不到兩個全音階（24 個半音）。

Conceptually, the most obvious sample point within a fundamental period is often referred to as the pitch mark. Usually pitch marks are selected as the local maxima or minima of the audio waveform. In the previous example of pitch determination for the tuning fork, we used two pitch marks that are local maxima. On the other hand, in the example of pitch determination for human speech, we used two pitch marks that are local minima instead since they are more obvious than local maxima. Reliable identification of pitch marks is an essential task for text-to-speech synthesis.

在觀察音訊波形時，每一個基本週期的開始點，我們稱為「音高基準點」（Pitch Marks，簡稱 PM），PM 大部分是波形的局部最大點或最小點，例如在上述音叉的範例中，我們抓取的兩個 PM 是局部最大點，而在我的聲音的範例中，由於 PM 在局部最大點並不明顯，因此我們抓取了兩個局部最小點的 PM 來計算音高。PM 通常用來調節一段聲音的音高，在語音合成方面很重要。

Due to the difference in physiology, the pitch ranges for males ane females are different:

- The pitch range for males is 35 ~ 72 semitones, or 62 ~ 523 Hz.
- The pitch range of females is 45 ~ 83 semitones, or 110 ~ 1000 Hz.

由於生理構造不同，男女生的音高範圍並不相同，一般而言：

- 男生的音高範圍約在 35 ~ 72 半音，對應的頻率是 62 ~ 523 Hz。
- 女生的音高範圍約在 45 ~ 83 半音，對應的頻率是 110 ~ 1000 Hz。

However, it should be emphasized that we are not using pitch alone to identify male or female voices. Moreover, we also use the information from timbre (or more precisely, formants) for such task. More information will be covered in later chapters.

但是我們分辨男女的聲並不是只憑音高，而還是依照音色（共振峰），詳見後續說明。

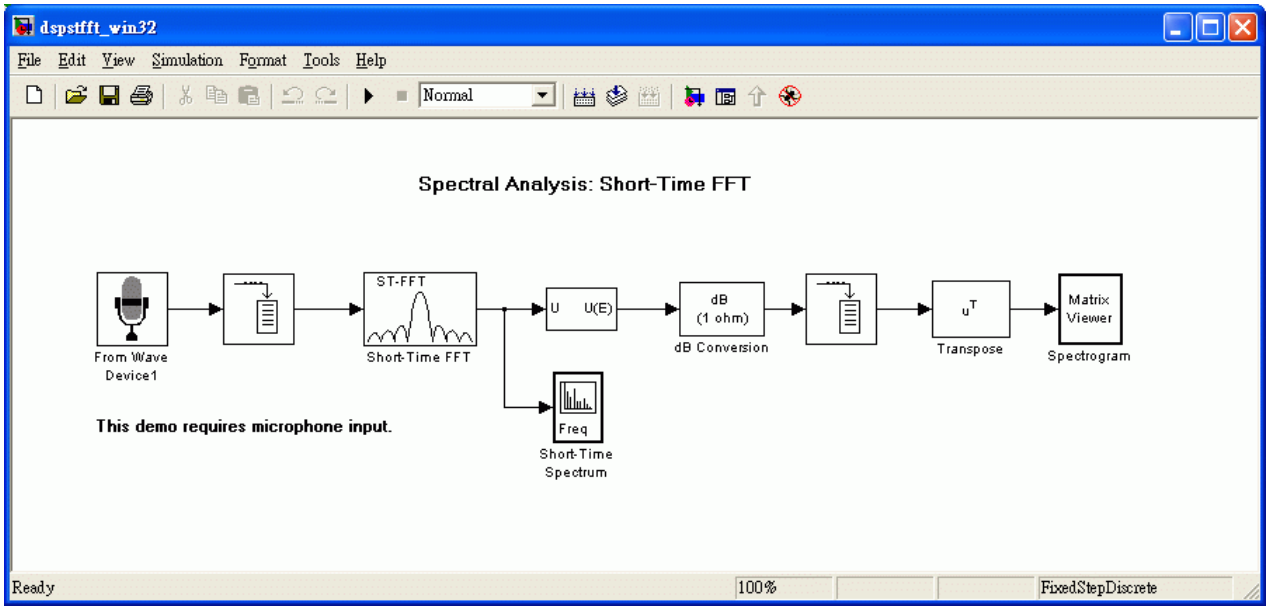
As shown in this section, visual identification of the fundamental frequency is not a difficult task for human. However, if we want to write a program to identify the pitch automatically, there are much more we need to take into consideration. More details will be followed in the next few chapters.
使用「觀察法」來算出音高，並不是太難的事，但是若要電腦自動算出音高，就需要更深入的研究。有關音高追蹤的各種方法，會在後續章節詳細介紹。

5-4 Timbre (音色)

Timbre is an acoustic feature that is defined conceptually. In general, timbre refers to the "content" of a frame of audio signals, which is ideally not affected much by pitch and intensity. Theoretically, for quasi-periodic audio signals, we can use the waveform within a fundamental period as the timbre of the frame. However, it is difficult to analysis the waveform within a fundamental period directly. Instead, we usually use the fast Fourier transform (or FFT) to transform the time-domain waveform into frequency-domain spectrum for further analysis. The amplitude spectrum in the frequency domain simply represent the intensity of the waveform at each frequency band.
「音色」（Timber）是一個很模糊的名詞，泛指音訊的內容，例如「天書」這兩個字的發音，雖然都是第一聲，因此它們的音高應該是蠻接近的，但是由於音色的不同，我們可以分辨這兩個音。直覺來看，音色的不同，代表基本週期的波形不同，因此我們可以使用基本週期的波形來代表音色。若要從基本週期的波形來直接分析音色，是一件很困難的事。通常我們的作法，是將每一個音框進行頻譜分析（Spectral Analysis），算出一個音框訊號如何可以拆解成在不同頻率的分量，然後才能進行比對或分析。在頻譜分析時，最常用的方法就是「快速傅立葉轉換」(Fast Fourier Transform)，簡稱 FFT，這是一個相當實用的方法，可以將在時域（Time Domain）的訊號轉換成在頻域（Frequency Domain）的訊號，並進而知道每個頻率的訊號強度。
If you want to experience real-time FFT demo, type the following command within the MATLAB command window:
若要看看 FFT 的實際展示，可以輸入下列指令：

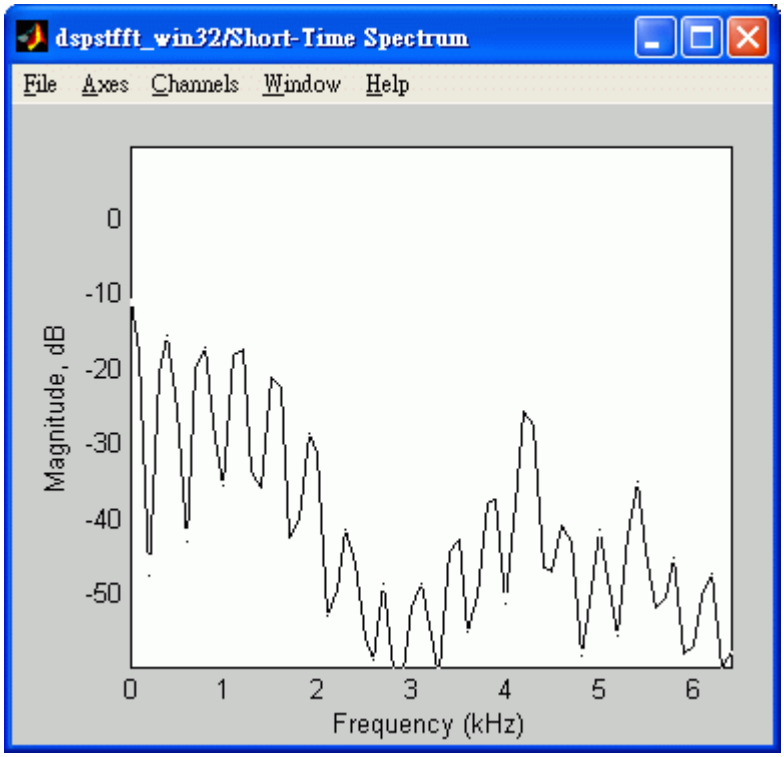
- dspstfft_nt (MATLAB 5)
- dspstfft_win32 (MATLAB 6 and 7)

The opened Simulink block system looks like this:
開啟的 Simulink 系統如下：



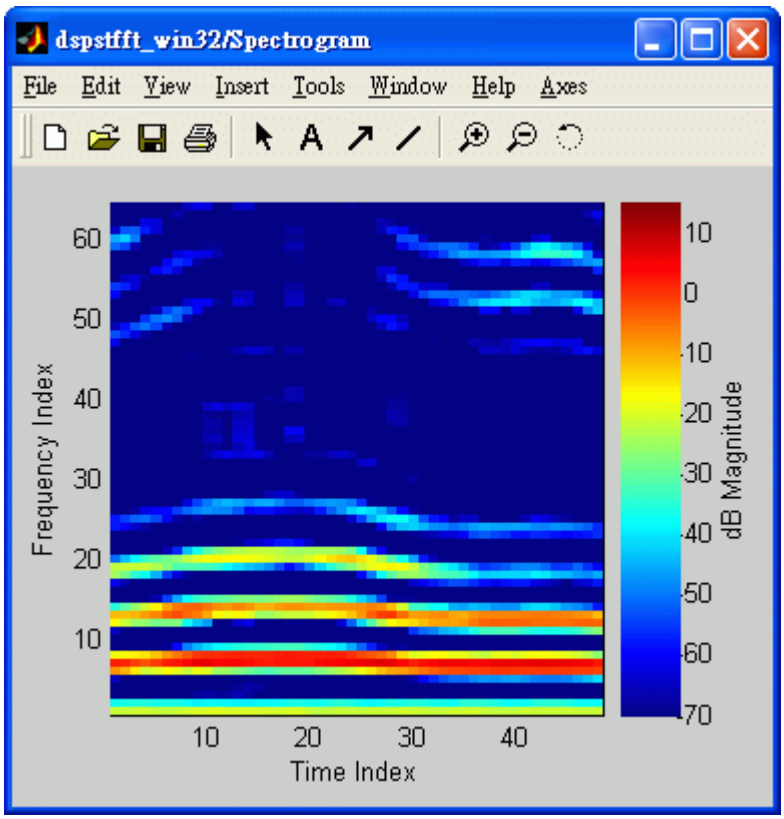
When you start running the system and speak to the microphone, you will able to see the time-varying spectrum:

當你啟動程式並開始對麥克風說話時，就會出現下列動態的「頻譜圖」（Spectrum），隨時間而呈現急遽的變化：



If we use different colors to represent the height of spectrum, we can obtain the spectrogram, as shown next:

若將頻譜圖「立」起來，並用不同的顏色代表頻譜圖的高低，就可以得到頻譜對時間所產生的影像，稱為 Spectrogram，如下：



Spectrogram represent the time-varying spectrum displayed in a image map. The same utterance will correspond to the same pattern of spectrogram. Some experienced persons can understand the contents of the speech by viewing the spectragram alone. This is call "spectrogram reading" and related contests and information can be found on the Internet. For instance:

- http://cslu.cse.ogi.edu/tutordemos/SpectrogramReading/spectrogram_reading.html
- http://home.cc.umanitoba.ca/~robh/howto.html

Spectrogram 代表了音色隨時間變化的資料，因此有些厲害的人，可以由 Specgrogram 直接看出語音的內容，這種技術稱為 Specgrogram Reading，有興趣的同學，可以在搜尋引擎上找到很多相關的網頁，也可以試試自己的功力。

第 5 章作業

1. (*) **Frequency-to-pitch conversion**： Write an m-file function *freq2pitch.m* that convert a given frequency in Hz into the pitch in semitones. The format is

   $$\text{pitch} = 69 + 12*\log_2(\text{freq}/440)$$

2. (*) **Pitch-to-frequency conversion**： Write an m-file function *pitch2freq.m* that converts a given pitch in semitones into the frequency in Hz. The format is

   $$\text{freq} = 440*2\text{^}((\text{pitch}-69)/12)$$

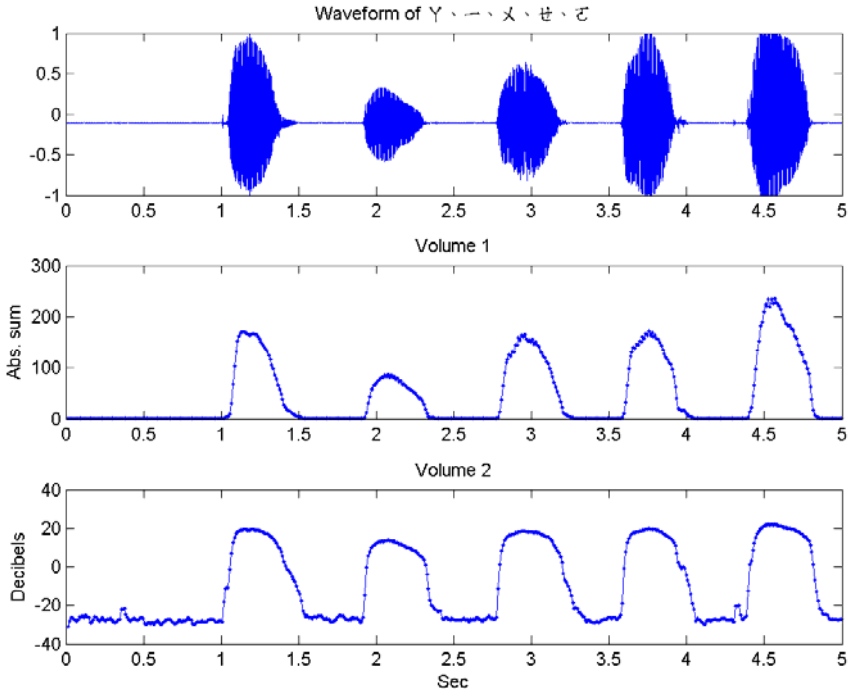
3. (**) **Visual inspection of pitch**： Please follow the example of pitch determination by visual inspection in our text to identify the pitch (in terms of semitones with two digits of precision after the decimal) in the following two audio clips. Be sure to plot your result, just like what we did in the text.
   a. Write a script *pitchTuningFork02.m* for the clip of another sound fork, tuningFork02.wav.
   b. Write a script *pitchVoice02.m* for the vowel part of the character "系" of the utterance "清華大學資訊系", csNthu.wav
4. (**) **Frame-to-volume computation**： Write an m-file function *frame2volume.m* which can compute the volume of a given frame. The format is

   ```
   volume = frame2volume(frame, method);
   ```

   where "frame" is the input frame and "method" is the method for volume computation (1 for abs. sum; 2 for log square sum), and "volume" is the output volume.
5. (**) **Wave-to-volume computation**： Write an m-file function which can compute the volume of a given wave signals. The format is

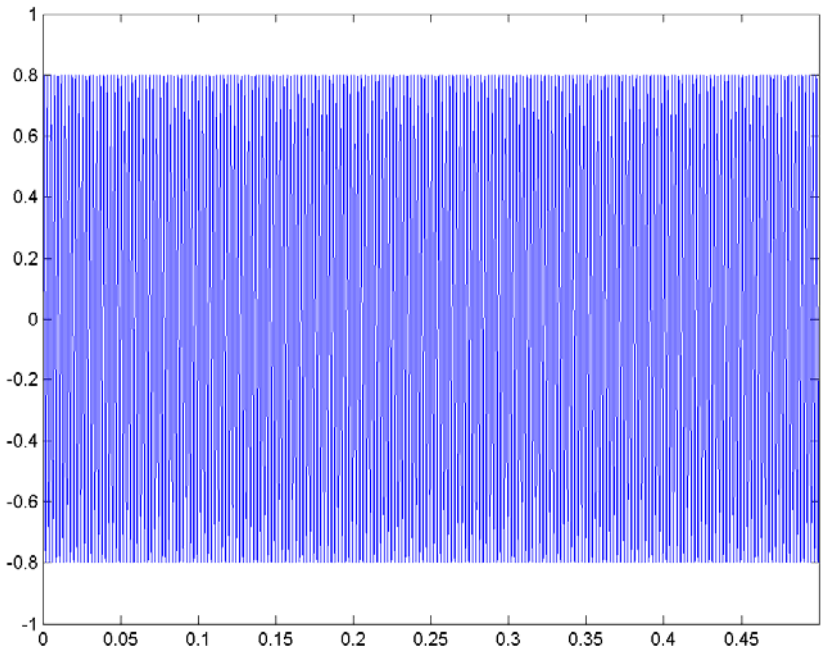
   ```
   volume = wave2volume(wave, frameSize, overlap, method);
   ```

   where "wave" is the input wave signals, "frameSize" is the frame size, "overlap" is the overlap, "method" is the method for computing the volume (1 for abs. sum; 2 for log square sum), and "volume" is the output volume vector.
6. (*) **Volume vs. timbre**： Write an m-file script plotVolVsTimbre.m that can record your utterance of "ㄚ、ㄧ、ㄨ、ㄝ、ㄛ" (example) with a sample rate of 16 KHz and a bit resolution of 16 bits. When doing the recording, please try your best to keep a constant perceived volume for all these five vowels. Then your program should plot the wave signals together with two volume vectors (with frame size = 512, hop size = 160) computed via the two methods mentioned in this chapter. The obtained plots should be similar to the following image:



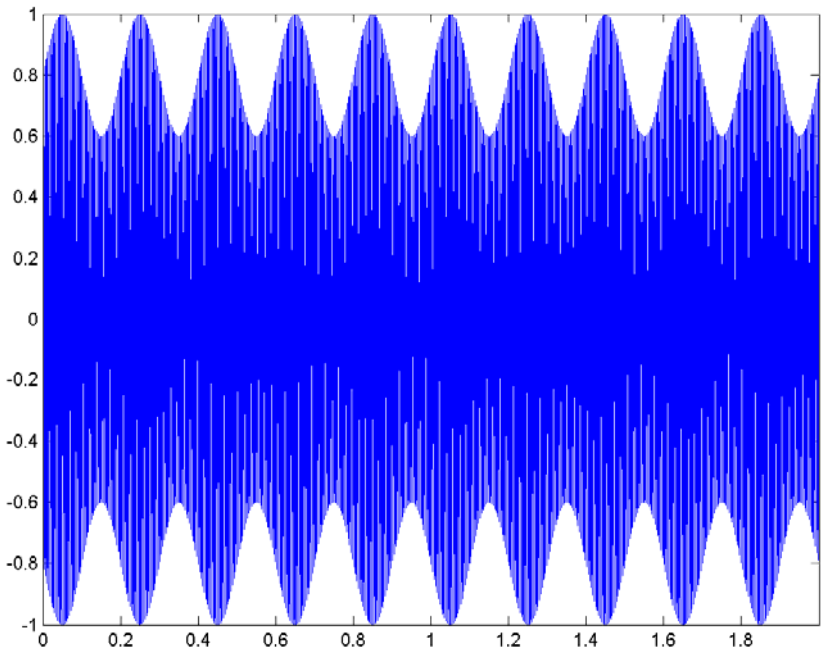
From your plot, can you deduct the relationship between the volume and the shapes of your mouth? Try other vowels to see if your argument still holds.

7. (**) **Use sinusoids to synthesize waveform of given volume and pitch：** In the following three sub-exercises, you are going to use the sine function to synthesize audio signals with given volume and pitch.
    a. Write an m-file script *playSineWave01.m* that uses a sinusoid of 0.8 amplitude to generate a 0.5-second mono signals with a pitch of 69 semitones, sample rate of 16 KHz. Your program should plot the waveform and play the signals. (Here is an example). The plot should look like this:

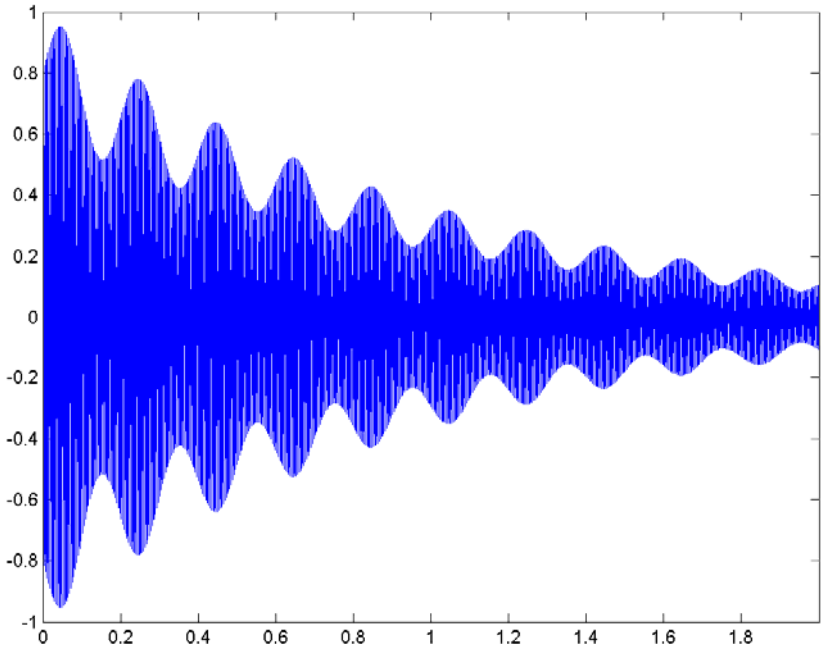


Does the pitch sound the same as the recording of a tuning fork tuningFork01.wav? (Hint: a sinusoid with frequency f can be expressed as y = sin(2*pi*f*t).)

b. Write an m-file script *playSineWave02.m* that uses the sinusoid to generate a mono wave signal of duration of 2 seconds, pitch of 60 semitones, sample rate of 16 KHz, bit resolution of 16 bits. Moreover, the waveform should be oscillate between 0.6 and 1.0 with a frequency of 5 Hz. Your program should plot the waveform and play the signal. (Here is an example). The plot should look like this:

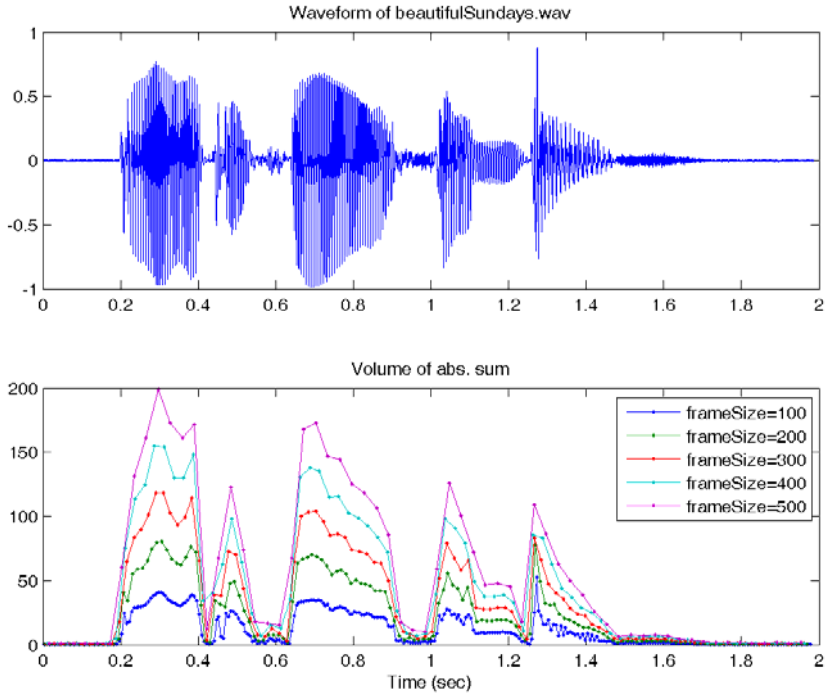


c. Write an m-file script *playSineWave03.m* to repeat the previous sub-problem, but the intensity of the waveform should decrease by following an exponential function exp(-t). Here is an example. Your plot should look like this:

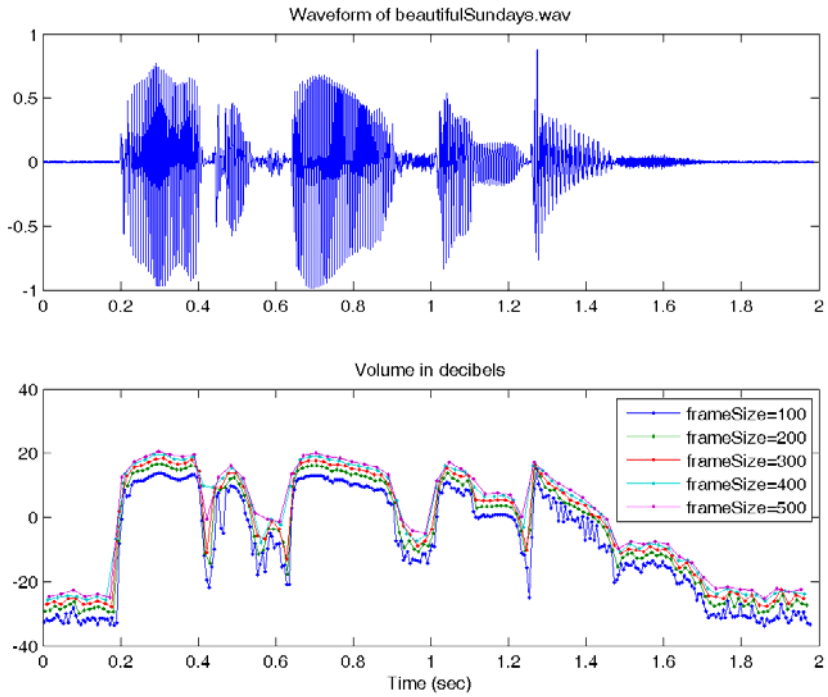


8. (***) **Impact of frame sizes on volume**： First of all, you need to record your utterance of "beautiful sundays" and save it to a wave file of 16 KHz, 16 bits, mono. My example is here. But please use your own recording instead of mine.

a. Write an m-file script *plotVolVsFrameSize01.m* that reads the wave file and plot the volumes (as the absolute sum of a frame) with respect to frame sizes of 100 癒 B200 癒 B300 癒 B400 癒 B500, and overlap of 0. Please use the same time axis for the first plot of the waveform, and the second plots of the 5 volume curves. You plot should look like this:



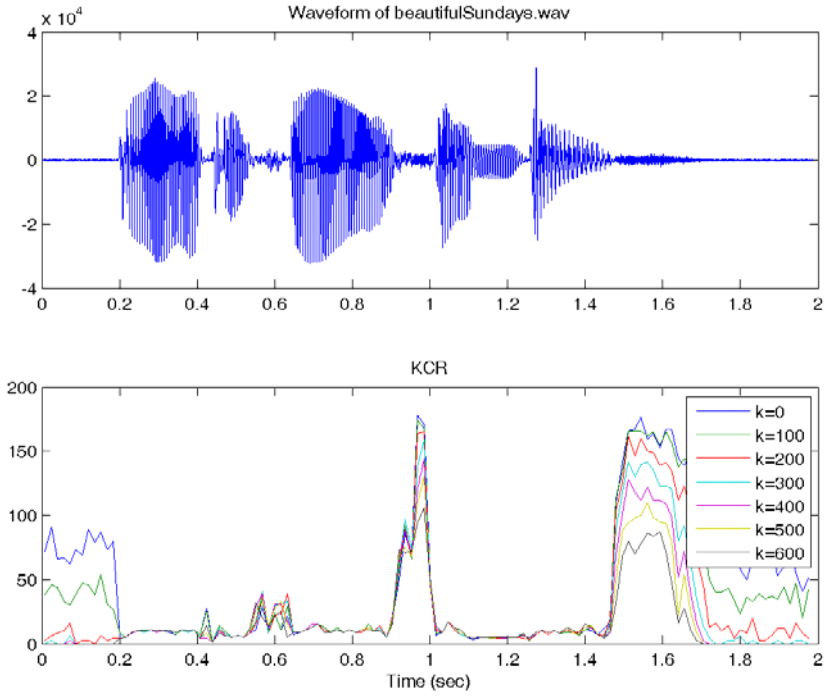
b. Write an m-file script *plotVolVsFrameSize02.m* to repeat the previous sub-problem, but compute the volumes in terms of decibels. Your plot should look like this:



9. Hint:

- You can do the recording with either MATLAB or CoolEdit. But CoolEdit may be easier for you to edit the recordings, e.g., cut off leading or trailing slience, etc.
- Since each volume vector corresponds to a different time vector, you can put all volume vectors in a volume matrix, and all time vectors in a time matrix, and plot 5 volume curves all at once. The length of these 5 curves are not the same, hence you need to pad the time/volume matrix with *NaN*.
- Here are some functions in the Audio Processing Toolbox that can make your job easier: *frame2volume.m*, *frame2sampleIndex.m*.

10. (***) **Impact of the values of K on KCR：** We can extend the definition of ZCR to KCR (k-crossing rate) which is the number of crossings over y = k in a frame. Write an m-file script *plotKcrVsK01.m* that read your recording of "beautiful sundays" (as integers) and plot 7 KCR curves with K equal to 0, 100, 200, ..., 600. The frame size is 256 and the overlap is 0. You program should plot the waveform as well as the KCR curves on the same time axis. The plot should look like this:



Do these KCR curves have different but consistent behavior on voiced and unvoiced sounds? Can you use KCR for unvoiced/voiced detection?

11. (***) **Playback of music notes：** This problem concerns the playback of music notes using the concatenation of sinusoids.
    . Write a m-file function note2wave01.m which can take a sequence of music notes and return the audio signals for playback. The format is

```
wave=note2wave01(pitch, duration, fs)
```

where "pitch" is the vector of note pitch in semitones (with 0 indicating silence), "duration" is the vector of note duration in seconds, "fs" is the sample rate of the output wave signals for playback. Using the following script to try your function and see if you can identify the song. (my result for your reference)

```
pitch=      [55 55 55 55 57 55 0 57 60 0    64 0    62 62 62 62 60 64 60 0 57 55 55];
duration=[23 23 23 23 23 35 9 23 69 18 69 18 23 23 23 12 12 23 35 9 12 12 127]/64;
fs=16000;
wave=note2wave01(pitch, duration, fs);
```

```
sound(wave, fs);
```

a. If you concatenate the sinusoids of two different notes directly, you can hear obvious noise due to the discontinuity between waveforms of these two notes. Can you find a method to eliminate the noise so it is more pleasant to hear it? (Hint: There are several methods to solve the problem.)

12. (**) **由一個音框計算一點音量：** 請寫一個函數 frame2volume，由一個音框來計算音量，其用法如下：

```
volume = frame2volume(frame, method);
```

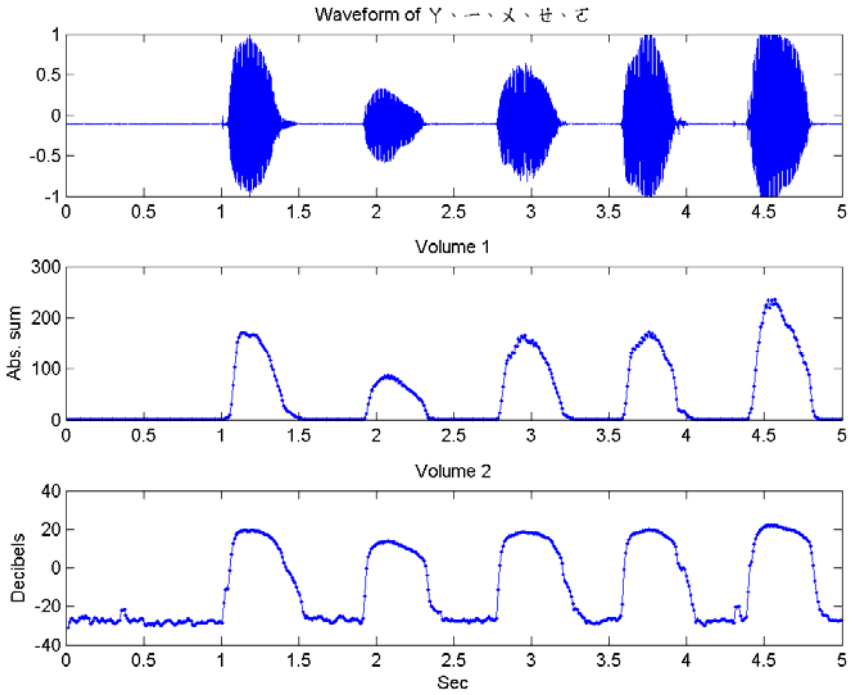
其中 frame 是一個音框向量，method 則是代表使用的方法（'absSum' 代表使用絕對值的總和，'logSquareSum' 代表使用平方和的對數），volume 則是音高。

13. (**) **由一段聲音來計算音量向量：** 請寫一個函數 wave2volume，由一段聲音來計算音量向量，其用法如下：

```
volume = wave2volume(wave, frameSize, overlap, method);
```

其中 wave 是一段音訊向量，frameSize 是音框點數，overlap 是相鄰音框的重疊點數，method 則是代表使用的方法（'absSum' 代表使用絕對值的總和，'logSquareSum' 代表使用平方和的對數），而 volume 則是音量向量。

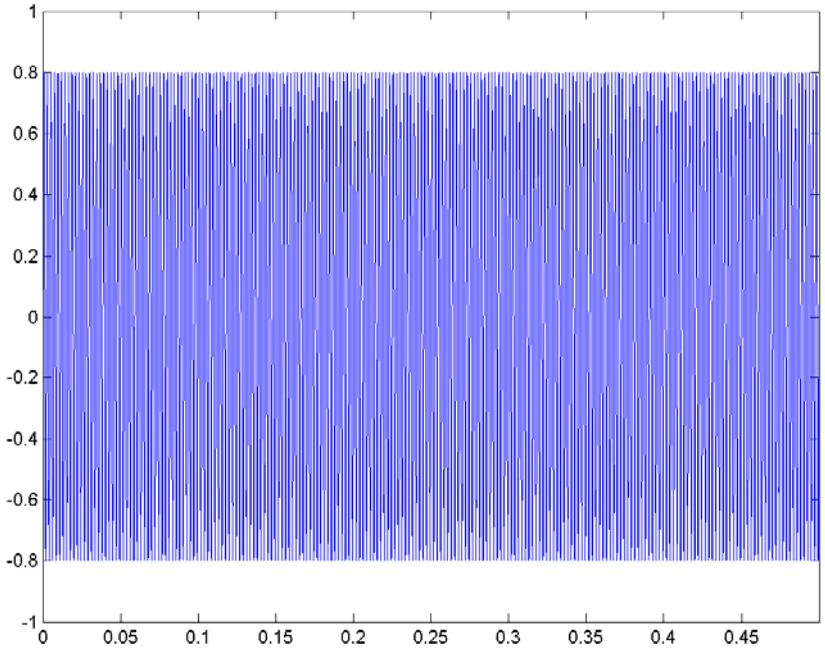
14. (*) **音量與音色的關係：** 請寫一段 MATLAB 程式，先進行錄音，取樣頻率為 16 KHz，解析度為 16 Bits，錄音內容是「ㄚ、ㄧ、ㄨ、ㄝ、ㄛ」。在進行錄音時，盡量用同樣的音量（憑個人的主觀感覺），每個音也要盡量平穩。然後將聲波和兩種音量圖（frame Size = 512，frame Step = 160）同時畫出來，所得到的圖形應該類似下圖：



從你畫出來的圖形中，你可以歸納出音量和嘴型之間的關係嗎？你可以再試試其他母音，看看你的推論是否適用於其他母音。

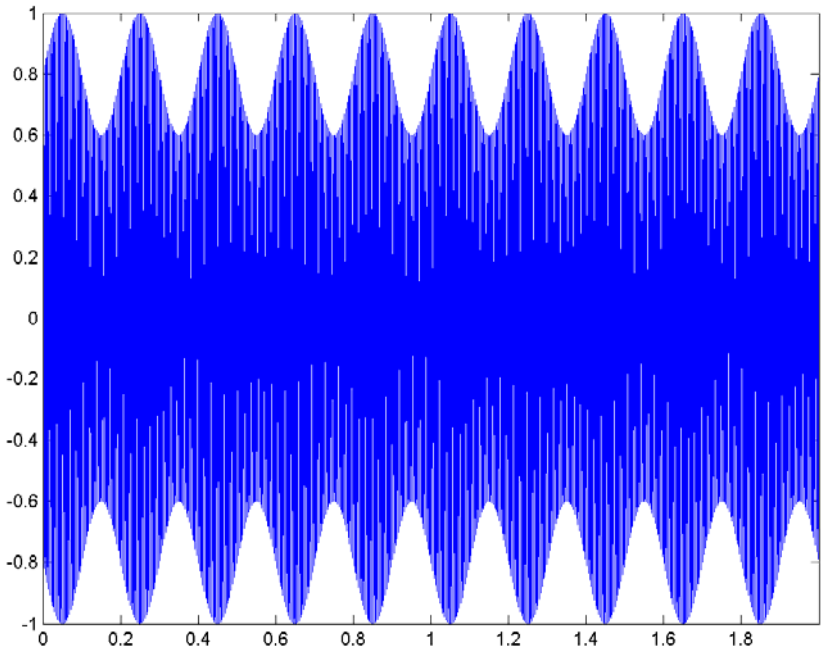
15. (**) **使用正弦波產生特定音高及音量的音訊：** 以下這三小題，都是用正弦波來產生特定音高及音量的音訊。

. 請用震幅為 0.8 的正弦波產生一個 0.5 秒的音訊，音高為 69 Semitone，取樣頻率為 16 KHz，單聲道。你的程式碼必須畫出此波形，並同時播放此聲音（偷聽一下）。所得到的圖形應該類似下圖：

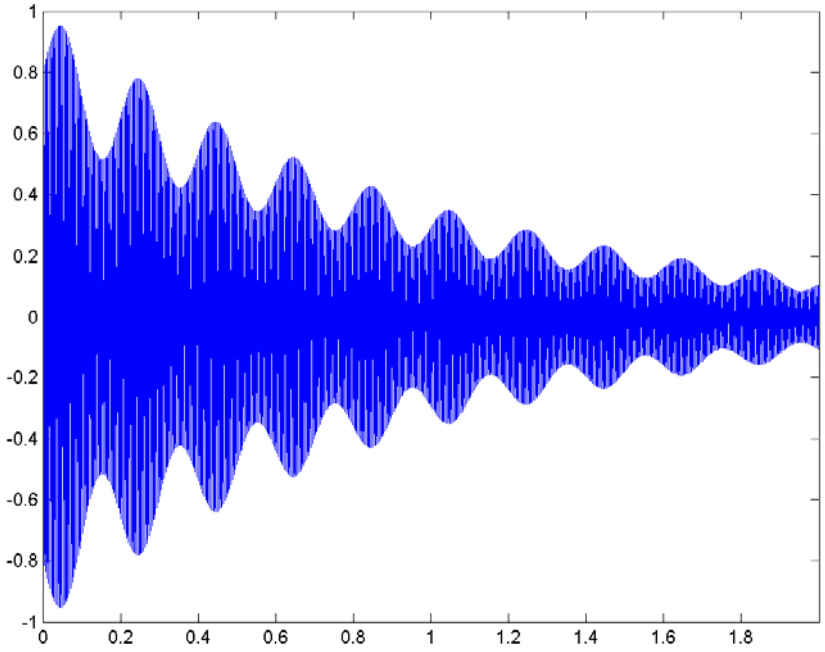


此音訊的音高和音叉的聲音接近嗎？（提示：一個頻率為 f 的正弦波可以表示成 y = sin(2*pi*f*t)。）

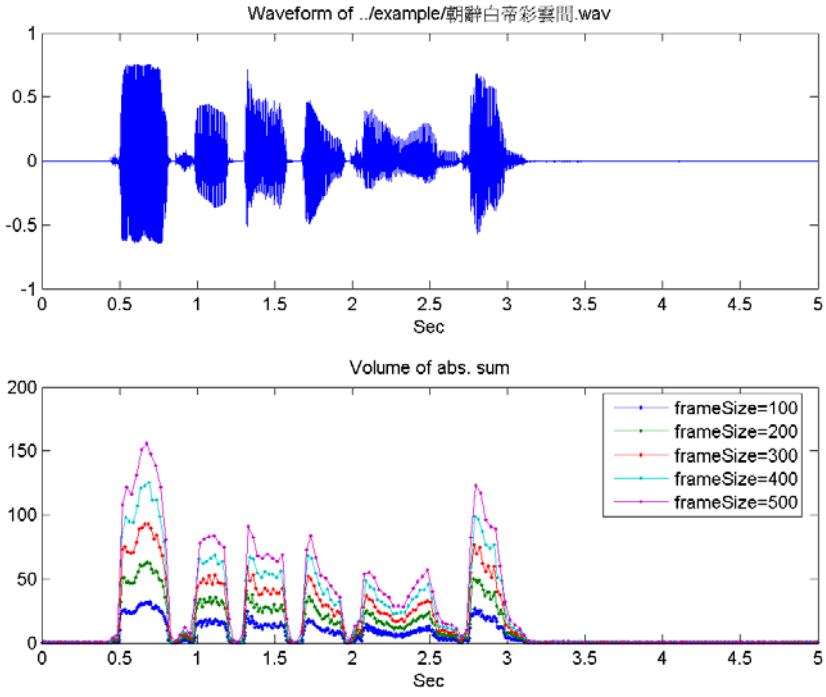
a. 請用正弦波產生一個 2 秒的音訊，音高為 60 Semitone，取樣頻率為 16 KHz，解析度為 16 Bits，單聲道，其波形的震幅隨著時間呈現另一個正弦波的變化，平均值是 0.8，但在 0.6 和 1.0 之間震盪，震盪的頻率是 5 Hz。你的程式碼必須畫出此波形，並同時播放此聲音（偷聽一下）。所得到的圖形應該類似下圖：



b. 重複上一小題，但是聲波強度隨時間而遞減，遞減函數是 exp(-t)。你的程式碼必須畫出此波形，並同時播放此聲音（偷聽一下）。所得到的圖形應該類似下圖：



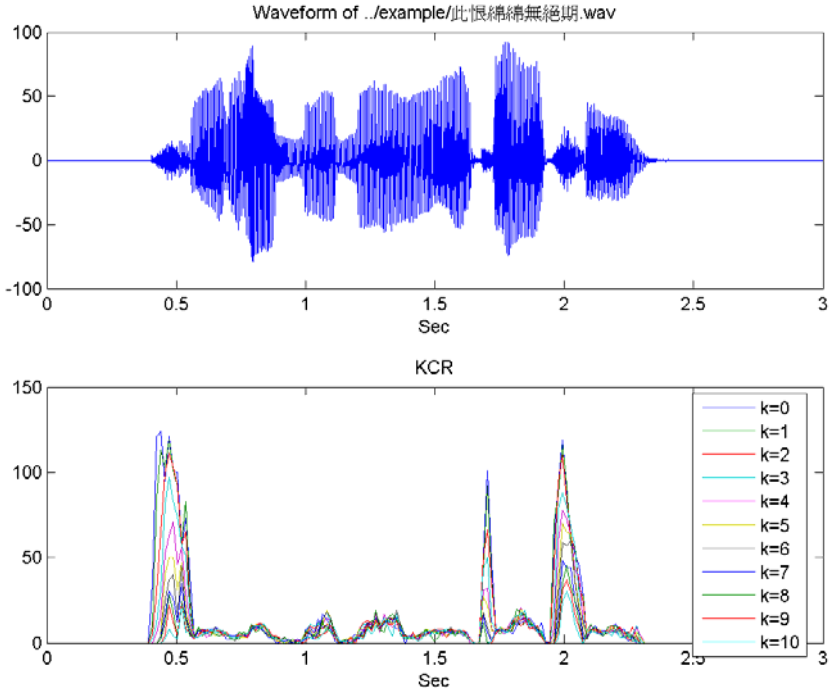
16. (***) **音框長度對音量計算的影響 1：** 請讀入此檔案「朝辭白帝彩雲間.wav」，並畫出當音框長度分別是 100、200、300、400、500 點時，音量（每個音框的絕對值的和）對時間的圖形。請將原聲波和這 5 條音量曲線畫在同一個圖內，所得到的圖形應該類似下圖：



請特別注意，如果你的波形有偏移的現象，要先進行零點校正，畫出的音量才會正確。（提示：因為音框長度不同，所得到的每一條音量曲線的點數也會不一樣，所以在畫圖時，要比較小心。可以將五條音量曲線放在一個矩陣內，長度不足的部分，則填入 nan，對應的時間也是如法泡製，然後再一次畫出。）

17. (***) **音框長度對音量計算的影響 2：** 請重複上題，但將音量的計算改成以分貝（Decibels）為單位，也就是說，先取每個音框平方值的總和，再取以 10 為底對數值，再乘以 10。
18. (***) **K 值對「過 K 率」的影響：** 我們可以將「過零率」（Zero-Crossing Rate）延伸成「過 K 率」（K-Crossing Rate，簡稱 KCR），也就是在每個音框中，訊號通過水平線 y

= K 的次數。請讀入此檔案「此恨綿綿無絕期.wav」，並將所有的音訊取樣值轉成整數，然後畫出當 K = 1 ~ 10 時，KCR 對時間的圖形。請將原聲波和這 10 條 KCR 曲線畫在同一個圖內，所得到的圖形應該類似下圖：



這幾條 KCR 曲線在氣音所出現的位置，是否有明顯不同？我們是否可用 KCR 來偵測氣音的存在？

19. (*) **頻率至音高的轉換：** 請寫一個函數 freq2pitch.m，來將頻率（單位是 Hertz）轉換成為音高（單位是 Semitone），公式如下：

$$pitch = 69 + 12*\log_2(freq/440)$$

20. (*) **音高至頻率的轉換：** 請寫一個函數 pitch2freq.m，來將音高（單位是 Semitone）轉換成為頻率（單位是 Hertz），公式如下：

$$freq = 440*2^{((pitch-69)/12)}$$

21. (**) **觀察法量測音高：** 請用觀察法來量測下列音訊的音高，請以 Semitone 為單位，並保留小數以下兩位的精確度。（請務必畫圖，類似我們課文中的範例。）

    . 另一支音叉的聲音。

    a. 「清華大學資訊系」的「系」的母音部分。

（提示：你必須使用到頻率轉成半音差的公式。）

22. (***) **音符的播放：** 本題牽涉到如何對單軌的音符進行播放。

    . 請寫一個函數 notePlay.m，其使用格式如下：

```
wave=notePlay(pitch, duration, fs)
```

其中 pitch 是一串音高（以 Semitone 為單位，若是 0，則代表沒有靜音），duration 是對應的音長（以秒為單位），fs 則是合成的輸出訊號 wave 的取樣頻率。請使用此函數來合成下列參數的 wave 並播放，聽得出來是什麼歌嗎？（偷聽一下...）

```
pitch=    [55 55 55 55 57 55 0 57 60 0   64 0   62 62 62 62 60 64 60 0 57 55 55];
duration=[23 23 23 23 23 35 9 23 69 18 69 18 23 23 23 12 12 23 35 9 12 12 127]/64;
fs=16000
```

    a. 如果只是將兩段正弦波接起來，那麼在音符交接之處，波形會不連續，就會聽到很明顯的雜音，尤其如果當音長很短時，這個問題特別嚴重。請問你有什麼好辦法來消除這個雜音呢？（提示：至少應該有兩種辦法。）

Chapter 6: End-Point Detection (EPD)

6-1 Introduction to End-Point Detection (端點偵測介紹)

The goal of end-point detection (EPD for short) is to identfy the important part of an audio segment for further processing. Hence EPD is also known as "speech detecton" or "voice activity detection" (VAD for short). EPD plays an important role in audio signal processing and recognition.

Based on the acoustic features used for EPD, we can classify the EPD methods into two types:

1. Time-domain methods:
   a. Volume: Volume is the most commonly used feature for EPD. However, it is usually hard to have a single universal threshold for EPD. In particular, a single volume threshold for EPD is likely to misclassify unvoiced sounds as silence for audio input from uni-directional microphone.
   b. Volume and ZCR: ZCR can be used in conjunction with volume to identify unvoiced sounds in a more reliable manner, as explained in the next section.

   The computation load is usually small so these methods can be ported to low-end platform such as micro-controllers.
2. Frequency-domain methods:
   a. Variance in spectrum: Voiced sounds have more regular amplitude spectra, leading to smaller spectral variances.
   b. Entropy in spectrum: Regular amplitude spectra of voices sounds also generate low entropy, which can be used as a criterion for EPD.

   These methods usually require more computing power and thus are not portable to low-end platforms.

Hint

To put it simply, time-domain methods use only the waveform of audio signals for EPD. On the other hand, if we need to use the Fourier transform to analyze the waveforms for EPD, then it is frequency-domain method. More information on spectrum and Fourier transform will be detailed in later chapters.

There are two types of errors in EPD, which cause different effects in speech recognition, as follows.

- False Rejection: Speech frames are erroneously identified as silence/noise, leading to decreased recognition rates.
- False Acceptance: Silence/noise frames are erroneously identified as speech frames, which will not cause too much trouble if the recognizer can take short leading/trailing silence into consideration.

The other sections of this chapter will introduce both time-domain and frequency-domain methods for EPD.

「端點偵測」（End-point Detection，簡稱 EPD）的目標是要決定音訊開始和結束的位置，所以又可以稱為 Speech Detection 或是 VAD (Voice Activity Detection)。端點偵測在音訊處理與辨識中，扮演一個重要的角色。

常見的端點偵測方法與相關的特徵參數，可以分成兩大類：

1. 時域（Time Domain）的方法：計算量比較小，因此比較容易移植到計算能力較差的微電腦平台。
   a. 音量：只使用音量來進行端點偵測，是最簡單的方法，但是會對氣音造成誤判。不同的音量計算方式也會造成端點偵測結果的不同，至於是哪一種計算方式比較好，並無定論，需要靠大量的資料來測試得知。
   b. 音量和過零率：以音量為主，過零率為輔，可以對氣音進行較精密的檢測。

2. 頻域（Frequency Domain）的方法：計算量比較大，因此比較難移植到計算能力較差的微電腦平台。
    a. 頻譜的變異數：有聲音的頻譜變化較規律，變異數較低，可作為判斷端點的基準。
    b. 頻譜的 Entropy：我們也可以使用使用 Entropy 達到類似上述的功能。

Hint

簡單地說，若只是對聲音波形做一些較簡單的運算，就是屬於時域的方法。另一方面，凡是要用到傅立葉轉換（Fourier Transform）來產生聲音的頻譜，就是屬於頻譜的方法。這種分法常被用來對音訊處的方法進行分類，但有時候有一些模糊地帶。有關於頻譜以及傅立葉轉換，會在後續的章節說明。

錯誤的端點偵測，在語音辨識上會造成兩種效應：

- False Rejection：將 Speech 誤認為 Silence/Noise，因而造成音訊辨識率下降
- False Acceptance：將 Silence/Noise 誤認為 Speech，此時音訊辨識率也會下降，但是我們可以在設計辨識器時，前後加上可能的靜音聲學模型，此時辨識率的下降就會比前者來的和緩。

以下各小節將針對這兩類的端點偵測方法來介紹。

6-2 EPD in Time Domain (端點偵測：時域的方法)

We shall introduce several time-domain methods for EPD in this section.

The first method uses volume as the only acoustic feature for EPD. This is the most intuitive method with least computation. We only need to determine a volume threshold and any frame with a volume less than the threshold is regarded as silence. However, how to determine a good volume threshold is not obvious. Besides empirically determining the threshold, the best way is to use a set of labelled training data to find the best value for achieving the minimum error.

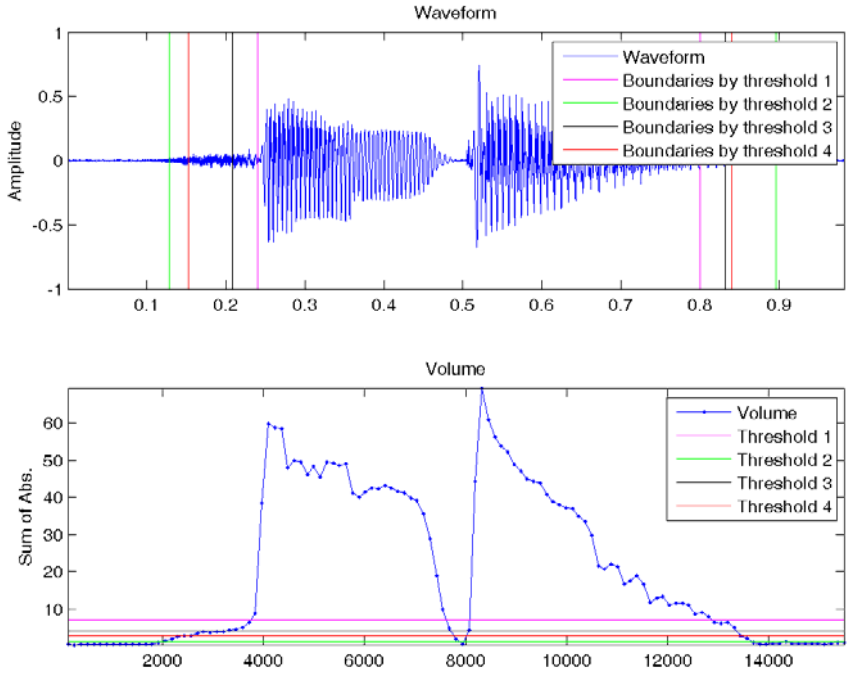
In the following example, we shall use four different ways to compute volume thresholds for EPD of the wave file sunday.wav:

Example 1**Input file** endPointDetection/epdVolTh01.m

```
waveFile='sunday.wav';
[wave, fs, nbits] = wavread(waveFile);
frameSize = 256;
overlap = 128;

wave=wave-mean(wave);                                % zero-mean substraction
frameMat=buffer2(wave, frameSize, overlap);  % frame blocking
frameNum=size(frameMat, 2);                          % no. of frames
volume=frame2volume(frameMat, 1);                    % volume
volumeTh1=max(volume)*0.1;                           % volume threshold 1
volumeTh2=median(volume)*0.1;                        % volume threshold 2
volumeTh3=min(volume)*10;                            % volume threshold 3
volumeTh4=volume(1)*5;                               % volume threshold 4
index1 = find(volume>volumeTh1);
index2 = find(volume>volumeTh2);
index3 = find(volume>volumeTh3);
index4 = find(volume>volumeTh4);
endPoint1=frame2sampleIndex([index1(1), index1(end)], frameSize, overlap);
endPoint2=frame2sampleIndex([index2(1), index2(end)], frameSize, overlap);
endPoint3=frame2sampleIndex([index3(1), index3(end)], frameSize, overlap);
endPoint4=frame2sampleIndex([index4(1), index4(end)], frameSize, overlap);
```

```
subplot(2,1,1);
time=(1:length(wave))/fs;
plot(time, wave);
ylabel('Amplitude'); title('Waveform');
axis([-inf inf -1 1]);
line(time(endPoint1(    1))*[1 1], [-1, 1], 'color', 'm');
line(time(endPoint2(    1))*[1 1], [-1, 1], 'color', 'g');
line(time(endPoint3(    1))*[1 1], [-1, 1], 'color', 'k');
line(time(endPoint4(    1))*[1 1], [-1, 1], 'color', 'r');
line(time(endPoint1(end))*[1 1], [-1, 1], 'color', 'm');
line(time(endPoint2(end))*[1 1], [-1, 1], 'color', 'g');
line(time(endPoint3(end))*[1 1], [-1, 1], 'color', 'k');
line(time(endPoint4(end))*[1 1], [-1, 1], 'color', 'r');
legend('Waveform', 'Boundaries by threshold 1', 'Boundaries by threshold 2', 'Boundaries by threshold 3',
'Boundaries by threshold 4');

subplot(2,1,2);
frameTime=frame2sampleIndex(1:frameNum, frameSize, overlap);
plot(frameTime, volume, '.-');
ylabel('Sum of Abs.'); title('Volume');
axis tight;
line([min(frameTime), max(frameTime)], volumeTh1*[1 1], 'color', 'm');
line([min(frameTime), max(frameTime)], volumeTh2*[1 1], 'color', 'g');
line([min(frameTime), max(frameTime)], volumeTh3*[1 1], 'color', 'k');
line([min(frameTime), max(frameTime)], volumeTh4*[1 1], 'color', 'r');
legend('Volume', 'Threshold 1', 'Threshold 2', 'Threshold 3', 'Threshold 4');
```

Output figure



In the above example, we have used four methods to compute the volume thresholds. These four methods all have their weakness, as explained next:

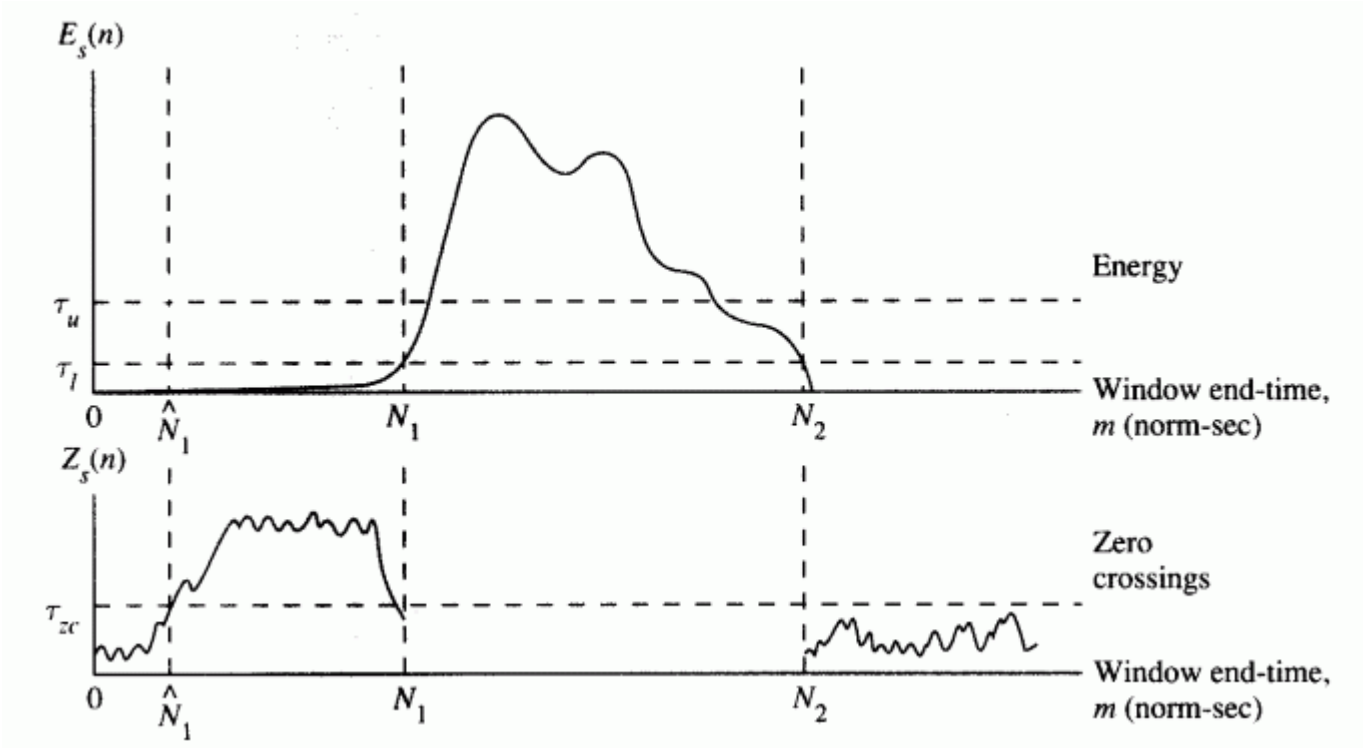
1. A ratio times the maximal volume: Not reliable if there is an impulse in volume due to plosive sounds.
2. A ratio times the median of the volume: Not reliable when silence occupy more than half of the audio signals.
3. The minimal volume times a constant: This could go wrong is the noise if too big. Moreover, it is likely for some recordings to have a frame of zero volume.
4. The volume of the first frame times a constant: This could go wrong if the first frame of the recording is unstable, which is not rare in practice.

The ratios or constants in the above four methods should be determined through labeled training data. It should be noted wave files of different characteristics (recordings via uni-directional or omni-directional microphones, different sample rates, different bit resolutions, different frame sizes and overlaps) will have a different best thresholds.

Of course, you also create a new threshold by using linear combinations of these thresholds, etc. From the above example, it is obvious that the leading unvoiced sound is likely to be misclassified as silence. Moreover, a single threshold might not perform well if the volume varies a lot. As a result, an improved method can be stated next:

1. Use a upper threshold τ_u to determine the inital end-points.
2. Extend the boundaries until they reach the lower threshold τ_l.
3. Extend the boundaries further until they reach the ZCR threshold τ_{zc}.

This method is illustrated as follows.



The above improved method uses only three thresholds, hence it is possible to use grid search to find the best values via a set of labeled training data.

Hint

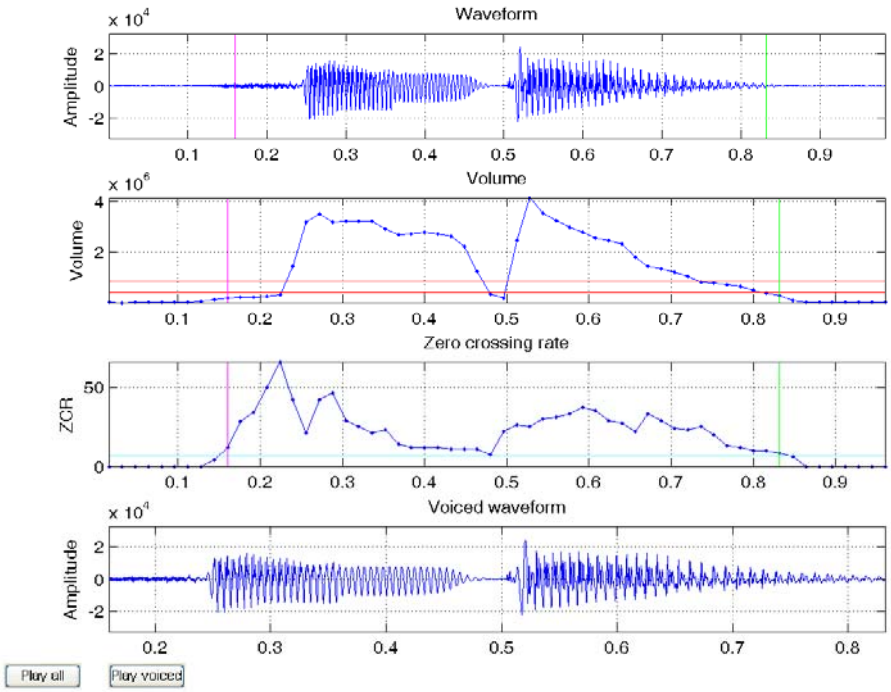
The above method is designed for speech recognition. For melody recognition, we do not need to consider unvoiced sounds since they do not have pitch at all.

If we apply the above method for EPD of sunday.wav, the result can plotted as follows:

Example 2**Input file** endPointDetection/epdVolZcr01.m

```
waveFile='sunday.wav';
plotOpt = 1;
[y, fs, nbits] = wavReadInt(waveFile);
endPoint = epdByVolZcr(y, fs, nbits, [], plotOpt);
```

Output figure



In the above plot, the red and green lines indicates the beginning and end of sound, respectively. This example uses the function endPointDetect.m in the Audio Processing Toolbox, which use volume and ZCR as mentioned above for EPD.

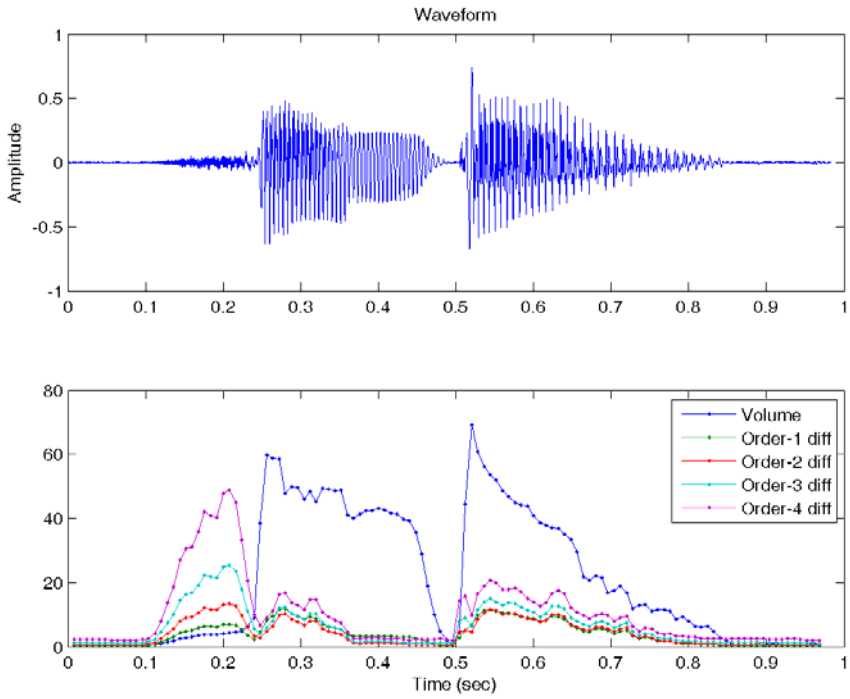
Now it should be obvious that the most difficult part in EPD is to distinguish unvoiced sounds from silence reilably. One way to achieve this goal is to use high-order difference of the waveform as a time-domain features. For instance, in the following example, we use order-1, 2, 3 difference on the waveform of beautifulSundays.wav:

Example 3**Input file** endPointDetection/highOrderDiff01.m

```
waveFile='sunday.wav';
[wave, fs, nbits] = wavread(waveFile);
frameSize = 256;
overlap = 128;

wave=wave-mean(wave);                                   % zero-mean substraction
frameMat=buffer2(wave, frameSize, overlap);  % frame blocking
frameNum=size(frameMat, 2);                             % no. of frames
volume=frame2volume(frameMat, 1);
sumAbsDiff1=sum(abs(diff(frameMat)));
sumAbsDiff2=sum(abs(diff(diff(frameMat))));
sumAbsDiff3=sum(abs(diff(diff(diff(frameMat)))));
sumAbsDiff4=sum(abs(diff(diff(diff(diff(frameMat))))));
```

```
subplot(2,1,1);
time=(1:length(wave))/fs;
plot(time, wave); ylabel('Amplitude'); title('Waveform');
subplot(2,1,2);
frameTime=frame2sampleIndex(1:frameNum, frameSize, overlap)/fs;
plot(frameTime', [volume; sumAbsDiff1; sumAbsDiff2; sumAbsDiff3; sumAbsDiff4]', '.-');
legend('Volume', 'Order-1 diff', 'Order-2 diff', 'Order-3 diff', 'Order-4 diff');
xlabel('Time (sec)');
```

Output figure



It is obvious that the high-order difference (HOD) on the waveform can let us identify the unvoiced sound more easily for this case. Therefore you can take the union of high-volume and high HOD regions to have most robust of EPD.

Hint

If you have counter examples which invalids the use of HOD, please let me know.

From the above example, a possible simple way of combining volume and HOD for EPD can be stated as follows:

1. Compute volume (VOL) and the absolute sum of order-n difference (HOD).
2. Select a weighting factor w within [0, 1] to compute a new curve VH = w*VOL + (1-w)*HOD.
3. Find a ratio ρ to compute the threshold τ of VH to determine the end-points. The threshold is equal to $VH_{min}+(VH_{max}-VH_{min})*\rho$.

The above method involves three parameters to be determined: n, w, ρ. Typical values of these parameters are n = 4, w = 0.5, and ρ = 0.125. However, these values vary with data sets. It is always adviceable to have these values tuned by using the target data set for a more robust result.

Of course, there are still plenty of other methods for EPD on time domain. The only limit is your imagination.

6-3 EPD in Frequency Domain (端點偵測：頻域的方法)

Voiced sounds have harmonic structures in the frequency-domain specta. Moreover, the energy distribution will be mostly biased toward the low-frequency bands. As a result, we can apply simple mathematical functions on the spectrum for EPD.
If you are not familiar with the definition of spectrum (or more specific, amplitude spectrum), you do not need to worry about this at this stage. All you need to know is that the amplitude spectrum is the distribution of energy with respective to frequency. More information about the spectrum and its mathematical definition will be covered in later chapters.
有聲音訊在頻譜上會有重複的諧波結構，因此我們也可以使用頻譜的變異數或是 Entropy 來進行端點偵測，相關論文及投影片請見此連結。
（待續）

/jang/books/audioSignalProcessing/paper/endPointDetection/index.asp

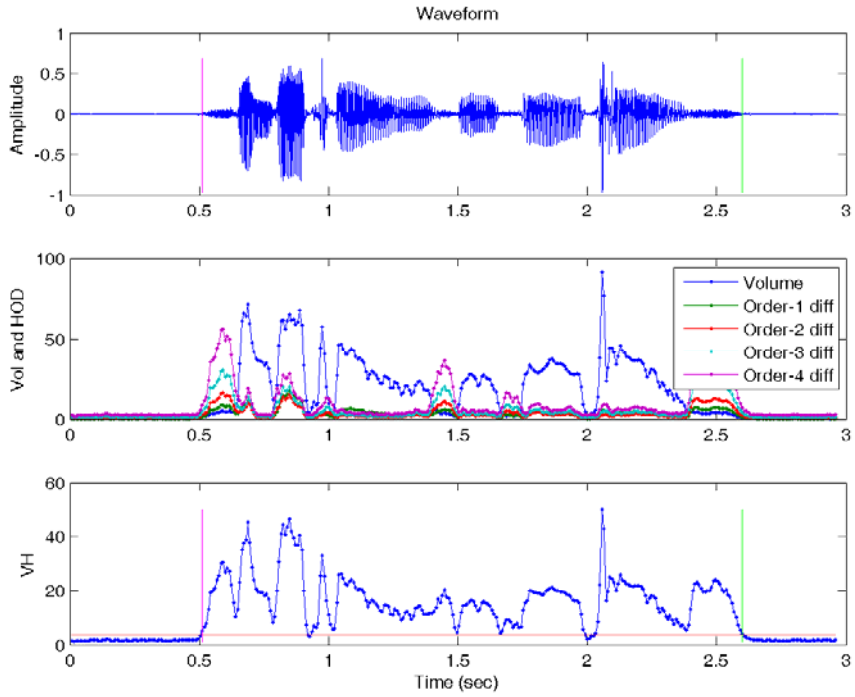
---

File listing:

File name	Size (Bytes)	Last modified
096.PDF	231553	2004/8/18 下午 11:41:15
endPointDetectionViaEntropy.ppt	513024	2003/3/27 下午 10:13:18
shenHL98-endpoint.pdf	366756	2004/8/18 下午 11:42:15

第 6 章作業

1. (**) **EPD by volume and HOD**：
   a. Record your voice of "Singapore is a fine place" and save it as "test.wav", with the format of 16 KHz, 16 bits, mono. (Hint: You can use waveFileRecord.m in the Audio Processing Toolbox for easy recording.)
   b. Write a script epdByVolHod01.m which do EPD by using volume and high-order difference. You should plot your result like this:



(Hint: Be sure to understand all the examples provided in the text before you attemp this exercise. In particular, you should be familiar with the use of "line" command for adding lines to the existing plot.)

c. Convert your script epdByVolHod01.m to a function epdByVolHod.m with the following usage:

```
[epInSampleIndex, epInFrameIndex] = epdByVolHod(wave, fs, nbits, epdParam, plotOpt)
```

where

- epInFrameIndex: two-element end-points in frame index
- epInSampleIndex: two-element end-points in sample index
- wave: input audio signals within [-1, 1];
- fs: sample rate
- epdParam: EPD parameter, including three fields:
  - epdParam.frameSize: frame size
  - epdParam.overlap: overlap
  - epdParam.diffOrder: the order of difference (default: 4)
  - epdParam.volWeight: the weighting factor for volume (default: 0.5)
  - epdParam.vhRatio: the constant for obtaining the VH threshold of $VH_{min}+(VH_{max}-VH_{min})$*epdParam.vhRatio (default 0.125)
- plotOpt: 0 for silent operation, 1 for plotting the result (as shown in the previous sub-problem).

Please test your program using the following script:

```
waveFile='test.wav';
epdParam.frameSize = 256;
epdParam.overlap = 0;
epdParam.diffOrder=4;
epdParam.volWeight=0.5;
epdParam.vhRatio = 0.125;
plotOpt = 1;
```

```
out = epdByVolHod(wave, fs, nbits, epdParam, plotOpt);
```

2. (Hint: You should finish part (b) before trying this one since it is trickier debugging an m-file function. When you are debugging your m-file function, be sure to issue "dbstop if error" or "dbstop if warning" to stop at the work space where errors/warnings occur. To clear the debugging flags, try "dbclear all". Also it would be better if you can follow the same flow as in epdByVol.m in the Audio Processing Toolbox.)
3. (*) **Recordings of digits and letters**： This is a recording task for digits and letters. Please refer to this page for details.
4. (***) **Programming contest: end-point detection**： Please read this page for details.

The followings are old Chinese version.

1. (*) **數字與字母的錄音**： 此作業要讓各位進行數字與字母的錄音，細節請看此連結。
2. (***) **程式競賽：端點偵測**： 請見此連結。

Audio Signal Processing and Recognition (音訊處理與辨識): Recording Task

Roger Jang (張智星)

In this task, you need to do the recordings for both digits and letters. The recording clips will be used for further exercises in the class, including end-point detection and speech recognition using DTW, etc. More specifically, you need to record 10 digits (0~9) and 26 letters (A~Z) twice to have 72 .wav files. Basically it will take about 10 minutes to finish the recording. Please follow the stey-by-step instructions closely.

1. Please download winrar and install it. (Change the file extension to exe and then execute directly.)
2. If this is your first time to do the recording task, please read important notes about recording. Please read them carefully. If you don't follow the rules, your recordings might not be in good shape and you need to do it again.
3. Downlaod the recording program digitLetterRecordingProgram.rar and uncompress it into a folder "digitLetterRecordingProgram". Read the instructions in "readme.txt" and then you can start recording. For easy reference, the contents of digitLetterRecordingProgram/readme.txt are listed here:
4. Please type "go" under MATLAB to start the recording of 0~9 digits and a~z letters. After recording, the program will generate a folder "waveFile/dddddd" where "dddddd" is your student ID, such as "921510". This directory should located at the same level as the f

Chapter 7: Pitch Tracking

如前一章所述，使用「觀察法」來算出音高，並不是太難，但是若要電腦自動算出音高，就需要對音訊進行進一步的自動分析。使用電腦對整段音訊進行抓取音高的過程，通常稱為「音高追蹤」(Pitch Tracking)，所抓出來的音高資訊，有下列應用：

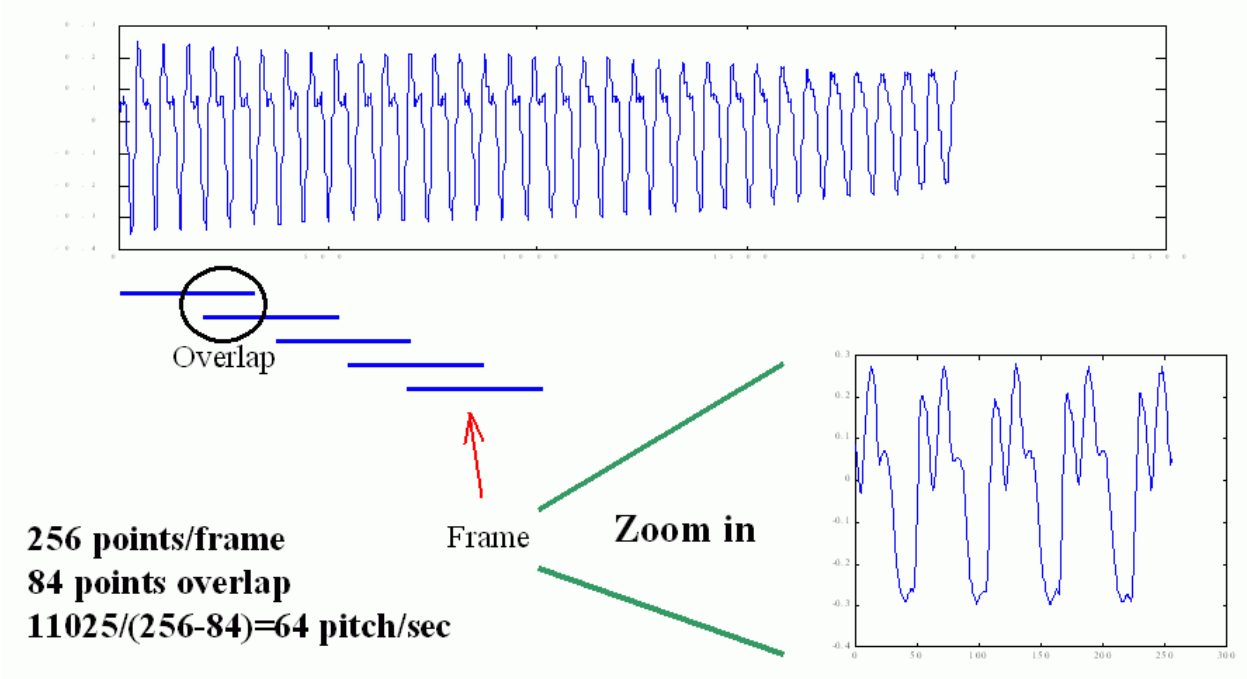
- 旋律辨識（Melody Recognition）：或稱為「哼唱選歌」，也就是如何由使用者的哼唱，找出音樂資料庫中間對應的歌。
- 國語的聲調辨識（Tone Recognition）：辨識使用者講一句話時，每一個字的聲調（一聲、二聲、三聲、四聲等）。
- 語音合成的韻律分析（Prosody Analysis）中的音高分析：如何在合成語音時，使用最自然的音高曲線。
- 語音評分中的音調評分（Intonation Assessment）：如何評估使用者說話的語音，其音高曲線是否標準。
- 語音辨識（Speech Recognition）：我們可以使用語句的音高來提高語音辨識的正確率。

- 總而言之，音高追蹤可說是音訊處理過程中，最基本也是最重要的一環，相關的研究，也進行了數十年，因此我們必須完全瞭解其原理，才能繼續進行其他相關的分析與處理。

音高追蹤的基本流程如下：

1. 將整段音訊訊號切成音框（Frames），相鄰音框之間可以重疊。
2. 算出每個音框所對應的音高。
3. 排除不穩定的音高值。（可由音量來篩選，或由音高值的範圍來過濾。）
4. 對整段音高進行平滑化，通常是使用「中位數濾波器」（Median Filters）。

在切音框的過程中，我們允許左右音框的重疊，因此我們定義「音框率」（Frame Rate）是每秒鐘所出現的音框個數，如果取樣頻率是 11025，音框長度是 256 點，重疊點數是 84，那麼音框率就是 11025/(256-84) = 64，換句話說，我們的電腦要能夠每秒鐘處理 64 個音框，才能達到「即時處理」的目的。示意圖如下：



我們讓音框重疊的目地，只是希望相鄰音框之間的變化不會太大，使抓出來的音高曲線更具有連續性。但是在實際應用時，音框的重疊也不能太大，否則會造成計算量的過大。在選擇音框的大小時，有下列考量因素：

- 音框長度至少必須包含 2 個基本週期以上，才能顯示語音的特性。已知人聲的音高範圍大約在 50 Hz 至 1000 Hz 之間，因此對於一個的取樣頻率，我們就可以計算出音框長度的最小值。例如，若取樣頻率 fs = 8000 Hz，那麼當音高 f = 50 Hz（例如男低音的歌聲）時，每個基本週期的點數是 fs/f = 8000/50 = 160，因此音框必須至少是 320 點；若音高是 1000 Hz（例如女高音的歌聲）時，每個基本週期的點數是 8000/1000 = 8，因此音框必須至少是 16 點。
- 音框長度也不能太大，太長的音框無法抓到音訊的特性隨時間而變化的細微現象，同時計算量也會變大。
- 音框之間的重疊完全是看電腦的運算能力來決定，若重疊多，音框率就會變大，計算量就跟著變大。若重疊少（甚至可以不重疊或跳點），音框率就會變小，計算量也跟著變小。

由一個音框計算出音高的方法很多，可以分為時域和頻域兩大類：

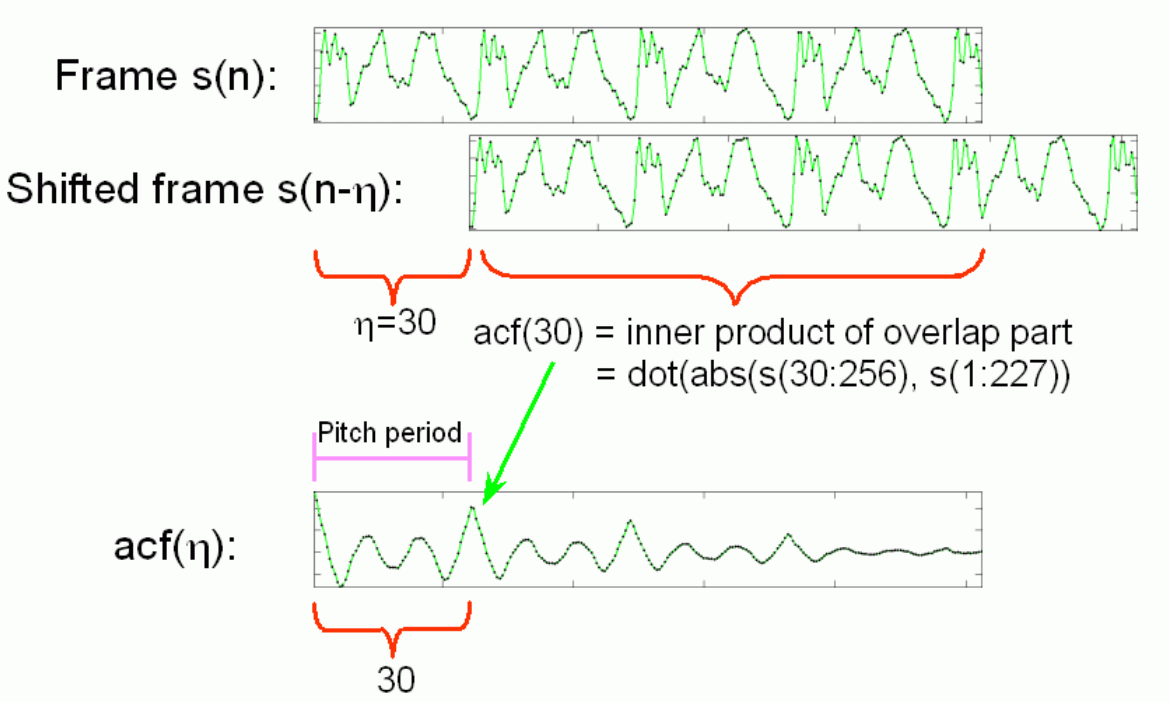
- 時域（Time Domain）
  - ACF: Autocorrelation function
  - SMDF: Average magnitude difference function
  - SIFT: Simple inverse filter tracking

- 頻域（Frequency Domain）
  - Harmonic product spectrum method
  - Cepstrum method

在本節中，我們先介紹幾種在時域上的音高追蹤的方法：

- ACF: Autocorrelation function
- SMDF: Average magnitude difference function
- SIFT: Simple inverse filter tracking

在這些方法中，ACF 和 SMDF 運算量較少，在實作上也比較容易。
我們先來看看 ACF，其示意圖如下：

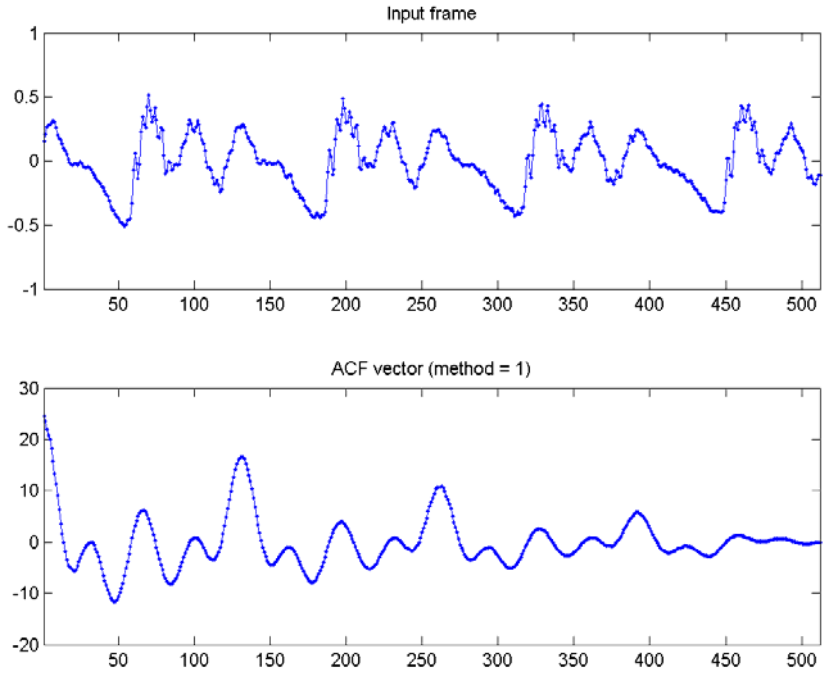


換句話說，將音框每次向右平移一點，和原本音框的重疊部分做內積，重複 n 次後會得到 n 個內積值這就是 ACF。
以我的聲音「sunday.wav」為例，如果從第 9000 點開始抓一個音框，相當於「day」的韻母左右的位置，音框長度是 512 (或 32 ms)，以此音框來進行 ACF，結果如下：

Example 1**Input file** pitchTracking/frame2acf01.m

```
waveFile='sunday.wav';
[y, fs, nbits]=wavread(waveFile);
index1=9000;
frameSize=512;
index2=index1+frameSize-1;
frame=y(index1:index2);
maxShift=length(frame);
plotOpt=1;
method=1;
acf=frame2acf(frame, maxShift, method, plotOpt);
```

Output figure



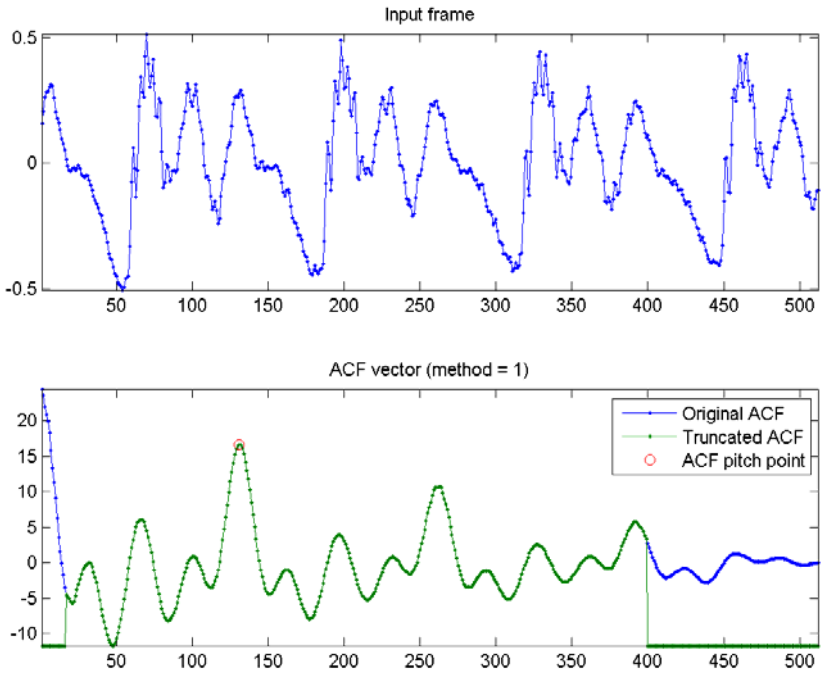
由上圖可以看出，ACF 的最大值發生在第一點，如果我們此點附近的值設定為零，就可浮現第二高點，索引值為 131，因此對應的音高是 fs/(131-1) = 16000/130 = 123.08 Hz，或 46.94 半音。在上述說明中，我們將索引值為 131 的第二最高點稱為「ACF 音高點」，只要能夠自動抓到此點，就可以直接計算音高。但要如何計算此點呢？這需要花一點時間多觀察各種音訊 ACF 曲線的形狀。一個簡單的方法，就是將 ACF 的開始數點設定成 0，再抓出最大值所在的點，即是「ACF 音高點」，範例如下：

Example 2**Input file** pitchTracking/frame2acfPitchPoint01.m

```
waveFile='sunday.wav';
[y, fs, nbits]=wavread(waveFile);
index1=9000;
frameSize=512;
index2=index1+frameSize-1;
frame=y(index1:index2);
maxShift=length(frame);
method=1;
acf=frame2acf(frame, maxShift, method);
acf2=acf;
maxFreq=1000;
acf2(1:fs/maxFreq)=min(acf);
minFreq=40;
acf2(fs/minFreq:end)=min(acf);
[maxValue, maxIndex]=max(acf2);

subplot(2,1,1);
plot(frame, '.-'); axis tight; title('Input frame');
subplot(2,1,2);
xVec=1:length(acf);
plot(xVec, acf, '.-', xVec, acf2, '.-', maxIndex, maxValue, 'ro');
```

```
axis tight; title(sprintf('ACF vector (method = %d)', method));
legend('Original ACF', 'Truncated ACF', 'ACF pitch point');
```

Output figure



在上述圖形中，上圖是音框波形，下圖是原始 ACF 曲線和修改後的 ACF 曲線，紅色圈圈即是「ACF 音高點」。

Hint

假設人聲最高頻率是 1000 Hz，而取樣頻率是 16000，那麼基本週期的點數是 16000/1000 = 16，因此我們可以將 ACF 的前 16 點設定成 0，以便抓取第二高點。

使用上述抓取 ACF 音高點的方法，我們就可以對一段聲音進行音高追蹤，並把抓到的音高播放出來，範例如下：

Example 3**Input file** pitchTracking/wave2pitchByAcf01.m

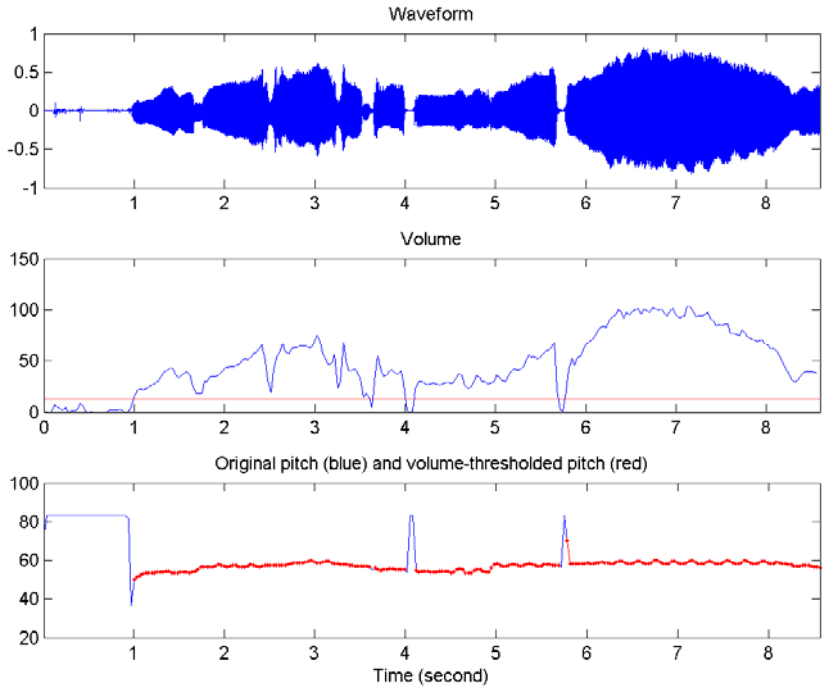
```
waveFile='soo.wav';
[y, fs, nbits]=wavread(waveFile);
y=y-mean(y);
frameDuration=32;               % in ms
frameSize=round(frameDuration*fs/1000);
overlap=0;
maxShift=frameSize;
maxFreq=1000;
minFreq=40;
n1=round(fs/maxFreq);       % acf(1:n1) will not be used
n2=round(fs/minFreq);       % acf(n2:end) will not be used
frameMat=buffer2(y, frameSize, overlap);
frameNum=size(frameMat, 2);
volume=frame2volume(frameMat);
volumeTh=max(volume)/8;
pitch=0*volume;
```

```
for i=1:frameNum
%           fprintf('%d/%d\n', i, frameNum);
            frame=frameMat(:, i);
            acf=frame2acf(frame, maxShift, 1);
            acf(1:n1)=-inf;
            acf(n2:end)=-inf;
            [maxValue, maxIndex]=max(acf);
            freq=fs/(maxIndex-1);
            pitch(i)=freq2pitch(freq);
end

frameTime=frame2sampleIndex(1:frameNum, frameSize, overlap)/fs;
subplot(3,1,1);
plot((1:length(y))/fs, y); set(gca, 'xlim', [-inf inf]);
title('Waveform');
subplot(3,1,2);
plot(frameTime, volume); set(gca, 'xlim', [-inf inf]);
line([0, length(y)/fs], volumeTh*[1, 1], 'color', 'r');
title('Volume');
subplot(3,1,3);
pitch2=pitch;
pitch2(volume
```

Output message

```
Warning: In the directory "d:\users\jang\matlab\toolbox\audioProcessing",
pitch2waveMex.mexw32 now shadows pitch2waveMex.dll.
 Please see the MATLAB 7.1 Release Notes.
> In wave2pitchByAcf01 at 43
  In goWriteOutputFile at 32
```

Output figure



在上述範例中，我們設定音量的門檻值為最大音量的 1/8，音量低於此門檻值的音框，其音高都會被設定成 0。由上圖的音高曲線可以看出來，大部分的音高都算對了，只有少部份的音高點錯了，這些錯誤的點，會造成音高曲線不平滑，在播放音高時特別刺耳。為了避免這種現象，在進行音高追蹤後，通常我們還會使用各種方法來對音高曲線進行平滑化，例如中位數濾波器（Median Filter）。讀者可以試聽看看上述範例的結果：

- 原始音訊檔案 soo.wav
- 音高曲線所對應的音訊檔案。

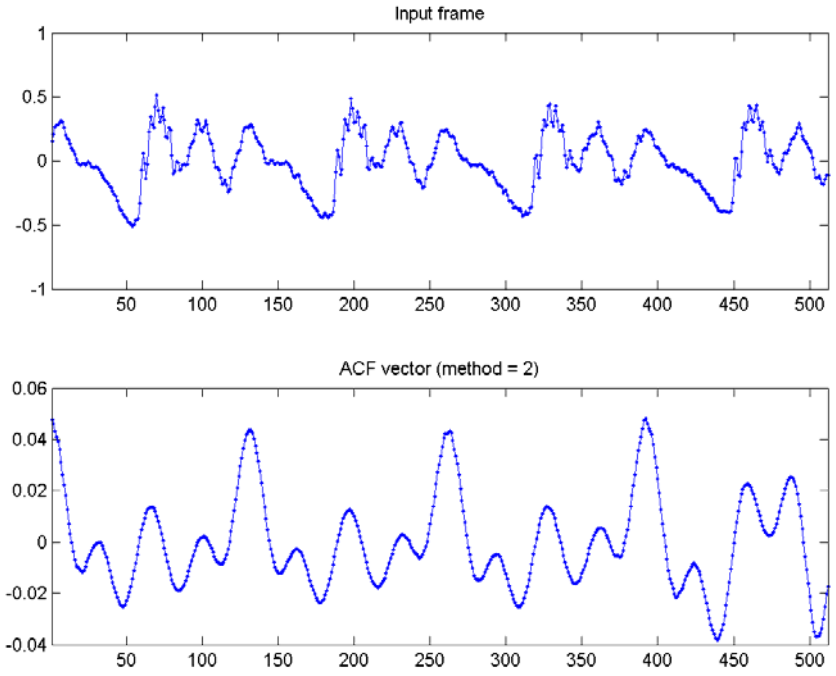
在實作 ACF 計算的過程中，通常還有下列幾中變形：

1. 上述 ACF 的計算方式，由於重疊的波形越來越短，因此 ACF 的曲線會越來越小。為了避免此種情況，我們可以求取正規化的 ACF，亦即將重疊部份的內積再除以重疊的點數，但出現的壞處是計算量的變大，而且曲線在重疊部份較少處，容易呈現不穩定的情況。範例如下：

   Example 4**Input file** pitchTracking/frame2acf02.m

```
waveFile='sunday.wav';
[y, fs, nbits]=wavread(waveFile);
index1=9000;
frameSize=512;
index2=index1+frameSize-1;
frame=y(index1:index2);
maxShift=length(frame);
plotOpt=1;
method=2;
acf=frame2acf(frame, maxShift, method, plotOpt);
```

   Output figure

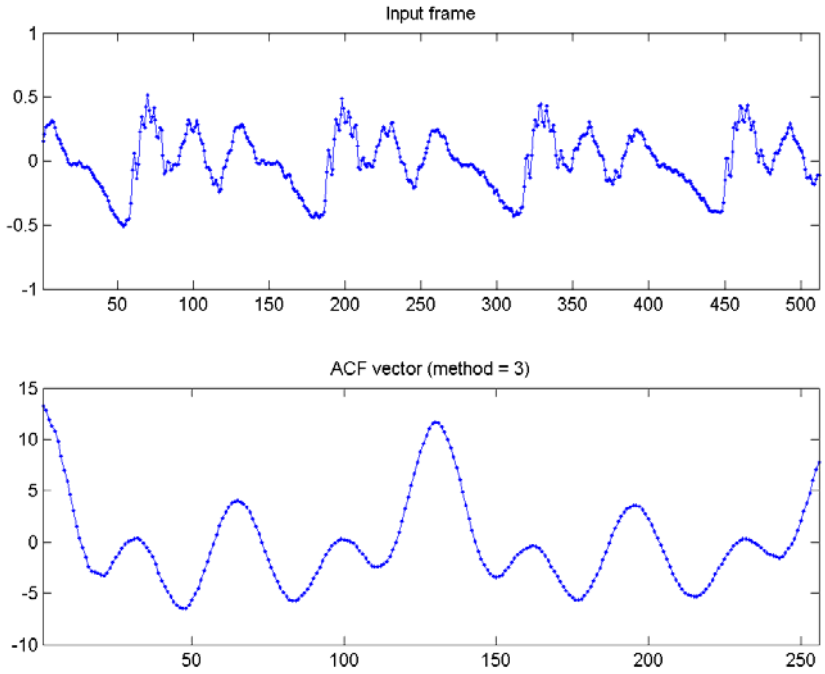


2. 另一個避免 ACF 曲線遞減的方式，是可以只對音框的前半部進行平移，讓重疊部份永遠是 256 點（音框長度的一半），得到的曲線就不會遞減，但是 ACF 曲線的長度變成原先的一半，只有 256 點。範例如下：

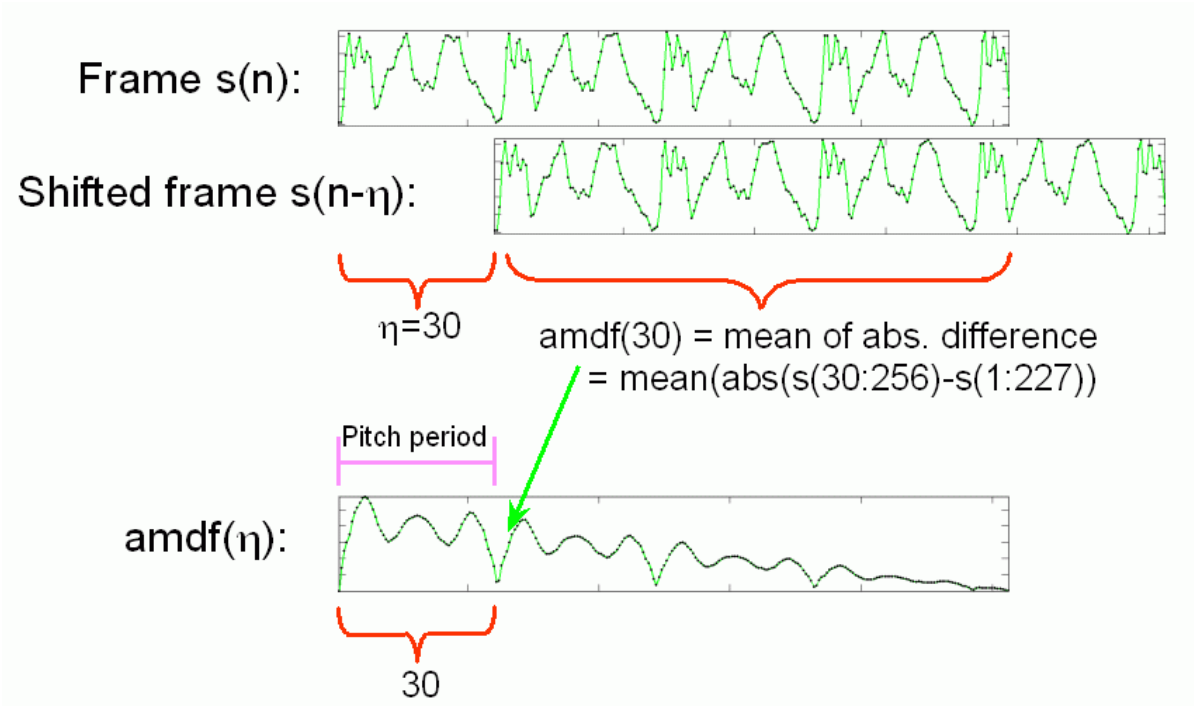
Example 5**Input file** pitchTracking/frame2acf03.m

```
waveFile='sunday.wav';
[y, fs, nbits]=wavread(waveFile);
index1=9000;
frameSize=512;
index2=index1+frameSize-1;
frame=y(index1:index2);
maxShift=length(frame)/2;
plotOpt=1;
method=3;
frame2acf(frame, maxShift, method, plotOpt);
```

Output figure



使用類似 ACF 的概念，我們可以發展出 AMDF 來求取基本週期，其示意圖如下：



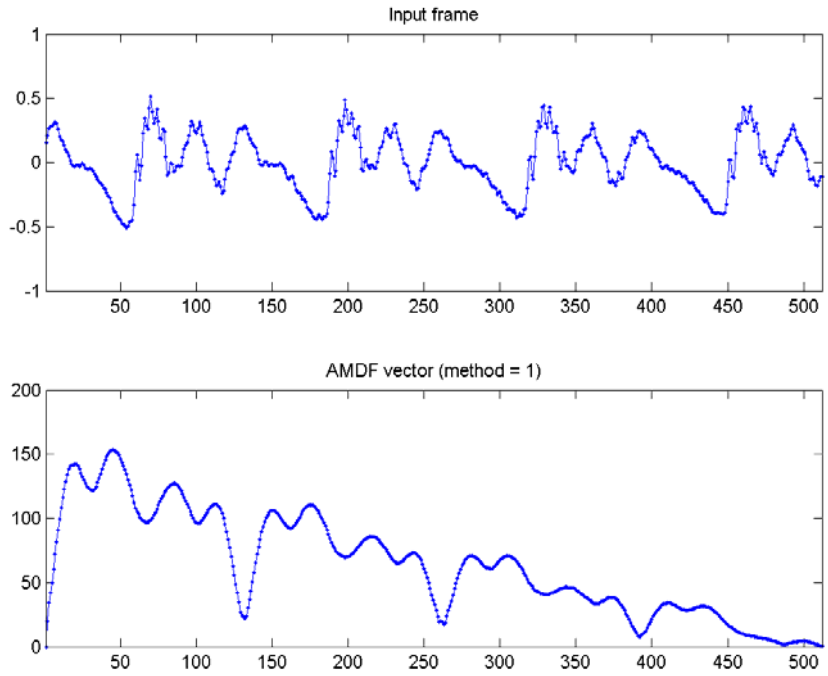
若使用與 ACF 範例相同的音框來進行 AMDF，結果如下：

Example 1**Input file** pitchTracking/frame2amdf01.m

```
waveFile='sunday.wav';
[y, fs, nbits]=wavread(waveFile);
index1=9000;
frameSize=512;
index2=index1+frameSize-1;
frame=y(index1:index2);
maxShift=length(frame);
```

```
plotOpt=1;
method=1;
frame2amdf(frame, maxShift, method, plotOpt);
```

Output figure



換句話說，將音框每次向右平移一點，和原本音框的重疊部分進行點對點的相減、取絕對值、相加，重複 n 次後會得到 n 個內積值這就是 AMDF。

由上圖可以看出，AMDF 的最小值發生在第一點，其值為零，我們可以尋求一個門檻值，抓出低於此門檻值的 AMDF 局部最低點（不包含第一點），再找出這些位置的索引值，取其最小者。以上圖而言，此值為 62，因此對應的音高就是 fs/(62-1) = 8000/61 = 131.15 Hz，或 48.04 半音，和使用 ACF 所得到的結果是相同的。

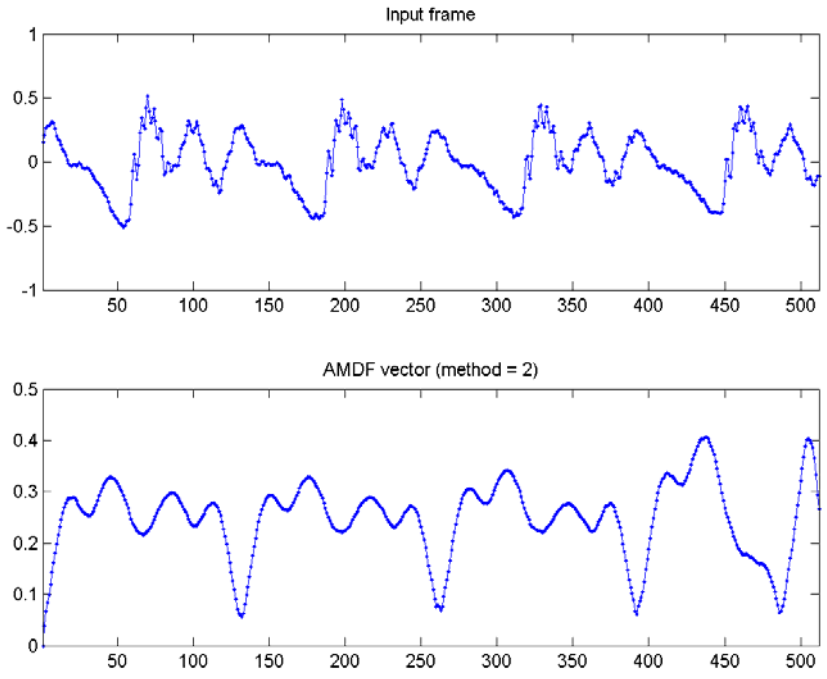
類似 ACF，AMDF 在實作上，也有下列變形：

1. 上述 AMDF 的計算方式，由於重疊的波形越來越短，因此 AMDF 的曲線會越來越小。為了避免此種情況，我們可以求取正規化的 AMDF，亦即將重疊部份的內積再除以重疊的點數，這就是傳統的 AMDF (Average Magnitude Difference Function)。但出現的壞處是計算量的變大，而且曲線在重疊部份較少處，容易受到雜訊影響而呈現不穩定的情況。範例如下：

   Example 2**Input file** pitchTracking/frame2amdf02.m

   ```
   waveFile='sunday.wav';
   [y, fs, nbits]=wavread(waveFile);
   index1=9000;
   frameSize=512;
   index2=index1+frameSize-1;
   frame=y(index1:index2);
   maxShift=length(frame);
   plotOpt=1;
   method=2;
   frame2amdf(frame, maxShift, method, plotOpt);
   ```

Output figure

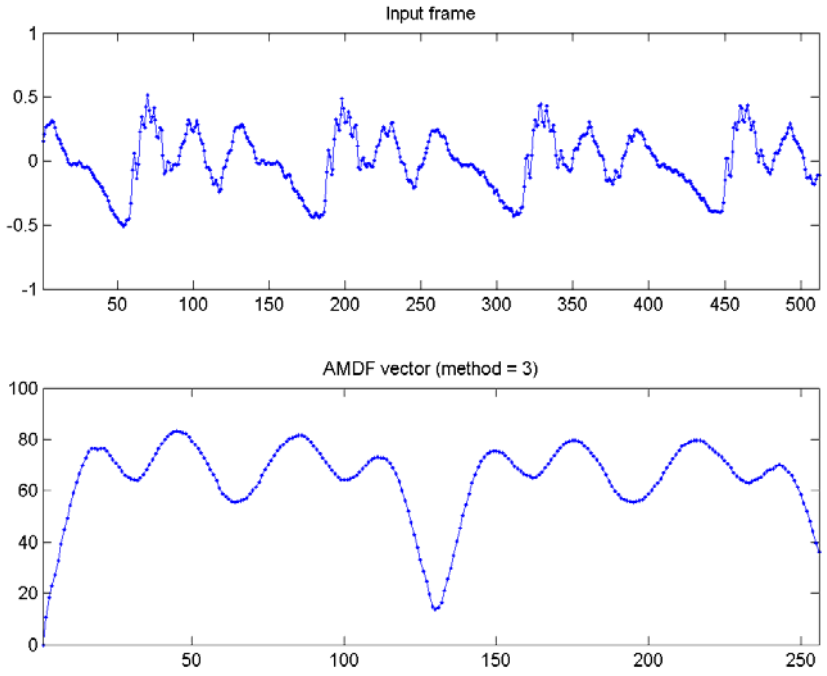


2. 另一個避免 AMDF 曲線遞減的方式，是可以只對音框的前半部進行平移，讓重疊部份永遠是 128 點（音框長度的一半），得到的曲線就不會遞減，但是 ACF 曲線的長度變成原先的一半，只有 128 點。範例如下：

Example 3**Input file** pitchTracking/frame2amdf03.m

```
waveFile='sunday.wav';
[y, fs, nbits]=wavread(waveFile);
index1=9000;
frameSize=512;
index2=index1+frameSize-1;
frame=y(index1:index2);
maxShift=length(frame)/2;
plotOpt=1;
method=3;
acf=frame2amdf(frame, maxShift, method, plotOpt);
```

Output figure

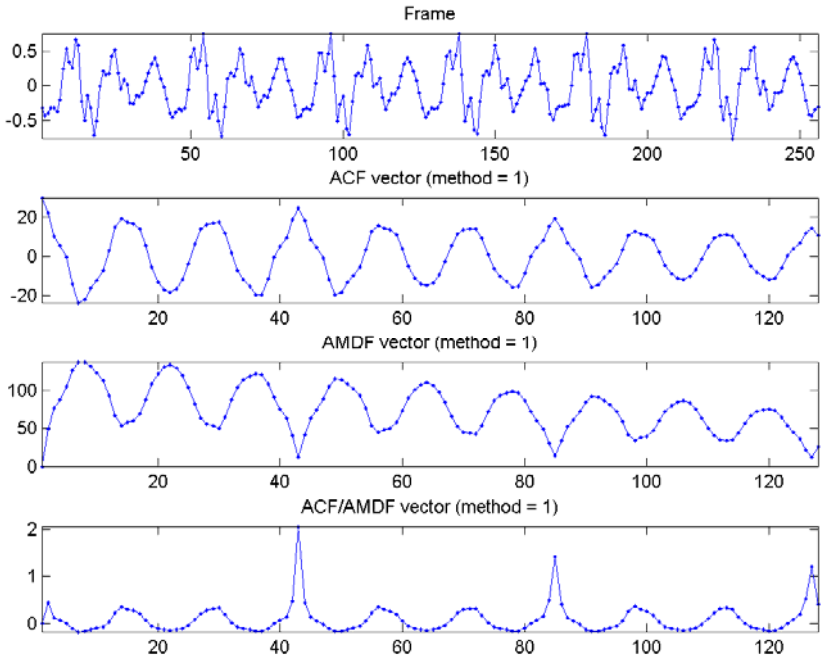


一般而言，AMDF 不牽涉到除法和乘法，因此比較適合用在微電腦嵌入式系統或是 MCU 的實作。如果 ACF 的峰點並不明顯，或是 AMDF 的谷點並不明顯，此時我們可以同時運用兩者，例如把 ACF 除以 AMDF，所得到的峰點就更為明顯了，請見下列範例：

Example 4**Input file** pitchTracking/frame2acfOverAmdf01.m

```
waveFile='soo.wav';
[y, fs, nbits]=wavread(waveFile);
frameSize=256;
frameMat=buffer(y, frameSize, 0);
frame=frameMat(:, 292);
method=1;
maxShift=length(frame)/2;
plotOpt=1;
frame2acfOverAmdf(frame, maxShift, method, plotOpt);
```

Output figure



在上述範例中，我們使用了蘇豐文老師的歌聲（蘇老師曾是台大合唱團男高音），很明顯的，無論 ACF 的最高點或是 AMDF 的最低點都不是很明顯，但是當我們將將兩者相除，所得到的 ACF/AMDF 曲線就有很明顯的最高點。

Hint

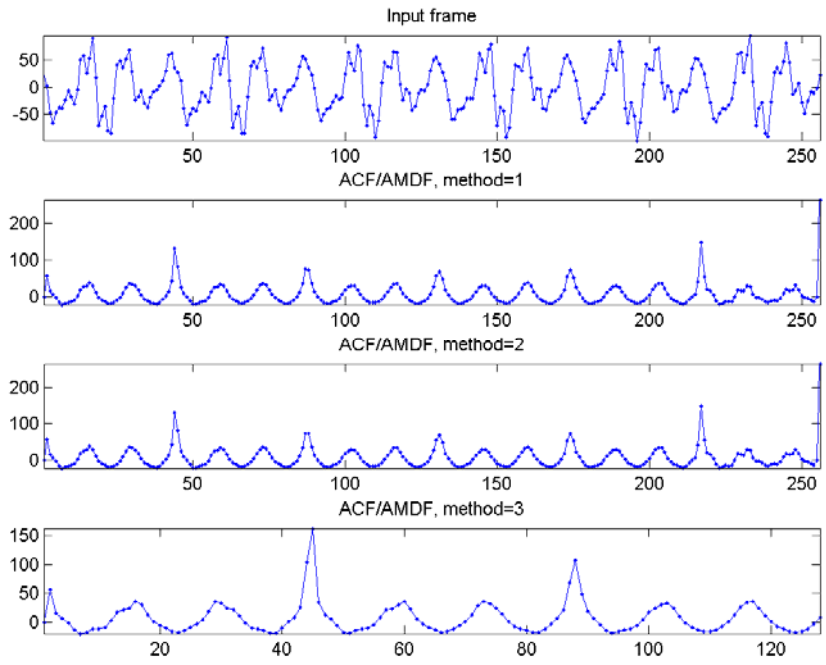
我原先也只是使用 ACF 在我們的「哼唱選歌」系統，經過了蘇老師的踢館，我才知道光使用 ACF 或是 AMDF 是不足以對付蘇老師的歌聲...

下一個範例則是使用不同的變形來計算 ACF/AMDF：

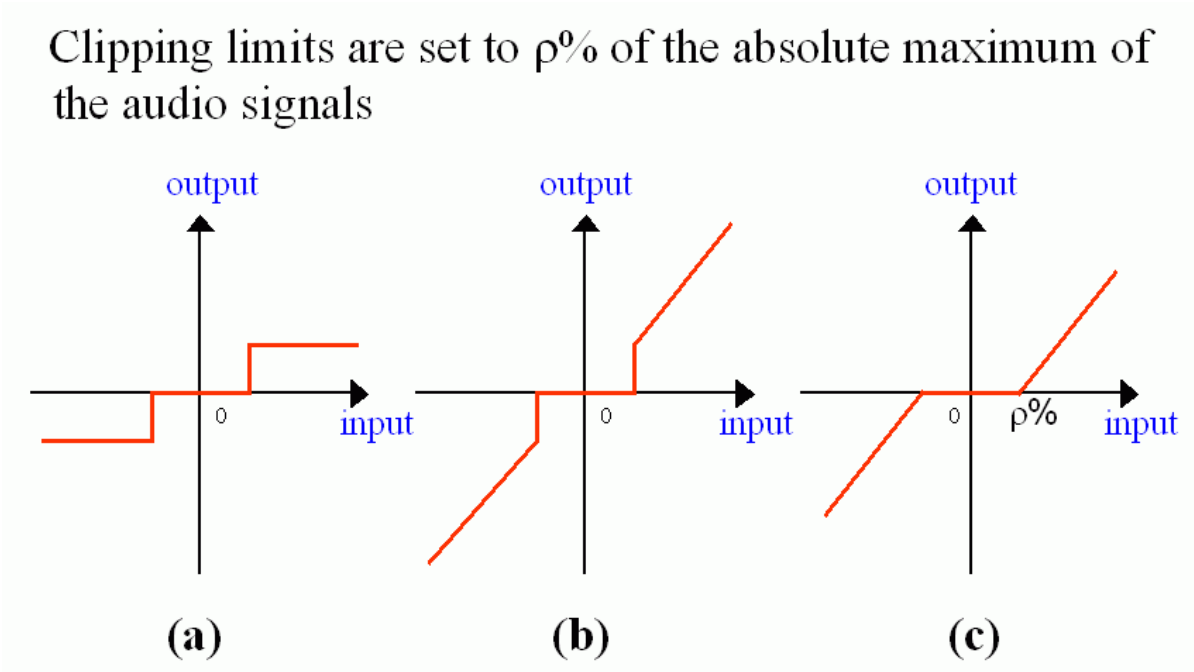
Example 5**Input file** pitchTracking/frame2acfOverAmdf02.m

```
waveFile='soo.wav';
[y, fs, nbits]=wavReadInt(waveFile);
framedY=buffer(y, 256, 0);
frame=framedY(:, 290);
subplot(4,1,1);
plot(frame, '.-');
title('Input frame'); axis tight
subplot(4,1,2);
method=1; out=frame2acfOverAmdf(frame, 256, method);
plot(out, '.-'); title('ACF/AMDF, method=1'); axis tight
subplot(4,1,3);
method=2; out=frame2acfOverAmdf(frame, 256, method);
plot(out, '.-'); title('ACF/AMDF, method=2'); axis tight
subplot(4,1,4);
method=3; out=frame2acfOverAmdf(frame, 128, method);
plot(out, '.-'); title('ACF/AMDF, method=3'); axis tight
```

Output figure

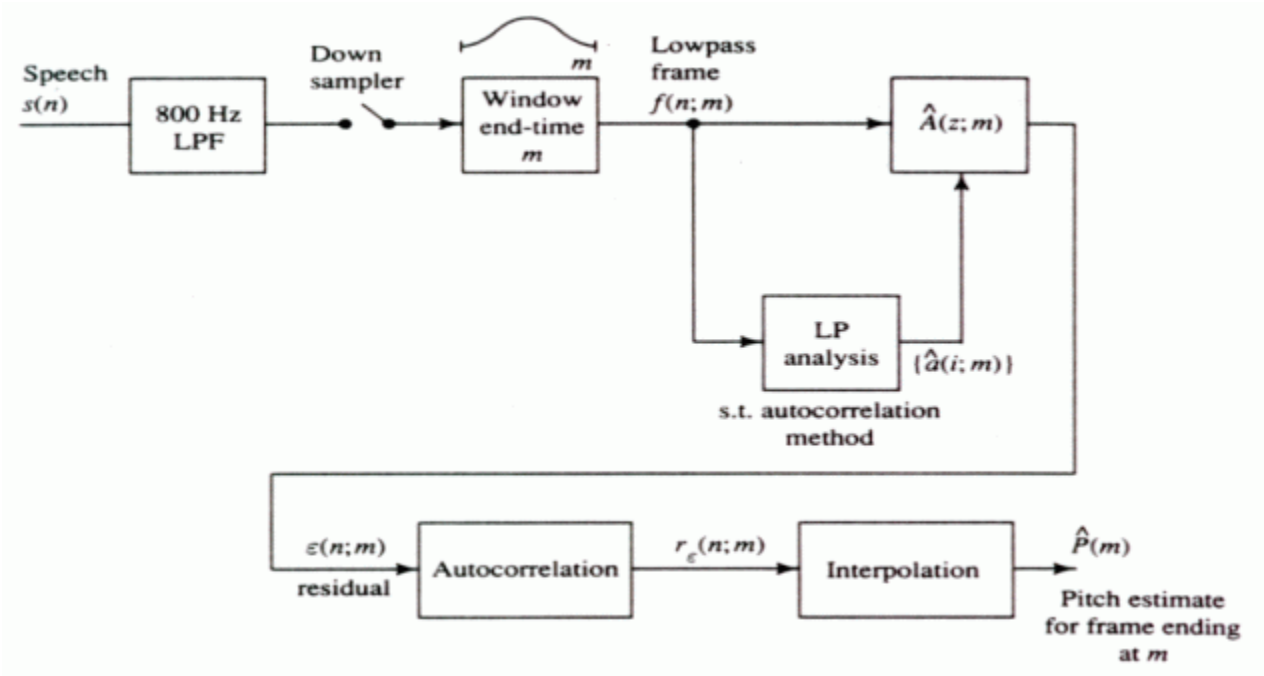


在上述範例中，method=1 和 method=2 所得到的 ACF/AMDF 曲線是一樣的。
有時候為了消除音訊在零點附近的雜訊，我們會在使用 ACF 或 AMDF 之前，先將音訊經過 Center Clipping 的處理，常用的 Center Clipping 如下：



7-4 SIFT

另外，我們在用 ACF 時，有時候會將訊號先經過 inverse filtering，企圖找到原先聲帶的原始訊號，這個原始訊號沒有經過口腔、鼻腔的作用，因此理論上會比較乾淨，ACF 所得到的效果會比較好，這個方法稱為 SIFT (Simple Inverse Filter Tracking)，其示意圖如下：



在上圖中，我們是將目前的訊號 s(n) 表示成前面 m 個訊號的線性組合：

$$s(n) = a_1 s(n-1) + a_2 s(n-2) + \ldots + a_m s(n-m) + e(n)$$

並利用最小平方法來找出最佳的 $\{a_1, a_2, \ldots, a_m\}$，使得 $\Sigma e^2(n)$ 為最小，此 e(n) 又是所謂的 excitation signal （原始激發訊號），再用此 e(n) 來進行 ACF，得到的效果較好。

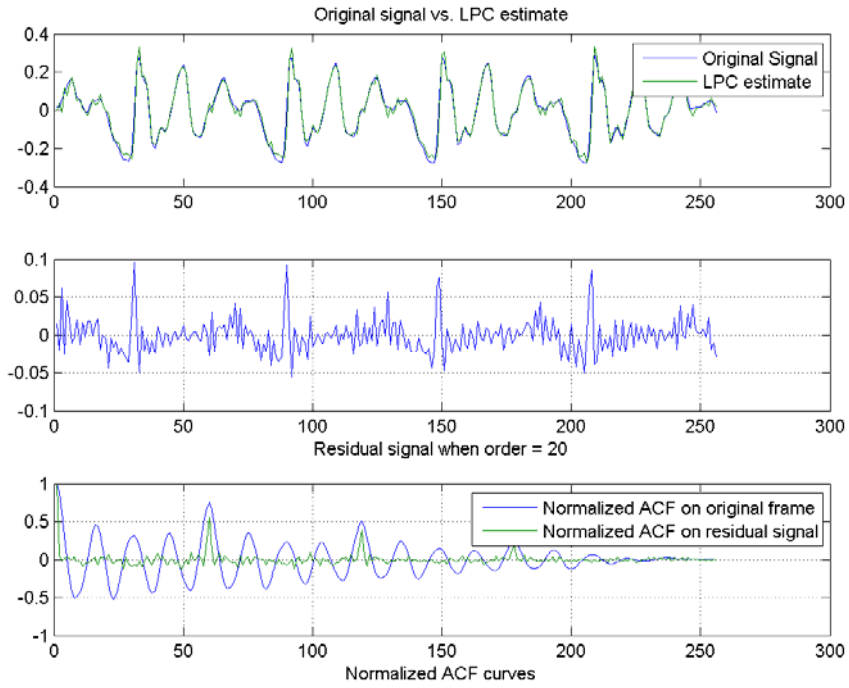
在以下的範例中，我們使用一個 order 為 20 的 LPC (linear predictive coefficients) 來進行 SIFT:

Example 1**Input file** pitchTracking/siftAcf01.m

```
waveFile = 'soo.wav';
[y, fs, nbits]=wavread(waveFile);
startIndex=15000;
frameSize=256;
endIndex=startIndex+frameSize-1;
frame=y(startIndex:endIndex);
order=20;
[frame2, error, coef]=sift(frame, order);          % Simple inverse filtering tracking
maxShift=frameSize;
method=1;
acf0=frame2acf(frame, maxShift, method);
acf1=frame2acf(error, maxShift, method);

subplot(3,1,1)
plot(1:frameSize, [frame, frame2]);
legend('Original Signal', 'LPC estimate');
title('Original signal vs. LPC estimate');
subplot(3,1,2);
plot(1:frameSize, error);
grid on
xlabel(['Residual signal when order = ', int2str(order)]);
subplot(3,1,3);
plot(1:frameSize, [acf0/max(acf0), acf1/max(acf1)]);
grid on
```

```
xlabel('Normalized ACF curves');
legend('Normalized ACF on original frame', 'Normalized ACF on residual signal');
```

Output figure



由上圖可知，經過了 SIFT，我們使用 residual signal 來進行 ACF，所得到的圖形的高點比較明顯，這會讓我們比較容易找到正確的音高點。

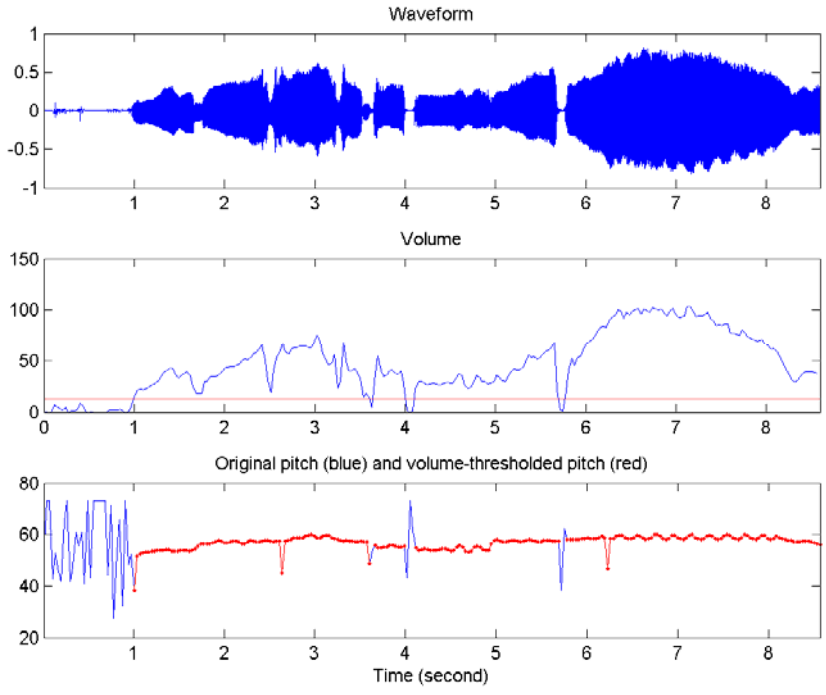
使用 SIFT 加上 ACF 來進行音高追蹤的範例如下：

Example 2**Input file** pitchTracking/wave2pitchBySiftAcf01.m

```
waveFile='soo.wav';
[y, fs, nbits]=wavread(waveFile);
y=y-mean(y);
frameDuration=32;               % in ms
frameSize=round(frameDuration*fs/1000);
overlap=0;
maxShift=frameSize;
maxFreq=1000;
minFreq=40;
n1=round(fs/maxFreq);       % acf(1:n1) will not be used
n2=round(fs/minFreq);       % acf(n2:end) will not be used
frameMat=buffer2(y, frameSize, overlap);
frameNum=size(frameMat, 2);
volume=frame2volume(frameMat);
volumeTh=max(volume)/8;
pitch=0*volume;
lpcOrder=20;                    % for sift
for i=1:frameNum
%           fprintf('%d/%d\n', i, frameNum);
            frame=frameMat(:, i);
```

```
        [frame2, error, coef]=sift(frame, lpcOrder);      % Simple inverse filtering tracking
        acf=frame2acf(error, frameSize, 1);
        acf(1:n1)=-inf;
        acf(n2:end)=-inf;
        [maxValue, maxIndex]=max(acf);
        freq=fs/(maxIndex-1);
        pitch(i)=freq2pitch(freq);
end

frameTime=frame2sampleIndex(1:frameNum, frameSize, overlap)/fs;
subplot(3,1,1);
plot((1:length(y))/fs, y); set(gca, 'xlim', [-inf inf]);
title('Waveform');
subplot(3,1,2);
plot(frameTime, volume); set(gca, 'xlim', [-inf inf]);
line([0, length(y)/fs], volumeTh*[1, 1], 'color', 'r');
title('Volume');
subplot(3,1,3);
pitch2=pitch;
pitch2(volume
```

Output figure



當遇到雜訊時，基本上是沒有音高存在的，所以 ACF 或是 SMDF 得到的值都是不合理的過低或過高值，因此在實作時，我們必須限定合理的音高範圍，如果超過此範圍，我們就認定沒有音高存在（通常是直接將音高設定為零。）。
有關於音高追蹤的各種其他方法，請見：

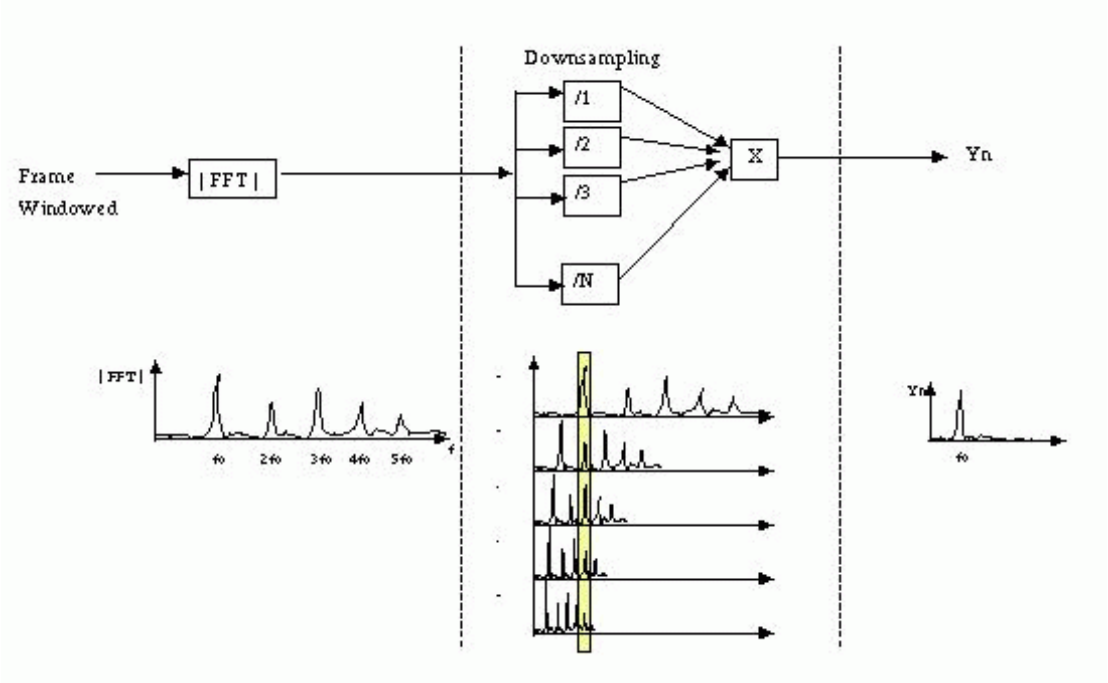
- Pitch Detection Methods Review
- Pitch Detection

7-5 HPS

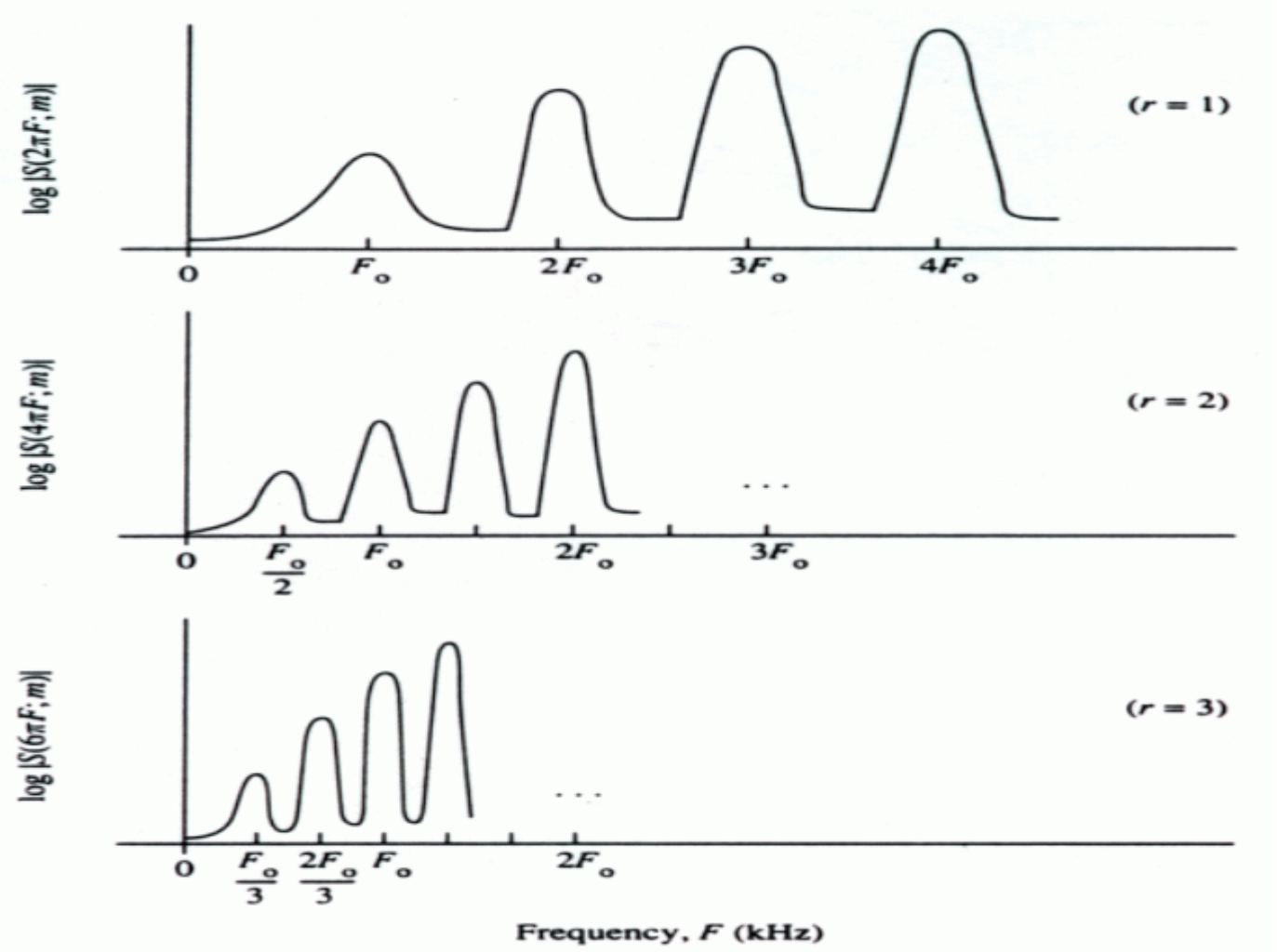
本節介紹在頻域上的音高追蹤的方法，包含

- Harmonic product spectrum (HPS)
- Cepstrum method

首先我們來看看 HPS，其示意圖如下：



其中在 down-sampling 的部分，我們將頻譜訊號進行向下取樣，r=1 代表原訊號，r=2 代表隔一點抓一點（訊號長度只有原先的 1/2），r=3 代表隔兩點抓一點（訊號長度只有原先的 1/3），依此類推，得到各個「壓縮」的版本，再將這些「壓縮」的訊號加起來，示意圖如下：



由於每一個壓縮後的訊號都會在基頻附近有一個高點，所以累加的結果，就會凸顯這個高點，比較容易看出基頻的位置。

以我的聲音「清華大學資訊系 3.wav」為例，如果從第 7200 點開始抓一個音框，相當於「華」的韻母左右的位置，音框長度是 256，以此音框來進行 HPS，結果如下：

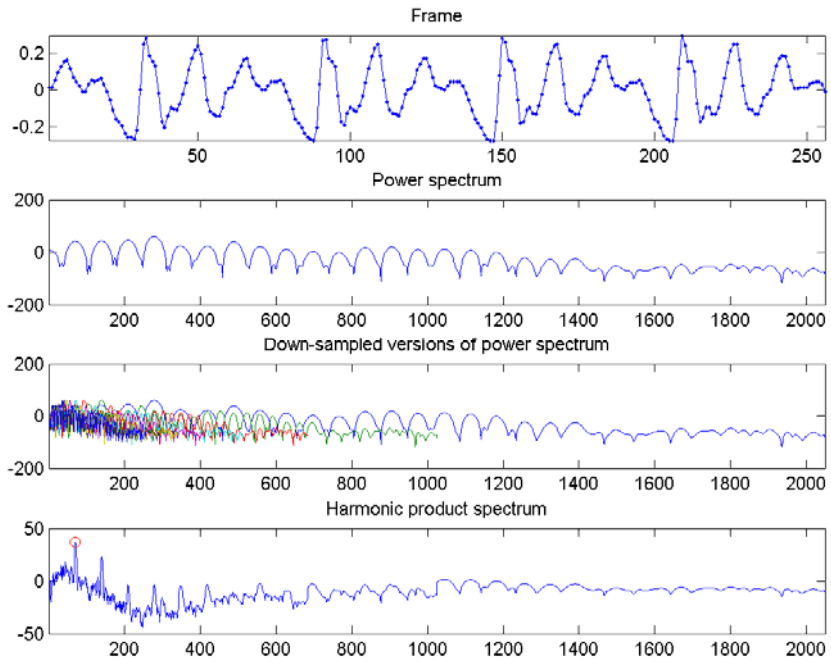
Example 1**Input file** pitchTracking/hps01.m

```
waveFile = 'soo.wav';
[y, fs, nbits]=wavread(waveFile);
startIndex=15000;
frameSize=256;
endIndex=startIndex+frameSize-1;
frame = y(startIndex:endIndex);
zeroPaddedFrameSize=16*frameSize;
output=frame2hps(frame, zeroPaddedFrameSize, 1);
[maxValue, maxIndex]=max(output);
line(maxIndex, output(maxIndex), 'marker', 'o', 'color', 'r');
fprintf('Pitch frequency = %f Hz\n', fs/zeroPaddedFrameSize*(maxIndex-1));
```

Output message

```
Pitch frequency = 188.415527 Hz
```

Output figure



由上述圖形可知，HPS 的最高點的索引值是 71，因此對應的音高就是： fs/(16*frameSize)*(71-1) = 11025/(16*256)*70 = 188.42 Hz，或 54.32 半音。

HPS 的特性說明如下：

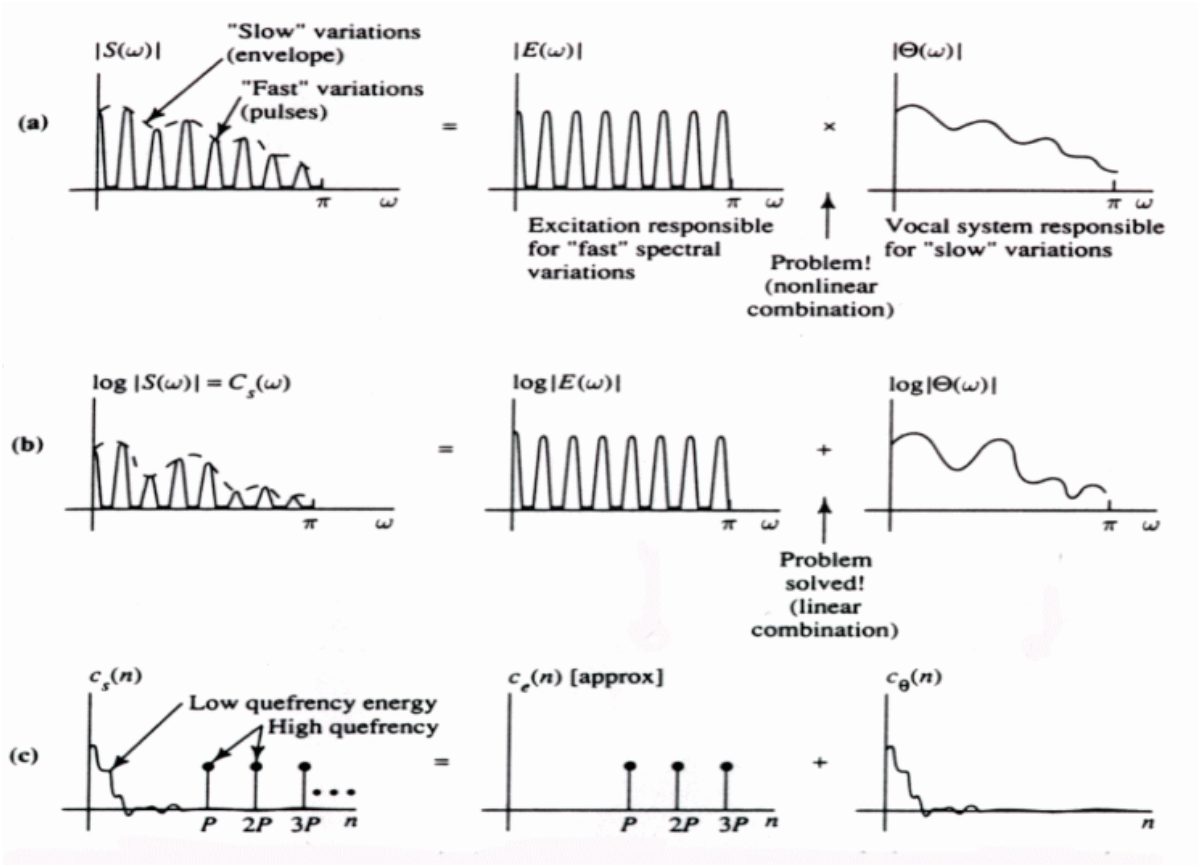
- HPS 所得到的音高解析度並不高，以上述範例而言，若不進行補零（Zero Padding），則無論最高點的位置為何，音高都會是 11025/256 = 43.07 Hz z 的倍數。若需要提高解析度，我們可以將音框進行補零，此時在頻譜就會有較高的解析度，得到的音高解析度也會提高。在上述範例中，我們進行補零直到音框長度為 16*256 = 4096，所得到的頻率解析度是 fs/4096 = 11025/4096 = 2.69 Hz。
- 由 HPS 得到的音高並不很穩定，因為頻譜容易受到共振峰的影響，所以有時候 HPS 得到的最大值並非對應於正確音高。若先拿掉共振峰的影響，再來進行 HPS，得到的效果可能更好。

7-6 Cepstrum

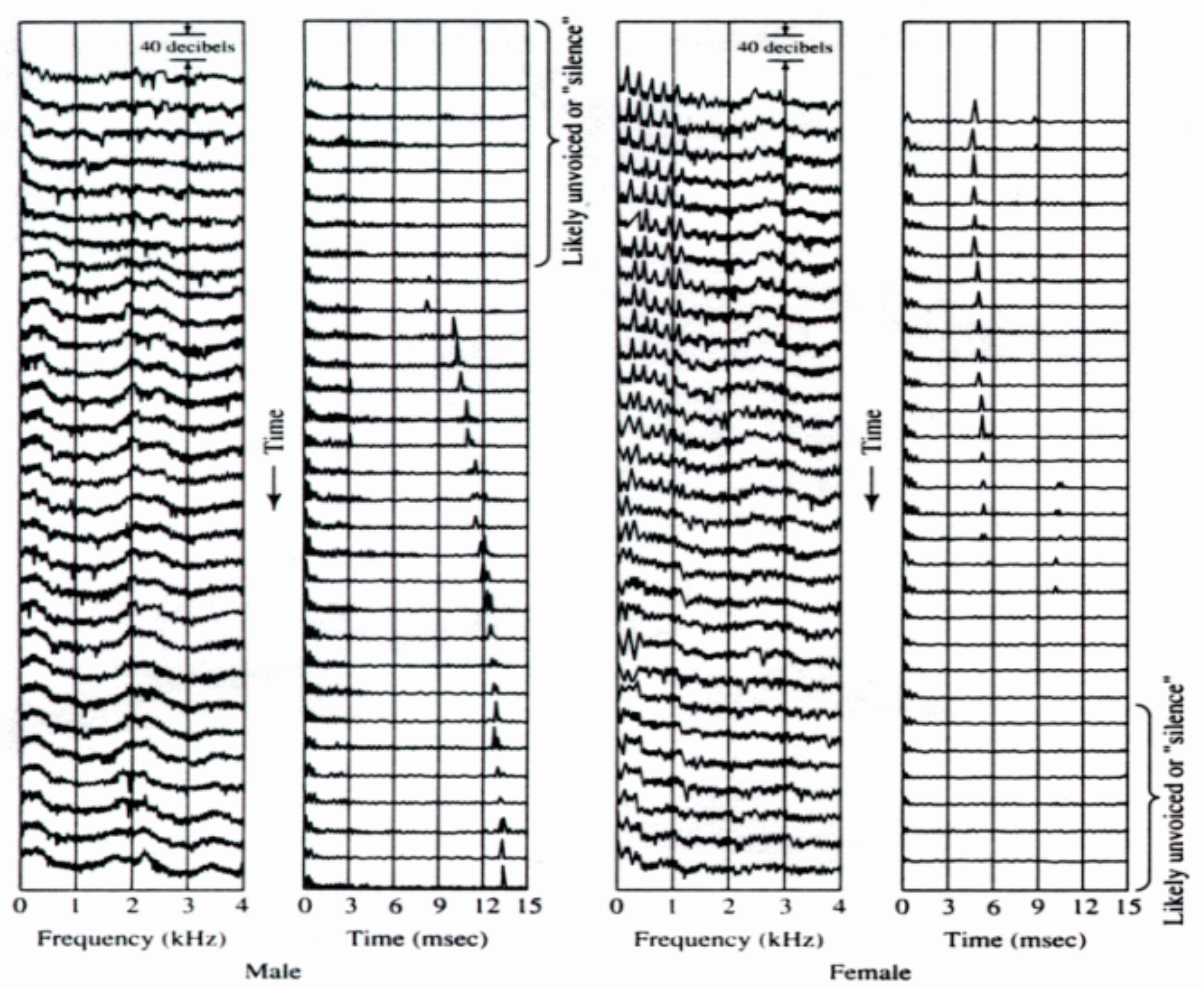
另一個在頻域上的音高追蹤的方法，則是使用「倒頻譜」（Cepstrum），其定義是

cepstrum = |ifft(log(|fft(frame)|))|

倒頻譜的物理意義，可用下圖說明：



典型的範例如下：



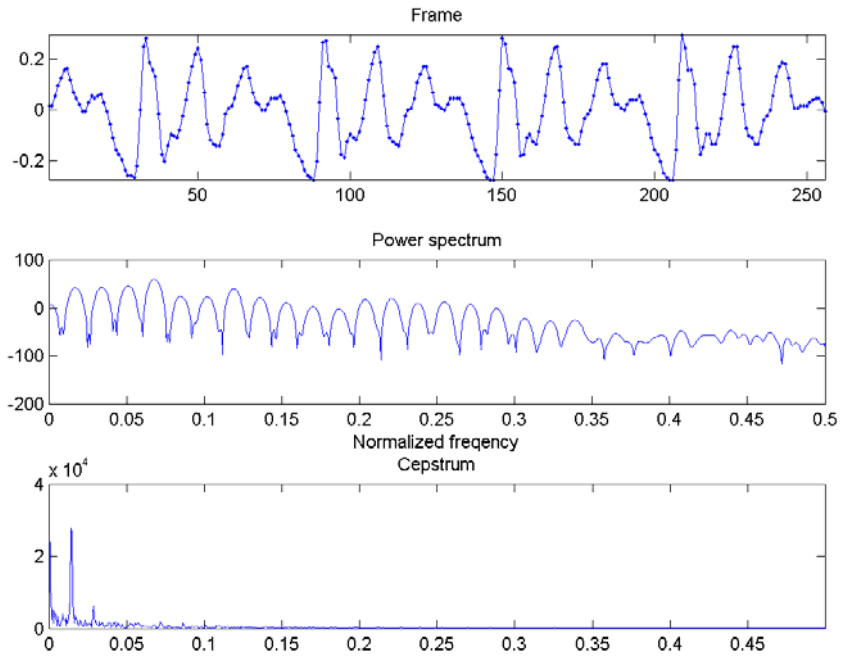
因此，只要使用 high-pass liftering，就可以把在 low quefrency 部分拿掉，凸顯高點，就可以抓出基頻的位置。

在下面這個範例，我們對一個音框進行倒頻譜的計算：

Example 1**Input file** pitchTracking/ceps01.m

```
waveFile = 'soo.wav';
[y, fs, nbits]=wavread(waveFile);
startIndex=15000;
frameSize=256;
endIndex=startIndex+frameSize-1;
frame = y(startIndex:endIndex);
zeroPaddedFrameSize=16*frameSize;
output=frame2ceps(frame, zeroPaddedFrameSize, 1);
[maxValue, maxIndex]=max(output);
line(maxIndex, output(maxIndex), 'marker', 'o', 'color', 'r');
%fprintf('Pitch frequency = %f Hz\n', fs/zeroPaddedFrameSize*(maxIndex-1));
```

Output figure



7-7 How to Increase Pitch Resolution (音高解析度的提升)

另一個常碰到的問題，是音高解析度的問題。通常一個半音差等於 100 個「音分」（Cents），若希望能得到高解析度的音高，我們可以調高取樣頻率以增加音訊在時域的解析度，這也會使 ACF 或是 SMDF 的解析度隨之提高，進而提高音高的解析度。從數學上來說，音高和基本頻率的關係如下：

$$p = 69 + 12*\log_2(f/440) = 69 + 12*\log_2((fs/L)/440)$$

其中 L 是基本週期的點數。當 L 增加 1 時，音高的改變量可以表示如下：

$$\Delta P = (69 + 12*\log_2((fs/L)/440)) - (69 + 12*\log_2((fs/(L+1))/440)) = -12*\log_2(1+1/L) = -12*\log_2(1+f/fs)$$

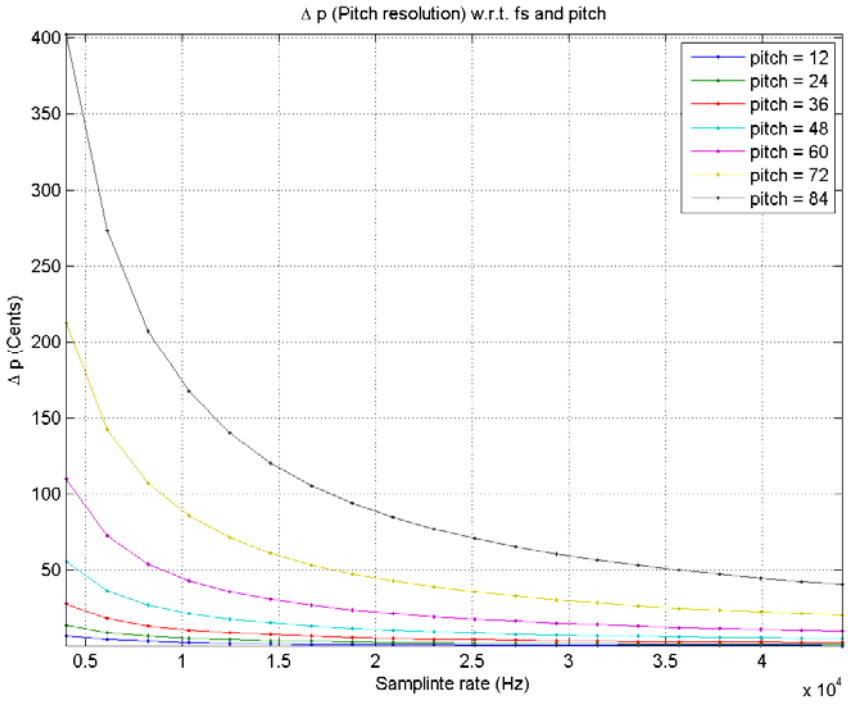
其中 f 是基本頻率，fs 則是取樣頻率，ΔP 隨 fs 的變化情況可以使用下列範例來畫圖說明：

Example 1**Input file** pitchTracking/pitchResolution01.m

```
% Pitch resolution w.r.t. sampling rate and pitch
fs=linspace(4000, 44100, 20)';
pitch=12*(1:7);
deltaP=[];
```

```
for i=1:length(pitch)
        f=440*2^((pitch(i)-69)/12);
        deltaP=[deltaP, 12*log2(1+f./fs)];
end
plot(fs, 100*deltaP, '.-');
axis tight; grid on
xlabel('Samplinte rate (Hz)');
ylabel('\Delta p (Cents)');
title('\Delta p (Pitch resolution) w.r.t. fs and pitch');
% Display legends
pitchStr={};
for i=1:length(pitch)
        pitchStr={pitchStr{:}, ['pitch = ', int2str(pitch(i))]};
end
legend(pitchStr);
```

Output figure



由以上圖形可以看出，當取樣頻率降低時，或是基本頻率提高時，音高的誤差就會變大（解析度降低）。通常基本頻率不是我們所能控制的，因此若要提高音高的解析度，對於在時域上的音高追蹤方法，我們可以提高取樣頻率，或是對聲波（或是 ACF、SMDF）進行向上取樣（Up Sampling）或是內差（Interpolation）。

同理，若要提高音高的解析度，對於在頻域上的音高追蹤方法，我們可以直接在每個音框進行補零（Zero Padding），此時在頻譜的解析度就會提高，所計算出來的音高解析度也就跟著提高了。

7-8 Software for Pitch Tracking (音高抓取的軟體)

可用來抓取聲音音高的軟體相當多，以下列出幾種常用軟體，各位同學可以自行下載試試看。

- Speech Filing System (SFS): http://www.phon.ucl.ac.uk/resource/sfs/
- Speech Analyzer
- Solo Explorer

- Praat
    1. (**) **計算 ACF**： 請寫一個函數 myFrame2acf，將一個音框轉換成 ACF，其用法如下：

       ```
       acf = myFrame2acf(frame);
       ```

       其中 acf 的長度應該是和 frame 一樣。（提示：請直接參考 Audio Toolbox 裡面的 frame2acf.m。你也可以使用 Signal Processing Toolbox 中的 xcorr 函數來完成此題。）
    2. (**) **計算 AMDF**： 請寫一個函數 myFrame2amdf，將一個音框轉換成 AMDF，其用法如下：

       ```
       amdf = myFrame2amdf(frame);
       ```

       其中 amdf 的長度應該是和 frame 一樣。（提示：請直接參考 Audio Toolbox 裡面的 frame2acf.m。）
    3. (**) **計算 AMDF/ACF**： 請寫一個函數 myFrame2amdfOverAcf，將一個音框轉換成 AMDF/ACF，其用法如下：

       ```
       amdfOverAcf = myFrame2amdfOverAcf(frame);
       ```

       其中 amdfOverAcf 的長度應該是和 frame 一樣。請利用此函數，畫出類似此範例的圖，看看此方法是否能夠比單獨使用 ACF 或是 AMDF 更有用。若此方法不適合用來進行音高追蹤，請問有何改進方法？（提示：請直接參考 Audio Toolbox 裡面的 frame2acfOverAmdf.m。）
    4. (**) **由 ACF 計算音高**： 請寫一個函數 myAcf2pitch，由一個 ACF 向量來計算音高，其用法如下：

       ```
       pitch = myAcf2pitch(acf, fs, plotOpt);
       ```

       其中 acf 是一個 ACF 向量，fs 則是取樣頻率，pitch 則是音高（以 Semitone 為單位），plotOpt 則決定是否畫圖，若其值不等於 0，則必須畫出 ACF 及你的程式所選出來的「ACF 音高點」。（提示：建議你在使用 frame2acf.m 計算 ACF 時，使用的方法是 method=1，然後設定 ACF 的前十點等於 0，再用 max() 函數來抓出最大值的位置，就可以計算音高。你會用到 Audio Toolbox 的 freq2pitch.m。畫圖部份，可以參考此範例。）
    5. (**) **由 AMDF 計算音高**： 請寫一個函數 myAmdf2pitch，由一個 AMDF 向量來計算音高，其用法如下：

       ```
       pitch = myAmdf2pitch(amdf, fs, plotOpt);
       ```

       其中 amdf 是一個 AMDF 向量，fs 則是取樣頻率，pitch 則是音高（以 Semitone 為單位），plotOpt 則決定是否畫圖，若其值不等於 0，則必須畫出 AMDF 及你的程式所選出來的「AMDF 音高點」。（提示：你會用到 Utility Toolbox 的 localMin.m/localMax.m，以及 Audio Toolbox 的 freq2pitch.m。）
    6. (**) **由一個音框計算一點音高**： 請寫一個函數 myFrame2pitch，由一個音框來計算音高，其用法如下：

       ```
       pitch = myFrame2pitch(frame, fs, method, plotOpt);
       ```

       其中 frame 是一個音框向量，fs 是取樣頻率，method 則是代表使用的方法（'acf' 代表使用 ACF，'amdf' 代表使用 AMDF，依此類推），pitch 則是音高（以 Semitone 為單位），plotOpt 則決定是否畫圖，若其值不等於 0，則必須畫出音框、ACF（或是 AMDF 等）及你的程式所選來計算音高的最佳點。另：你的函數必須要有 selfdemo 的功能，以便展示，請參考 Audio Toolbox 裡面的 frame2acf.m 或 frame2amdf.m 等函數。（提示：你會用到 Audio Toolbox 裡面的 freq2pitch.m、frame2acf.m、frame2amdf.m 等函數，以及前面幾題的 myAcf2pitch.m、myAmdf2pitch.m 等函數。）
    7. (***) **音高的計算、顯示及播放之一**： 請寫一段 MATLAB 程式 showPitch01.m，其功能如下：

- 讀入檔案 DoReMi.wav。
- 切出音框，其中音框大小是 256 點，相鄰音框不重疊。
- 對每一個音框計算音量，並使用之前所學的端點偵測的方法，求出音量門檻值。
- 對每一個音框計算下列任一個向量：ACF、AMDF、ACF/AMDF、AMDF/ACF。
- 根據上一步驟，算出每一個音框所對應的音高值，並轉成以 semitone 為單位。（提示：第一次進行時，可以直接抓出 ACF 或是 ACF/AMDF 的最大值所在的索引，或是 AMDF 或 AMDF/ACF 最小值所在的索引，看看效果如何。後續修正時，可以抓取數個最大值或最小值，然後再各種方法從這些候選者中選出最可能的答案。）
- 對音高向量進行後處理，使音高曲線是一段平滑的曲線，同時在音高不存在時，其值為零，可能方法如下：
  - 若某一個音框的音量低於音量門檻值，可直接將此音框的音高設定為 0。可以使用前幾章所學的「端點偵測」的各種方法來設定音量門檻值。
  - 若某一點音高偏離人聲的音高範圍，表示可能是雜訊或是氣音，此時可以將音高設定為零。
  - 對於單獨暴增或暴減的音高點，可將其值設為左右點的平均值。
  - 對音高向量進行平滑化，可用 Median Filter，相關的指令是 median。
  - 其他你可以想到的各種方法。
- 畫出結果，圖形視窗內必須有三個圖，第一個圖是原始聲音訊號，第二個圖是音量圖，第三個圖是音高圖，這三個圖的 X 軸必須都是以秒為單位。
- 播放你所找到的音高。（可使用 Audio Processing Toolbox 的 pvPlay.m。）

你所畫出來的音高向量必須盡量連續，而播放出來的音高也必須盡量接近原歌者的音高。

8. (***) **音高的計算、顯示及播放之二：** 請寫一段 MATLAB 程式 showPitch02.m，功能和前一題一樣，但是改用蘇豐文老師的歌聲。
9. (***) **由一段聲音來計算音高向量：** 請寫一個函數 myWave2pitch，由一段聲音來計算音高向量，其用法如下：

```
pitch = myWave2pitch(wave, fs, frameSize, overlap, method);
```

其中 wave 是一段音訊向量，fs 是取樣頻率，frameSize 是音框點數，overlap 是相鄰音框的重疊點數，method 則是代表使用的方法（'acf' 代表使用 ACF，'amdf' 代表使用 AMDF），而 pitch 則是音高（以 Semitone 為單位）。請注意，你找出來的音高向量應該是一段連續的曲線，你可以使用各種方法來修整 pitch，使得所找出來的 pitch 合於常理，請見上題的方法說明。（提示：你會用到 Audio Toolbox 以及前面所寫的函數！）

10. (***) **音高的播放：** 請寫一個函數 myPitchPlay.m，可輸入一段音高向量，然後計算以正弦波重現此音高向量的波形，以便播放，用法如下：

```
wave = myPitchPlay(pitch, frameRate, fs);
```

其中 pitch 是音高向量（若有元素為零，代表靜音），frameRate 是每秒鐘的音框個數（每個音框會產生一個音高點），fs 則是合成訊號 wave 的取樣頻率。此函數同時會以 sound(wave, fs) 來播放合成的聲音。請特別注意，在連接正弦波時，必須盡量讓波形連續，否則會出現規律性的雜音。

a. 請用此函數來合成一段「Do, Re, Mi, Fa, So, La, Si, Do」的聲音。（提示：通常一秒有兩拍，而上述音階對應到的半音差是「60, 62, 64, 65, 67, 69, 71, 72」。）
b. 請使用此函數來播放前一題所計算出來的音高向量。

Chapter 8: 音高追蹤的應用

8-1 旋律辨識

旋律辨識（Melody Recognition）是根據音高來辨識音樂的技術，其基本目的乃是要讓使用者在哼唱一首歌之後，系統即可以從使用者的歌聲計算出音高向量，接者以此音高向量在音樂資料庫中比對，找出最接近的歌。由於這種找歌方式並不牽涉到音樂的 Metadata（例如歌曲名稱、歌詞、歌手、專輯、作詞者、作曲者等文字資訊），所以又稱為「內容式的音樂檢索」（Content-based Music Retrieval）。

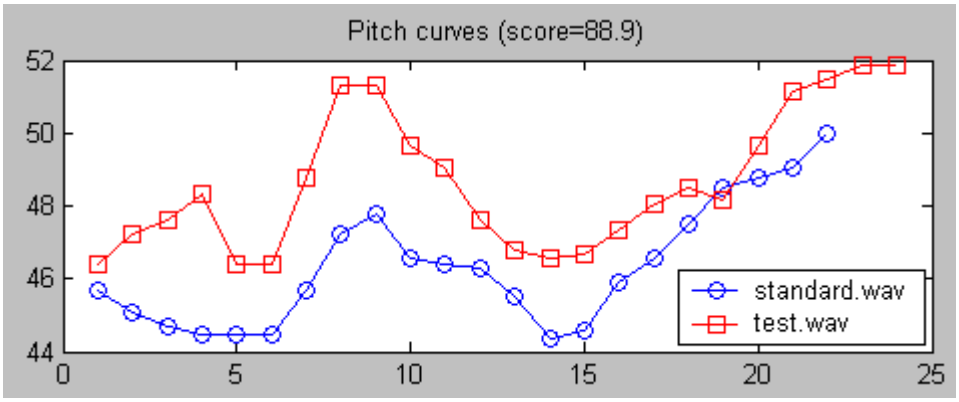
（待完成，請稍候，謝謝！）

8-2 音調評分

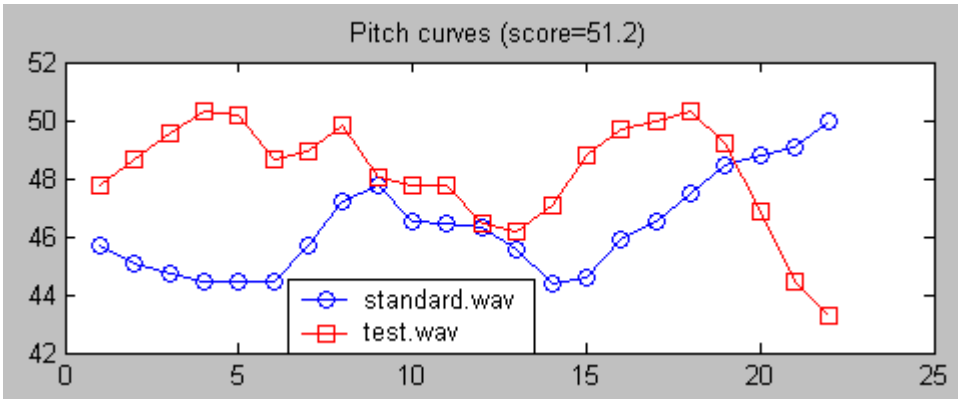
「音調評分」（Intonation Assessment）的目的是要以電腦來自動評斷一個人發音的音調是否標準，並提示相異之處。舉例來說，外國人到了台灣學國語，最難學的就是國語的四聲，所以他們會說「窩腰刀胎杯刊顛鷹」（我要到台北看電影），我們一聽，就知道是外國人在講國語，因為完全沒有音調（全部都是一聲）。同樣地，我們在學英文時，碰到較長的句子，也很難把英文的音調抓準，大部分只能平平地唸，因此老外一聽到老中講音調怪怪的英文，就知道你不是 native speaker。「音調評分」的目的，就是希望以電腦的快速運算能力，來顯示你講的英文語調和老外的標準語調到底差多少，並同時顯示差異之處，以達到電腦輔助語言學習（CALL, Computer-Assisted Language Learning）的宗旨。

以下是一個簡單的範例：

- 老外講的標準語句：Can I take your order?
- 使用我講的好的測試語句，下圖是電腦畫的音調曲線，其中老外的音調曲線是用圓圈（藍色）來標示，我的音調曲線則是用方塊（紅色）來標示。由於我講的音調還可以，尾音有跟著上揚，所以兩條曲線的趨勢基本上是一致的，因此我得到了 88.9 分。



- 使用我講的壞的測試語句，下圖是電腦畫的音調曲線，其中老外的音調曲線是用圓圈（藍色）來標示，我的音調曲線則是用方塊（紅色）來標示。由於我講的音調是肯定句的音調，尾音是下降的，和老外尾音上揚的的音調差很多，兩條曲線的趨勢並不一致，因此我只得到了 51.2 分。



可能的實際應用如下：

- **英文覆讀機、電子英文學習機**：我們已經將此音調計算及評分機制實現在 8-bit 8051 平台和凌陽 16-bit SPCE061 平台，這兩款微控制器的價格都在一美元左右，因此很適合用在移動式的語言學習機。例如用在英文覆讀機或電子英文學習機，使用者不但可以聽到例句的發音，更可以對著機器講同樣一句話，系統就會根據音調相似度來評分，同時在螢幕上畫出兩條音調曲線，讓使用者可以知道哪裡的音調不對，進而可以反覆練習。同時也可以運用利用音調評分的機制來設計從基本到進階的音調自我測驗，以便循序漸進、自我挑戰、精益求精。
- **語言學習軟體**：一般人常問的問題是：是否有可能針對語音內容的相似度進行評分呢？基本上，若要對語音內容進行評分，就要牽涉到語音辨識的計算，因此計算量比較大，比較難在低階的 MCU 平台實現。但由於一般個人電腦的運算能力已經很強大，因此我們可以在一般個人電腦上進行完整的語音評分（音調評分只是其中一部份），可以用在各種語言學習軟體，比對的參數比較廣泛，可以包含音量、音高、過零率、有聲與無聲音、倒頻譜參數、音節持續時間等，以達到更客觀的評分與分析。

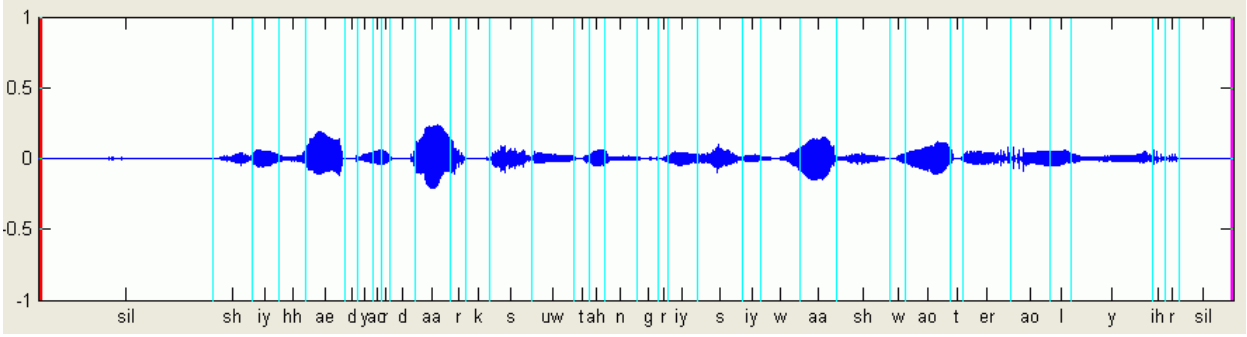
8-3 語音評分

由於個人電腦的普及和運算速度的提高，「電腦輔助語言學習」（CALL, Computer Assisted Language Learning）的研發已經到達可以商業化的階段，其中又以「電腦輔助發音訓練」（CAPT, Computer Assisted Pronunciation Training）最受到矚目，因為其實用性很高，互動性很強，可以大量彌補口說語言師資的不足。一般而言，我們又把「電腦輔助發音訓練」簡稱為「語音評分」或「發音評分」等。

以口說英文為例，語音評分的目的是要以電腦來自動評斷一個人的一句英文發音是否標準，並和老外講的同一句話來進行比較，以圖表列出相近及相異之處，並以聲音或動畫來提示正確發音，讓使用者反覆練習，以改進個人的英語發音。語音評分的流程可以說明如下：

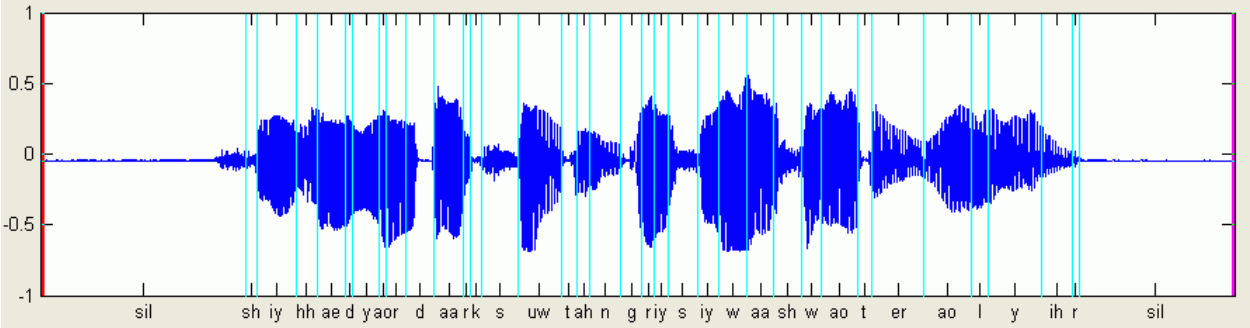
1. 對標準語句及測試語句抽取出語音的特徵參數（通常是 MFCC, Mel-frequency Cepstral Coefficients）。
2. 以 Viterbi Decoding 來進行 Forced Alignment，以便切出來每一個子音及母音。此部分需用到語者無關（Speaker-independent）的英文語音辨識核心。
3. 對每一個子音及母音進行評分因素的擷取，包含音量、音高、音長等，以及之前已經取得的 MFCC。
4. 對每一個評分因素進行個別評分，然後進行加權平均，以得到最後的評分結果。這些評分的因素有：
    a. 音色：語音的內容及發音的準確性，此部份的分數通常是經由計算 Acoustic Models 的機率值，並和類似音進行排名而得的分數，而不是直接和標準發音進行比較所得的分數。這是因為標準發音的範例可能有限（例如只有男生或女生），因此直接進行音色的相似度比對可能會造成使用者期望的誤差。
    b. 音調：經由每一個音節的音高曲線來和目標發音進行相似度比對。若是中文，就要加上聲調辨識，以便決定使用者的發音是否符合華語的聲調和變調規則。
    c. 韻律（或是音長）：經由每一個音節的發音長度，來和目標語句進行相似度比對。
    d. 強度：經由每一個音節的發音音量，來和目標語句進行相似度比對。

這邊有一個簡單的範例，我們先使用老外講的標準語句：She had your dark suit in greasy wash water all year，經由 Forced Alignment 處理後，可以得到下列結果：



由上圖可看出，經由 Forced Alignment 之後，電腦已經將每一個音標所在的區域自動標示出來，一旦這些標示是對的，以後的步驟就很簡單，我們就可以針對每一個音標來進行個別評分，然後再計算總分。因此這部分「切音」的結果可說是影響評分系統的最重要因素。（為了使切音的結果正確，我們的英文辨識引擎使用了兩個語料，一個是傳統的英文語料 TIMIT，另一個是台灣地區的英文語料，此部分的語料收集是由工研院負責統籌，參與錄音與整理語料的學校包含台灣大學、清華大學、交通大學、成功大學、師範大學）。

使用我講的好的測試語句，下圖是波形及經由 Forced Alignment 的結果，基本上的切音位置都是正確的：

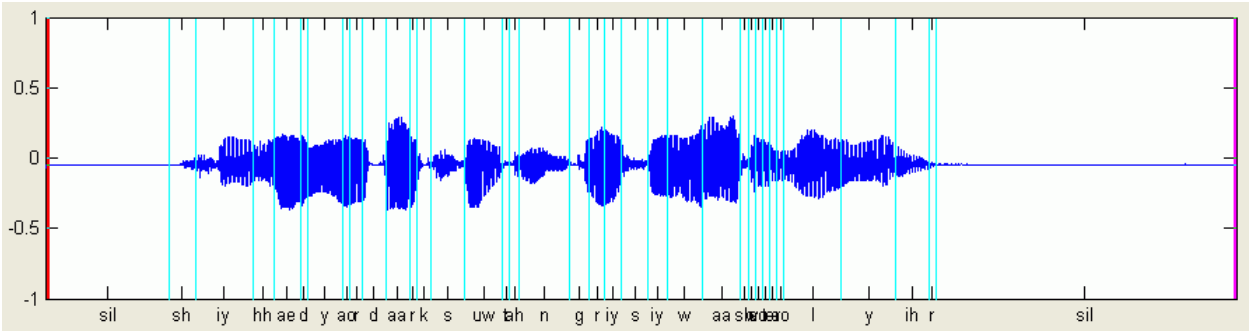


得到的評分結果是：

Pitch	(22.40%):	93.64
Magnitude	(7.45%):	79.68
Rhythm	(17.24%):	85.25
Pronunciation	(52.91%):	76.29

Score: 83.10

若使用我講的差的測試語句，下圖是波形及經由 Forced Alignment 的結果：



得到的評分結果是：

Pitch	(22.40%):	91.58
Magnitude	(7.45%):	85.48
Rhythm	(17.24%):	80.31
Pronunciation	(52.91%):	72.77

Score: 80.98

在這一句英文中，我故意漏掉「wash」這個英文字，因此會導致在進行切音的位置錯誤，由上圖可以看出，wash 的前半部被放在 water 語音的位置，而 water 則整個被壓縮了。由於唸法的不完整，造成切字的錯誤，因此整段話的分數就會比較低。

相關應用方面，可以列出如下：

- **語言學習軟體**：由於一般個人電腦的運算能力已經很強大，因此我們可以在一般個人電腦上進行完整的語音評分，可以用在各種語言學習軟體，以達到更客觀的評分與分析，以進行電腦輔助口說英語教學。

- **英文覆讀機、電子英文學習機、PDA**：由於語音評分需要的計算量相當大，因此比較難在於低階的嵌入式系統（例如 8 位元或是 16 位元的 MCU）。但如果將平台放寬到 32 位元的平台（例如 ARM），我們就可以進行比較完整的語音評分，可以隨時隨地進行電腦輔助口說英語教學。（在低階的平台上，還是可以進行音調評分。）

8-4 音腔評分

（趕工中，請稍候，謝謝！）

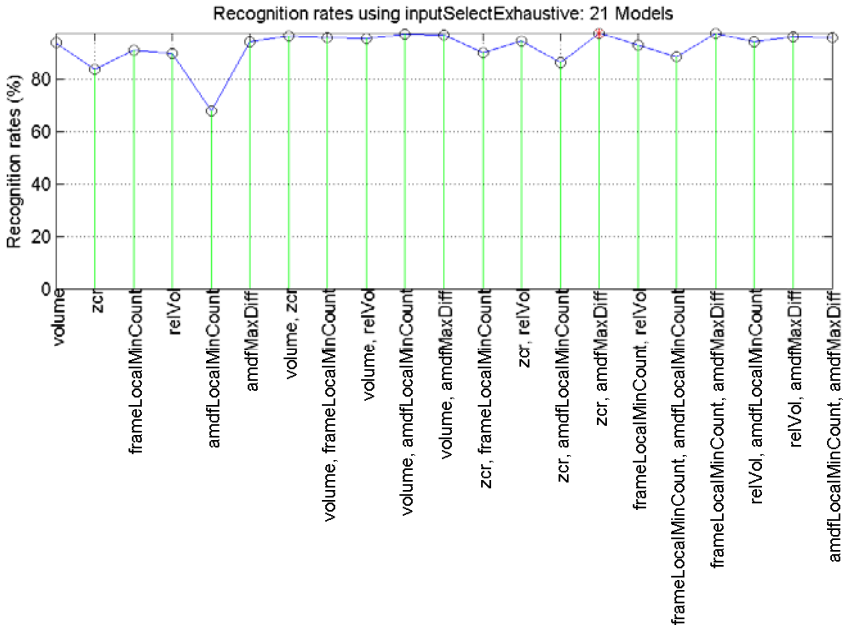
8-5 國語音調辨識

（待完成，請稍候，謝謝！）

第 8 章作業

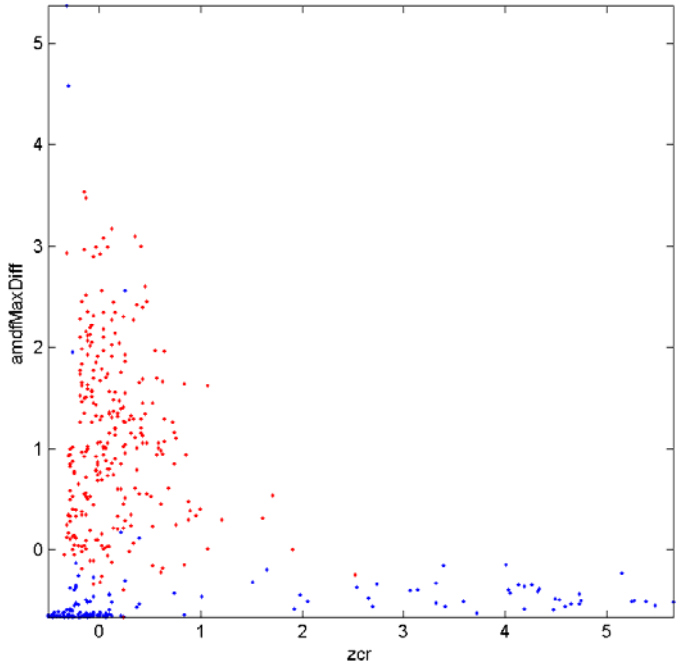
1. (**) 「一個音框是否有音高」的辨識： 請先由此比賽取得資料集 DS.mat（執行 goCollectData.m 所得到的輸出檔案），請用你自己的錄音（有標示音高的檔案，至少 5 個）即可。請寫一個 MATLAB 程式 pitchExistTest01.m，先載入資料 DS.mat，然後對資料集進行正規化後（將每個特徵的數值範圍進行線性調整，使其機率分佈接近於平均值為零、標準差為 1 的高斯機率密度分佈，可用 dataNormalize.m 來對資料正規化）後，來進行下列作業 . :

   a. 請使用 KNNR 分類器以及 Leave-one-out 的效能指標，來選取最好的兩個特徵（因為資料量不大，可用窮舉法）。請畫出 1 個圖，代表特徵依次被選取的過程。所得到的圖形應該類似下圖：



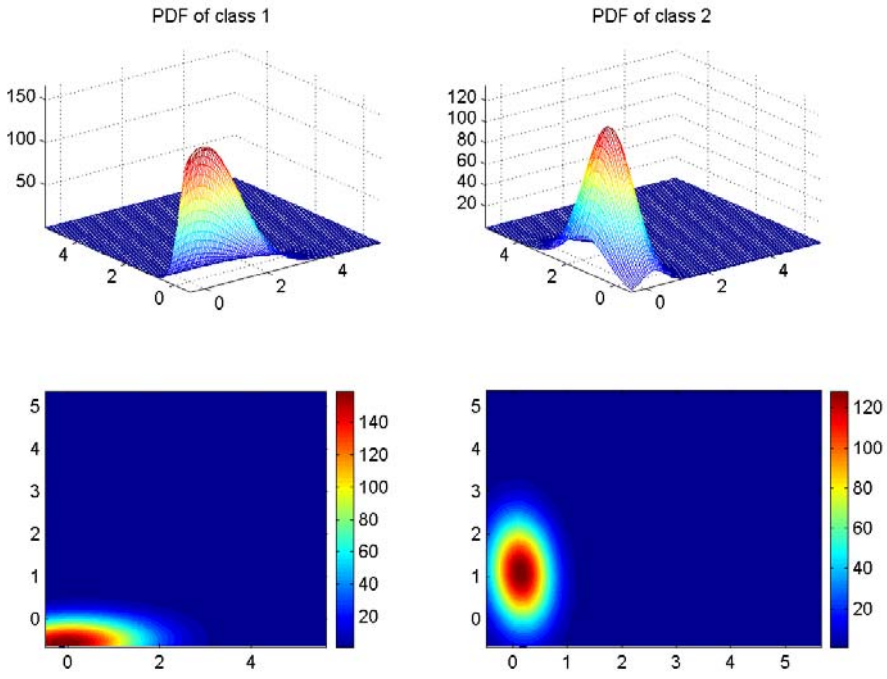
（提示：可用 inputSelectExhaustive.m 來進行輸入特徵的窮舉法選取。）

   b. 請顯示上一小題之二維資料的分佈概況。所得到的圖形應該類似下圖：



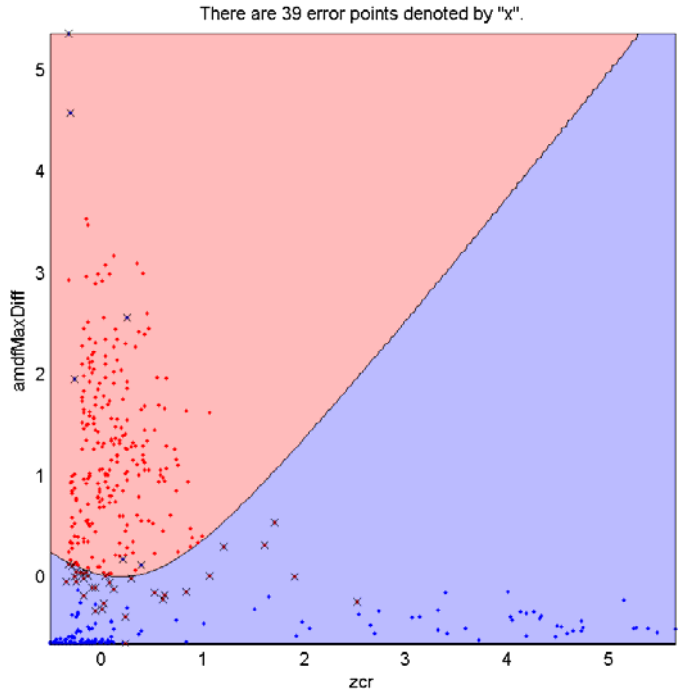
（提示：可用 dcprDataPlot 來畫出資料的分佈圖。）

c. 請針對上述二維的正規化資料，使用 SGC (Single-Gaussian Classifier)來對資料進行分類，請畫出每個類別的 Gaussian 曲面，所得到的圖形應該類似下圖：



（提示：可參考此範例。）

d. 請畫出出上述分類法所對應的 Decision Boundaries，以及分類正確及錯誤的資料點。所得到的圖形應該類似下圖：



（提示：可參考此範例。）

2. (***) **程式競賽：音高追蹤：** 請見此連結。
3. (***) **程式競賽：判斷一個音框是否具有音高：** 請見此連結。

Chapter 9: Digital Signals and Systems (數位訊號與系統)

幾乎在所有自然界的訊號都是連續的，但是在電腦世界的訊號，由於都是以位元為單位來儲存，所以這些訊號都是離散的，簡稱「離散時間訊號」（Discrete Time Signal），我們通常可以將這一列的訊號表示成 x(nT)，T 是取樣週期，n 是整數，nT 則代表時間。為了簡化起見，我們也可以使用 x[n] 代表離散時間訊號在 nT 的值。以下將介紹幾個常見的離散時間訊號。

單位脈衝訊號（Unit Impulse Signal）只有在 n = 0 的時候有一個值為 1，其他都是零。可用數學式表示如下：

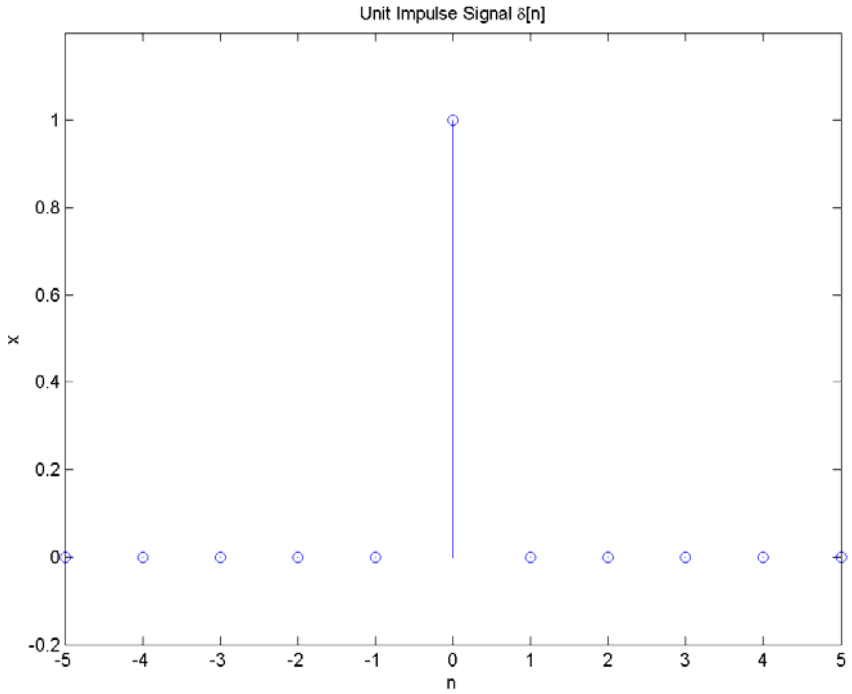
$$\delta[n] = 1, \text{ if } n=0$$
$$\delta[n] = 0, \text{ otherwise}$$

其圖形可見下列範例：

Example 1**Input file** digitalSignalsAndSystems/impulse01.m

```
% Plot an unit impulse signal

n = -5:5;
x = 0*n;
index=find(n==0);
x(index)=1;

% plot
stem(n, x);
axis([-inf, inf, -0.2, 1.2]);
xlabel('n'); ylabel('x');
title('Unit Impulse Signal \delta[n]');
```

Output figure



如果單位脈衝訊號在時間上延遲了 k 個取樣點，就可以得到下列數學式：

$$\delta[n-k] = 1, \text{ if } n=k$$

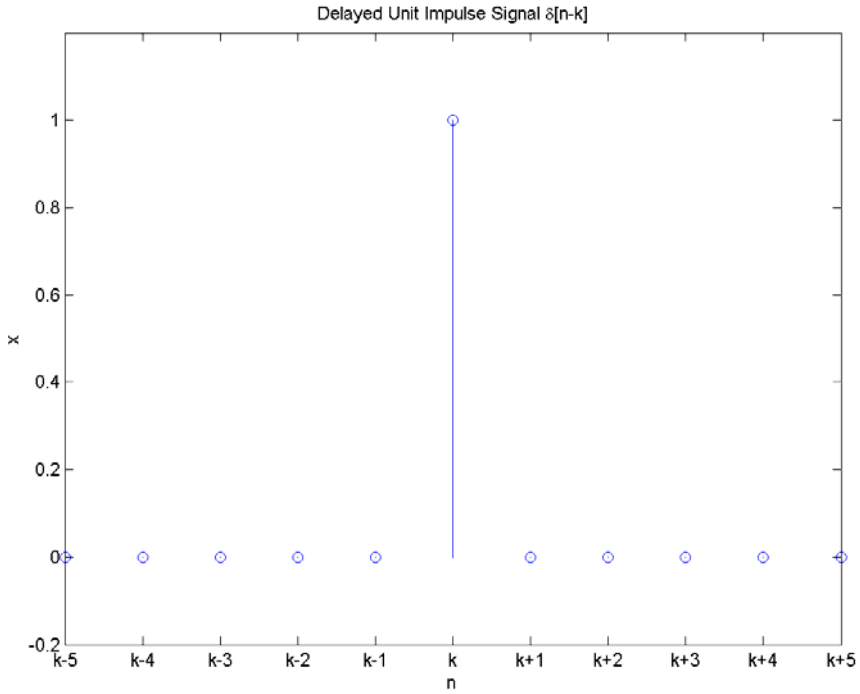
$$\delta[n-k] = 0, \text{ otherwise}$$

其圖形可見下列範例：

Example 2**Input file** digitalSignalsAndSystems/impulse02.m

```
% Plot an unit impulse signal

n = -5:5;
x = 0*n;
index=find(n==0);
x(index)=1;

% plot
stem(n, x);
axis([-inf, inf, -0.2, 1.2]);
xlabel('n'); ylabel('x');
title('Delayed Unit Impulse Signal \delta[n-k]');
set(gca, 'xticklabel', {'k-5', 'k-4', 'k-3', 'k-2', 'k-1', 'k', 'k+1', 'k+2', 'k+3', 'k+4', 'k+5'});
```

Output figure



單位脈衝訊號具有「篩選」的特性，任何訊號 x[n]（n = -∞～∞）與 δ[n-k] 相乘，就只剩下第 k 項，可用數學式描述如下：

$$x[k] = \Sigma_{n=-\infty}^{\infty} \delta[n-k] \cdot x[n]$$

利用這個特性，我們可以將任何一個離散時間訊號表示成單位脈衝訊號的組合，在計算旋積（Convolution）時，非常便利。

單位步進訊號（Unit Step Signal）只有在 n >= 0 時其值為 1，其他都是零。可用數學式表示如下：

$$u[n] = 1, \text{ if } n>=0$$

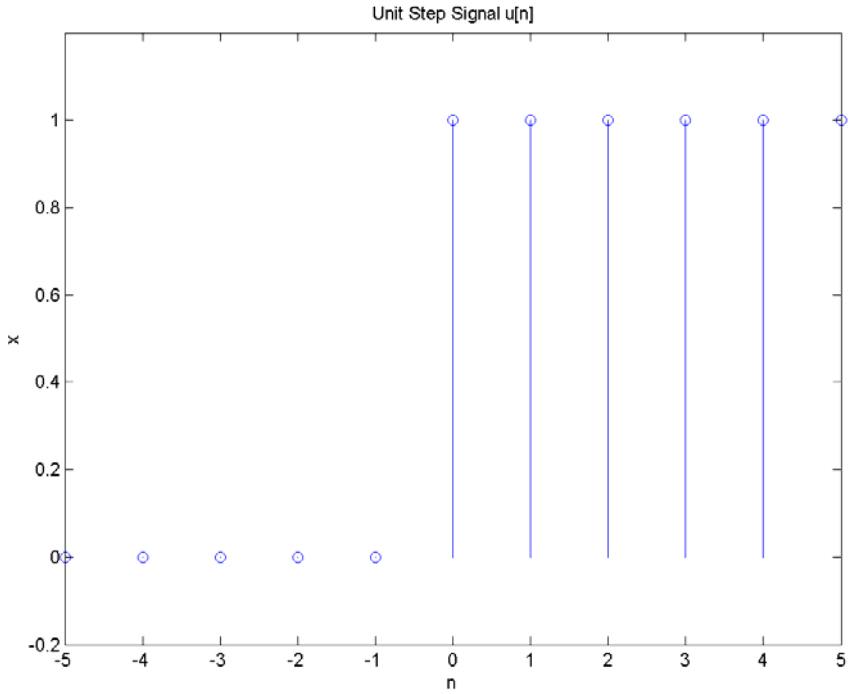
$$u[n] = 0, \text{ otherwise}$$

其圖形可見下列範例：

Example 3**Input file** digitalSignalsAndSystems/unitStep01.m

```
% Plot an unit impulse signal

n = -5:5;
x = 0*n;
index=find(n>=0);
x(index)=1;

% plot
stem(n, x);
axis([-inf, inf, -0.2, 1.2]);
xlabel('n');
ylabel('x');
title('Unit Step Signal u[n]');
```

Output figure

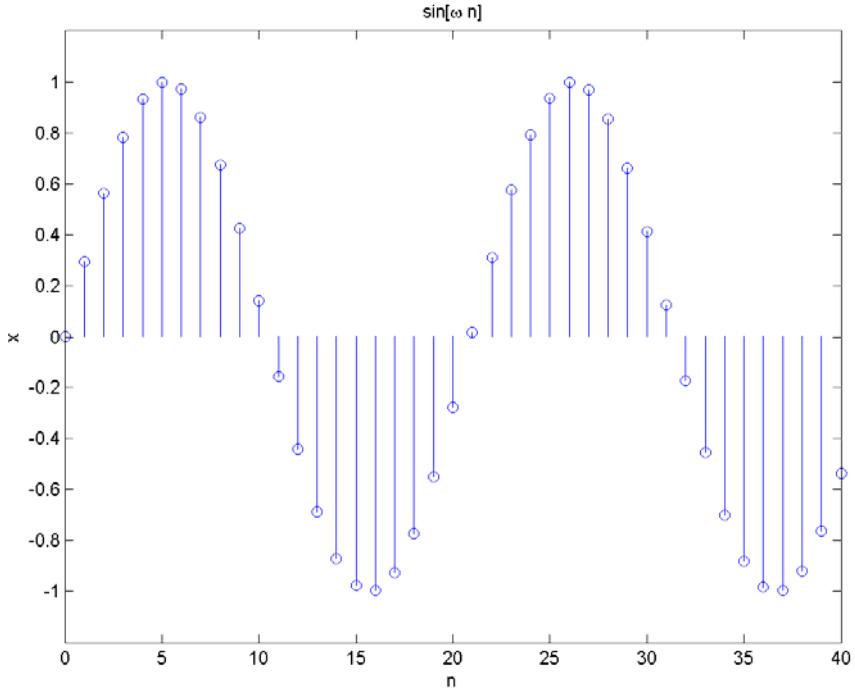


另外常見的離散時間訊號，還有弦波訊號等，其圖形可見下列範例：

Example 4**Input file** digitalSignalsAndSystems/sinusoid01.m

```
% Plot a sinusoidal signal

n = 0:40;
omega=0.3;
x = sin(omega*n);

% plot
stem(n, x);
axis([-inf, inf, -1.2, 1.2]);
xlabel('n');
ylabel('x');
title('sin[\omega n]');
```

Output figure



對於任一個離散時間訊號 x[n] 而言，我們可以使用一個電腦系統來處理此訊號，得到的輸出為 y[n]，這個關係可以用數學式表示如下：

$$y[n] = L\{x[n]\}$$

換句話說，此系統 $L\{\cdot\}$ 的輸入是一個函數 x[n]，n = 0～∞，而對應的輸出也是一個函數 y[n]，n = 0～∞。

Hint

為簡化討論，我們假設在 n 小於零時，x[n] = 0。換句話說，輸入訊號 x[n] 在 n ≧ 0 時才開始發生作用，因此 y[n] 也只有在 n ≧ 0 時，才有非零的值。

如果 $L\{\cdot\}$ 滿足下列兩個等式，此類系統稱為「線性系統」（Linear Systems）：

1. $y[n] = L\{x[n]\} \rightarrow ky[n] = L\{kx[n]\}$
2. $y_1[n] = L\{x_1[n]\}, y_2[n] = L\{x_2[n]\} \rightarrow y_1[n] + y_2[n] = L\{x_1[n] + x_2[n]\}$

上述兩等式，也可以寫成一個等式：

$$y_1[n] = L\{x_1[n]\}, y_2[n] = L\{x_2[n]\} \rightarrow ay_1[n] + by_2[n] = L\{ax_1[n] + bx_2[n]\}, \text{ for all a and b.}$$

上述等式稱為「疊加原則」（Superposition Principle），換句話說，只要滿足疊加原則的系統，就是線性系統。

如果 $L\{\cdot\}$ 滿足下列等式，此類系統稱為「非時變系統」（Time-invariant Systems）：

$$y[n] = L\{x[n]\} \rightarrow y[n-k] = L\{x[n-k]\}, \text{ for all k.}$$

如果一個系統是線性，而且也是非時變，我們稱其為「線性非時變系統」（Linear Time-invariant Systems），簡稱 LTI 系統。

在以下的討論中，我們均假設我們所遇到的系統都是 LTI 系統。

對於任意的訊號 x[n]，我們可以將其表示成為單位脈衝訊號（Unit Impulse Signal）的組合，如下：

$$x[n] = \Sigma_{k=0}^{\infty} x[k]\delta[n-k]$$

上述式子可以看成是「k 代表時間，x[k] 固定不動，δ[n-k] 隨不同的 n 而移動」。同理，上式也可以寫成：

$$x[n] = \Sigma_{k=0}^{\infty} x[n-k]\delta[k]$$

此時可以看成是「k 代表時間，δ[k] 固定不動，x[n-k] 隨不同的 n 而移動」。

因此對於一個 LTI 系統 $L\{\cdot\}$，當其輸入為 x[n] 時，其輸出為 y[n]，可用下列關係式來描述：

$$\begin{aligned} y[n] &= L\{x[n]\} \\ &= L\{\Sigma_{k=0}^{\infty} x[k]\delta[n-k]\} \end{aligned}$$

$$= \Sigma_{k=0}^{\infty} x[k] L\{\delta[n-k]\}$$
$$= \Sigma_{k=0}^{\infty} x[k] h[n-k]$$

其中 h(n-k)=L{δ(n-k)} 是系統對於位於 n=k 的單位脈衝訊號所產生的響應（Response）。換句話說，一個 LTI 系統的輸出，完全可以由輸入訊號 x[n] 和系統的脈衝響應（Impulse Response）h[n] 來決定。換句話說，一個 LTI 系統的脈衝響應就決定了這個系統的全部特性！

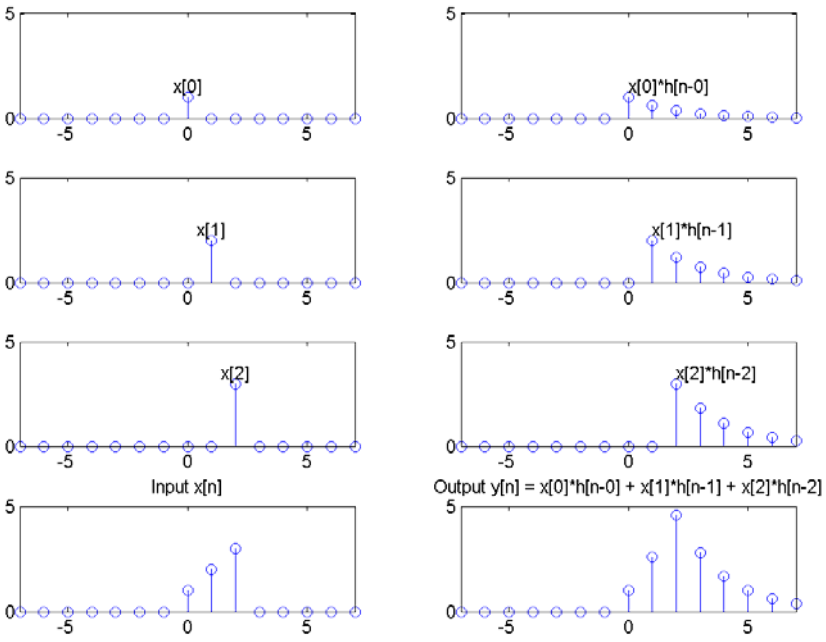
上述公式，可用下列示意圖來說明：

Example 1**Input file** digitalSignalsAndSystems/convolution01.m

```
% Plot the operation of convolution

n = -7:7;
x = [0 0 0 0 0 0 0 1 2 3 0 0 0 0 0];

subplot(4,2,7);
stem(n, x);
limit=[min(n), max(n), 0, 5];
axis(limit);
title('Input x[n]');

subplot(4,2,1);
x0=0*x;
x0(8)=x(8);
stem(n, x0);
axis(limit);
h=text(0, x0(8), 'x[0]'); set(h, 'horiz', 'center', 'vertical', 'bottom');

subplot(4,2,2);
y0=0*x;
index=find(x0);
for i=index:length(n)
        y0(i)=x0(index)*exp(-(i-index)/2);
end
stem(n, y0);
axis(limit);
h=text(0, x0(8), 'x[0]*h[n-0]'); set(h, 'vertical', 'bottom');

subplot(4,2,3);
x1=0*x;
x1(9)=x(9);
stem(n, x1);
axis(limit);
h=text(1, x1(9), 'x[1]'); set(h, 'horiz', 'center', 'vertical', 'bottom');

subplot(4,2,4);
y1=0*x;
index=find(x1);
for i=index:length(n)
        y1(i)=x1(index)*exp(-(i-index)/2);
end
```

```
stem(n, y1);
axis(limit);
h=text(1, x1(9), 'x[1]*h[n-1]'); set(h, 'vertical', 'bottom');

subplot(4,2,5);
x2=0*x;
x2(10)=x(10);
stem(n, x2);
axis(limit);
h=text(2, x2(10), 'x[2]'); set(h, 'horiz', 'center', 'vertical', 'bottom');

subplot(4,2,6);
y2=0*x;
index=find(x2);
for i=index:length(n)
        y2(i)=x2(index)*exp(-(i-index)/2);
end
stem(n, y2);
axis(limit);
h=text(2, x2(10), 'x[2]*h[n-2]'); set(h, 'vertical', 'bottom');

subplot(4,2,8);
stem(n, y0+y1+y2);
axis(limit);
title('Output y[n] = x[0]*h[n-0] + x[1]*h[n-1] + x[2]*h[n-2]');
```

Output figure



如果採用本節的第二個方程式，則 y[n] 可以表示成

$$
\begin{aligned}
y[n] &= L\{x[n]\} \\
&= L\{\Sigma_{k=0}^{\infty} x[n-k]\delta[k]\} \\
&= \Sigma_{k=0}^{\infty} x[n-k]L\{\delta[k]\} \\
&= \Sigma_{k=0}^{\infty} x[n-k]h[k]\}
\end{aligned}
$$

由於上述的運算常被用到，因此我們將之定義成「旋積」（Convolution）：

$$y[n] = x[n]*h[n] = \Sigma_{k=0}^{\infty} x[k]h[n-k]\} = \Sigma_{k=0}^{\infty} x[n-k]h[k]\}$$

旋積運算具有下列特性：

1. 交換律：

$$x[n]*y[n] = y[n]*x[n]$$

2. 結合律：

$$(x[n]*y[n])*z[n] = x[n]*(y[n]*z[n])$$

3. 分配律：

$$x[n]*(y[n]+z[n]) = x[n]*y[n]+x[n]*z[n]$$

4. 延遲性質：

$$y[n]=x[n]*h[n] \rightarrow y[n-n_1-n_2]=x[n-n_1]*h[n-n_2]$$

如果我們將某一類特定的輸入函數送進某個系統後，如果所得到的輸出函數仍然具有輸入函數的格式，則此函數稱為此系統的「固有函數」（Eigen Function）。本節將說明，指數函數（含弦波函數）就是 LTI 系統的固有函數。

如果我們的輸入函數是一個指數函數：

$$x[n]=e^{pn}$$

那麼根據之前推導的公式，可得到對應的輸出是：

$$
\begin{aligned}
y[n] &= x[n]*h[n] \\
&= \Sigma_k x[n-k]\ h[k] \\
&= \Sigma_k e^{p(n-k)}\ h[k] \\
&= \Sigma_k e^{pn-pk}\ h[k] \\
&= e^{pn}\Sigma_k e^{-pk}\ h[k] \\
&= e^{pn}H(e^p)
\end{aligned}
$$

換句話說，輸出函數 y[n] 等於原輸入函數乘上一個常數 $H(e^p)$（$= \Sigma_k e^{-pk}\ h[k]$，是一個不隨時間而變化的函數），因此我們說「指數函數是 LTI 系統的固有函數」。

從這裡推廣，任何輸入函數，只要能夠表示成指數函數的線性組合，就可以使用疊加原理，先算出這些指數函數的對應輸出，再進行線性組合，就可以得到原輸入函數的對應輸出，可用數學式表示如下：

$$x[n] = ae^{pn} + be^{qn} \rightarrow y[n] = aH(e^p) + bH(e^q)$$

對於弦波函數，我們可以使用 Euler Identity 來代換成指數函數：

$$e^{j\theta} = \cos(\theta) + j\sin(\theta)$$

換句話說：

$$
\begin{aligned}
\cos(\theta) &= (e^{j\theta} + e^{-j\theta})/2 = Re\{e^{j\theta}\} \\
\sin(\theta) &= (e^{j\theta} - e^{-j\theta})/(2j) = Im\{e^{j\theta}\}
\end{aligned}
$$

因此，當輸入是

$$x[n] = \cos(\omega n) = Re\{e^{j\omega n}\}$$

對應的輸出則是

$$
\begin{aligned}
y[n] &= Re\{e^{j\omega n}H(e^{j\omega})\} \\
&= Re\{e^{j\omega n}\ |H(e^{j\omega})|\ e^{j\theta}\},\ \theta = \angle H(e^{j\omega}) \\
&= Re\{|H(e^{j\omega})|\ e^{j(\omega n+\theta)}\}
\end{aligned}
$$

$$= |H(e^{j\omega})| \cos(\omega n + \theta)$$

換句話說，輸出函數還是 cos，只不過相位偏移了 $\theta = \angle H(e^{j\omega})$，震幅也乘上了 $|H(e^{j\omega})|$。幾個相關名詞，說明如下。

- 如果我們不看相位，那麼此輸出的乘數 $|H(e^{j\omega})|$ 代表此系統對不同角頻率 ω 的訊號的擴大或壓縮特性，因此我們通常稱 $|H(e^{j\omega})|$ 為「響度頻率響應」（Magnitude Frequency Response）。
- $\angle H(e^{j\omega})$ 則稱為「相位頻率響應」（Phase Frequency Response）。（一般而言，較少被用到，因為人耳對於相位的感覺，並不敏銳。）
- $H(e^{j\omega})$ 統稱是此系統的「頻率響應」（Frequency Response）。

同理，當輸入是

$$x[n] = \sin(\omega n) = \mathrm{Im}\{e^{j\omega n}\}$$

則輸出是

$$\begin{aligned} y[n] &= \mathrm{Im}\{e^{j\omega n} H(e^{j\omega})\} \\ &= \mathrm{Im}\{e^{j\omega n} |H(e^{j\omega})| e^{j\theta}\}, \theta = \angle H(e^{j\omega}) \\ &= \mathrm{Im}\{|H(e^{j\omega})| e^{j(\omega n+\theta)}\} \\ &= |H(e^{j\omega})| \sin(\omega n + \theta) \end{aligned}$$

因此，我們可以得到一個結論：對於一個 LTI 系統而言，當輸入函數是一個單一弦波函數時，輸出也一定是一個頻率相同的弦波函數，只有震幅和相位會被系統所改變，而改變的方式完全依照系統的頻率響應 $H(e^{j\omega})$ 而定。

上述的兩種弦波輸入，我們可以改用一個虛數的指數函數來概括：

$$x[n] = e^{j\omega n}$$

那麼對應的輸出是

$$y[n] = e^{j\omega n} H(e^{j\omega})$$

其中 $H(e^{j\omega})$ 稱為 h[n] 的「離散時間傅立葉轉換」（Discrete-time Fourier Transform），可以表示如下：

$$H(e^{j\omega}) = \Sigma_k h[k] e^{-j\omega k}$$

事實上，上述的輸入和輸出在現實世界並不存在，因為一個系統的輸入和輸出，不可能是複數。但是這些數學運算在理論上是成立的，因此我們可以使用這些式子來讓我們的推導更簡潔一些。

Chapter 10: Fourier Transform (傅立葉轉換)

Old Chinese version

我們在前一章中，已經說明了一個 LTI 系統的頻率響應，也就是其單位脈衝響應的離散時間傅立葉轉換，可用來表示對於特定頻率弦波所產生的震幅和相角改變。此外，對於任意的離散時間訊號 x[n]，其「離散時間傅立葉轉換」（簡稱 DTFT）及其反轉換可以表示如下：

$$X(e^{j\omega}) = \Sigma_k e^{-j\omega k} x[k]$$

$$x[n] = (2\pi)^{-1} \int_{2\pi} X(e^{j\omega}) e^{j\omega n} \, d\omega$$

由上述的第一個式子即可推導出第二個式子。在第二個式子中，由於 $X(e^{j\omega})$ 的週期性是 2π，因此積分區間可以是任何長度為 2π 的區間。

由第二個式子，可以我們可以將 x[n] 拆解成無窮個基本函數 $e^{j\omega n}$ 的線性組合，而每一個基本函數的震幅則是由 $X(e^{j\omega})$ 來控制。換句話說，$X(e^{j\omega})$ 即代表 x[n] 在連續頻率 ω 的分量，可以進一步說明如下。

由基本微積分可知，對於一個函數 f(x) 的積分，我們可以使用累加來逼近：

$$\int_0^{2\pi} f(x)\, dx = \lim_{N\to\infty} \Sigma_{k=0}^{N-1} f(k\cdot(2\pi/N))\cdot(2\pi/N)$$

因此如果 x[n] 是實數，我們就可以將其拆解開來，表示如下：

$$\begin{aligned} x[n] &= \mathrm{Re}\{(2\pi)^{-1} \int_{2\pi} X(e^{j\omega}) e^{j\omega n} \, d\omega\} \\ &= (2\pi)^{-1} \int_{2\pi} \mathrm{Re}\{X(e^{j\omega}) e^{j\omega n}\} \, d\omega \end{aligned}$$

$$= (2\pi)^{-1}\int_{2\pi} |X(e^{j\omega})| \cos(\omega n + \theta)\, d\omega,\ \theta = \angle X(e^{j\omega})$$
$$= N^{-1} \Sigma_{k=0}^{N-1} [|X(e^{j\omega})| \cos(\omega n + \theta)]_{\omega=2\pi k/N}\, d\omega,\ N \to \infty$$

換句話說，x[n] 已經被拆解成 N 個餘弦函數的線性組合，而這些餘弦函數的角頻率是從 0 到 $2\pi(N-1)/N$，震幅則是 $|X(e^{j\omega})|/N$ 在每個角頻率的值。（如果 x[n] 是複數，我們也可以進行此種拆解，只不過物理意義較不明顯。）

由上一節及本節的討論，我們就可以瞭解，DTFT 具有兩個非常重要的意義：

1. 如果 h[n] 是一個 LTI 系統的脈衝響應（Impulse Response），那麼 $H(e^{j\omega})=\Sigma_k h[k]e^{-j\omega k}$ 就可以代表此系統對於角頻率等於 ω 的訊號所造成的增益 $|H(e^{j\omega})|$ 和相位 $\angle H(e^{j\omega})$。
2. 如果訊號 x[n] 是一個任意訊號，那麼 $X(e^{j\omega})=\Sigma_k x[k]e^{-j\omega k}$ 就可以代表此訊號在不同角頻率 ω 的分量大小 $|H(e^{j\omega})|$ 和相位 $\angle H(e^{j\omega})$。

以下列出離散時間傅立葉轉換的一些常用到的重要性質。

1. 線性性質：
$$z[n] = a\, x[n] + b\, y[n] \longleftrightarrow Z(e^{j\omega}) = a\, X(e^{j\omega}) + b\, Y(e^{j\omega})$$
2. 週期性：
$$X(e^{j(\omega + 2k\pi)}) = X(e^{j\omega})$$
換句話說，DTFT 的週期為 2π。
3. 延遲性質：
$$y[n] = x[n-k] \longleftrightarrow Y(e^{j\omega}) = e^{j\omega k} X(e^{j\omega})$$
4. 旋積：
$$y[n] = x[n]*h[n] \longleftrightarrow Z(e^{j\omega}) = X(e^{j\omega})Y(e^{j\omega})$$
也就是說，在時域的旋積等於在頻域的乘積。

如果 x[n] 是一個實數序列，那麼我們可以將對應的 DTFT $X(e^{j\omega})$ 拆解成實部和虛部的組合，如下：
$$X(e^{j\omega}) = \Sigma_k e^{-j\omega k} x[k]$$
$$= \Sigma_k x[k] \cos(\omega k) - j\, \Sigma_k x[k] \sin(\omega k)$$
$$= X_R(e^{j\omega}) + j\, X_R(e^{j\omega})$$

其中
$$X_R(e^{j\omega}) = \Sigma_k x[k] \cos(\omega k)$$
$$X_I(e^{j\omega}) = -\Sigma_k x[k] \sin(\omega k)$$

這些函數滿足下列性質：

- 共軛對稱：$X^*(e^{j\omega}) = X(e^{-j\omega})$
- $X_R(e^{j\omega})$ 是偶函數
- $X_I(e^{j\omega})$ 是奇函數
- $|X(e^{j\omega})|$ 是偶函數
- $\angle X(e^{j\omega})$ 是奇函數

在前一節中，我們可以使用「離散時間傅立葉轉換」（簡稱 DTFT）來將一段數位訊號轉換成各個頻譜的分量，但這是一個連續的函數，並不適合在電腦中處理，因此本節將介紹「離散傅立葉轉換」（Discrete Fourier Transform），簡稱 DFT，其功能是將一段數位訊號轉換成其各個離散頻率的弦波分量，以便後續使用電腦進行各種處理。

如果我們的訊號可以表示成 x[n], n = 0~N-1，那麼 DFT 的公式如下：
$$X[k]=(1/N)*\Sigma_{n=0}^{N-1} x[n]*\exp(-j*2\pi*n*k/N),\ k=0, ..., N-1$$

這些傅立葉係數 X[k] 所代表的資訊是 k 的函數，而 k 直接和頻率有正比關係，因此這些係數 X[k] 通稱為「頻譜」（Spectrum），而對於 X[k] 的分析，我們通稱為「頻譜分析」（Spectral Analysis）。

我們也可以由這些傅立葉係數 X[k]，來反推原始訊號 x[n]，如下：
$$x[n]=\Sigma_{k=0}^{N-1} X[k]*\exp(j*2\pi*n*k/N),\ n=0, ..., N-1$$

Hint

DTFT 和 DFT 很類似，兩者都用來處理離散時間的訊號，只是前者產生一個角頻率的連續函數，而後者產生離散頻率的訊號，更適合使用電腦來進行處理。

這邊有幾點要說明：

- 如果原始訊號 x[n] 有 N 點，那麼轉換出來的訊號 X[k] 也會有 N 點。
- 一般而言，X[k] 是一個複數，其大小是 |(X[k])| （abs(X[k]) in MATLAB），相位是 ∠X[k] (angle(X[k]) or atan(imag(X[k])/real(X[k]) in MATLAB)。
- 如果原始訊號 x[n] 都是實數，那麼 X[k] 和 X[N-k] 會是共軛複數，滿足 |(X[k])| = |(X[N-k])| 以及 ∠X[k] = -∠X[N-k]。

如果 x[n] 是實數，我們可以將 x[n] 表示如下：

$$\begin{aligned} x[n] &= X[0] \\ &+ X[1]*\exp(j*2\pi*n*1/N) + X[N-1]*\exp(j*2\pi*n*(N-1)/N) \\ &+ X[2]*\exp(j*2\pi*n*2/N) + X[N-2]*\exp(j*2\pi*n*(N-2)/N) \\ &+ X[3]*\exp(j*2\pi*n*3/N) + X[N-3]*\exp(j*2\pi*n*(N-3)/N) \\ &+ \ldots \end{aligned}$$

對上述第 k 項而言，我們有

$$\begin{aligned} & X[k]*\exp(j*2\pi*n*k/N) + X[N-k]*\exp(j*2\pi*n*(N-k)/N) \\ &= X[k]*\exp(j*2\pi*n*k/N) + X[N-k]*\exp(j*2\pi*n)*\exp(-j*2\pi*n*k/N) \\ &= X[k]*\exp(j*2\pi*n*k/N) + X[N-k]*\exp(-j*2\pi*n*k/N) \\ &= 2*\mathrm{Re}(X[k]*\exp(j*2\pi*n*k/N)) \end{aligned}$$

如果我們以 m_k 表示 X[k] 的大小，以 p_k 表示 X[k] 的相位，那麼上式可以化簡如下：

$$\begin{aligned} & 2*\mathrm{Re}(m_k*\exp(j*p_k)*\exp(j*2\pi*n*k/N)) \\ &= 2*\mathrm{Re}(m_k*\exp(j*(2\pi*n*k/N + p_k)) \\ &= 2*m_k*\cos(2\pi*n*k/N + p_k) \end{aligned}$$

一般而言，N 是 2 的倍數，因此 x[n] 可以表示成

$$x[n] = X[0] + 2*\Sigma_{k=1}^{N/2-1} m_k*\cos(2\pi*n*k/N + p_k) + m_{N/2}*\cos(\pi*n + p_{N/2})$$

換句話說，我們可以將原始訊號拆解成一個直流訊號 X[0] 再加上 N/2 個弦波的組合，這些弦波的震幅就是 X[k] 的大小，而相位則是 X[k] 的相位。因此對於實數的 x[n] 而言，我們只需要看單邊的 X[k]，k = 0 ~ N/2，此種可稱為「單邊頻譜」。組成這些單邊頻譜的弦波共有 1+N/2 個，頻率由小到大分別是 fs/N*(0:N/2)，這些弦波稱為「基本弦波」。

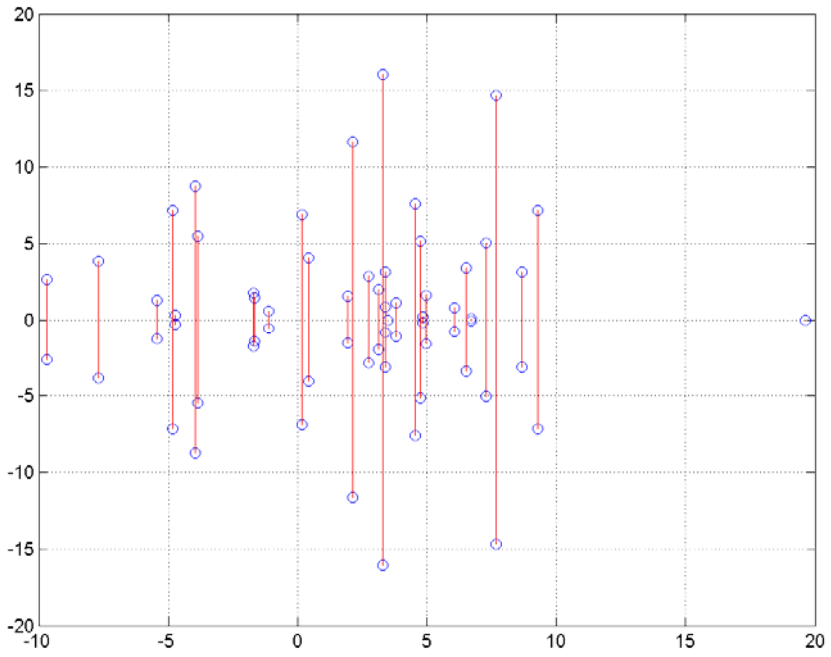
若是套用上述公式來計算 DFT，所需要的複雜度是 $O(n^2)$，但在 1965 年，有兩位學者提出來一套更精簡的演算法，所需的複雜度只有 O(n log n)，這一套演算法稱為「快速傅立葉轉換」（Fast Fourier Transform，簡稱 FFT），換句話說，FFT 是用來計算 DFT 的快速方法。若使用 MATLAB，相關的指令也是 fft。

首先，我們使用 MATLAB 的 fft 指令來驗證實數訊號之 DFT 係數的共軛性，如下：

Example 1**Input file** ft/fftSym01.m

```
% This example demonstrates the pair-wise conjugate of DFT (此範例展示實數訊號之 DFT 係數的共軛性)

N=64;                       % Length of vector
x=randn(N, 1);
z=fft(x);
plot(z, 'o'); grid on
%compass(z);
% Connect conjugate pairs (將上下對稱的共軛點連起來)
for i=2:N/2+1
        twoPoint=z([i, N-i+2]);
        line(real(twoPoint), imag(twoPoint), 'color', 'r');
end
```

Output figure



由上圖可以看出，DFT 係數會對稱於複數平面的實數軸，代表這些係數都會以共軛複數的方式成對出現。

Hint

對於實數訊號 x[n] 而言：

- 當 N 為偶數時，只有 X[0] 和 X[N/2] 是單一實數，其餘均為成對出現之共軛複數。
- 當 N 為奇數時，只有 X[0] 是單一實數，其餘均為成對出現之共軛複數。

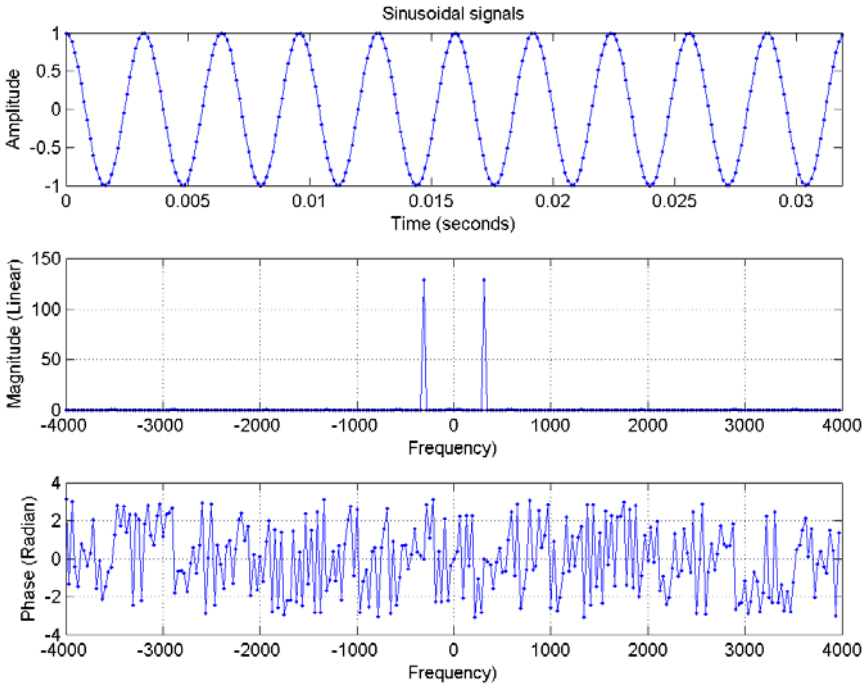
如果 x[n] 恰巧是這些基本弦波中的其中一個，那麼計算出來的雙邊頻譜，應該只有兩個係數不為零，我們可用 MATLAB 驗證如下：

Example 2**Input file** ft/fft01.m

```
% This example demonstrates the two-side DFT of a sinusoidal function (此範例展示一個簡單正弦波的傅立葉轉換，以雙邊頻譜來顯示)
% Since the sinusoidal function has a frequency to be a multiple of fs/N, the two-side DFT have only two nonzero terms. (此正弦波的頻率恰巧是 freqStep 的整數倍，所以雙邊頻譜應該只有兩個非零點)

N = 256;                    % length of vector (點數)
fs = 8000;                  % sample rate (取樣頻率)
freqStep = fs/N;            % freq resolution in spectrum (頻域的頻率的解析度)
f = 10*freqStep;            % freq of the sinusoid (正弦波的頻率，恰是 freqStep 的整數倍)
time = (0:N-1)/fs;          % time resolution in time-domain (時域的時間刻度)
y = cos(2*pi*f*time);       % signal to analyze
Y = fft(y);                 % spectrum
Y = fftshift(Y);            % put zero freq at the center (將頻率軸的零點置中)

% Plot time data
subplot(3,1,1);
```

```
plot(time, y, '.-');
title('Sinusoidal signals');
xlabel('Time (seconds)'); ylabel('Amplitude');
axis tight

% Plot spectral magnitude
freq = freqStep*(-N/2:N/2-1);          % freq resolution in spectrum (频域的频率的解析度)
subplot(3,1,2);
plot(freq, abs(Y), '.-b'); grid on
xlabel('Frequency)');
ylabel('Magnitude (Linear)');

% Plot phase
subplot(3,1,3);
plot(freq, angle(Y), '.-b'); grid on
xlabel('Frequency)');
ylabel('Phase (Radian)');
```

Output figure



由上圖可以看出：

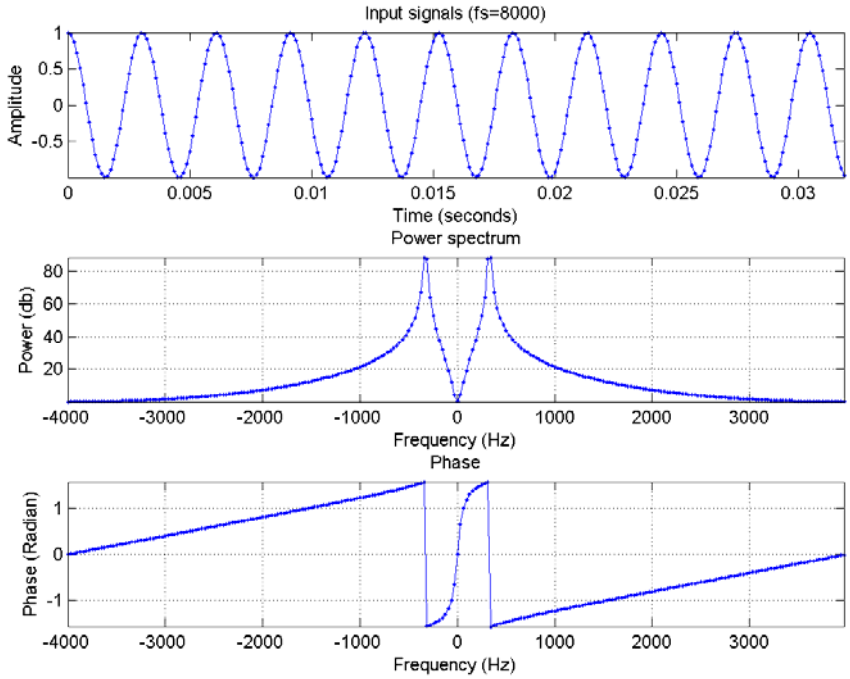
- 能量頻譜只有在兩點不為零，其他各點理論上應該是零，但由於計算精度之故，實際值很接近於零，但不一定是零。
- 相位頻譜沒有一定規則，像是亂數。這是由於大部分的能量頻譜很小，因此實際的相位頻譜並沒有任何意義。

如果 x[n] 的頻率不是這些弦波中的其中一個，那麼 DFT 還是會將此訊號拆解成基本弦波的組合，因此能量頻譜就會分散在各個基本弦波，如下所示：

Example 3**Input file** ft/fft02.m

```
% This example demonstrates the one-side DFT of a sinusoidal function (此範例展示一個簡單正弦波的傅立葉轉換，以雙邊頻譜來顯示)
% Since the sinusoidal function has a frequency not a multiple of fs/N, the two-side DFT smears. (此正弦波的頻率不是 freqStep 的整數倍，所以雙邊頻譜會「散開」(Smearing))

N = 256;                                % length of vector (點數)
fs = 8000;                              % sample rate (取樣頻率)
freqStep = fs/N;                        % freq resolution in spectrum (頻域的頻率的解析度)
f = 10.5*freqStep;                      % freq of the sinusoid (正弦波的頻率，不是 freqStep 的整數倍)
time = (0:N-1)/fs;                      % time resolution in time-domain (時域的時間刻度)
signal = cos(2*pi*f*time);              % signal to analyze
[mag, phase, freq]=fftTwoSide(signal, fs, 1);   % compute and plot the two-side DFT
```

Output figure



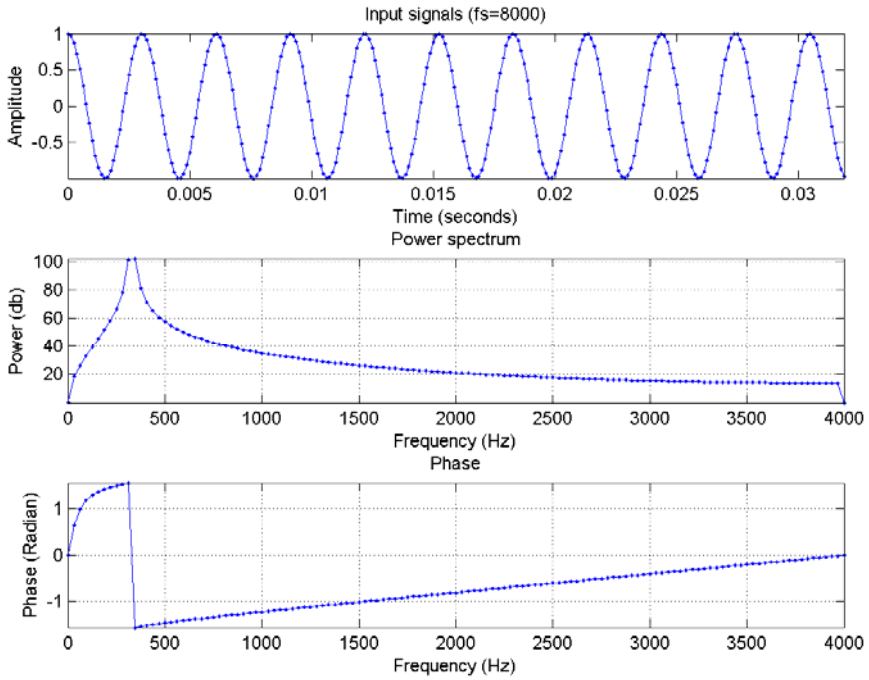
在上述範例中，我們使用 fftTwoSide.m 函數（位於 Audio Toolbox）來進行雙邊頻譜的計算，並將同時畫出時域訊號、頻譜能量、頻譜相位三個圖。
由於對於實數的 x[n] 而言，雙邊頻譜能量圖會左右對稱，而雙邊頻譜相位圖則是左右反對稱，因此我們可以只看單邊頻譜即可，範例如下：

Example 4**Input file** ft/fft03.m

```
% Same as fft02.m but use one-side DFT instead (同 fft02.m，但以單邊頻譜來顯示)
N = 256;                                % length of vector (點數)
fs = 8000;                              % sample rate (取樣頻率)
freqStep = fs/N;                        % freq resolution in spectrum (頻域的頻率的解析度)
f = 10.5*freqStep;                      % freq of the sinusoid (正弦波的頻率，不是 freqStep 的整數倍)
time = (0:N-1)/fs;                      % time resolution in time-domain (時域的時間刻度)
signal = cos(2*pi*f*time);              % signal to analyze
```

```
[mag, phase, freq]=fftOneSide(signal, fs, 1);     % Compute and plot one-side DFT
```

Output figure



在上述範例中，我們使用 fftOneSide.m 函數（位於 Audio Toolbox）來進行單邊頻譜的計算，並將同時畫出時域訊號、頻譜能量、頻譜相位三個圖。

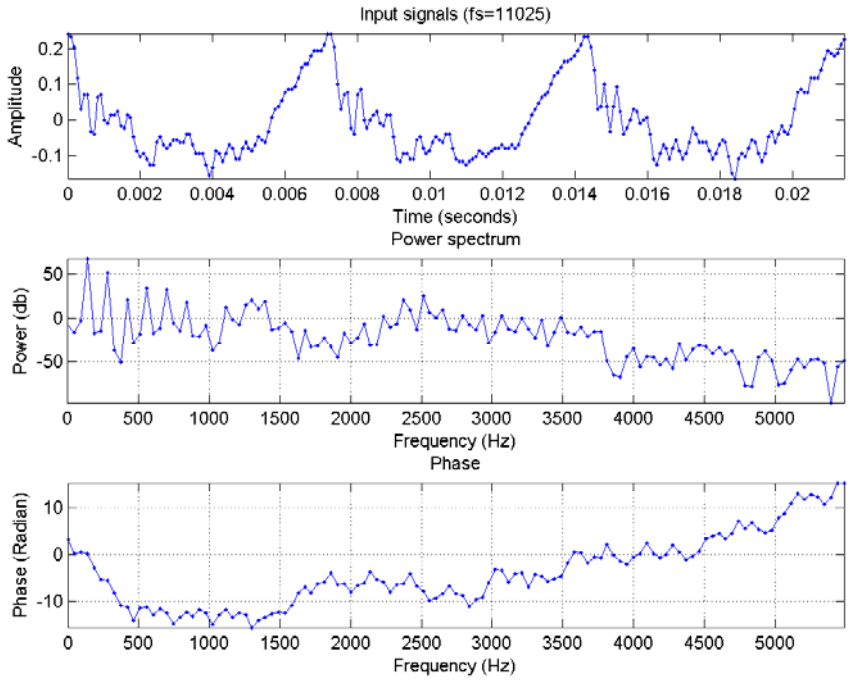
Hint

一般實際世界中的訊號，都是實數訊號，因此我們只要使用 fftOneSide.m 來計算單邊頻譜即可。下面這個範例，顯示一個語音音框的單邊頻譜：

Example 5**Input file** ft/fft04.m

```
% This example demonstrates the DFT of a real-world audio signal (顯示一個語音音框的單邊頻譜)
[y, fs]=wavread('welcome.wav');
signal=y(2047:2047+237-1);
[mag, phase, freq]=fftOneSide(signal, fs, 1);
```

Output figure



在上述範例中，我們可以看到很明顯的諧波出現在頻譜能量，這是由於訊號的週期性所造成的結果。（為了凸顯諧波現象，我們的音框剛好包含了三個完整的基本週期。）

當 k 越大時，代表對應的弦波是高頻訊號，因此我們可以捨棄高頻訊號，只用幾個低頻弦波，來合成原始訊號，達到下列兩項目的：

- 資料壓縮
- 濾除高頻訊號

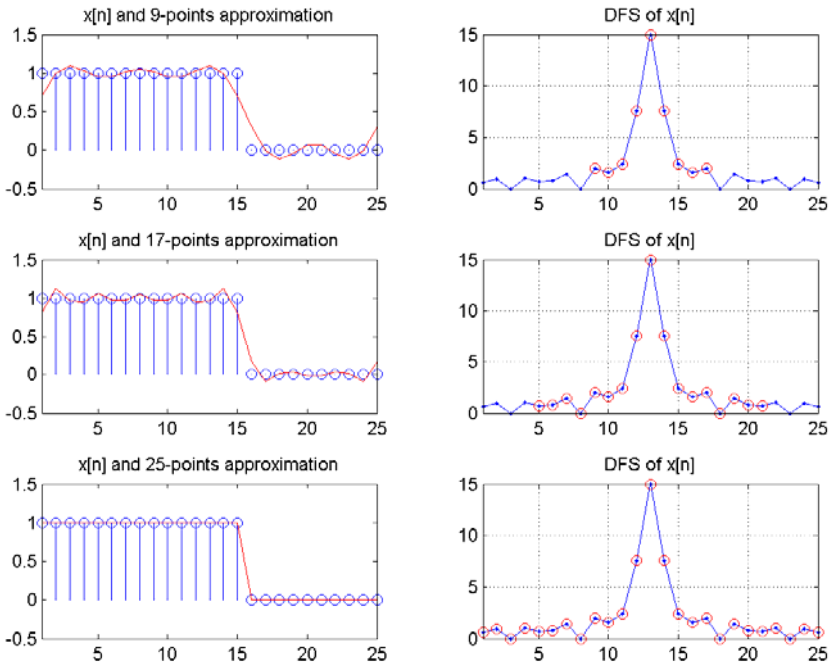
以下這個範例，顯示如何以弦波來合成一個方波：

Example 6**Input file** ft/fftApproximate01.m

```
% This example demos the effect of square wave approximation by DFT

figure
L = 15; N = 25;
x = [ones(1,L), zeros(1,N-L)];
frameSize=length(x);

runNum=3;
for i=1:runNum,
        pointNum=ceil(frameSize/(2*runNum)*i);      % Actually 2*pointNum-1 coefs are taken
        X = fft(x);
        magX = abs(X);

        remainIndex=[1:pointNum, frameSize-pointNum+2:frameSize];
        X2=0*X;
        X2(remainIndex)=X(remainIndex);
        x2=ifft(X2);
        x2=real(x2);

        subplot(3,2,2*i-1);
```

```
        stem(x);
        hold on
        plot(x2, 'r');
        hold off
        title(sprintf('x[n] and %d-points approximation', 2*pointNum-1));
        axis([-inf,inf,-0.5,1.5])

        subplot(3,2,2*i);
        shiftedMagX=fftshift(magX);
        plot(shiftedMagX, '.-'); axis tight
        title('DFT of x[n]')
        hold on
        temp=ifftshift(1:frameSize);
        ind=temp(remainIndex);
        plot(ind, shiftedMagX(ind), 'or'); grid on
        hold off
end
```

Output figure



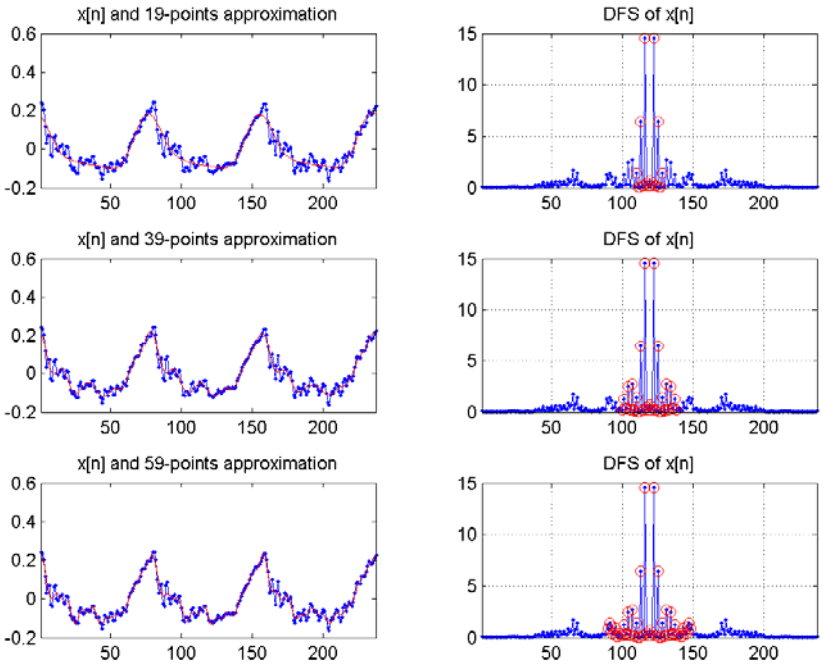
由此可見當我們所用的 DFT 係數越多，所合成的波形就會越接近原來的波形。
下面這個範例，則是以弦波來合成一個實際說話聲音的波形：

Example 7**Input file** ft/fftApproximate02.m

```
% This example demos the effect of FFT approximation

[y, fs]=wavread('welcome.wav');
x=y(2047:2047+237-1);

figure
```

```
frameSize=length(x);

runNum=3;
for i=1:runNum,
	pointNum=ceil(frameSize/(8*runNum)*i);	% Actually 2*pointNum-1 coefs are taken
	X = fft(x);
	magX = abs(X);

	remainIndex=[1:pointNum, frameSize-pointNum+2:frameSize];
	X2=0*X;
	X2(remainIndex)=X(remainIndex);
	x2=ifft(X2);
	x2=real(x2);

	subplot(3,2,2*i-1);
	plot(x, '.-');
	hold on
	plot(x2, 'r');
	hold off
	title(sprintf('x[n] and %d-points approximation', 2*pointNum-1));
	set(gca, 'xlim', [-inf inf]);

	subplot(3,2,2*i);
	shiftedMagX=fftshift(magX);
	plot(shiftedMagX, '.-');
	title('DFT of x[n]')
	hold on
	temp=ifftshift(1:frameSize);
	ind=temp(remainIndex);
	plot(ind, shiftedMagX(ind), 'or'); grid on
	hold off
	set(gca, 'xlim', [-inf inf]);
end
```

Output figure

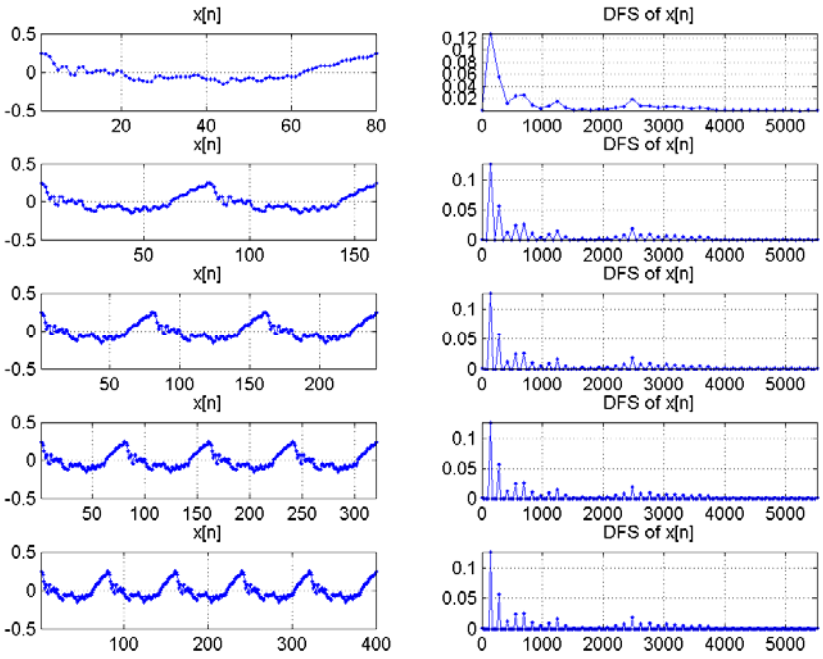


由此可見在實際的聲音波形中，我們只要少數幾個低頻的係數，就可以合成類似原來的訊號，由此也可以知道很多高頻訊號是沒有作用的。
對於週期性的波形，DFT 會插入零點，如下：

Example 8**Input file** ft/fftRepeat01.m

```
% This example demos the effect of FFT for purely periodic signals
[y, fs]=wavread('welcome.wav');
x=y(2047:2126);                 % A full fundamental period

runNum=5;
for i=1:runNum
        repeatedX = x*ones(1,i);
        signal = repeatedX(:);
%       signal=zeros(runNum*length(x), 1);              % Zero-padding version
%       signal(1:length(repeatedX))=repeatedX(:);       % Zero-padding version
        [mag, phase, freq, powerDb]=fftOneSide(signal, fs);
        mag=mag/length(signal);     % Divided by vector length to normalize magnitude (due to the
formula used by MATLAB)

        subplot(runNum,2,2*i-1);
        plot(signal, '.-'); grid on
        title('x[n]'); set(gca, 'xlim', [-inf inf]);

        subplot(runNum,2,2*i);
        plot(freq, mag, '.-'); grid on;
%       set(gca, 'yscale', 'log');
        title('DFT of x[n]'); axis tight;
end
```

Output figure



為什麼 DFT 會插入零點呢？若不使用詳細的數學分析，各位同學是否可以以直覺的方式來推導出這個結果呢？

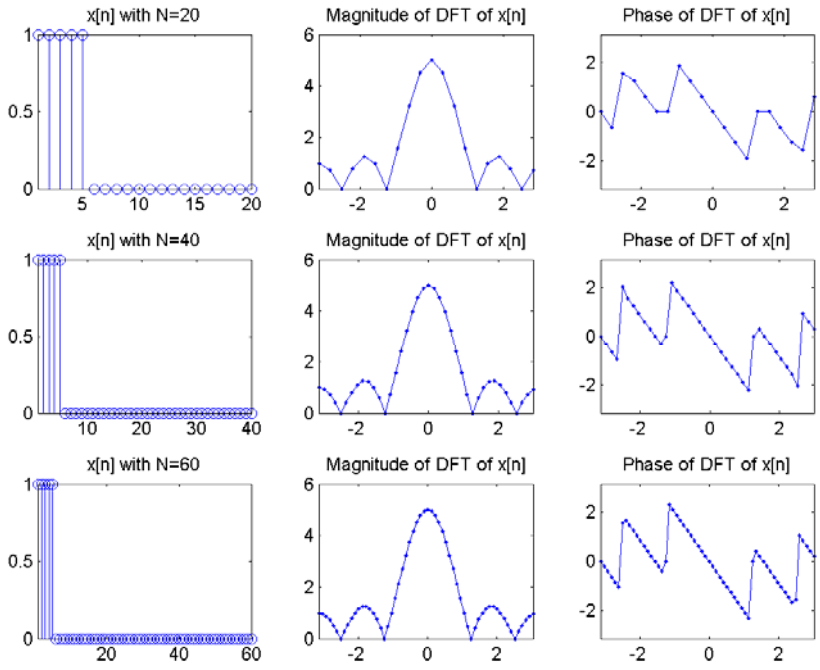
若在時域波形加上零點，則在頻域所產生的效果是內差以增加點數，如下所示：

Example 9**Input file** ft/fftZeroPadding01.m

```
% This example demos the effect of zero-padding of DFT

for i=1:3
        L = 5; N = 20*i;
        x = [ones(1,L), zeros(1,N-L)];
        subplot(3,3,i*3-2);
        stem(x);
        title(sprintf('x[n] with N=%d',N));
        set(gca, 'xlim', [-inf inf]);

        omega=((1:N)-ceil((N+1)/2))*(2*pi/N);
        X = fft(x);
        magX = fftshift(abs(X));
        subplot(3,3,i*3-1);
        plot(omega, magX, '.-');
        title('Magnitude of DFT of x[n]')
        set(gca, 'xlim', [-inf inf]);

        phase=fftshift(angle(X));
        subplot(3,3,i*3);
        plot(omega, phase, '.-');
        title('Phase of DFT of x[n]')
        set(gca, 'xlim', [-inf inf]);
```

```
        set(gca, 'ylim', [-pi pi]);
end
```

Output figure



換句話說，加上零點，在實質上並沒有增加訊號的資訊，但是 DFT 的點數必須隨之增加，因此在頻譜的效應則是進行內差以增加點數。

Hint

一般在進行音訊處理時，如果音框的點數不是 2^n 的格式，我們通常就是使用補零的方式，使最後的點數變成 2^n，這樣在進行 DFT 時，才比較方便採取比較快速的 FFT 計算方式。

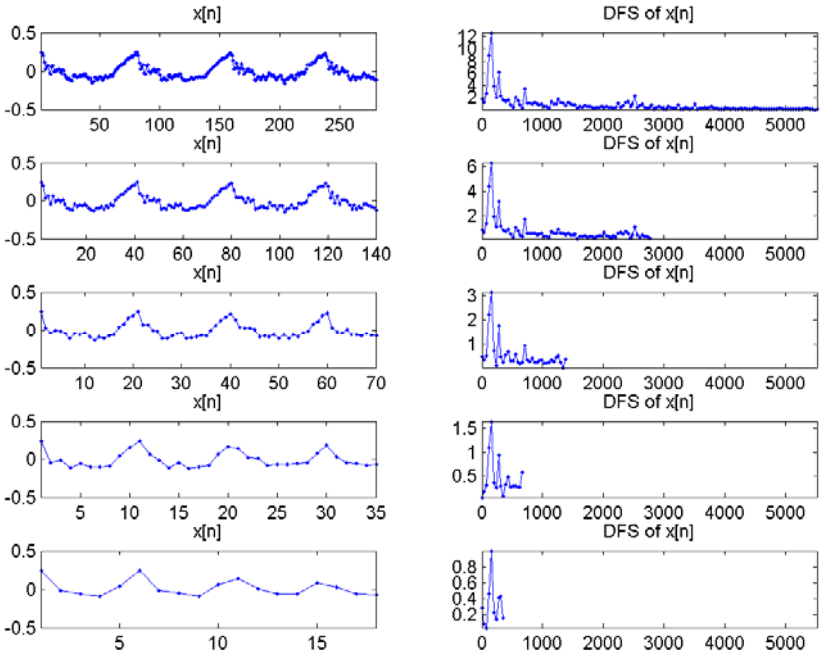
若在時域進行 Downsample，則在頻域所產生的效果如下：

Example 10**Input file** ft/fftReSample01.m

```
% This example demos the effect of FFT approximation

[y, fs]=wavread('welcome.wav');
x=y(2047:2126);
x=y(2047:2326);
n=length(x);
F = (0:n/2)*fs/n;

runNum=5;
for i=1:runNum,
        newX=x(1:2^(i-1):length(x));
        newFs=fs/(2^(i-1));
        X = fft(newX);
        magX = abs(X);
        frameSize=length(newX);

        subplot(runNum,2,2*i-1);
```

```
        plot(newX, '.-');
        title('x[n]');
        set(gca, 'xlim', [-inf inf]);

        subplot(runNum,2,2*i);
        freq = (0:frameSize/2)*newFs/frameSize;
        magX = magX(1:length(freq));
        M=nan*F;
        M(1:length(magX))=magX;
        plot(F, M, '.-');
        title('DFT of x[n]')
        axis tight;
end
```

Output figure



當我們 Downsample 進行越多次，曲線會越平滑，代表高頻部分已經慢慢不見了！
此外，我們可以將原先的時域訊號表示成數個弦波函數的線性組合，並使用最小平方法（lease-squares estimate）來進行曲線擬合（Curve fitting）以找出最佳的線性係數，這些線性係數和 fft 所得到的結果是一樣的，範例如下：

Example 11**Input file** ft/fftViaLse01.m

```
% FFT via least-squares estimate
N=8;
fs=1;
time=(0:N-1)'/fs;
x=rand(N,1)*2-1;

A=ones(N,1);
```

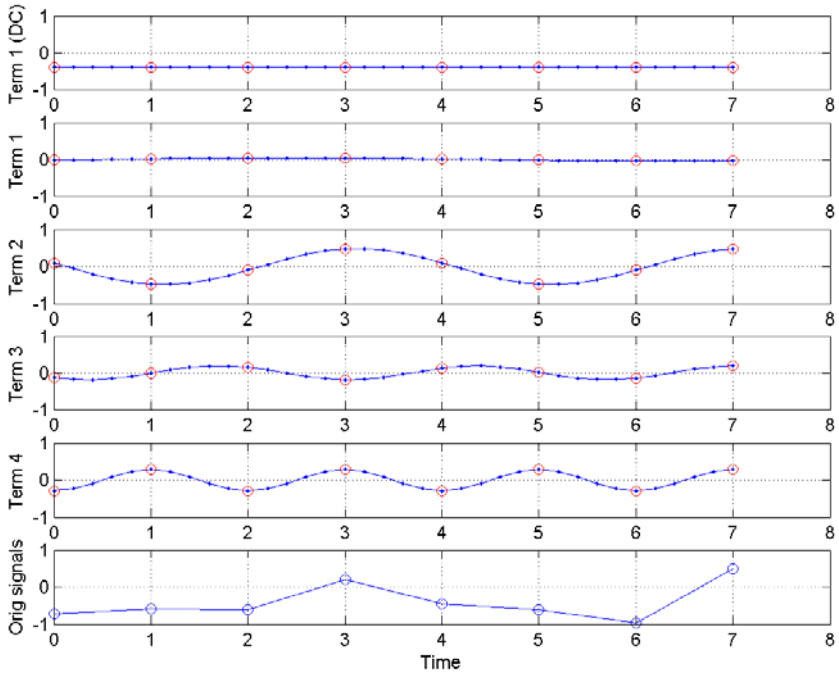
```
for i=1:N/2
        A=[A, cos(2*pi*(i*fs/N)*time), sin(2*pi*(i*fs/N)*time)];
end
th=A\x;

plotNum=fix(N/2)+2;
subplot(plotNum, 1, 1);
N2=(N-1)*5+1;                % Insert 4 points between every 2 points for better observation (兩點間插入四點，以便觀察波形)
timeFine=linspace(min(time), max(time), N2);
x2=th(1)*ones(N,1);
plot(timeFine, th(1)*ones(N2,1), '.-', time, x2, 'or');                % Plot the first term (畫出第一項)
ylabel('Term 0 (DC)'); axis([0 N/fs -1 1]); grid on

for i=1:N/2                  % Plot terms 2 to 1+N/2 (畫出第二至第 1+N/2 項)
        freq=i*fs/N;
        y=th(2*i)*cos(2*pi*freq*time)+th(2*i+1)*sin(2*pi*freq*time);    % a term (每一項)
        x2=x2+y;
        fprintf('i=%d, sse=%f\n', i, norm(x-x2)/sqrt(N));
        subplot(plotNum, 1, i+1);
        yFine=th(2*i)*cos(2*pi*(i*fs/N)*timeFine)+th(2*i+1)*sin(2*pi*(i*fs/N)*timeFine); % Finer verison for plotting
        plot(timeFine, yFine, '.-', time, y, 'or'); ylabel(sprintf('Term %d', i));
        axis([0 N/fs -1 1]); grid on
end

% Plot the original signal (畫出原來訊號)
subplot(plotNum, 1, plotNum)
plot(time, x, 'o-'); axis([0 N/fs -1 1]); grid on
xlabel('Time'); ylabel('Orig signals');

% Transform LSE result back to fft format for comparison (將 th 轉回 fft 並比較結果)
F=fft(x);
F2=[];
F2(1)=th(1)*N;
for i=1:N/2
        F2(i+1)=(th(2*i)-sqrt(-1)*th(2*i+1))*N/2;
        if (i==N/2)
                F2(i+1)=2*F2(i+1);
        end
end
% symmetric of DFT (DFT 的對稱性)
for i=N/2+2:N
        F2(i)=F2(N-i+2)';
end

error1=sum(abs(F2-F.'));     % F.' is simple transpose (F.' 是不進行共軛轉換的轉置)
error2=sum(abs(F2-F'));                % F' is conjugate transpose (F' 是進行共軛轉換的轉置)
```

```
fprintf('Errors after transforming LSE to DFT coefficients (將 LSE 轉換成 DFT 係數的誤差):
error1=%f, error2=%f\n', error1, error2);
fprintf('Due to the symmetry of DFT, one of the above error terms should be zero. (由於 DFT 的對稱性，上述誤差應該有一項為零。)\n');
```

Output message

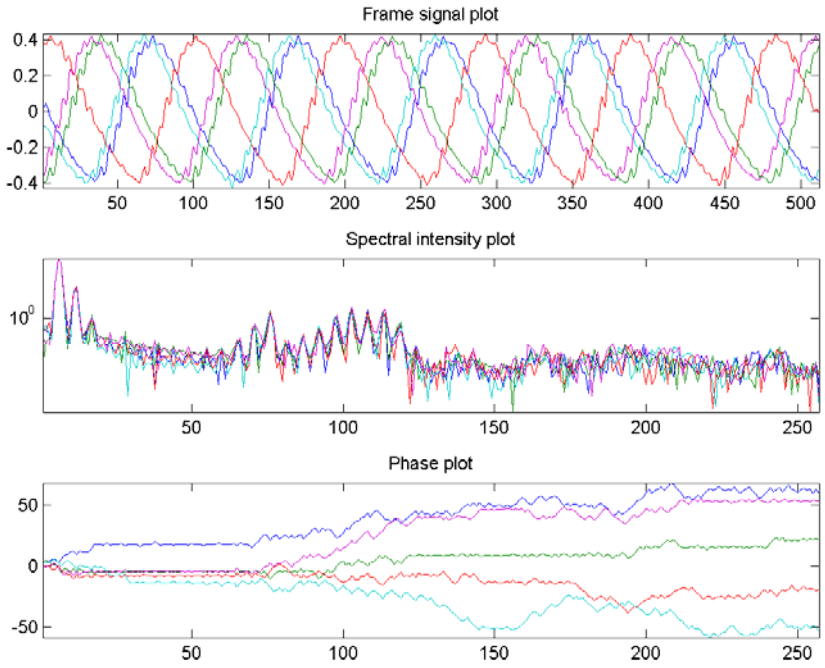
```
i=1, sse=0.462366
i=2, sse=0.310406
i=3, sse=0.281771
i=4, sse=0.000000
Errors after transforming LSE to DFT coefficients (將 LSE 轉換成 DFT 係數的誤差):
error1=0.000000, error2=10.527154
Due to the symmetry of DFT, one of the above error terms should be zero. (由於 DFT 的對稱性，上述誤差應該有一項為零。)
```

Output figure



1. (**) **計算中文母音頻譜之一：** 請寫一個 MATLAB 程式 vowelSpectrumPlot01.m，完成下列功能：
    a. 進行三秒錄音，錄音規格是 16KHz/16Bits/Mono，錄音內容是中文母音「ㄧ」。（請盡量維持穩定的音高及音量，可以試聽此範例檔案。）
    b. 使用 buffer2.m 及 frame2volume.m 來切出音框並計算音量，相關規格是 frameSize=32ms, overlap=0ms。（提示：你必須先將 frameSize 及 overlap 轉成點數。）請抓出音量最大的音框，在其前後各抓出 2 個音框，總共有 5 個音框。
    c. 使用 fft 來計算這五個音框的頻譜，含大小和相位，並畫出所有相關圖形：
        1. subplot(3,1,1)：五個音框的訊號圖。
        2. subplot(3,1,2)：五個音框的單邊頻譜強度圖。
        3. subplot(3,1,3)：五個音框的單邊頻譜相位圖。

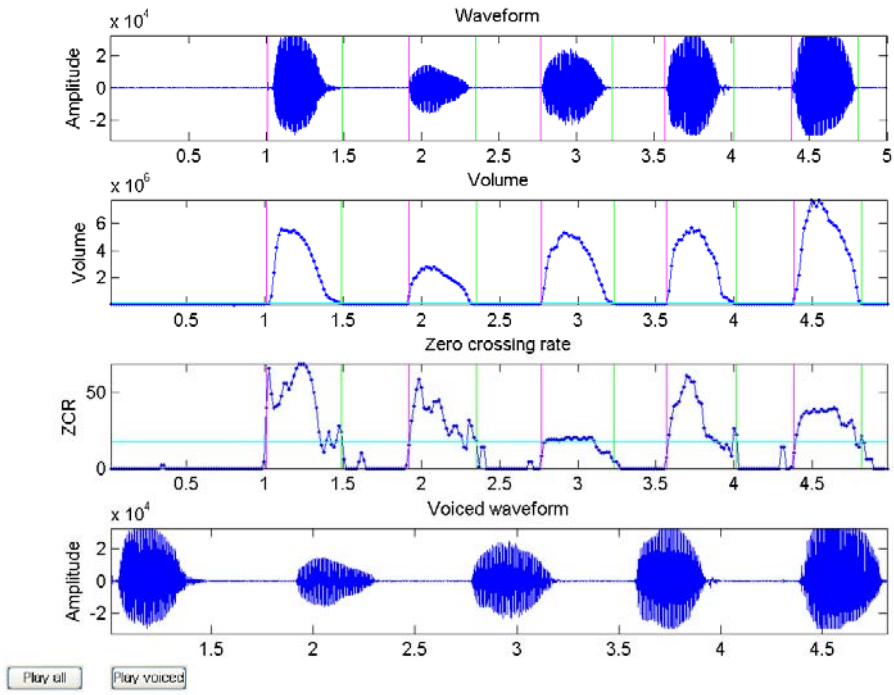
畫出的圖形應該類似下圖：



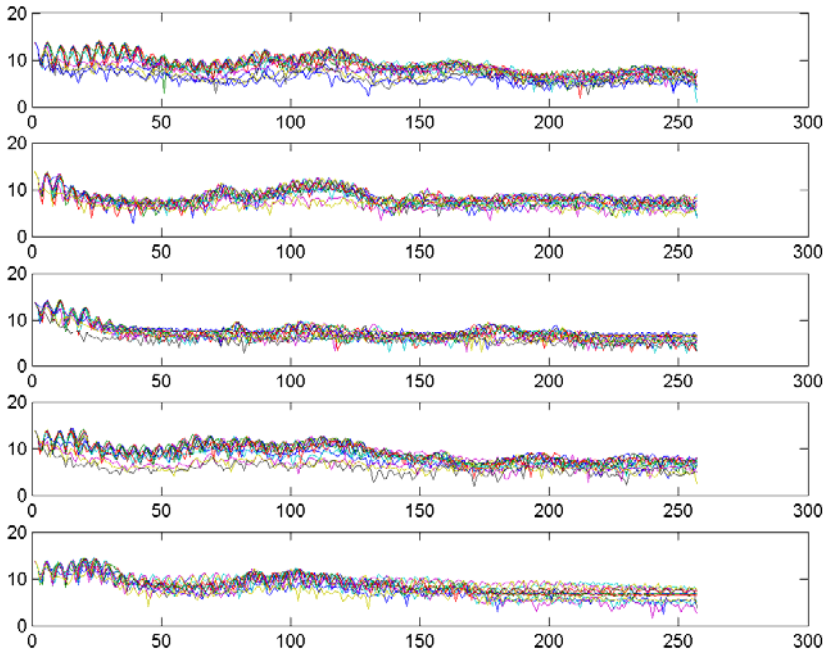
（提示：(a)對音框進行 fft 時，可以先乘上 Hamming window，可以讓算出來的頻譜比較乾淨。程式碼是：frame=frame.*hamming(length(frame))。(b)在畫出單邊頻譜強度圖，可以使用 log 刻度，比較容易看出諧波結構，方法是在畫第二個圖後，下達下列命令：set(gca, 'yscale', 'log')。(c) 為使相位產生連續的曲線，可以使用 unwarp 指令。）

d. 請觀察看看所畫出的三個圖，那一個圖的五條曲線的相似度最高？（提示：由於我們的發音都是「ㄧ」，因此相似度最高的圖所用的特徵，就是穩定不變的特徵，可用於語音辨識。）

2. (**) **計算中文母音頻譜之二：** 請寫一個 MATLAB 程式 vowelSpectrumPlot02.m，功能如同上一題，但將錄音內容改成中文母音「ㄧ ˊ」。此題可用來探討在音高變化的情況下，頻譜的變化情況。
3. (***) **計算中文母音頻譜並進行分類：** 請寫一個 MATLAB 程式 vowelSpectrumRecogChinese01.m，完成下列功能：
   a. 進行五秒錄音，錄音規格是 16KHz/16Bits/Mono，錄音內容是中文母音「ㄚ、ㄧ、ㄨ、ㄝ、ㄛ」。（發音請盡量平穩，並在母音之間稍許停頓，以便後續進行自動切音，可以試聽此範例檔案。）
   b. 使用 endPointDetect.m (in Audio Toolbox) 來找出這五個音的開始和結束位置。請調整相關端點偵測參數，使得你的程式能夠自動地正確切出這五個音。你可以設定 plotOpt=1，以便使用 endPointDetect.m 來畫出端點偵測結果，正確圖形類似下圖：

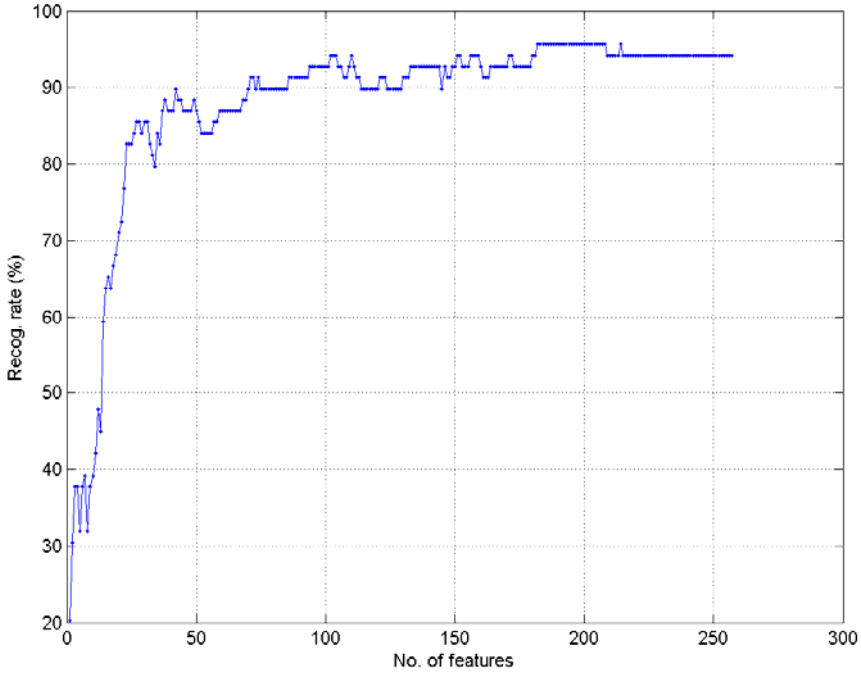


c. 使用 buffer2.m 來對這五個音切出音框（frameSize=32ms, overlap=0ms，但你必須先將 frameSize 及 overlap 轉成點數）並計算頻譜強度，請將這五個音的頻譜強度畫出來，每一個母音對應到 m 條曲線，m 是此母音的音框個數，請比較此 m 條曲線的相似度。畫出圖形應該類似下圖：



d. 請用 knnrLoo.m (in DCPR Toolbox) 來算出使用 KNNR 將資料分成五類母音的 leave-one-out 辨識率。（可以設定 k = 1。）

e. 請將特徵數目由 1 增加到 128，畫出 KNNR 的 leave-one-out 辨識率對特徵數目的曲線。畫出圖形應該類似下圖：



f. 請用 PCA 或 LDA 將這些頻譜強度資料投影到二度空間，並將圖形畫出來，觀察看看是否同一類母音，其資料點有聚集在一起的傾向。

4. (**) **計算英文母音頻譜並進行分類：** 請寫一個 MATLAB 程式 vowelSpectrumRecogEnglish01.m.m，功能如同上一題，但將錄音內容改成英文母音「i、e、 、o、u...」。

Chapter 11: Digital Filters

本節將介紹在數位訊號處理常用到的濾波器（Filters），並簡單介紹其應用。一般人大概都聽過濾波器，但是對於相關的數學分析，可能都覺得很複雜，敬而遠之。本節將從應用面來著手，並說明相關的 MATLAB 指令，所以並不會牽涉到複雜的分析。

簡單的說，一個濾波器可以用兩個向量 **a** 和 **b** 來代表，其中 **a** 的長度是 p，**b** 的長度是 q，而且 **a** 的第一個元素 a_1 永遠是 1，如下：

$$\mathbf{a} = [1, a_2, \ldots a_p]$$
$$\mathbf{b} = [b_1, b_2, \ldots b_q]$$

當我們把具有參數 **a** 和 **b** 的濾波器應用到一段離散時間訊號 x[n]，所得到的結果是 y[n]，可以表示如下：

$$y[n] = b_1x[n] + b_2x[n-1] + \ldots + b_qx[n-q+1]$$
$$-a_2y[n-1] - a_3y[n-2] - \ldots - a_px[n-p+1]$$

這個式子看起來有點複雜，下面我們來看起個特例，就會比較清楚。

首先，如果

$$\mathbf{a} = [1]$$
$$\mathbf{b} = [1/5, 1/5, 1/5, 1/5, 1/5]$$

那麼濾波器的輸出就是

$$y[n] = (x[n] + x[n-1] + x[n-2] + x[n-3] + x[n-4])/5$$

這就是一個簡單的濾波器，換句話說，濾波器的輸出值，等於原訊號在過去五點的平均值，因此這個濾波器就有「低通」（Low Pass）的效果，換句話說，比較尖銳的訊號（高頻的部分），經過了五點求平均，會變的比較平滑，因此整個訊號的高頻部分會被壓低，而低頻部分則比較不受到影響，所以這一類濾波器就稱為「低通濾波器」（Low Pass Filter）。低通濾波器的效果，就好像用紙杯罩著嘴巴講話，只剩下低沈、模糊、悶悶的聲音。

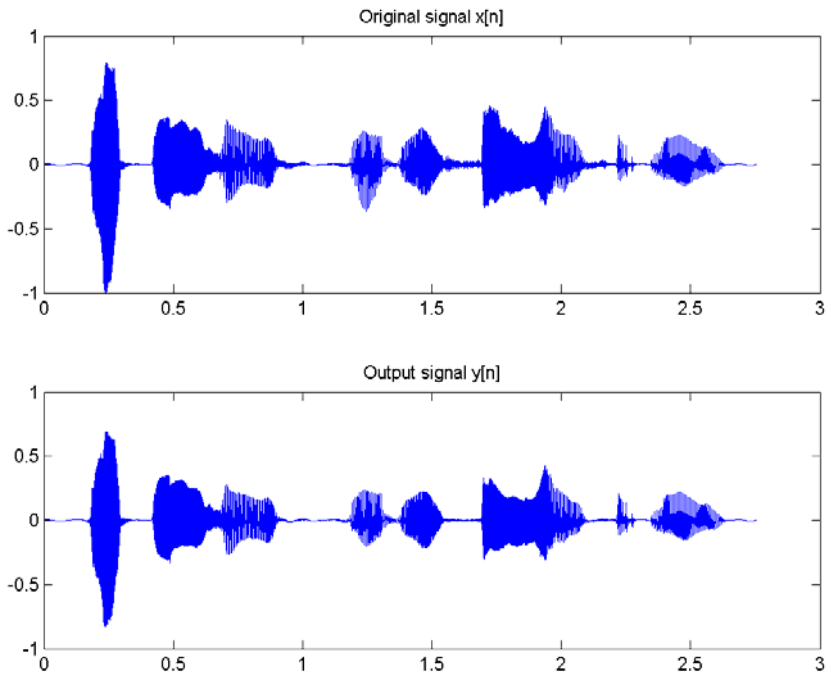
Hint

為什麼罩上紙杯講話，會有低通濾波器的效果？請想想看...
以下是一個範例：

Example 1**Input file** filter/lpf01.m

```
waveFile='whatMovies.wav';
[x, fs, nbits]=wavread(waveFile);
% Filter parameters
a = [1];
b = [1, 1, 1, 1, 1]/5;
y = filter(b, a, x);
% Plot the result
time = (1:length(x))/fs;
subplot(2,1,1);
plot(time, x); title('Original signal x[n]');
subplot(2,1,2);
plot(time, y); title('Output signal y[n]');
wavwrite(y, fs, nbits, 'lpf01.wav');      % Save the output signal
```

Output figure



在上述範例中，y = filter(b, a, x) 即是由原始訊號 x 及濾波器參數 a 和 b，來產生濾波器的輸出訊號 y。由上述範例圖形可以看出，在高頻的部分（尤其是氣音的部分），訊號震幅明顯降低，這就是低通濾波器的效果。我們也可以直接聽看看訊號的差異：

- 原訊號 x[n]：example/filter/whatMovies.wav
- 濾波器輸出訊號 y[n]：example/filter/lpf01.wav

我們在看另一組濾波器：

a = [1]
b = [1, -1]

那麼濾波器的輸出就是

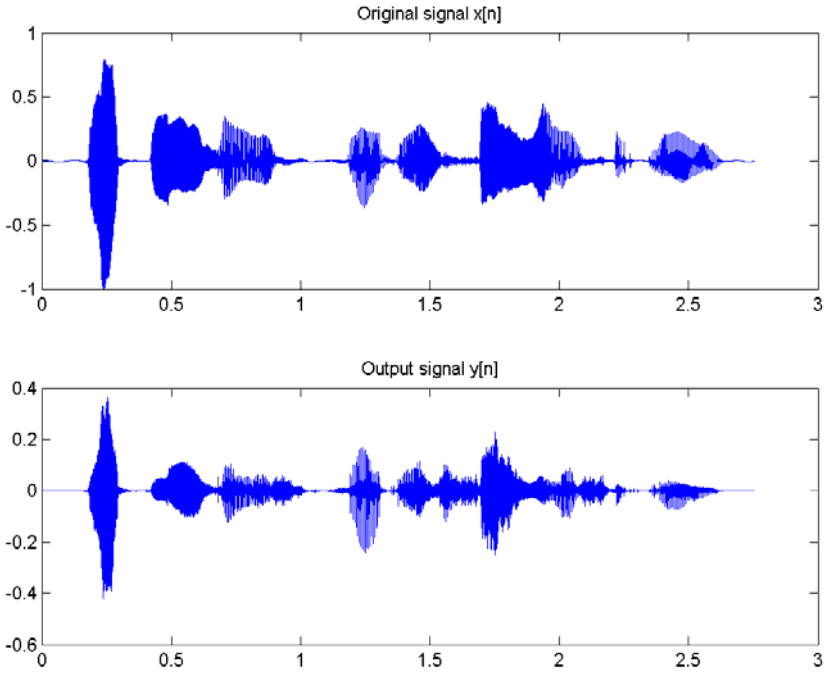
y[n] = x[n] - x[n-1]

換句話說，濾波器的輸出值，等於原訊號目前的值減掉原訊號在前一個時間點的值，因此變化劇烈的訊號（即高頻訊號），其差異會被突顯出來，而變化不劇烈的訊號（即低頻訊號），其差異會被壓抑，所以這一類的濾波器就稱為「高通濾波器」（High Pass Filter），請見下列範例：

Example 2**Input file** filter/hpf01.m

```
waveFile='whatMovies.wav';
[x, fs, nbits]=wavread(waveFile);
% Filter parameters
a = [1];
b = [1, -1];
y = filter(b, a, x);
% Plot the result
time = (1:length(x))/fs;
subplot(2,1,1);
plot(time, x); title('Original signal x[n]');
subplot(2,1,2);
plot(time, y); title('Output signal y[n]');
wavWrite(y, fs, nbits, 'hpf01.wav');    % Save the output signal
```

Output figure



由上述範例圖形可以看出，在高頻的部分（尤其是氣音的部分），訊號震幅相對明顯變大，這就是高通濾波器的效果。我們也可以直接聽看看訊號的差異：

- 原訊號 x[n]：example/filter/whatMovies.wav
- 濾波器輸出訊號 y[n]：example/filter/hpf01.wav

經過了高通濾波器之後，訊號聽起來的感覺，好像就是小型收音機（沒有重低音喇叭）的播放效果，比較尖銳刺耳。

除了高通和低通濾波器外，濾波器也可以產生各種不同的音效，例如

a = [1]

b = [1, 0, 0, 0, ..., 0, 0.8] （中間夾雜了 3199 個零）

那麼濾波器的輸出就是

$$y[n] = x[n] + 0.8*x[n-3200]$$

換句話說，這個濾波器的功能就是「單次迴音器」，會產生一次迴音，若取樣頻率 fs = 16kHz，則迴音和原音的時間差是 3200/fs = 3200/16000 = 0.2 秒。請見下列範例：

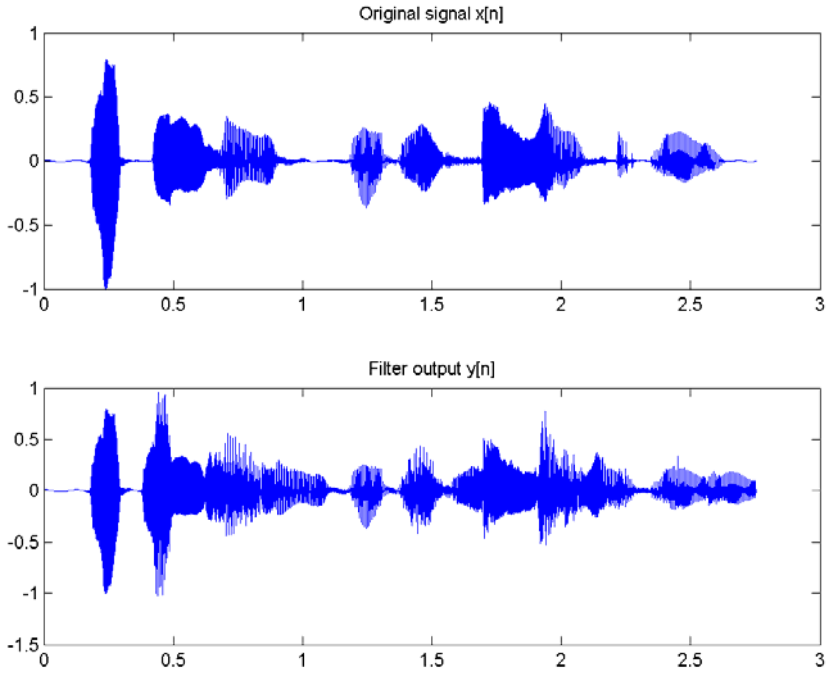
Example 3**Input file** filter/echo01.m

```
waveFile='whatMovies.wav';
[x, fs, nbits]=wavread(waveFile);
% Filter parameters
delay=0.2;
gain=0.8;
a = [1];
b = [1, zeros(1, round(delay*fs)-1), gain];
y = filter(b, a, x);
% Plot the result
time = (1:length(x))/fs;
subplot(2,1,1);
plot(time, x); title('Original signal x[n]');
subplot(2,1,2);
plot(time, y); title('Filter output y[n]');
wavWrite(y, fs, nbits, 'echo01.wav');  % Save the output signal
```

Output message

```
Warning: Data clipped during write to file:echo01.wav
> In wavwrite>PCM_Quantize at 241
  In wavwrite>write_wavedat at 267
  In wavwrite at 112
  In echo01 at 15
  In goWriteOutputFile at 32
```

Output figure



在上述範例中，gain = 0.8 代表迴音的衰減比值，而 delay = 0.2 則是迴音的時間差。我們可以聽看看原訊號和濾波器輸出的差異：

- 原訊號 x[n]：example/filter/whatMovies.wav
- 濾波器輸出訊號 y[n]：example/filter/echo01.wav

上述濾波器只能產生一個迴音，比較不像一般多重迴音。若要產生多重迴音，可用下列濾波器：

a = [1, 0, 0, 0, ..., 0, -0.8] （中間夾雜了 3199 個零）

b = [1]

那麼濾波器的輸出就是

y[n] = x[n] + 0.8*y[n-3200]

這個濾波器的功能就是「多重迴音器」，會產生多重迴音，若取樣頻率 fs = 16kHz，則迴音之間的時間差是 3200/fs = 3200/16000 = 0.2 秒。請見下列範例：

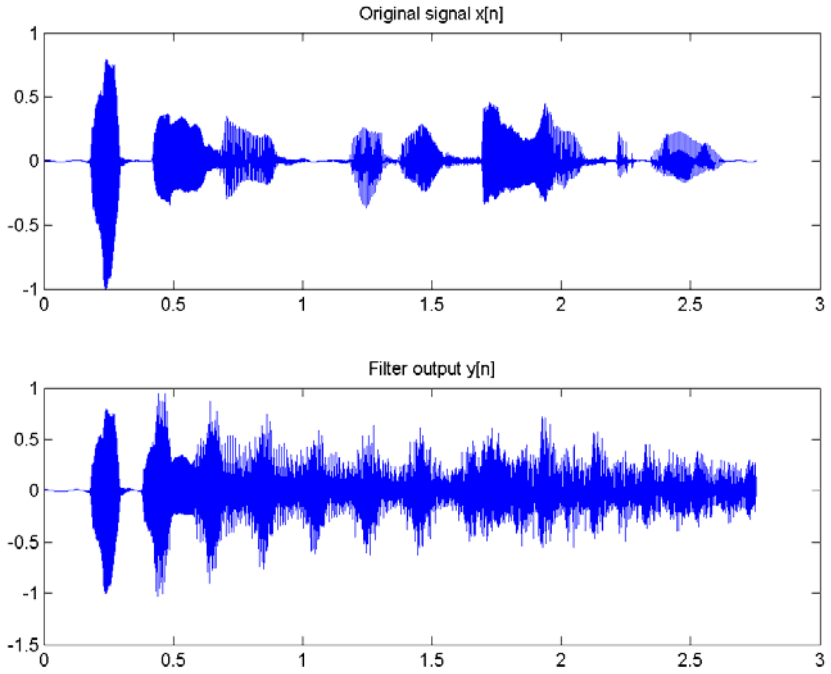
Example 4**Input file** filter/echo02.m

```
waveFile='whatMovies.wav';
[x, fs, nbits]=wavread(waveFile);
% Filter parameters
delay=0.2;
gain=0.8;
a = [1 zeros(1, round(delay*fs)), -gain];
b = [1];
y = filter(b, a, x);
% Plot the result
time = (1:length(x))/fs;
subplot(2,1,1);
plot(time, x); title('Original signal x[n]');
subplot(2,1,2);
plot(time, y); title('Filter output y[n]');
wavWrite(y, fs, nbits, 'echo02.wav');  % Save the output signal
```

Output message

```
Warning: Data clipped during write to file:echo02.wav
> In wavwrite>PCM_Quantize at 241
  In wavwrite>write_wavedat at 267
  In wavwrite at 112
  In echo02 at 15
  In goWriteOutputFile at 32
```

Output figure



我們可以聽看看原訊號和濾波器輸出的差異：

- 原訊號 x[n]：example/filter/whatMovies.wav
- 濾波器輸出訊號 y[n]：example/filter/echo02.wav（是否有「魔音傳腦」的感覺？）

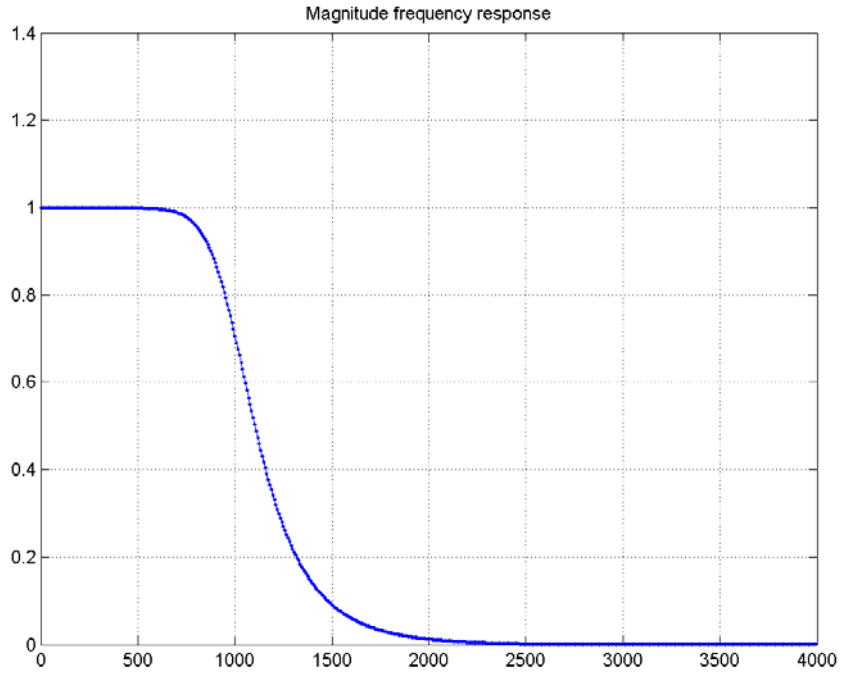
在上一節中，我們介紹了幾種基本的濾波器，並說明其應用。本節將介紹如何以簡單的 MATLAB 指令來設計濾波器，並顯示其效果。

我們可以使用 butter 指令來設計一個 Butterworth 濾波器，範例如下：

Example 1**Input file** filter/butter01.m

```
fs=8000;              % Sampling rate
filterOrder=5;                % Order of filter
cutOffFreq=1000; % Cutoff frequency
[b, a]=butter(filterOrder, cutOffFreq/(fs/2), 'low');
% === Plot frequency response
[h, w]=freqz(b, a);
plot(w/pi*fs/2, abs(h), '.-'); title('Magnitude frequency response');
grid on
```

Output figure



在上述範例中，我們使用 butter 指令來設計一個 Butterworth 低通濾波器，其格式如下：

[b, a] = butter(order, Wn, function)

對於輸入參數，我們可以說明如下：

- order 是濾波器的階數，階數越大，濾波效果越好，但是計算量也會跟著變大。所產生的濾波器參數 a 和 b 的長度，等於 order+1。
- Wn 是正規化的截止頻率，介於 0 和 1 之間，當取樣頻率是 fs 時，所能處理的最高頻率是 fs/2，所以如果實際的截止頻率是 f = 1000，那麼 Wn = f/(fs/2)。
- function 是一個字串，function = 'low' 代表是低通濾波器，function = 'high' 代表是高通濾波器。

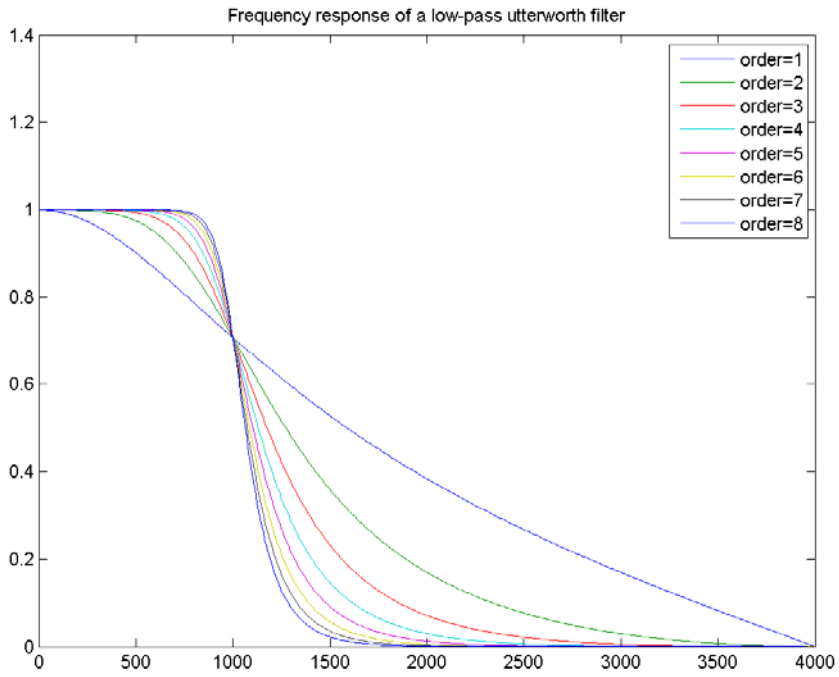
上述範例所產生的四個圖形，事實上是同一個圖，只是分別在 x 軸或 y 軸使用對數刻度，所以造成不同的效果。這些圖都稱為濾波器的「頻率響應」（Frequency Response），顯示不同頻率的訊號經過此濾波器時，所乘上的衰減率。上述範例中，我們是要設計一個截止頻率為 1000 Hz 的濾波器，由頻率響應可以看出，這果然是一個低通濾波器。

當濾波器的階數越高時，，因為濾波器參數 a 和 b 的長度變長，濾波的效果越明顯，但是計算量也會跟著提高；反之，若階數越低，濾波器參數 a 和 b 的長度變短，計算量降低，但是濾波的效果也會變差，請見下列範例：

Example 2**Input file** filter/butter02.m

```
fs=8000;                    % Sampling rate
cutOffFreq=1000;            % Cutoff frequency
allH=[];
for filterOrder=1:8;
        [b, a]=butter(filterOrder, cutOffFreq/(fs/2), 'low');
        % === Plot frequency response
        [h, w]=freqz(b, a);
        allH=[allH, h];
end
plot(w/pi*fs/2, abs(allH)); title('Frequency response of a low-pass utterworth filter');
legend('order=1', 'order=2', 'order=3', 'order=4', 'order=5', 'order=6', 'order=7', 'order=8');
```

Output figure



在上述範例中，可以很明顯地看出，當階數由 1 慢慢增大成 8 時，濾波器的效果也會越來越明顯。我們可以將音訊通過截止頻率為 1000 Hz 的低通濾波器，看是否能夠把過濾高音，範例如下：

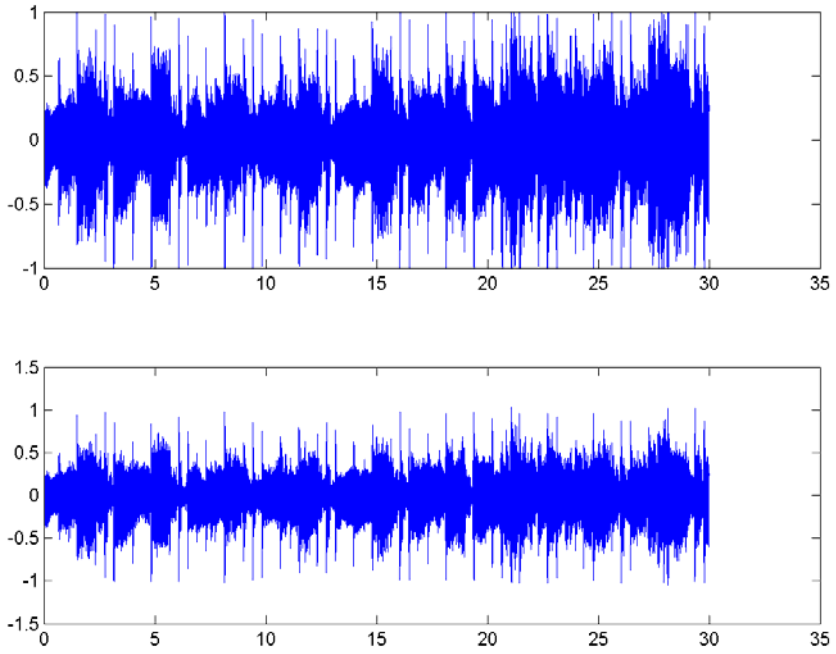
Example 3**Input file** filter/butter03.m

```
cutOffFreq=1000; % Cutoff frequency
filterOrder=5;                  % Order of filter
[x, fs, nbits]=wavRead('wubai_solicitude.wav');
[b, a]=butter(filterOrder, cutOffFreq/(fs/2), 'low');
x=x(60*fs:90*fs);  % 30-second signal
y=filter(b, a, x);
% ====== Save output files
wavwrite(x, fs, nbits, 'wubai_solicitude_orig.wav');
wavwrite(y, fs, nbits, sprintf('wubai_solicitude_%d.wav', cutOffFreq));
% ====== Plot the result
time=(1:length(x))/fs;
subplot(2,1,1);
plot(time, x);
subplot(2,1,2);
plot(time, y);
```

Output message

```
Warning: Data clipped during write to file:wubai_solicitude_1000.wav
> In wavwrite>PCM_Quantize at 241
  In wavwrite>write_wavedat at 267
  In wavwrite at 112
  In butter03 at 9
  In goWriteOutputFile at 32
```

Output figure



我們可以聽看看原訊號和濾波器輸出訊號的差異：

- 原訊號 x[n]：example/filter/wubai_solicitude_orig.wav
- 濾波器輸出訊號 y[n]：example/filter/wubai_solicitude_1000.wav

可以很明顯地聽到，高音部分都幾乎被刪除了。

如果我們將截止頻率設定成 100 Hz，此時濾波器的訊號，就幾乎只能聽到低音鼓的聲音，範例如下：

Example 4**Input file** filter/butter04.m

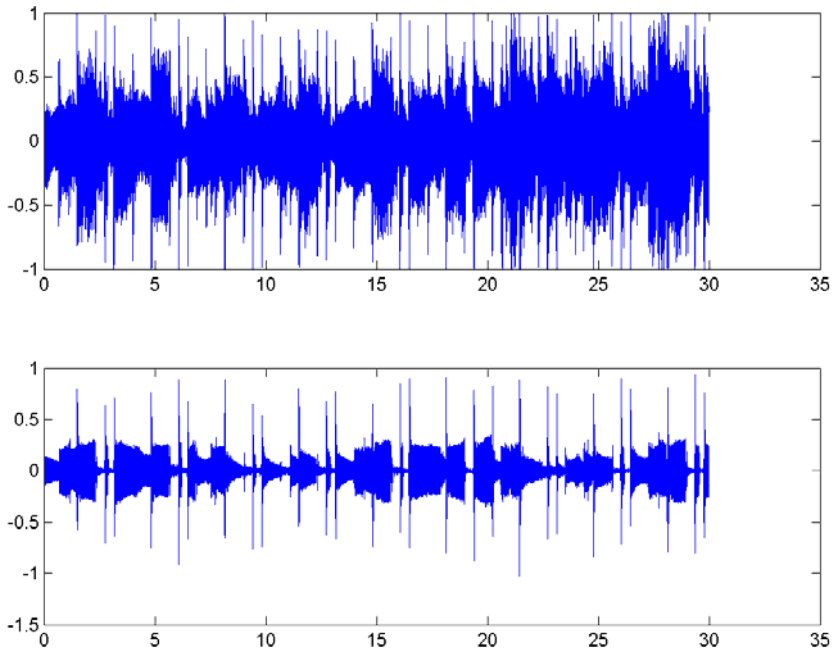
```
cutOffFreq=100;                % Cutoff freq (截止頻率)
filterOrder=5;                 % Order of filter (濾波器的階數)
[x, fs, nbits]=wavRead('wubai_solicitude.wav');
[b, a]=butter(filterOrder, cutOffFreq/(fs/2), 'low');
x=x(60*fs:90*fs);  % 30 seconds of singing (30  秒歌聲)
y=filter(b, a, x);
% ====== Save wav files (存檔)
wavwrite(x, fs, nbits, 'wubai_solicitude_orig.wav');
wavwrite(y, fs, nbits, sprintf('wubai_solicitude_%d.wav', cutOffFreq));
% ====== Plotting (畫圖)
time=(1:length(x))/fs;
subplot(2,1,1);
plot(time, x);
subplot(2,1,2);
plot(time, y);
```

Output message

```
Warning: Data clipped during write to file:wubai_solicitude_100.wav
> In wavwrite>PCM_Quantize at 241
```

```
In wavwrite>write_wavedat at 267
In wavwrite at 112
In butter04 at 9
In goWriteOutputFile at 32
```

Output figure



- 原訊號 x[n]：example/filter/wubai_solicitude_orig.wav
- 濾波器輸出訊號 y[n]：example/filter/wubai_solicitude_100.wav

很明顯地，經過了濾波器的作用，只留下低音鼓的聲音，而且由這些規律出現的低音鼓聲音，我們就可以進行節拍追蹤，找出這一段音樂的節拍。（當然，這只是一個開始，若要進行節拍追蹤，還有很多細節要處理。）

如果你還是聽不出來低音鼓在原來音樂的位置，可以嘗試逐次聽聽下列檔案（最好使用 CoolEdit 來聽，可以同步顯示播放進度），就應該可以慢慢抓到低音鼓的位置：

- 截止頻率 = 100 Hz：example/filter/wubai_solicitude_100.wav
- 截止頻率 = 200 Hz：example/filter/wubai_solicitude_200.wav
- 截止頻率 = 300 Hz：example/filter/wubai_solicitude_300.wav
- 截止頻率 = 400 Hz：example/filter/wubai_solicitude_400.wav
- 截止頻率 = 500 Hz：example/filter/wubai_solicitude_500.wav
- 截止頻率 = 1000 Hz：example/filter/wubai_solicitude_1000.wav
- 原訊號：example/filter/wubai_solicitude_orig.wav

第 11 章作業

1. (**) **Use filters to obtain pure-tone signals**： Write a MATLAB script to separate a mixed signal of 3 pure-tone components, as follows.
   a. Load the audio file mixSinusoid.wav which is composed of three pure-tone signals with some noise. Can you identify the pitch of these three pure-tone signals by aural perception?

b. Use fftOneSide.m to plot the one-side magnitude spectrum with respect to frequency. What are the frequencies of these three components?
c. Use three Butterworth filters (low-pass, band-pass, and hig-hpass) to recover these three components. What is the cutoff frequency you used for designing each filter? Plot these three components for the first 1000 points.
d. Play these three signals to see if you can hear pure tone. Add these three signals together and play it to see if you can resynthesize the original signal.

Chapter 12: Speech Features

12-1 共振峰

Old Chinese version

一般語音訊號的母音部分，都會出現重複性很高的基本週期，在一個基本週期內的波形，就是代表語音的內容或音色。因此在理論上，我們可以進行下列分析：

1. 在一個音框內抓出一個代表性的基本週期。
2. 求取這個基本週期的 DFT。
3. 使用 DFT 來計算頻譜能量，並用這些頻譜能量來代表這個音框的特徵，以進行語音辨識。

一般來說，頻譜能量的個數仍然太多，因此一個簡單的方式，是採用頻譜能量圖的局部最大點，稱為共振峰（Formants），來作為語音特徵。以下這個範例，就是抓出一個音框內的一個完整的基本週期，然後算出頻譜能量及對應的共振峰： Error: D:\users\jang\books\audioSignalProcessing\example\chap11-speechFeature\fftFormant01.m does not exist!

在語音辨識（Speech Recognition）和語者辨識（Speaker Recognition）方面，最常用到的語音特徵就是「梅爾倒頻譜係數」（Mel-scale Frequency Cepstral Coefficients，簡稱 MFCC），此參數考慮到人耳對不同頻率的感受程度，因此特別適合用在語音辨識。我們將逐一說明 MFCC 的計算過程。

1. **預強調**（Pre-emphasis）：將語音訊號 s(n) 通過一個高通濾波器：

$$H(z)=1-a*z^{-1}$$

其中 a 介於 0.9 和 1.0 之間。若以時域的運算式來表示，預強調後的訊號 $s_2(n)$ 為

$$s_2(n) = s(n) - a*s(n-1)$$

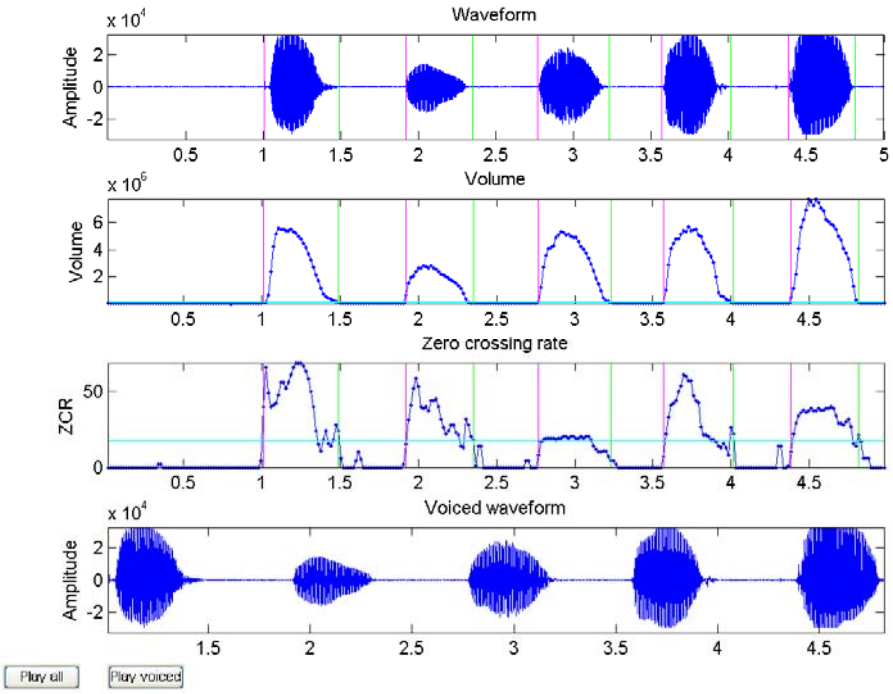
這個目的就是為了消除發聲過程中聲帶和嘴唇的效應，來補償語音信號受到發音系統所壓抑的高頻部分。（另一種說法則是要突顯在高頻的共振峰。）下面這個範例可以示範預強調所產生的效果： Error: D:\users\jang\books\audioSignalProcessing\example\chap11-speechFeature\preEmphasis01.m does not exist!

第 12 章作業

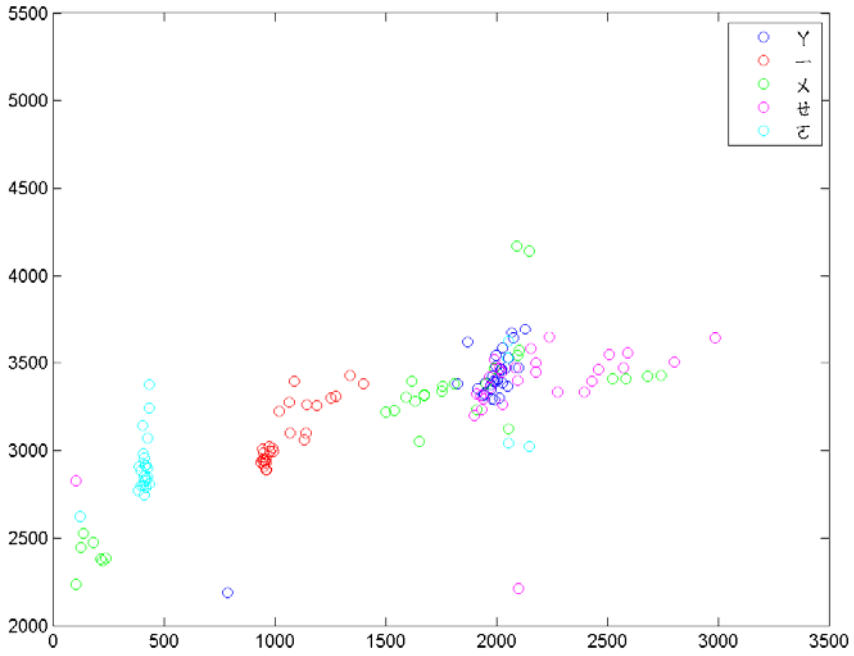
Old Chinese version

1. (**) **計算中文母音共振峰並畫圖**： 請寫一個 MATLAB 程式 vowelFormantRecogChinese01.m，完成下列功能：
   a. 進行五秒錄音，錄音規格是 16KHz/16Bits/Mono，錄音內容是中文母音「ㄚ、ㄧ、ㄨ、ㄝ、ㄛ」。（發音請盡量平穩，並在母音之間稍許停頓，以便後續進行自動切音，可以試聽此範例檔案。）
   b. 使用 wave2formant.m (in ASR Toolbox) 來抓出兩個共振峰，相關規格是 formantNum=2, frameSize=20ms, frameStep=10ms, lpcOrder=12，這些參數也是 wave2formant.m 的預設參數。

c. 使用 endPointDetect.m (in Audio Toolbox) 來找出這五個音的開始和結束位置。請調整相關端點偵測參數，使得你的程式能夠自動地正確切出這五個音。你可以設定 plotOpt=1，以便使用 endPointDetect.m 來畫出端點偵測結果，正確圖形類似下圖：



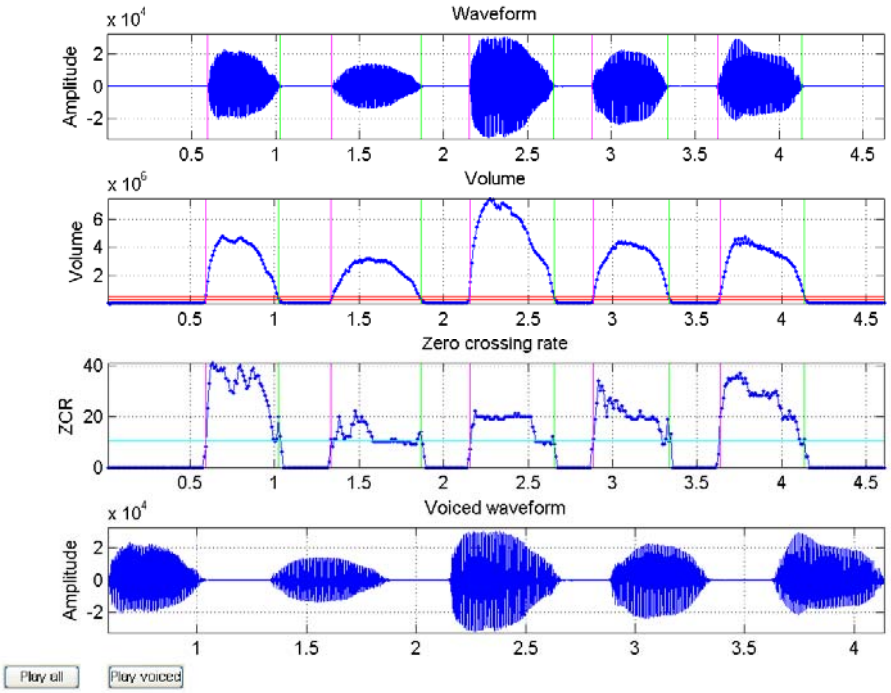
d. 並用不同的顏色，在二度空間畫出這五個音的共振峰。畫出圖形應該類似下圖：



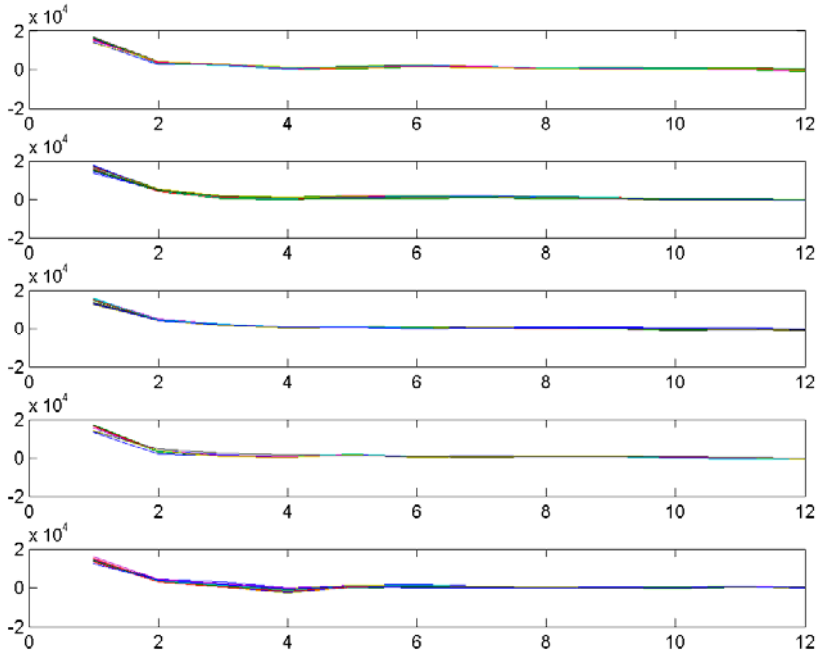
e. 請用 knnrLoo.m (in DCPR Toolbox) 來算出使用 knnr 將資料分成五類的 leave-one-out 辨識率。

2. (**) **Frame to MFCC conversion**： Write a MATLAB function frame2mfcc.m that can compute 12-dimensional MFCC from a given speech/audio frame. Please follow the steps in the text.
Solution: Pleae check out the function in the Audio Processing Toolbox.

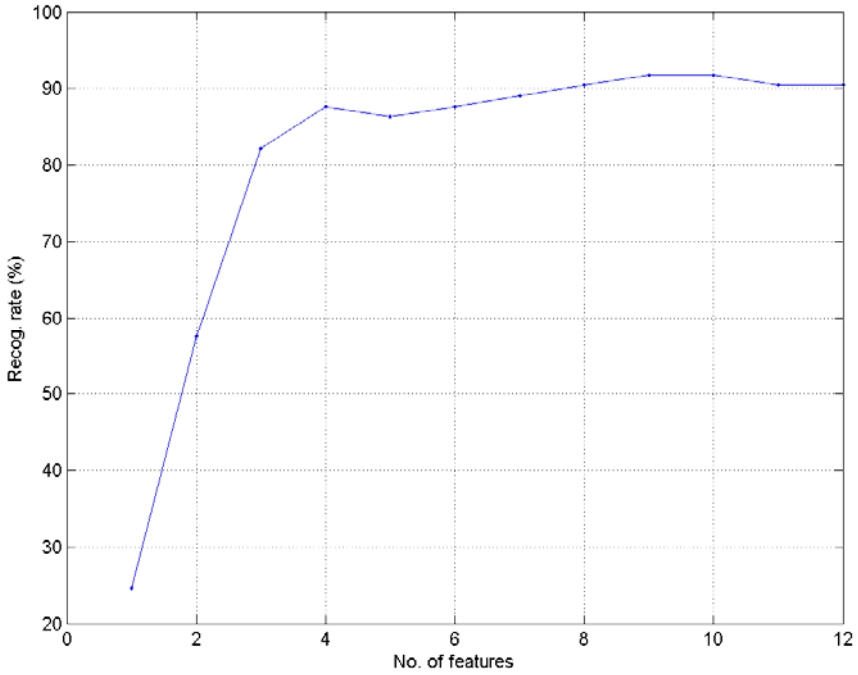
3. (***) **Use MFCC for classifying vowels：** Write an m-file script to do the following tasks:
   a. Record a 5-second clips of the Chinese vowel「ㄚ、ㄧ、ㄨ、ㄝ、ㄛ」(or the English vowels "a, e, i, o, u") with 16KHz/16Bits/Mono. (Please try to maintain a stable pitch and volume, and keep a short pause between vowels to facilitate automatic vowel segmentation. Here is a sample file for your reference.)
   b. Use `epdByVol.m` (in Audio Processing Toolbox) to detect the starting and ending positions of these 5 vowels. If the segmentation is correct, you should have 5 sound segments from the 3rd output argument of `epdByVol.m`. Moreover, you should set plotOpt=1 to verify the segmentation result. Your plot should be similar to the following:



   If the segmentation is not correct, you should adjust the parameters to `epdByVol.m` until you get the correct segmentation.
   c. Use buffer2.m to do frame blocking on these 5 vowels, with frameSize=32ms and overlap=0ms. Please generate 5 plots of MFCC (use `frame2mfcc.m` or `wave2mfcc.m`) corresponding to each vowel. Each plot should contains as many curves of MFCC vectors as the number of frames in this vowel. Your plots should be similar to those shown below:

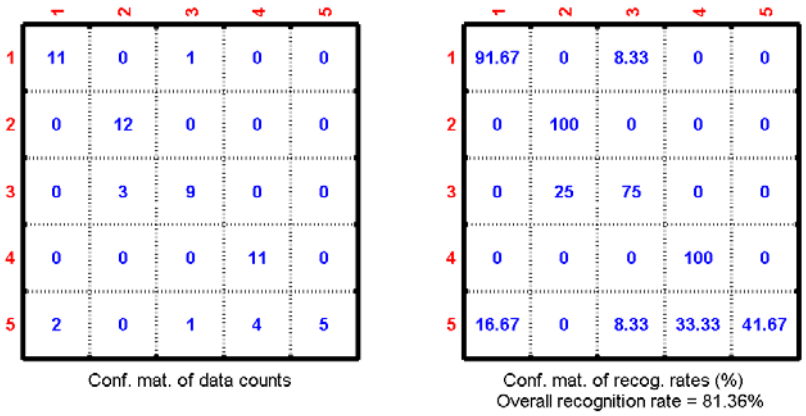


d. Use `knnrLoo.m` (in DCPR Toolbox) to compute the leave-one-our recognition rate when we use MFCC to classify each frame into 5 classes of different vowels. In particular, we need to change the dimension of the feature from 1 to 12 and plot the leave-one-out recognition rates using KNNR with k=1. What is the maximum recognition rate? What is the corresponding optimum dimension? Your plot should be similar to the next one:

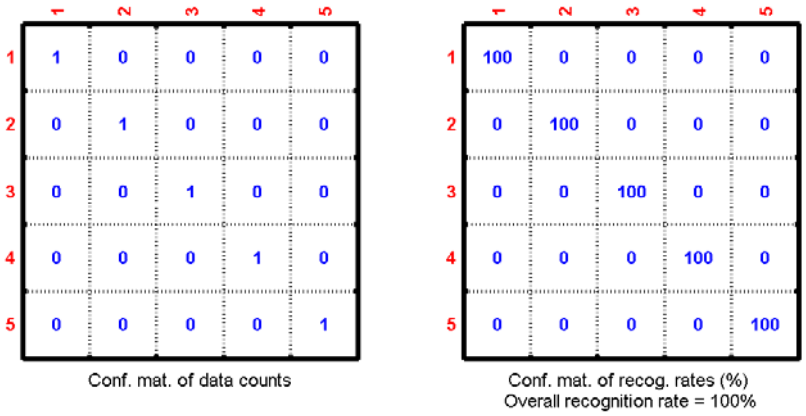


e. Record another clip of the same utterance and use it as the test data. Use the original clip as the train data. Use the optimum dimension in the previous subproblem to compute the the frame-based recognition rate of KNNR with k=1.

What is the frame-based recognition rate? Plot the confusion matrix, which should be similar to the following figure:

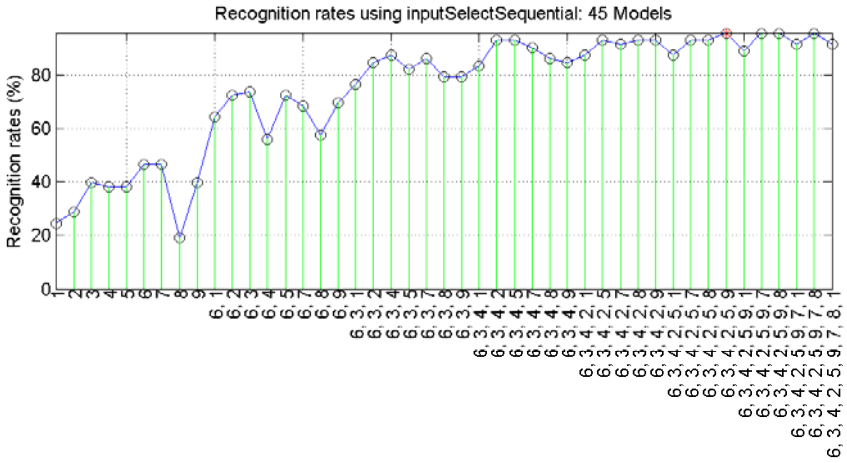


f. What is the vowel-based recognition rate? Plot the confusion matrix, which should be similar to the following figure:



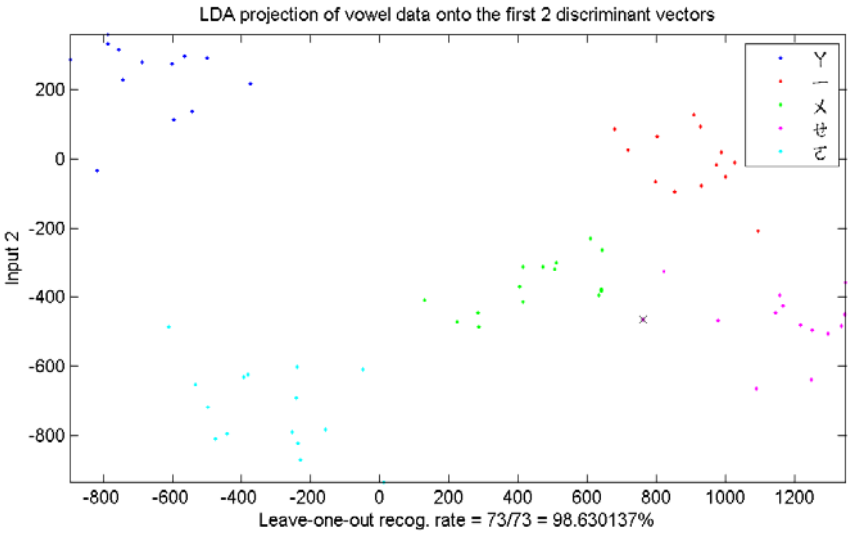
(Hint: The vowel-based recognition result is the voting of frame-based results. You can use `mode` to compute the result of voting.)

g. Perform feature selection based on sequential forward selection to select up to 12 features. Plot the leave-one-out recognition rates with respective to the selected features. Your plot should be similar to the next one:



What is the maximum recognition rate? What features are selected?

h. Use LDA to project 12-dimentional data onto 2D plane and plot the data to see if the data has the tendency of natural clustering based on their classes. Your plot should be similar to the next one:



What is the LOO recognition rate after LDA projection?

i. Repeat the previous sub-problem using PCA.

Chapter 13: Speaker Recognition (語者辨識)

第 13 章作業

1. (***) **Programming contest: use GMM for speaker identification**： Please see the detailed descriptions。

Programming Contests for Audio Signal Processing

Roger Jang（張智星）

本作業的目的在使同學瞭解如何以 GMM（Gaussian Mixture Model）的方法來進行語者辨識（Speaker Identification），並嘗試如何調整各種參數來提高辨識率。

1. 需要下載的資料：
   - exampleProgram.rar：所有的範例程式碼。
   - 語音訓練資料為各位同學所錄製的唐詩三百首，下載網址會由助教通知。
2. 如何執行範例程式：
   - 先修改 go.m 裡面的 waveDir 變數，將其設定為錄音檔按所在之目錄。
   - 在 MATLAB 輸入 go 即可。此範例共用了每個人的十句話，單數句（共 5 句）用來訓練 GMM，雙數句（共 5 句）用來測試用，inside test 辨識率約 100.00%，outside test 辨識率約 99.00%。（若稍微修改 go.m，讓程式使用每個人的所有語料，單數句用來訓練 GMM，雙數句用來測試，則 inside test 辨識率約 98.63%，outside test 辨識率約 97.26%。）
3. 範例程式說明：
   - go.m：主程式，包含載入語音訓練資料、求取特徵向量、訓練 GMM 模型，以及計算 inside test 及 outside test 的辨識率。
   - wave2mfcc.m：將語音訊號轉換成 MFCC（Mel-frequency cepstral coefficients）的函數。（由此函數轉出的特徵向量共有 26 維，1-12 維是 MFCC，13 維是 log energy，14-25 維是 delta MFCC，26 維是 delta log energy。但實際轉出的參數是由 setFeatureParam.m 所控制的。）
   - gmmTrain.m 及 gmmEval.m：GMM 的訓練及計算。（這些函數是放在 dcpr 裡面。）
   - speakerID.m：利用訓練完的 GMM 來進行語者辨識，並算出辨識率。
4. 如何更改範例程式，以得到較好的辨識率(請參考 "Robust Text-Independent Speaker Identification using Gaussian Mixture Speaker Models")：
   - 為了展示方便，go.m 只有採用少數語句，若要測試所有語句，運算時間就會變長很多，所以你要找一台快速的電腦。
   - 在端點偵測方面，可將含有 silence 的音框都拿掉，甚至連氣音都應該拿掉。
   - 在辨識參數方面，可以嘗試：
     - 改變 Gaussian component 的個數。（但不能盲目提高 Gaussian component 的個數，否則會發生 inside test 的辨識率極高，但 outside test 的辨識率卻極低的情況。）
     - 改變不同的 k-means 的啟始方法，使得 GMM 的訓練能夠得到較好的啟始點。
     - 改用 center-splitting 的 VQ 方法（vqLbg.m）。
     - 提高 gmmTrain 的迴圈次數，看是否能得到更大的 log probability。
     - 設法不讓 variance 變的太小。
   - 在特徵向量擷取方面，可以進行下列事項：
     - 採用 HTK 的 feature extraction 函數（htkWave2mfccMex.dll）。
     - 只取用 2-12 維的 MFCC。
     - Cepstral mean normalization
     - Difference coefficients

        - Frequency warping
    - 在加速計算方面：
        - 可以將呼叫 vqKmeans.m 之處改為呼叫 vqKmeansMex.dll，將會使速度加快。
        - 若嫌 gmmTrain.m 太慢，你也可以自己寫一個 C 的版本。
5. 效能測試：助教會進行 inside test 和 outside test，以兩者之平均值做為效能指標。
6. 應該上傳的檔案：
    - gmmParam.mat：GMM 的參數檔案
    - 若有修改 wave2mfcc.m，請一併上傳。
    - method.txt：簡要說明你的方法
    - 任何其他你修改過的程式。

第 13 章作業

1. (***) **Programming contest: use GMM for speaker identification**： Please see the detailed descriptions。

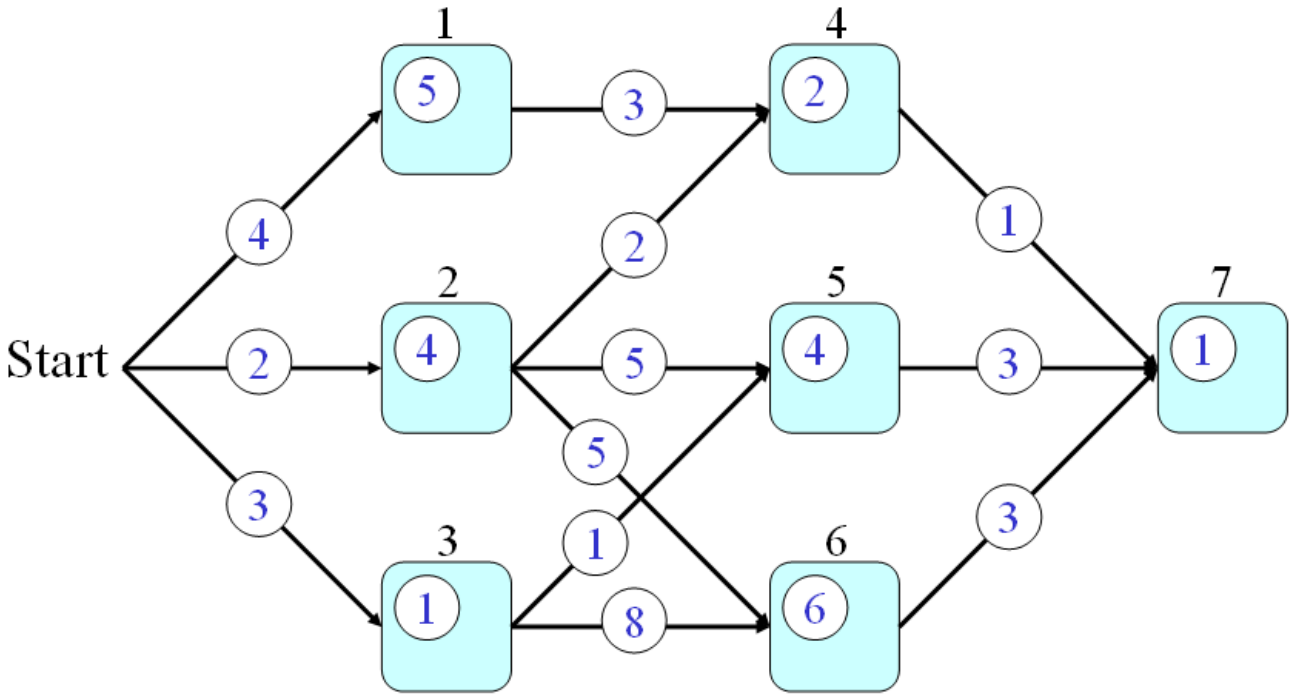
Chapter 14: Dynamic Programming

「動態規劃」（Dynamic Programming，簡稱 DP）是一個很有效的方法來求得一個問題的最佳解，DP 的精神是來自於 Richard Bellman 所提出的 Principle of Optimality：

An optimal policy has the property that whatever the initial state and the initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

簡單地說，就是在一條最佳路徑上，其中任一條子路徑（Partial Path）也都必須是相關子問題的最佳路徑，否則原路徑就不是最佳路徑。這是一個很明顯的道理，可以很直覺地使用矛盾法來加以證明，可是在實際應用上面，可以有許多複雜的變形。

舉例來說，以下列的圖形為例：



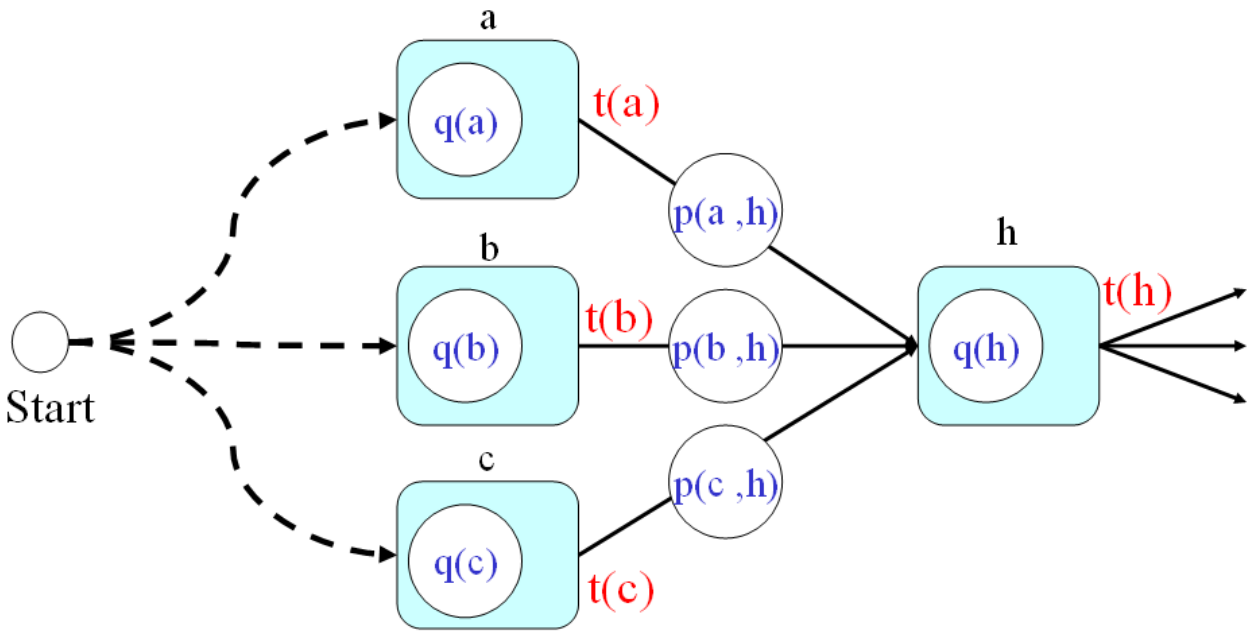
假設：

- 每一條連結（Link）代表一條公路，連結上的數字則代表通過此公路所需的時間。我們使用 p(a, b) 來代表通過連接 Node a 和 Node b 的公路所需的時間。
- 每一個節點（Node）代表一個城市，節點上的數字則代表通過此程式所需的時間。我們使用 q(a) 來代表通過 Node a 所需的時間。

假設起點是 Start，終點是 Node 7，請問我們如何找到一條路徑，使得從起點到終點所花的時間最短？這是一個很典型的 DP 問題，我們可以定義一個函數 t(h)，代表從 Start 到 Node h （需通過 Node h）的最短時間，此時 t(h) 滿足下列遞迴關係式：

$$t(h) = \min\{t(a)+p(a,h),\ t(b)+p(b,h),\ t(c)+p(c,h)\} + q(h)$$

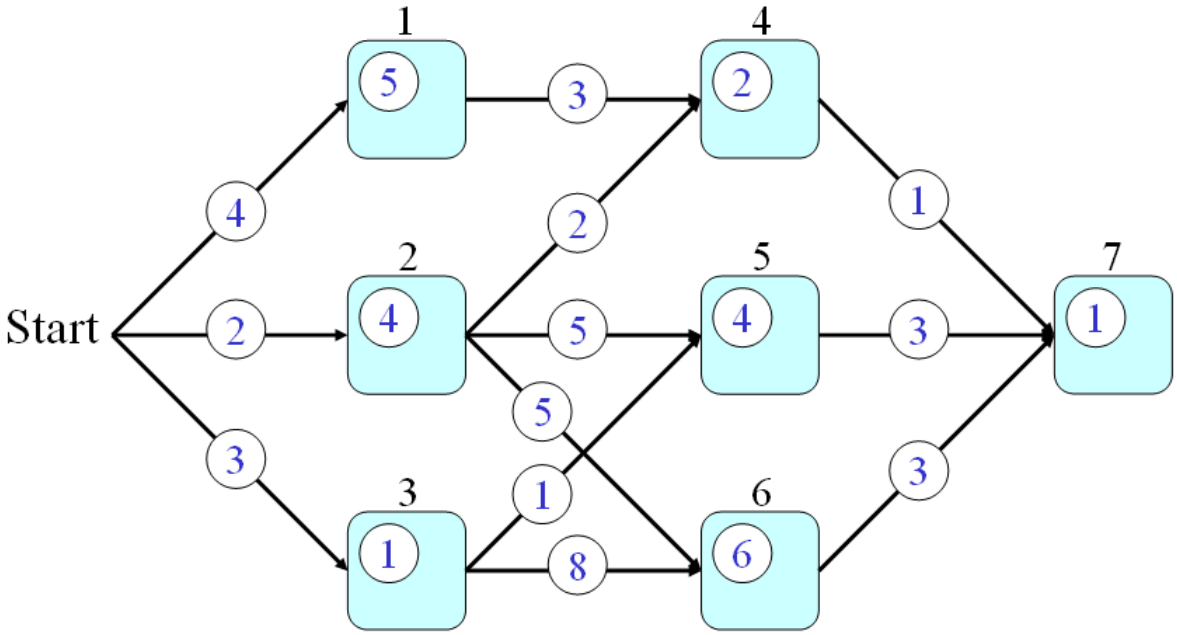
在上述方程式中，我們假設連接到 Node h 的節點有三個，分別是 a, b, c。示意圖如下：



因此，我們計算的次序如下：

1. t(1) = 4+5 = 9
2. t(2) = 2+4 = 6
3. t(3) = 3+1 = 4
4. t(4) = min(9+3, 6+2)+2 = 10
5. t(5) = min(6+5, 4+1)+4 = 9
6. t(6) = min(6+5, 4+8)+6 = 17
7. t(7) = min(10+1, 9+3, 17+3)+1 = 12

示意圖如下：



Hint

你可以點選上圖，就可以使用滑鼠點選，看到 DP 逐步計算的結果。

換句話說，在我們計算的過程中，可以反覆利用先前所計算出來的結果，而省掉許多不必要的運算。

DP 的計算，具有下列特性：

1. 在計算 DP 的過程中，必須在每個節點記錄最佳路徑的來源，才能在走到終點後，經由回溯（Back Tracking）來找到整個過程的最佳路徑。
2. 若要尋求第二最佳路徑，則必須使用較複雜的 Top-N 計算方法，而不能只靠上述簡單的 DP 算法。

LCS 是目標是要找出在兩個字串中，共同出現且前後次序一致的子字串。假設 LCS(**a**, **b**) 是字串 **a** 和 **b** 的最長共同子字串的長度，則我們可以使用 DP 的方法來進行 LCS 的計算，遞迴式如下：

1. 若 x=y，則 LCS(**a**x, **b**y) = LCS(**a**, **b**)+1
2. 若 x≠y，則 LCS(**a**x, **b**y) = max(LCS(**a**x, **b**), LCS(**a**, **b**y))

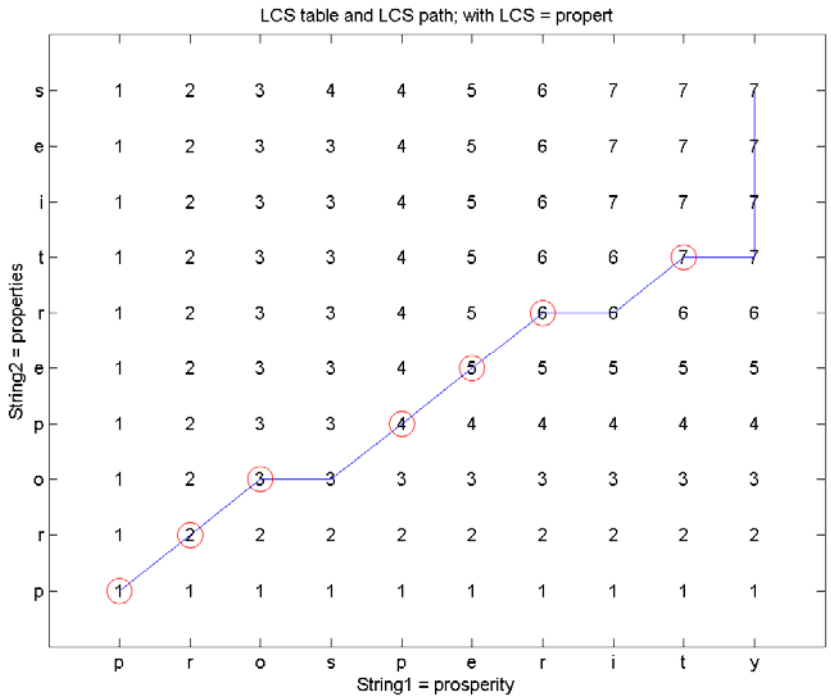
相關的邊界條件則是：LCS(**a**, []) = 0, LCS([], **b**) = 0。

以下是 LCS 的範例：

Example 1**Input file** dp/lcs01.m

```
str1 = 'prosperity';
str2 = 'properties';
plotOpt = 1;
[lcscount, lcsPath, lcsStr, lcsTable] = lcs(str1, str2, plotOpt);
```

Output figure



ED 另一種計算計算兩個字串之間距離的方法，其目標是在計算第一個字串要經過多少次的刪除（Delete）、插入（Insert）、代換（Substitute）等三個基本字串運算，才能得到第二個字串。假設 ED(**a**, **b**) 是字串 **a** 和 **b** 的編輯，則我們還是可以使用 DP 的方法來進行 ED 的計算，遞迴式如下：

1. 若 x=y，則 ED(**a**x, **b**y) = ED(**a**, **b**)
2. 若 x≠y，則 ED(**a**x, **b**y) = min(ED(**a**, **b**), ED(**a**x, **b**), ED(**a**, **b**y))

相關的邊界條件則是：ED(**a**, []) = len(**a**), ED([], **b**) = len(**b**)。

Hint

一般我們在 DOS 使用 fc 或是在 UNIX 使用 diff 來比較兩個文字檔案的不同處，就是使用 ED 來進行快速比對。

DTW 是 Dynamic Time Warping 的簡稱，中文可以翻譯成「動態時間扭曲」或是「動態時間校正」，這是一套根基於「動態規劃」（Dynamic Programming，簡稱 DP）的方法，可以有效地將搜尋比對的時間大幅降低。

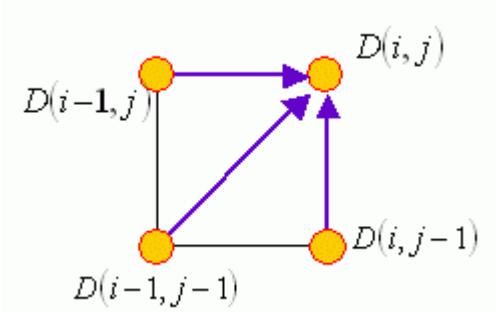
DTW 的目標就是要找出兩個向量之間的最短距離。一般而言，對於兩個 n 維空間中的向量 **x** 和 **y**，它們之間的距離可以定義為兩點之間的直線距離，稱為歐基里得距離（Euclidean Distance）：

$$dist(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}|^2$$

但是如果向量的長度不同，那它們之間的距離，就無法使用上述的數學式來計算。一般而言，假設這兩個向量的元素位置都是代表時間，由於我們必須容忍在時間軸的偏差，因此我們並不知道兩個向量的元素對應關係，因此我們必須靠著一套有效的運算方法，才可以找到最佳的對應關係。

假設有兩個向量 **t** 和 **r**，長度分別是 m 和 n，那麼 DTW 的目標，就是要找到一組路徑 $(p_1, q_1), (p_2, q_2), ..., (p_k, q_k)\}$，使得經由上述路徑的「點對點」對應距離和 $\Sigma_{i=1}^{k}$ ｜t(p_i) - r(q_i)｜ 為最小，而且，此路徑必須滿足下列條件：

- 端點關係：$(p_1, q_1) = (1, 1), (p_k, q_k) = (m, n)$。此端點關係代表這是「頭對頭、尾對尾」的比對。
- 局部關係：假設最佳路徑上任一點可以表示成 (i, j)，那麼其前一點路徑只有三種可能：(i-1, j), (i, j-1), (i-1, j-1)。此局部關係定義了路徑的連續性，而且也規定了 **t** 的任一個元素至少對應一個 **r** 的元素，反之亦然。



但是，我們要如何很快地找到這條最佳路徑呢？我們可以根據 DP 的原理，來將 DTW 描述成下列四大步驟：

1. 目標函數之定義：定義 D(i, j) 是 t(1:i) 和 r(1:j) 之間的 DTW 距離，對應的最佳路徑是由 (1, 1) 走到 (i, j)。
2. 目標函數之遞迴關係：D(i, j) = ｜t(i) - r(j)｜ + min{D(i-1, j), D(i-1, j-1), D(i, j-1)}
3. 端點條件：D(1, 1) = ｜t(1) - r(1)｜
4. 最後答案：D(m, n)

在上述的方法描述中，我們是有點濫用數學符號，嚴格地說，D(i, j) 應該表示成為 D(**t**(1:i), **r**(1:j))，才能準確地描述 D(·,·) 和 **t**、**r** 的關係。另外，D() 具有下列對稱的性質：D(**t**(1:i), **r**(1:j)) = D(**r**(1:j), **t**(1:i))。

在實際運算時，我們通常事先建立一個矩陣 D，其維度為 m×n，先根據端點條件來填入 D(1, 1)，然後再根據遞迴關係，逐行或逐列算出 D(i, j) 的值，最後就可以得到我們所要的答案 D(m, n)。

如果我們除了要知道 DTW 距離之外，也希望把相關最佳路徑找出來，此時在計算遞迴關係式時，就要記錄每一個最小點所對應的路徑，直到我們求出 D(m, n)，在反覆回推前一個最佳路徑的位置，如此一再反覆，才能算出整個最佳路徑，這個步驟在 DP 裡面稱為 Back Tracking。

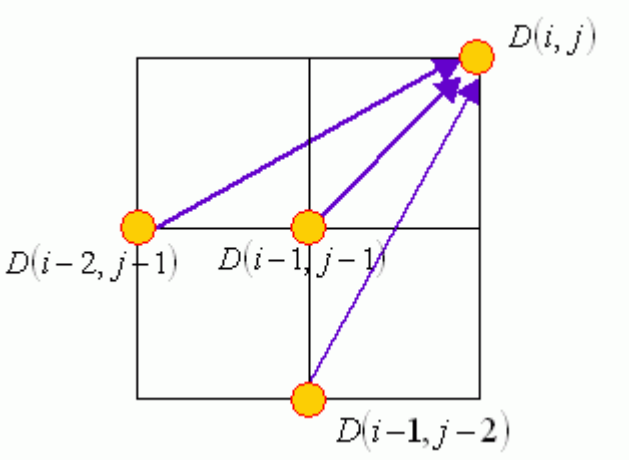
有上述方法可得知，DTW 的計算複雜度大約是 m×n，比用暴力法是有效率多了。

此外，在上述方法中，我們定義的子路徑是由 (1, 1) 走到 (i, j)，我們也可以反向操作，定義子路徑是由 (i, j) 走到 (m, n)，此時對應的 DTW 解法可以描述成下列四大步驟：

1. 目標函數：定義 D(i, j) 是 t(i:m) 和 r(j:n) 之間的 DTW 距離，對應的最佳路徑是由 (i, j) 走到 (m, n)。
2. 遞迴關係：D(i, j) = ｜t(i) - r(j)｜ + min{D(i+1, j), D(i+1, j+1), D(i, j+1)}
3. 端點條件：D(m, n) = ｜t(m) - r(n)｜
4. 最後答案：D(1, 1)

有這個方法所得到的結果，應該是和前述的方法完全相同。

另一個常用到的 local path constraint，是 27°-45°-63°，如下圖所示：

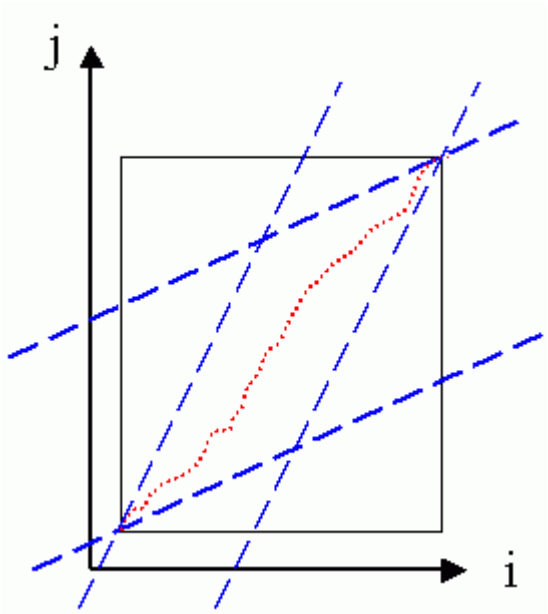


此種 local path constraint，會有下列特色：

- 會「跳點」，因此若有一點雜訊，最佳的 DTW 路徑將會跳掉此點。

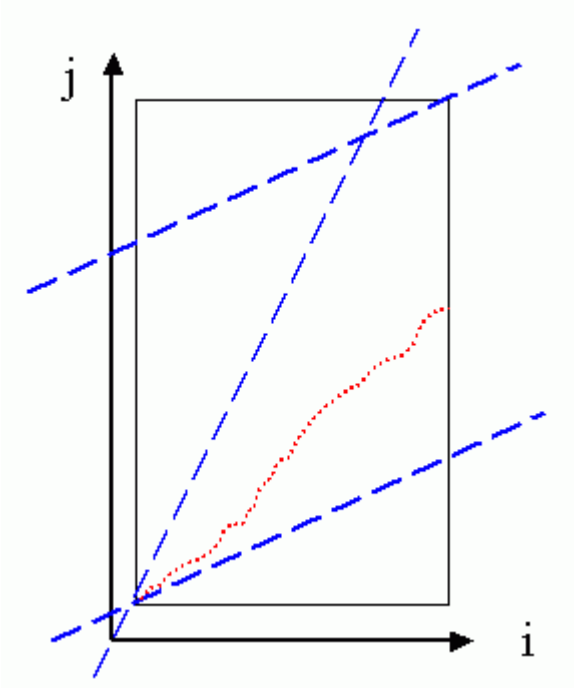
- 如果最佳路徑是對應於總距離的最小值，那麼最佳路徑會盡量走 27 或 63 度，以使對應到的點數降低。

如果我們要求是「頭對頭、尾對尾」的比對，那麼由上述的 local path constraint，我們就可以推斷出 global path constraint，如下所示：



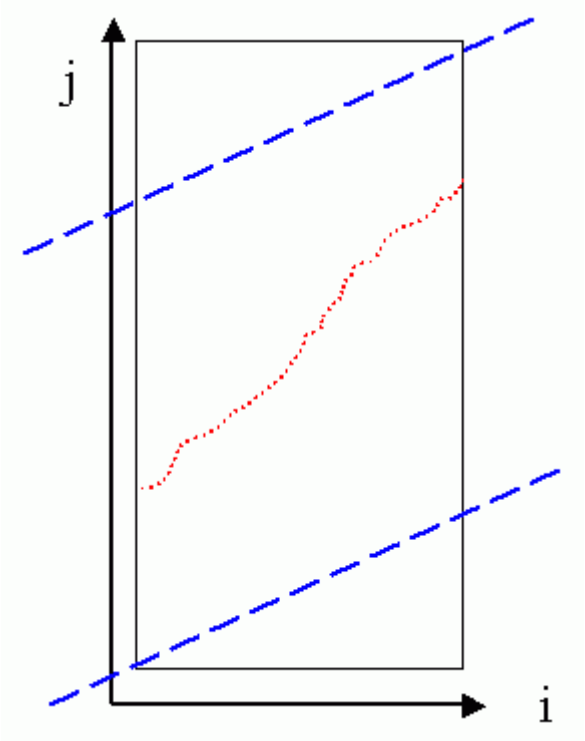
換句話說，若要使 DTW 的比對有意義，那麼兩者長度的比值必須介於 0.5 和 2.0 之間，否則我們就不可能找出一條合法的路徑。如果我們事先能夠定義 global path constraint，那麼在進行 DTW 計算時，就可以省掉很多不需要計算的部分。

如果我們要求的是「頭對頭、尾自由」的比對，例如用在「哼唱選歌」的「從頭比對」時，那麼對應的 global path constraint 如下：



這時候可以省略的計算，就會變少。

如果我們要求的是「頭尾都自由」的比對時，對應的 global path constraint 如下：



很明顯的，可以省略的計算，就更少了。
相關投影片如下：

- 旋律辨識所用的 DTW
- 語音辨識所用的 DTW

第 14 章作業

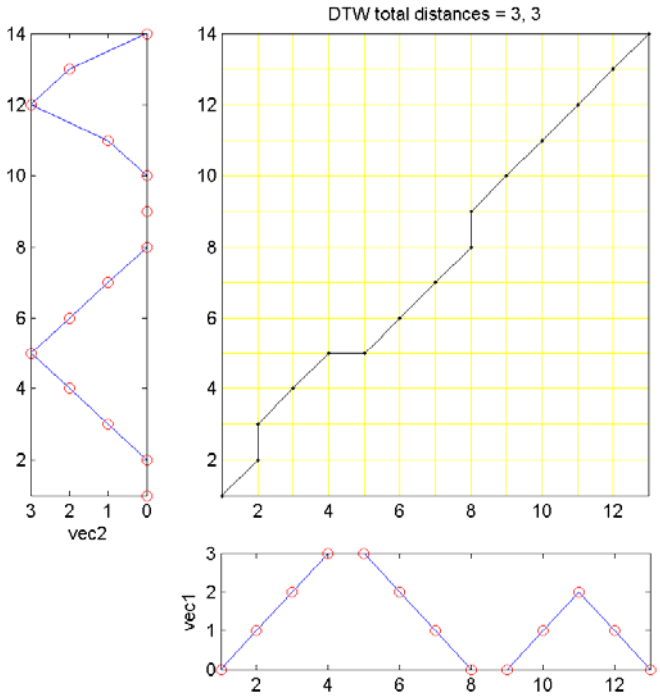
1. (***) **Function for edit distance**： Please write an m-file function editDistance.m for computing edit distance, with the usage:
   [minDist, edPath, edTable] = editDistance(str1, str2)
2. (***) **MATLAB function for type-2 DTW**： Write an m-file function dtw2m.m that is equivalent to dtw2mex.dll (or dtw1mex.mexwin32) in the DCPR toolbox. The usage is
   [minDistance, dtwPath, dtwTable] = dtw2m(vec1, vec2);
   You can use the following example as a script to test your program:
   Example 1**Input file** dp/dtw2test01.m

```
vec1=[0 1 2 3 3 2 1 0 0 1 2 1 0];
vec2=[0 0 1 2 3 2 1 0 0 0 1 3 2 0];
[minDist1, dtwPath1, dtwTable1] = dtw2m(vec1, vec2);
[minDist2, dtwPath2, dtwTable2] = dtw2mex(vec1, vec2);
fprintf('minDist1=%g, minDist2=%g, error=%g\n', minDist1, minDist2, abs(minDist1-minDist2));
dtwPlot(vec1, vec2, dtwPath1, dtwPath2);
```

   Output message

```
minDist1=3, minDist2=3, error=0
```

   Output figure



For a more extensive test with timing measurement, try the following example:
Example 2**Input file** dp/dtw2test02.m

```
% Compare the speed/distance difference between various versions of type-1 DTW

% ====== Output test
beginCorner=1; endCorner=1;
testNum=100;
fprintf('%d runs of output tests:\n', testNum);
for i=1:testNum
   vec1=randn(1, 25);
   vec2=randn(1, 30);
   [minDist1, dtwPath1, dtwTable1] = dtw2m(vec1, vec2, beginCorner, endCorner);
   [minDist2, dtwPath2, dtwTable2] = dtw2mex(vec1, vec2, beginCorner, endCorner);
   if ~isequal(minDist1, minDist2) | ~isequal(dtwPath1, dtwPath2) | ~isequal(dtwTable1,
dtwTable2)
                figure('name', 'dtw2m()');    dtwplot(vec1, vec2, dtwPath1);
                figure('name', 'dtw1mex()'); dtwplot(vec1, vec2, dtwPath2);
                fprintf('Press any key to continue...\n'); pause
   end
end
fprintf('\tDone!\n');

% ====== Speed test
testNum=200;
mat1=randn(testNum, 25);
mat2=randn(testNum, 30);
fprintf('%d runs of timing tests:\n', testNum);
tic
```

```
for i=1:testNum, dtw2m(mat1(i,:), mat2(i,:)); end
time1=toc;
tic
for i=1:testNum, dtw2mex(mat1(i,:), mat2(i,:)); end
time2=toc;
fprintf('\ttime1=%g, time2=%g, ratio=time1/time2=%g\n', time1, time2, time1/time2);
```

Output message

```
100 runs of output tests:
  Done!
200 runs of timing tests:
  time1=1.50159, time2=0.0140599, ratio=time1/time2=106.8
```

For simplicity, you should only consider the following situations:

- You only need to consider the case of "anchored beginning and anchored end".
- You need to add the global constraints where the feasible region is a parallelogram.
- You can assume the inputs are row vectors.

The behavior of dtw2mex.dll should be exactly the same as those described in the text. If you find any discrepancies, feel free to let me know.

3. (***) **Implementation of a speaker-dependent voice-command system using DTW**：In this exercise, you are going to use MATLAB to implement a speaker-dependent voice-command system. Basically, the user can record a 3-second utterance and the system will identify the most likely command via DTW. To create such a system, we have two stage for registration and recognition, respectively. During the **registration stage**, we need to prepare a database where the user can register their utterances for recognition. This involves the following steps:
   i. Prepare a text file "command.txt" manually. The file should contains at least 10 commands. An English example follows:

```
one
two
three
five
...
```

   You can also use the following Chinese example:

```
牛肉麵
珍珠奶茶
貢丸湯
韭菜盒子
肉粽
...
```

   (In fact, you can use any language you like to prepare the text file.)
   vii. Write a script for the users to record these commands twice. To increase the variability, you should record each command once in a run, for two runs. (You can use "textread" command to read the text file.)
   viii. After each recording, you should perform endpoint detection and feature extraction, and store all the information in a structure array `recording` of size 2*n, where n is the number of voice comands in "command.txt". The structure array `recording` contains the following fields:
      - recording(i).text: Text for the i-th recording

- recording(i).mfcc: MFCC for the i-th recording

ix. Save the variable `recording` to `recording.mat`.

During the **recognition stage**, the user can hit any key to start 3-second recording, and the system will demonstrate the top-10 results on the screen according to the DTW distance. The recognition stage invloves the following steps:

x. Load `recording.mat`.
xi. Obtain the user's 3-sec utterance.
xii. Perform endpoint detection and feature extraction.
xiii. Compare the MFCC with that of each recording in the array `recording` using DTW.
xiv. Rank the distance.
xv. List the top-10 result (by showing the texts) according to the distance.
xvi. Go back to step ii unless ctrl-c is hit.

Some hints follow:

- All recordings are of the format: 16KHz, 16Bits, mono.
- The system should allow the user to do repeated recordings until ctrl-C is hit.
- You can use epdByVolHod.m for endpoint detection. After each recording, the result of endpoint detection should be displayed on the screen immediately to facilitate debugging. (It is in the Audio Processing Toolbox.)
- You can use wave2mfcc.m for feature extraction. (It is in the Audio Processing Toolbox.)
- You can use either dtw1mex.dll or dtw2mex.dll for DTW. (Both are in the DCPR toolbox.)

When you demo your system to TA, make sure you can have at least 3 utterances to have the correct answer in top-1 results.

4. (***) **Programming contest: Use DTW for speaker-dependent speech recognition**：
Please follow this link to have more descriptions.

Chapter 15: Hidden Markov Models (HMM)

前一章所介紹的 DTW，大部分是用於語者相關（Speaker Dependent）的語音辨識，這一類的應用大部分需要使用者自行錄音，然後再以自己的聲音來比對之前錄好的語音資料，雖然在概念上很簡單，但是此方法比較適合同一位語者的聲音來進行比較，因此應用範圍比較狹隘，譬如目前手機 Name Dialing 等等。但是如果我們要做到語者無關（Speaker Independent）的語音辨識，最常見的方法，就是「隱藏式馬可夫模型」（Hidden Markov Models），簡稱 HMM。

HMM 是根基於統計的機率模型，特別適用於具有大量訓練資料的語音辨識系統，若以機率模型的類別來分，HMM 又可以分成兩大類：

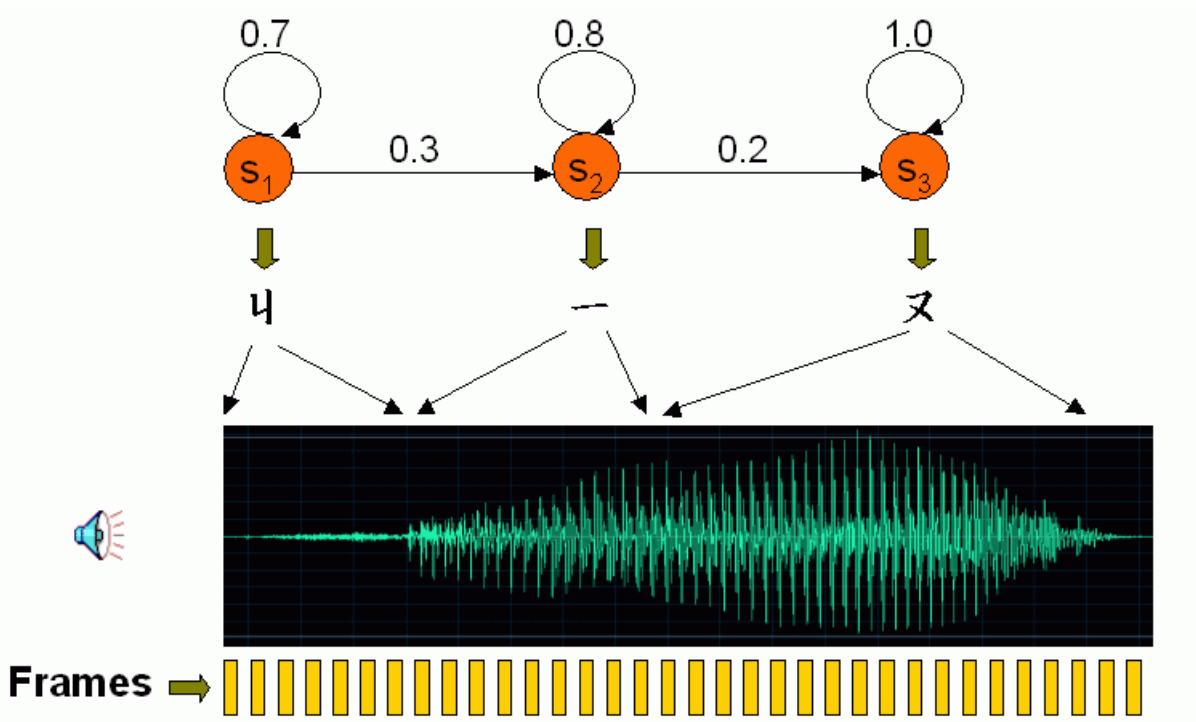
- Discrete HMM：簡稱 DHMM，其中機率的算法完全是依靠查表法，而表格的取得則是靠大量語音資料的統計而得。
- Continuous HMM：簡稱 CHMM，其中機率的算法是根據連續的機率密度函數，例如 Gaussian Mixture Models（GMM）等等。這些機率密度函數都是靠大量的語音資料來建模（Modeling）所得，所用的方法大部分是 Maximum Likelihood Estimate（MLE）。

DHMM 是 Discrete Hidden Markov Model 的簡稱，對語音辨識的應用而言，我們的目標，就是事先使用大量語料，對每一個辨識語句建立一個 DHMM，然後當我們取得使用者輸入的測試語句後，即對測試語句進行端點偵測及 MFCC 特徵擷取，然後再將 MFCC 送至各個辨識語句所形成的 DHMM，算出每一個模型的機率值，機率值最大的 DHMM，所對應的辨識語句就是此系統的辨識結果。

為方便說明，我們假設我們的目標示要建立一個 0 ~ 9 的數字語音辨識系統，建立流程可以分成兩大步驟：

1. 收集語料：由於我們的目標是語者無關的語音辨識，所以我們所收集的語料要盡量涵蓋各種可能的變化，例如：
    - 錄音的人：各種不同的人，包含男女老少、高矮胖瘦等，以及各種南腔北調。
    - 錄音裝置：取用不同的裝置，例如不同的麥克風、不同的音效卡等。
    - 周遭環境：除了要在安靜的辦公室進行錄音外，也可以在吵雜的環境錄音，以增強語音辨識系統對於雜訊的抵抗力。
2. 辨識系統的設計和測試：這也是我們下面的重點。

在進行 DHMM 的設計時，我們必須對每一個數字建立一個 HMM 模型，以數字「九」的發音為例，相關的 HMM 模型可以圖示如下：



換句話說，我可以把「九」的發音切分成三個「狀態」（States），分別是代表 ㄐ、ㄧ、ㄡ 的發音，每一個狀態可以包含好幾個音框，而每一個音框隸屬於一個狀態的程度，是用一個機率值來表示，稱為「狀態機率」（State Probability）。此外，當一個新的音框進來時，我們可以將此音框留在目前的狀態，也可以將此音框送到下一個狀態，這些行為模式也都是用機率來表示，統稱為「轉移機率」（Transition Probability），其中我們使用「自轉移機率」（Self-transition Probability）來代表新音框留在目前狀態的機率，而用「次轉移機率」（Next-transition Probability）來代表新的音框跳到下一個狀態的機率。

由於每一個音框都會轉成一個語音特徵向量，我們首先使用 VQ 來將這些特徵向量轉成符號（Symbols），換句話說，每一個音框所對應的 VQ 群聚的索引值（Index），即是此音框的符號。若以數學來表示，k = O(i) 即代表 frame i 被分到 cluster k。

為了簡化討論，首先我們定義一些常用的數量：

- frameNum：一段語音所切出的音框個數。
- dim：每一個音框所對應的語音特徵向量的維度（例如基本的 MFCC 有 12 維）。
- stateNum：一個 DHMM 的狀態個數
- symbolNum：符號的個數，也就是 VQ 後的群數

HMM 的參數，就是指「狀態機率」和「轉移機率」，說明如下：

- 我們通常以矩陣 A 來表示轉移機率，其維度是 stateNum x stateNum，其中 A(i, j) 即是指由狀態 i 跳到狀態 j 的機率值。例如在上圖中，由狀態 1 跳到狀態 2 的機率是 0.3，因此 A(1, 2) = 0.3。一般而言，A(i, j) 滿足下列條件：
    - 在進行次轉移時，我們只允許搜尋路徑跳到下一個相鄰的狀態，因此 A(i, j) = 0 if j≠i and j≠i+1。

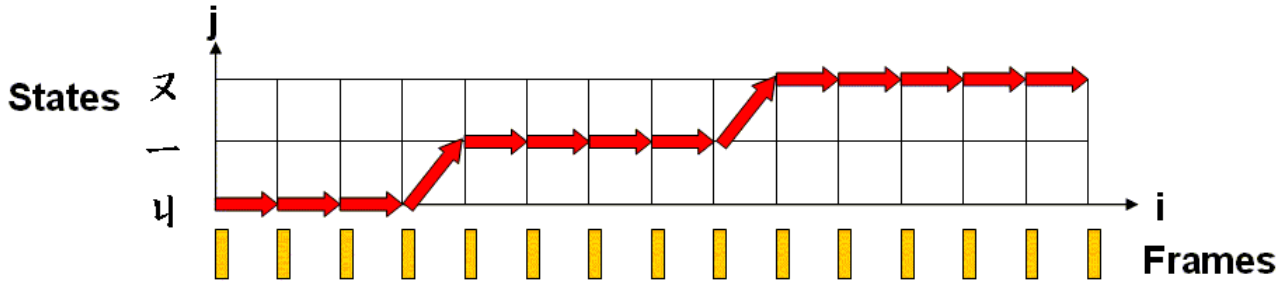
- 對於某一個狀態而言，所有的轉移機率總和為 1，因此 A(i, i) + A(i, i+1) = 1。

- 我們通常以矩陣 B 來表示狀態機率，其維度是 symbolNum x stateNum，B(k,j) 即是指符號 k 隸屬於狀態 j 的機率值。換句話說，B 定義了由「符號」到「狀態」的機率，因此給定第 i 個音框，我們必須先由 O(i) 找出這個音框所對應的符號 k，然後再由 B(k,j) 找出此音框屬於狀態 j 的機率。

假設我們已經知道「九」的 HMM 所包含相關的參數 A 和 B，此時當我們錄音得到一段語音，我們如何算出來此語音隸屬於「九」的機率或程度？換句話說，我們如何將每個音框分配到各個狀態之中，使得我們能夠得到整段語音最高的機率值？最常用的方法稱為 Viterbi Decoding，這是根基於「動態規劃」（Dynamic Programming）的方法，可用數學符號定義如下：

1. 目標函數：定義 D(i, j) 是 t(1:i) 和 r(1:j) 之間的最大的機率，t(1:i) 是前 i 個音框的特徵向量所成的矩陣，r(1:j) 則是由前 j 個狀態所形成的 DHMM，對應的最佳路徑是由 (1, 1) 走到 (i, j)。
2. 遞迴關係：D(i, j) = B(O(i), j) + max{D(i-1, j)+A(j, j), D(i-1, j-1)+A(j-1, j)}
3. 端點條件：D(1, j) = π(1, j) + B(O(1), j), j = 1 ~ stateNum
4. 最後答案：D(m, n)

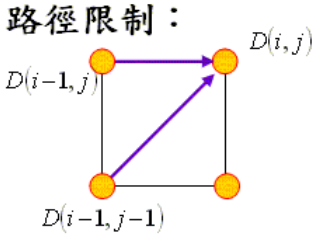
示意圖如下：



遞迴算式（based on log prob.）：

$$D(i,j)=B(O(i),j)+\max\begin{cases}D(i-1,j)+A(j,j)\\D(i-1,j-1)+A(j-1,j)\end{cases}$$

路徑限制：



請特別注意，在上述數學式中，我們所用的所有機率都是「對數機率」（Log Probabilities），所以原來必須連乘的數學方程式，都變成了連加，具有下列好處：

- 以加法來取代乘法，降低計算的複雜度。
- 避開了由於連乘所造成的數值誤差。

假設轉移機率 A 和狀態機率 B 已知，那麼經由 Viterbi Decoding，我們可以找到最佳路徑（即是最佳分配方式），使整段路徑的機率值為最大，此機率值及代表輸入語音隸屬於此 DHMM 的機率，若是機率越高，代表此輸入語音越可能是「九」。

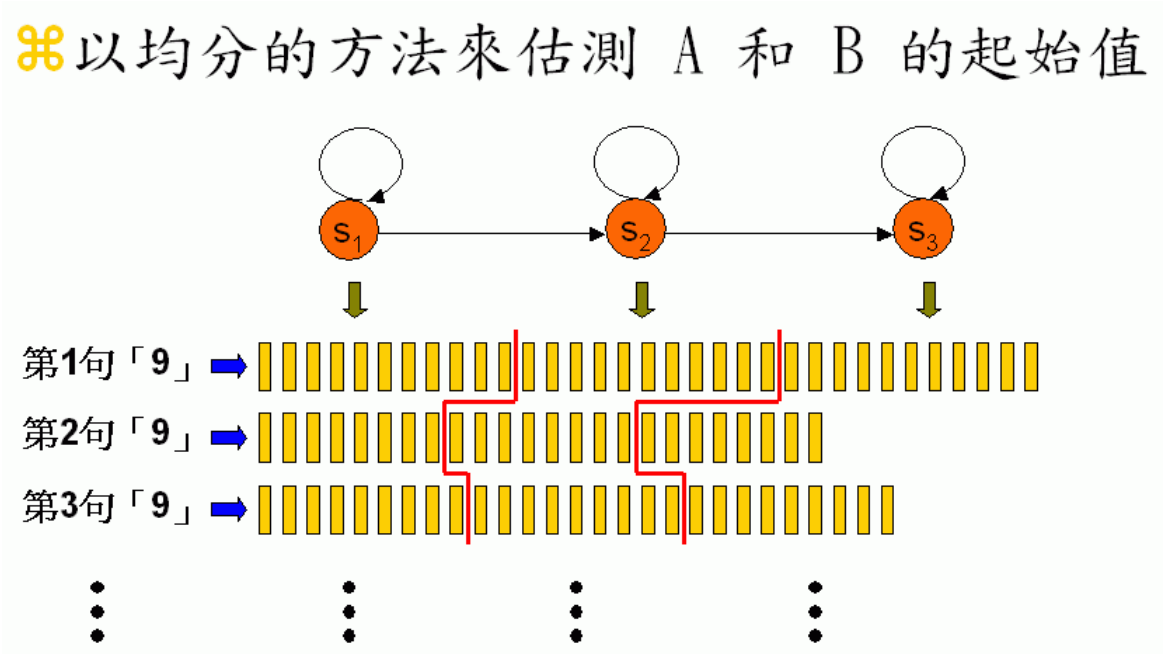
有關於 B(O(i),j)的定義，都是「音框 i 隸屬於狀態 j 的機率」，但是在 DHMM 和 CHMM 的算法差異很大，說明如下：

- 在 DHMM，若要計算 B(O(i),j)，我們必須先找到音框 i 所對應的符號 O(i)，然後在經由查表法查到此符號 O(i) 屬於狀態 j 的機率是 B(O(i), j)。
- 在 CHMM，B(O(i),j) 是由一個連續的機率密度函數所定義，請見下一小節說明。

但是，如何經由大量語料來估測每個 HMM 模型的最佳參數值 A 和 B 呢？首先，我們必須先定義什麼是「最佳參數」：對某一個特定模型而言，最佳參數值 A 和 B 應能使語料產生最大的對數機率總和。這個定義和 MLE 完全相同，也非常合理。

計算最佳參數的方法，稱為 Re-estimation，其原理非常類似 batch k-means (Forgy's method) 的方法，先猜 A 和 B 的值，再反覆進行 Viterbi Decoding，然後再重新估算 A 和 B 的值，如此反覆計算，我們可以證明在疊代過程中，機率總和會一路遞增，直到逼近最佳值。（但是，就如同 k-means Clustering，我們並無法證明所得到的機率值是 Global Maximum，因此在訓練的過程中，可以使用不同的啟始參數，看看是否在反覆疊代後，能夠得到更好的機率總和。）求取參數的方法說明如下：

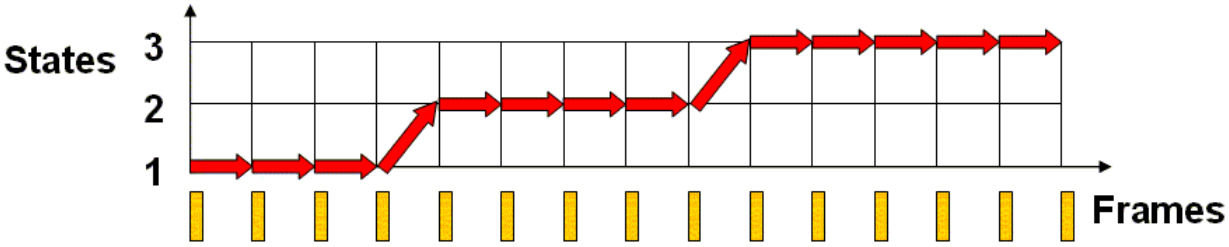
1. 將所有的訓練語句切出音框，並將每一個音框轉成語音特徵向量，例如 39 維的 MFCC。
2. 對所有語音特徵向量進行 VQ，找出每一個向量所對應的 symbol（此即為對應的中心點或 codeword）
3. 先猜一組 A 和 B 的啟始值。如果沒有人工的標示資料，我們可以使用簡單的「均分法」，示意圖如下：



4. 反覆進行下列兩步驟，直到收斂
   - Viterbi decoding：在 A 和 B 固定的情況下，利用 Viterbi decoding，找出 n 句「九」的語料所對應的 n 條最佳路徑。
   - Re-estimation：利用這 n 條最佳路徑，重新估算 A 和 B。

估算 A 的方法，可由下列示意圖說明：

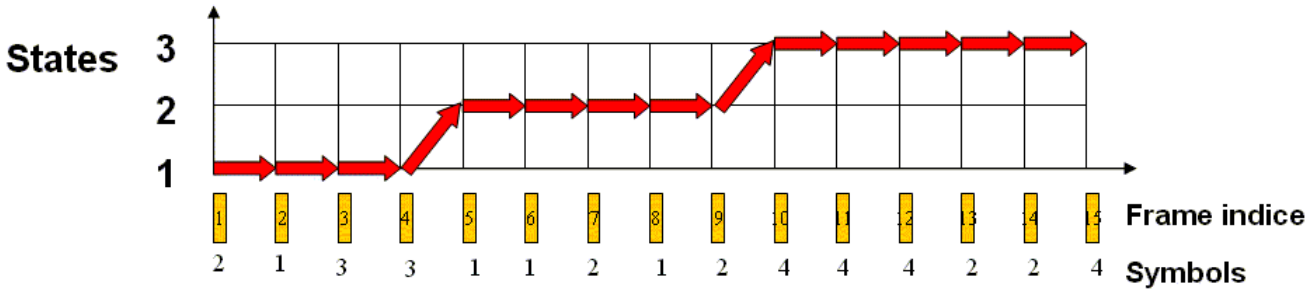
- 只估算 A(i, i) 和 A(i, i+1)，其餘為零。一條路徑的範例：
  - A(1, 1)=3/4, A(1, 2)=1/4
  - A(2, 2)=4/5, A(2, 3)=1/5
  - A(3, 3)=1



- 最後總結果是N條路徑（同屬這個模型）的統計結果

估算 B 的方法，可由下列示意圖說明：

- 欲計算B(i, j)，一條路徑的範例如下：
  - State 1: B(1, 1)=1/4, B(2, 1)=1/4, B(3, 1)=2/4
  - State 2: B(1, 2)=3/5, B(2, 2)=2/5
  - State 3: B(2, 3)=2/6, B(4, 3)=4/6
- 音框i所對應的符號是k=O(i)，因此此音框i屬於狀態j的機率是B(k, j)=B(O(i), j)。



- 最後總結果是N條路徑（同屬這個模型）的統計結果

如果我們使用 D(i,j) 代表音框 i 屬於狀態 j 的機率，則 D(i,j) = B(O(i),j)，針對上述範例中的一條路徑，我們可以得到：

- State 1: D(1,1)=B(2,1)=1/4, D(2,1)=B(1,1)=1/4, D(3,1)=B(3,1)=2/4, D(4,1)=B(3,1)=2/4
- State 2: D(5,2)=B(1,1)=3/5, D(6,2)=B(1,1)=3/5, D(7,2)=B(2,1)=2/5, D(8,2)=B(1,1)=3/5, D(9,2)=B(2,1)=2/5
- State 2: D(10,3)=B(4,3)=4/6, D(11,3)=B(4,3)=4/6, D(12,3)=B(4,3)=4/6, D(13,3)=B(2,3)=2/6, D(14,3)=B(2,3)=2/6, D(15,3)=B(4,3)=4/6

假設我們的目標函數可以寫成 P(A, B, Path)，則上述求參數的方法會讓 P(A, B, Path)逐次遞增，直到收斂，因為：

1. 在 A, B 固定時，Viterbi decoding 會找出最佳的路徑，使 P(A, B, Path) 有最大值

2. 在路徑（Path）固定時，Re-estimation 會找出最佳的 A, B 值以使 P(A, B, Path) 有最大值

上述第一點是無庸置疑，因為 Viterbi Decoding 會使每一條路徑的機率值為最大，因此其總和 P(A, B, Path) 也是最大。至於第二點的證明，也是非常直覺，可以說明如下。

以上述範例而言，此條路徑的總機率可以拆解成在三個 state 中的機率：

- State 1: D(1,1)A(1,1)D(2,1)A(1,1)D(3,1)A(1,1)D(4,1)A(1,2) = $A(1,1)^3A(1,2)D(1,1)D(2,1)D(3,1)D(4,1)$
- State 2: D(5,2)A(2,2)D(6,2)A(2,2)D(7,2)A(2,2)D(8,2)A(2,2)D(9,2)A(2,3) = $A(2,2)^4A(2,3)D(5,2)D(6,2)D(7,2)D(8,2)D(9,2)$
- State 3: D(10,3)A(3,3)D(11,3)A(3,3)D(12,3)A(3,3)D(13,3)A(3,3)D(14,3)A(3,3)D(15,3)A(3,3) = $A(3,3)^6D(10,3)D(11,3)D(12,3)D(13,3)D(14,3)D(15,3)$

對於 N 條路徑而言，所得到的總機率就是這 N 條路徑的機率乘積。因此若要求 A 和 B 的最佳值，我們必須同時考慮這 N 條路徑。

以下將求取 A 和 B 的最佳值。對任一個 state 而言，假設共有 a+b 個音框屬於這個 state，其中

- p: self-transition prob.（未知）
- q: next-transition prob.（未知）
- a: self-transition count（已知）
- b: next-transition count（已知）

則我們可以形成下列最佳化問題：

$$\text{Max } J(p, q) = p^a q^b \text{ s.t. } p \geqq 0, q \geqq, \text{ and } p + q = 1$$

由於算數平均數大於或等於幾何平均數：

$$x_1 + x_2 + ... + x_n \geqq n (x_1 x_2 ... x_n)^{1/n}$$

同時上述等式，只發生在 $x_1 = x_2 = ... = x_n$。利用此不等式，我們可以得到

$$p/a + p/a + ... + q/b + q/b ... \geqq (a + b) [(p/a)^a(q/b)^b]^{1/(a+b)}$$

$$1 \geqq (a + b) [(p/a)^a(q/b)^b]^{1/(a+b)}$$

因此 $J(p, q) = p^a q^b$ 的最大值是 $a^ab^b/(a+b)^{a+b}$，此最大值發生在 p/a = q/b，也就是 p=a/(a+b), q=b/(a+b)。

對任一個 state 而言，假設這個 state 內的音框只屬於三個 cluster，機率分別是 p, q, r:

- State prob: p, q, r（未知）
- Symbol count: a, b, c（已知）

則我們可以形成下列最佳化問題：

$$J(p, q, r) = p^a q^b r^c$$

其中 p, q, r 必須滿足 p + q + r = 1。同理，利用算數平均數大於或等於幾何平均數，我們可得到最佳參數值 p=a/(a+b+c), q=b/(a+b+c) , r=c/(a+b+c)。

本章節說明，請參考下列投影片：

- Discrete HMM
-

15-3 Continuous HMM

Once we grasp the concept of DHMM, it is straightforward to extend the concept to CHMM. The only difference between CHMM and DHMM is their representation of state probabilities:

- DHMM uses a VQ-based method for computing the state probability. For instance, frame i has to be converted into the corresponding symbol k=O(i), and the probability of symbol k to state j is retrieved from B(k, j)
- of the matrix B.

- CHMM uses a continious probability density function (such as GMM) for computing the state probability. In other words, B(O(i), j) in CHMM is prepresented by a continuous probability density function:

$$B(O(i), j) = p(x_i, \theta_j)$$

where $p(\cdot,\cdot)$ is a PDF (such as GMM), x_i is the feature vector of frame i, and θ_j is the parameter vector of this PDF of state j. The method for identifying the optimum of θ_j is based on re-estimation of MLE (Maximum Likelihood Estimate).
In summary, the parameters of CHMM can be represented by the matrix A and the parameters $\theta = \{\theta_j | j = 1\sim m\}$. The method for finding the optimum values A and θ is again re-estimation, in which we need to guess the initial values of A and θ, perform Viterbi Decoding, and then use the optimum mapping paths to compute A and θ again. This procedure is repreated until the values of A and θ converge. It can be proved that during the iteration, the overall probability is monotonic increasing. However, we cannot guarantee the obtained maximum is the global maximum.
The procedure for finding the parameters of CHMM is summarized next.
    1. Convert all utterances into acoustic features, say, 39-dimensional MFCC.
    2. Guess the initial values of A and θ. If there is no manual transcription, we can adopt the simple strategy of "equal division".
    3. Iterate the following steps until the values of A and θ converge.
        - Viterbi decoding: Given A and θ of a CHMM, find the optimum mapping paths of all the corresponding utterances of this CHMM.
        - Re-estimation: Use the optimum mapping paths to estimate the values of A and θ.

Note that in each iteration of the above procedure, optimum value of matrix A is identified by frame counting, which is the same as that used in DHMM. On the other hand, the optimum value of θ is identified via MLE. Since $p(\cdot,\cdot)$ in CHMM is a continuous function that can approximate the true PDF better, we can expect to achieve a better recognition rate than DHMM.

第 15 章作業

1. (*)**簡森不等式：數學推導一：** 對於 $0 \leqq \lambda \leqq 1$，請證明下列「簡森不等式」（Jensen's Inequality）：

$$\ln(\lambda x_1+(1-\lambda)x_2) \geqq \lambda \ln(x_1) + (1-\lambda)\ln(x_2)$$

2. (*)**簡森不等式：數學推導二：** 請用歸納法，證明下列一般化的「簡森不等式」（Jensen's Inequality）：

$$\ln(\Sigma_{i=1}^{n}\lambda_i x_i) \geqq \lambda_i \Sigma_{i=1}^{n} \ln(x_i)$$

其中 $0 \leqq \lambda_i \leqq 1$, i=1~n, 而且 $\Sigma_{i=1}^{n}\lambda_i = 1$。

3. (*)**算數平均數大於或等於幾何平均數：數學推導：** 請使用上一題的「簡森不等式」，證明「算數平均數大於或等於幾何平均數」，公式如下：

$$(\Sigma_{i=1}^{n} x_i)/n \geqq (\Pi_{i=1}^{n} x_i)^{1/n}$$

其中 $x_i \geqq 0$, i=1~n。

4. (*)**轉換機率最佳值：數學推導：** 請用直接微分的方式，求取下列函數的最大值，及所對應的 p, q 值：

$$J(p, q) = p^a q^b$$

其中 $0 \leqq p, q \leqq 1$ 且 $p+q=1$。

5. (**)**狀態機率最佳值：數學推導：** 請用 Lagrange's Multiplier 的方式，求取下列函數的最大值，及所對應的 p, q, r 值：

$$J(p, q, r) = p^a q^b r^c$$

其中 $0 \leqq p, q, r \leqq 1$ 且 $p+q+r=1$。

6. (**)**DHMM 最佳路徑：** 請寫一個函數 dhmmEval.m，來進行 DHMM 最佳路徑的計算，此函數的用法如下：

[maxLogProb, dpPath] = dhmmEval(initPI, A, B, O)

輸入參數的說明如下：

a. A 代表轉移機率，其維度是 stateNum x stateNum
b. B 代表符號對狀態的機率，其維度是 clusterNum x stateNum
c. O 代表每一個音框所對應的符號，其維度是 frameNum x 1

輸出參數的說明如下：

d. maxLogProb 是一個純量，最佳路徑的纍加對數機率。
e. dpPath 是一個向量，維度是 2 x frameNum，其中的每一個行向量即代表最佳路徑的一點，課文中的範例而言，dpPath = [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15; 1 1 1 1 2 2 2 2 2 3 3 3 3 3 3]。

請用下列程式骨架來撰寫你的程式：

原始檔（dhmmEval.m）：（灰色區域按兩下即可拷貝）

```
function [maxLogProb, dpPath, dpTable] = dhmmEval(initPI, A, B, O)
% dhmmEval: Evaluation of DHMM
%           Usage: [maxLogProb, dpPath, dpTable] = viterbiDecoding(initPI, A, B, O)
%                  initPI: Initial state log probability, 1 x stateNum
%                  A: Transition log probability, stateNum x stateNum
%                  B: State log probability, symbolNum x stateNum
%                  O: Observation sequence of this utterance, frameNum x 1
%                  maxLogProb: Maximum log probability of the optimal path
%                  dpPath: optimal path with size frameNum x 1

frameNum = length(O);
stateNum = length(A);
dpTable = -inf*ones(frameNum, stateNum);

if (stateNum>frameNum); error('Number of frames is less than the number of states!'); end

% ====== Fill the first row (matrix view)
dpTable(1,:)=initPI+B(O(1),:);
% ====== Fill the first column (matrix view)
for i=2:frameNum
  ...
end
% ====== Fill each row (matrix view)
for i=2:frameNum
  for j=2:stateNum
          ...
  end
end

% ====== Back track to find the optimum path
...
```

欲測試你的程式，請先下載 dhmmEvalMex.dll，再用下列程式 dhmmEvalTest.m 來測試你的結果：

原始檔（dhmmEvalTest.m）：（灰色區域按兩下即可拷貝）

```
% ====== Set up some parameters
frameNum=100;
stateNum=10;
symbolNum=64;
initPI=log([1, zeros(1, stateNum-1)]);
selfTransProb=0.85;
A=diag(selfTransProb*ones(stateNum,1));
A((stateNum+1):(stateNum+1):stateNum^2)=1-selfTransProb; A=A+eps;
A=log(A);
B=rand(symbolNum, stateNum); B=B*diag(1./sum(B));
B=log(B);
O=ceil(rand(frameNum,1)*symbolNum);

% ====== Start functionality and timing tests
n=100;
tic
for j=1:n
  [maxLogProb1, dpPath1, dpTable1] = dhmmEvalMex(initPI, A, B, O);
end
fprintf('dhmmEvalMex ==> %.2f, maxLogProb = %.9f\n', toc, maxLogProb1);
tic
for j=1:n
  [maxLogProb2, dpPath2, dpTable2] = dhmmEval(initPI, A, B, O);
end
fprintf('dhmmEval ==> %.2f, maxLogProb = %.9f\n', toc, maxLogProb2);

fprintf('Difference in maxLogProb = %g\n', abs(maxLogProb1-maxLogProb2));
fprintf('Difference in dpPath = %g\n', sum(sum(abs(dpPath1-dpPath2))));
fprintf('Difference in dpTable = %g\n', sum(sum(abs(dpTable1-dpTable2))));
```

若程式正確，則最後三列敘述所列印出的值應該都很小，小於 10 的 -6 次方。請問 dhmmEvalMex.dll 的執行速度是你寫的 dhmmEval.m 的幾倍？

7. (***)**程式競賽：使用 DHMM 進行語者無關的語音辨識：** 請見此連結。

Chapter 16: Methods for Melody Recognition

本章介紹旋律辨識（Melody Recognition）的各種方法。一個旋律辨識系統，包含有下列三部分：

1. 輸入端：系統所接受到的輸入，例如使用者的哼唱歌聲，或是使用者輸入的音符。一般而言，系統必須先將此輸入轉成可比對格式，例如音高向量，或是音符向量，才能送到下一階段進行比對。
2. 資料庫：資料庫包含系統內部可供比對的歌曲，同樣的，這些歌曲也必須事先處理成可比對的格式，最簡單的格式，就是單音的資料，只包含音高及音長的資訊，而且同一時間點，最多只有一種發音，這就是所謂的「單音音樂」（Monophonic Music），例如單軌的 MIDI 或是人聲的清唱等，都屬於此類。相對而言，一般我們常聽到的 MP3 流行音樂或是古典交響樂，在同一個時間點通常會有多個樂器同時發音，所以是屬於「多音音樂」（Polyphonic music）。
3. 比對方式：使用輸入向量來比對資料庫歌曲的方式，一般可以分成兩大類：
   - 切音符的方法：輸入訊號和資料庫歌曲都以音符（包含音高和音長的資訊）為單位來進行比對，這種方法的好處是比對速度比較快，但是「切音符」（Note Segmentation）本身可能就帶來誤差，導致比對的辨識率也會降低。典型的方法有編輯距離（Edit Distance）等。

- 不切音符的方法：輸入訊號和資料庫歌曲都以音高向量為單位，每一秒可以包含 8～32 個音高點，這種方法的好處是比對辨識率比較高，但是所花的計算量也比較大。典型的方法有線性伸縮（Linear Scaling）、type-1 & type-2 DTW（Dynamic Time Warping）等。
- 混合法：輸入訊號不切音符，但資料庫的歌曲則是以音符為單位來儲存資料，典型的方法是 type-3 DTW 以及 HMM（Hidden Markov Models）等方法。

本章將針對這幾種旋律辨識常用的方法，來進行說明。

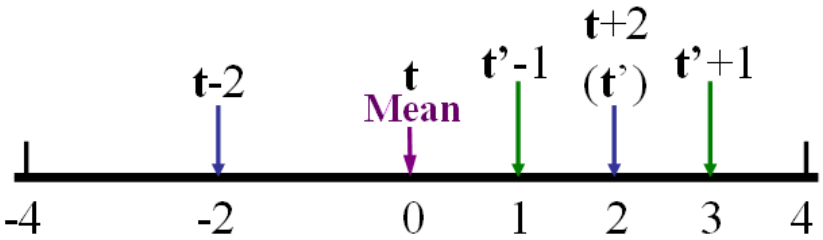
無論是使用什麼比對方法，我們都必須先進行音調移位（key transposition），例如，男生的歌聲具有較低的 key，而女生的歌聲具有較高的 key，但歌曲資料庫的音樂卻具有固定的 key。因此，對於一段使用者輸入的音高向量（或是進行音符切割後的音符資訊），我們都必須先進行音高的上下平移，以便和資料庫中的歌曲得到最佳的比對。

根據比對方法的不同，音調移位也有不同的作法，可以區分如下：

- 一次到位法：若使用線性伸縮的方法，我們可以使用 median 或是 mean 來直接找出最佳的平移值，因此在計算量較低。（請見後續各小節說明。）
- 試誤法：若使用 type-1 或是 type-2 DTW 等方法，由於無法事先得知最佳的平移值，因此只有採用試誤法（trial and error）來逐次逼近，計算量較大。

若採取試誤法，則我們可以採用任何一種一維函數的最佳化方法，常用的方法有線性搜尋（暴力搜尋）及二元搜尋等兩種方法，可用簡單範例說明如下：

- 線性搜尋（暴力搜尋）：
  1. 將輸入音高向量 **t** 的平均值平移到和標準向量 r（資料庫的歌曲）一樣。（若輸入向量的長度是 m，則標準向量的平均值也是取前 m 個元素來計算。）
  2. 將 **t** 分別平移到 [-2.0, -1.8, -1.6, ... 1.6, 1.8, 2.0]，並和 **r** 比對，算出距離的最小值或是相似度的最大值。
- 二元搜尋：
  1. 將輸入音高向量 **t** 的平均值平移到和標準向量 r（資料庫的歌曲）一樣。（若輸入向量的長度是 m，則標準向量的平均值也是取前 m 個元素來計算。）
  2. 設定平移量 s = 2。
  3. 將 **t** 分別平移到 [-s, 0, s]，和 **r** 比對，算出距離的最小值或是相似度的最大值，並進行 **t** 的平移。
  4. 設定 s = s/2，回到前一個步驟，直到收斂。二元搜尋法的示意圖如下：



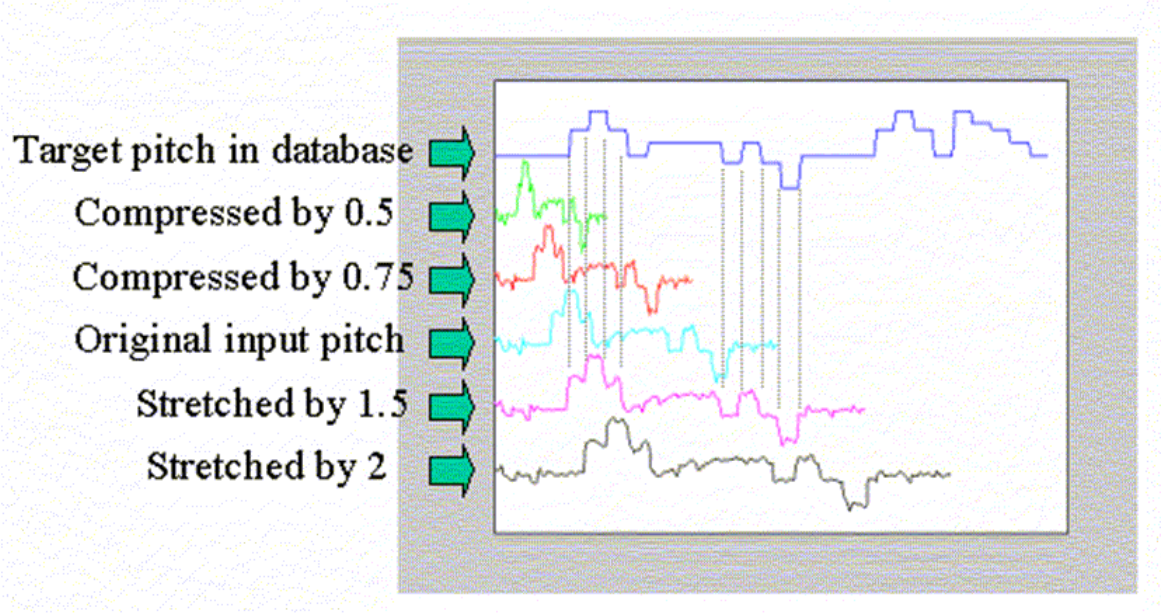
我們可以使用 Utility Toolbox 裡面的 binSearch4optim 函式來進行上述的二元搜尋。例如，若要搜尋 humps 函數在 [0, 1] 之間的最小值，並限制函數求值（Function Evaluation）的次數是 11 次，可見下列範例： Error: D:\users\jang\books\audioSignalProcessing\example\mr\binSearch4optim01.m does not exist!

在不切音符的情況下，最簡單的辨識方法，就是線性伸縮（Linear Scaling），流程如下：

1. 使用內差法，將使用者輸入的音高向量進行線性拉長或壓縮，例如伸縮比例可以是從 0.5 到 2.0，跳距是 0.1，共生出 16 個版本。
2. 將這 16 個版本和資料庫中的每一首歌曲進行比對，得到 16 個距離，其中的最小值，即是輸入向量和此首歌的距離。

3. 對所有資料庫歌曲進行比對，最短距離者，即是使用者所唱的歌。

下面是一個線性伸縮的示意圖，共伸縮五次，當伸縮比例是 1.5 時，可以達到最佳的比對效果：



在實做上，還有下列細節要考慮：

1. 內差法的選用：可以使用簡單的線性內差。
2. 距離的選擇：通常我們使用 L_1 norm，也就是計算每個對應元素絕對差值的和，或是使用 L_2 norm，又稱為歐基理德距離，也就是計算每個對應元素差值的平方和，再開平方，但在實做上，我們通常只在比較距離的大小，因此常常省略開平方的動作，以節省計算。

   Hint

   L_p norm of vectors x and y = $[\Sigma|x_i-y_i|^p]^{1/p}$
3. 距離的正規化：我們會將總距離除以點數，得到正規化的距離，以消除因伸縮造成點數不同所帶來的影響。
4. 音高的校正：每一個人唱歌的 key 不同（通常女生的 key 比較高，男生的 key 比較低），因此在進行比對之前，要先進行校正。一般而言，校正的目的是要達到兩個向量之間距離的最小值，因此對於不同的距離計算方式，我們就有不同的校正法則：
   - 若使用 L_1 norm，我們可以採用「中位數校正」，亦即將輸入向量的中位數校正成對應資料庫向量的中位數。
   - 若使用 L_2 norm，我們可以採用「平均值校正」，亦即將輸入向量的平均值校正成對應資料庫向量的平均值。
5. 對於休止符的處理：為了保持音符的特性，我們通常會將休止符（包含使用者的輸入和資料庫的歌曲）代換成前一個音。

線性伸縮用於旋律辨識的特性，可以說明如下：

- 如果使用者哼唱的歌聲不是忽快忽慢，那麼線性伸縮都可以達到不錯的辨識效果。
- 線性伸縮可以使用「一次到位」的音高校正，所以在計算上比較簡單。（相對而言，DTW 無法使用「一次到位」的音高校正，所以在計算上會比較繁複。）

以下是使用真實音高向量的範例：

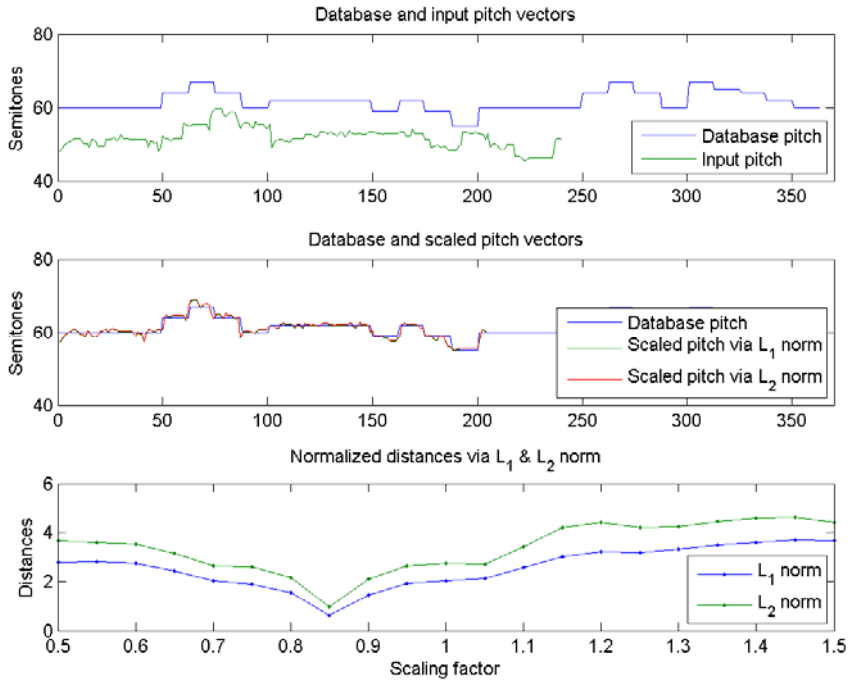
Example 1**Input file** mr/linScaling01.m

```
inputPitch=[48.044247 48.917323 49.836778 50.154445 50.478049 50.807818 51.143991 51.486821
51.486821 51.486821 51.143991 50.154445 50.154445 50.154445 49.218415 51.143991 51.143991
50.807818 49.524836 49.524836 49.524836 49.524836 51.143991 51.143991 51.143991 51.486821
51.836577 50.807818 51.143991 52.558029 51.486821 51.486821 51.486821 51.143991 51.143991
51.143991 51.143991 51.143991 51.143991 51.143991 51.143991 51.143991 49.218415 50.807818
```

```
50.807818 50.154445 50.478049 48.044247 49.524836 52.193545 51.486821 51.486821 51.143991
50.807818 51.486821 51.486821 51.486821 51.486821 51.486821 55.788268 55.349958 54.922471
54.922471 55.349958 55.349958 55.349958 55.349958 55.349958 55.349958 55.349958 55.349958
53.699915 58.163541 59.213095 59.762739 59.762739 59.762739 59.762739 58.163541 57.661699
58.163541 58.680365 58.680365 58.680365 58.163541 55.788268 54.505286 55.349958 55.788268
55.788268 55.788268 54.922471 54.505286 56.237965 55.349958 55.349958 55.349958 55.349958
54.505286 54.505286 55.349958 48.917323 50.478049 50.807818 51.143991 51.143991 51.143991
50.807818 50.807818 50.478049 50.807818 51.486821 51.486821 51.486821 51.486821 51.486821
51.486821 52.558029 52.558029 52.558029 52.558029 52.193545 51.836577 52.193545 53.310858
53.310858 53.310858 52.930351 52.930351 53.310858 52.930351 52.558029 52.193545 52.930351
53.310858 52.930351 51.836577 52.558029 53.699915 52.930351 52.930351 52.558029 52.930351
52.930351 52.558029 52.558029 52.558029 53.310858 53.310858 53.310858 53.310858 52.930351
52.930351 52.930351 52.558029 52.930351 52.930351 52.930351 52.930351 52.930351 52.930351
52.930351 53.310858 53.310858 53.310858 52.193545 52.193545 52.193545 54.097918 52.930351
52.930351 52.930351 52.930351 52.930351 51.143991 51.143991 51.143991 48.917323 49.524836
49.524836 49.836778 49.524836 48.917323 49.524836 49.218415 48.330408 48.330408 48.330408
48.330408 48.330408 49.524836 49.836778 53.310858 53.310858 53.310858 52.930351 52.930351
52.930351 53.310858 52.930351 52.930351 52.558029 52.558029 51.143991 52.930351 49.218415
49.836778 50.154445 49.836778 49.524836 48.621378 48.621378 48.621378 49.836778 49.836778
49.836778 49.836778 46.680365 46.680365 46.680365 46.163541 45.661699 45.661699 45.910801
46.163541 46.163541 46.163541 46.163541 46.163541 46.163541 46.163541 46.163541 46.163541
46.163541 46.163541 46.163541 50.807818 51.486821 51.486821 51.143991];
dbPitch =[60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60
60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 64 64 64 64 64 64 64 64 64 64 64 64 64 67 67 67
67 67 67 67 67 67 67 67 67 64 64 64 64 64 64 64 64 64 64 64 64 64 60 60 60 60 60 60 60 60 60 60 60 60
60 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62
62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 59 59 59 59 59 59 59 59 59 59 59 59 59 62 62 62 62 62
62 62 62 62 62 62 62 59 59 59 59 59 59 59 59 59 59 59 59 59 55 55 55 55 55 55 55 55 55 55 55 55 55 60
60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60
60 60 60 60 60 60 60 60 60 60 60 60 60 60 64 64 64 64 64 64 64 64 64 64 64 64 64 67 67 67 67 67 67 67
67 67 67 67 67 64 64 64 64 64 64 64 64 64 64 64 64 64 60 60 60 60 60 60 60 60 60 60 60 60 60 67 67 67
67 67 67 67 67 67 67 67 67 65 65 65 65 65 65 65 65 65 65 65 65 65 64 64 64 64 64 64 64 64 64 64 64 64
62 62 62 62 62 62 62 62 62 62 62 62 62 60 60 60 60 60 60 60 60 60 60 60 60 60];
resolution=21;
sfBounds=[0.5, 1.5];            % Scaling-factor bounds
distanceType=1;                 % L1-norm
[minDist1, scaledPitch1, allDist1]=linScalingMex(inputPitch, dbPitch, sfBounds(1), sfBounds(2),
resolution, distanceType);
distanceType=2;                 % L2-norm
[minDist1, scaledPitch2, allDist2]=linScalingMex(inputPitch, dbPitch, sfBounds(1), sfBounds(2),
resolution, distanceType);
allDist2=sqrt(allDist2);        % To reduce computation, the L2-distance returned by linScalingMex is
actually the square distance, so we need to take the square root.
axisLimit=[0 370 40 80];
subplot(3,1,1);
plot(1:length(dbPitch), dbPitch, 1:length(inputPitch), inputPitch);
title('Database and input pitch vectors'); ylabel('Semitones');
legend('Database pitch', 'Input pitch', 'location', 'SouthEast');
```

```
axis(axisLimit);
subplot(3,1,2);
plot(1:length(dbPitch), dbPitch, 1:length(scaledPitch1), scaledPitch1, 1:length(scaledPitch2),
scaledPitch2);
legend('Database pitch', 'Scaled pitch via L_1 norm', 'Scaled pitch via L_2 norm', 'location', 'SouthEast');
title('Database and scaled pitch vectors'); ylabel('Semitones');
axis(axisLimit);
subplot(3,1,3);
ratio=linspace(sfBounds(1), sfBounds(2), resolution);
plot(ratio, allDist1, '.-', ratio, allDist2, '.-');
xlabel('Scaling factor'); ylabel('Distances'); title('Normalized distances via L_1 & L_2 norm');
legend('L_1 norm', 'L_2 norm', 'location', 'SouthEast');
```

Output figure



Hint

請注意，為了節省計算，上述範例中 linScalingMex 的距離計算，是使用 L_2 norm 的平方，而不是使用 L_2 norm。所以最後在畫圖前，我們還要進行開平方，以便得到正確的 L_2 norm。

DTW 也是常用於旋律辨識的方法，其最主要優點就是辨識率高，而最大缺點則是在於計算時間較久。因此有很多研究，都是在強調如何不影響辨識率的情況下，來加速 DTW 的運算。以下是幾個 DTW 的使用範例，來說明 DTW 的特性。

有關於 DTW 的技術細節，可見前幾章的說明。以下就是示範幾個 DTW 的範例。首先，我們可以畫出 type-1 DTW 的路徑，如下： Error: D:\users\jang\books\audioSignalProcessing\example\mrDtwPlot01_type1.m does not exist!

只要我們掌握了 DP 的遞迴原則，就可以根據需要，對 DTW 進行各種變形。在本節中，我們介紹另一種 DTW，其輸入格式具有下列特性：

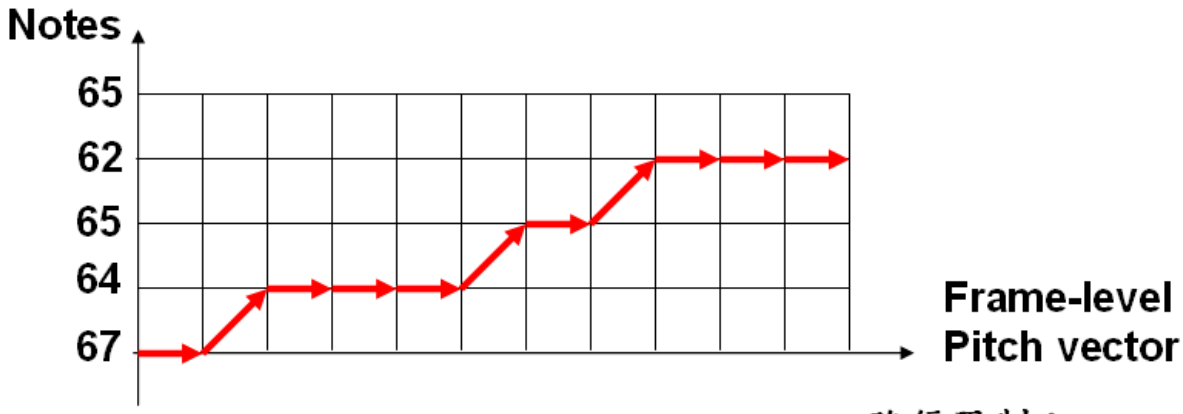
- 使用者輸入：以音框為基礎的音高向量，不進行音符切割。（通常這類資料稱為 mid 格式。以我們進行的人工標示音高而言，每一點的時間長度是 256/8000 = 1/31.25 = 0.032 s = 32 ms。）

- 資料庫格式：以音符為比對單位，但只考慮音高。（通常這類資料稱為 note 格式，以向量 [音高, 音長, 音高, 音長...] 來表示。以我們哼唱選歌的資料而言，音長的單位都是 1/64 秒。）

假設使用者輸入的音高向量是 t 而標準答案的音符向量是 r，並假設 D(i, j) 是 t(1:i) 和 r(1:j) 之間的最短距離，則我們有下列遞迴式：

D(i, j) = min(D(i-1,j), D(i-1, j-1))+|t(i)-r(j)|

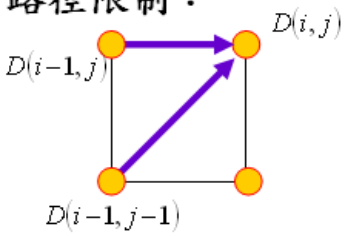
請見下列示意圖：



遞迴算式（based on log prob.）：

$$D(i,j)=|t(i)-r(j)|+\min\begin{cases}D(i-1,j)\\D(i-1,j-1)\end{cases}$$

路徑限制：



為便於說明，我們簡稱這一類方法為 type-3 DTW。此方法有下列特性：

- 由於資料庫格式是以音符為單位，所以計算量小於 type-1 及 type-2 DTW。
- 資料庫格式並沒有用到音符的音長資訊，所以理論上來說，辨識率應該低於 type-1 及 type-2 DTW。
- 無法進行一次到位的音高平移，這點和 type-1 及 type-2 DTW 是一樣的。

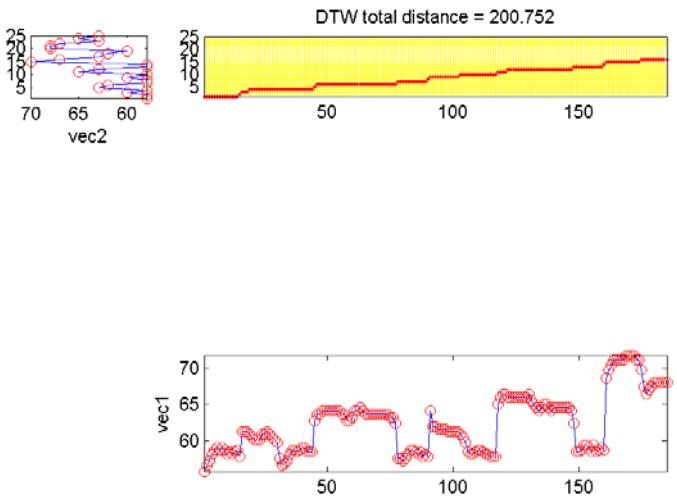
在以下的範例，我們使用 type-3 DTW 來進行音高向量對音符（只用音高）的「對位」（Alignment）：

Example 1**Input file** mr/dtw3path01.m

```
pv=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 47.485736 48.330408 48.917323 49.836778 50.478049 50.807818
50.478049 50.807818 50.478049 49.836778 50.154445 49.836778 50.154445 50.478049 49.524836 0 0
52.930351 52.930351 52.930351 52.558029 52.193545 51.836577 51.836577 51.836577 52.558029
52.558029 52.930351 52.558029 52.193545 51.836577 51.486821 49.218415 48.330408 48.621378
48.917323 49.836778 50.478049 50.478049 50.154445 50.478049 50.807818 50.807818 50.154445
50.154445 50.154445 0 0 0 54.505286 55.349958 55.349958 55.788268 55.788268 55.788268 55.788268
55.788268 55.788268 55.788268 55.788268 55.349958 55.349958 54.505286 54.505286 54.922471
55.788268 55.788268 56.237965 55.788268 55.349958 55.349958 55.349958 55.349958 55.349958
55.349958 55.349958 55.349958 55.349958 55.349958 54.922471 54.922471 54.097918 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 49.218415 49.218415 48.917323 49.218415 49.836778 50.478049 50.478049 50.154445
49.836778 50.154445 49.524836 49.836778 49.524836 0 0 55.788268 53.699915 53.699915 53.310858
53.310858 53.310858 53.310858 52.930351 52.930351 52.930351 52.930351 52.930351 52.558029
52.193545 51.486821 50.154445 49.836778 49.836778 50.154445 50.478049 50.478049 50.154445
49.836778 49.836778 49.524836 49.524836 49.524836 0 0 0 0 56.699654 57.661699 58.163541
58.163541 57.661699 57.661699 57.661699 57.661699 57.661699 57.661699 57.661699 57.661699
58.163541 57.173995 56.699654 56.237965 55.788268 56.237965 56.699654 56.699654 56.237965
55.788268 56.237965 56.237965 56.237965 56.237965 56.237965 56.237965 56.237965 55.788268
```

```
54.097918 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 50.154445 50.154445 50.478049 51.143991 51.143991
50.807818 50.154445 51.143991 50.154445 50.478049 50.807818 50.478049 0 0 0 60.330408 61.524836
62.154445 62.807818 62.807818 62.807818 62.807818 62.807818 63.486821 63.486821 63.486821
63.486821 62.807818 62.807818 61.524836 59.213095 58.163541 58.680365 59.213095 59.762739
59.762739 59.762739 59.762739 59.762739 59.762739];
pv(pv==0)=[];                                   % Delete rests (刪除休止符)
note=[60 29 60 10 62 38 60 38 65 38 64 77 60 29 60 10 62 38 60 38 67 38 65 77 60 29 60 10 72 38 69 38
65 38 64 38 62 77 0 77 70 29 70 10 69 38 65 38 67 38 65 38];
note(2:2:end)=note(2:2:end)/64;                 % The time unit of note duration is 1/64 seconds
frameSize=256; overlap=0; fs=8000;
timeStep=(frameSize-overlap)/fs;
pv2=note2pv(note, timeStep);
noteMean=mean(pv2(1:length(pv)));  % Take the mean of pv2 with the length of pv
pv=pv-mean(pv)+noteMean;                        % Key transposition
notePitch=note(1:2:end);                % Use pitch only (只取音高)
notePitch(notePitch==0)=[];             % Delete rests (刪除休止符)
[minDistance, dtwPath] = dtw3mex(pv, notePitch, 1, 0);
dtwPlot(pv, notePitch, dtwPath);
```

Output figure



在上述範例中，我們在進行 dtw3 的比對前，做了兩件事情：

1. 音調移位：我們假設歌唱者的速度和樂譜的速度是一樣的，因此我們將 note 先轉成 mid 格式，再取用和 PV 同樣的長度來計算其平均值為 noteMean，最後再將 PV 移到同樣的平均值。這是一個簡化的處理，因為我們並無法使用「一次到位」的音調移位。
2. 休止符的處理：我們是把 PV 和 Note 中的休止符都砍掉來進行比對。這也是一個簡化的處理，後續會提到如何使用休止符來提高比對效果。

經過上述範例的對位後，我們可以將每個音高點所對應的音符音高畫出來，如下：

Example 2**Input file** mr/dtw3inducedPitch01.m

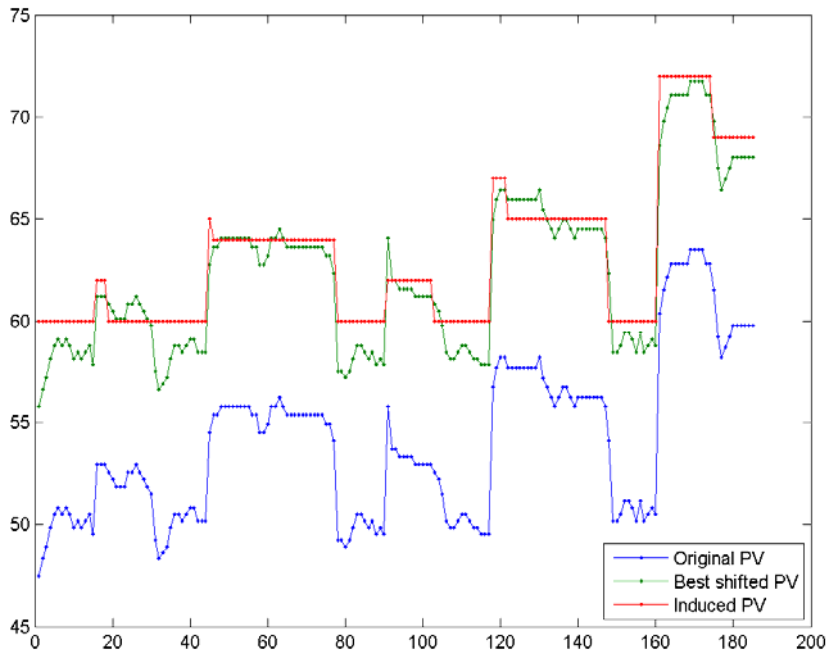
```
pv=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 47.485736 48.330408 48.917323 49.836778 50.478049 50.807818
50.478049 50.807818 50.478049 49.836778 50.154445 49.836778 50.154445 50.478049 49.524836 0 0
52.930351 52.930351 52.930351 52.558029 52.193545 51.836577 51.836577 51.836577 52.558029
52.558029 52.930351 52.558029 52.193545 51.836577 51.486821 49.218415 48.330408 48.621378
48.917323 49.836778 50.478049 50.478049 50.154445 50.478049 50.807818 50.807818 50.154445
50.154445 50.154445 0 0 0 54.505286 55.349958 55.349958 55.788268 55.788268 55.788268 55.788268
55.788268 55.788268 55.788268 55.788268 55.349958 55.349958 54.505286 54.505286 54.922471
55.788268 55.788268 56.237965 55.788268 55.349958 55.349958 55.349958 55.349958 55.349958
55.349958 55.349958 55.349958 55.349958 55.349958 54.922471 54.922471 54.097918 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 49.218415 49.218415 48.917323 49.218415 49.836778 50.478049 50.478049 50.154445
49.836778 50.154445 49.524836 49.836778 49.524836 0 0 55.788268 53.699915 53.699915 53.310858
53.310858 53.310858 53.310858 52.930351 52.930351 52.930351 52.930351 52.930351 52.558029
52.193545 51.486821 50.154445 49.836778 49.836778 50.154445 50.478049 50.478049 50.154445
49.836778 49.836778 49.524836 49.524836 49.524836 0 0 0 0 56.699654 57.661699 58.163541
58.163541 57.661699 57.661699 57.661699 57.661699 57.661699 57.661699 57.661699 57.661699
58.163541 57.173995 56.699654 56.237965 55.788268 56.237965 56.699654 56.699654 56.237965
55.788268 56.237965 56.237965 56.237965 56.237965 56.237965 56.237965 56.237965 55.788268
54.097918 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 50.154445 50.154445 50.478049 51.143991 51.143991
50.807818 50.154445 51.143991 50.154445 50.478049 50.807818 50.478049 0 0 0 60.330408 61.524836
62.154445 62.807818 62.807818 62.807818 62.807818 62.807818 63.486821 63.486821 63.486821
63.486821 62.807818 62.807818 61.524836 59.213095 58.163541 58.680365 59.213095 59.762739
59.762739 59.762739 59.762739 59.762739 59.762739];
pv(pv==0)=[];                                   % Delete rests (删除休止符)
origPv=pv;
pvLen=length(origPv);
note=[60 29 60 10 62 38 60 38 65 38 64 77 60 29 60 10 62 38 60 38 67 38 65 77 60 29 60 10 72 38 69 38
65 38 64 38 62 77 0 77 70 29 70 10 69 38 65 38 67 38 65 38];
note(2:2:end)=note(2:2:end)/64;                 % The time unit of note duration is 1/64 seconds
timeStep=256/8000;
pv2=note2pv(note, timeStep);
noteMean=mean(pv2(1:length(pv)));
shiftedPv=pv-mean(pv)+noteMean;                 % Key transposition
notePitch=note(1:2:end);                % Use pitch only (只取音高)
notePitch(notePitch==0)=[];             % Delete rests (删除休止符)
[minDistance, dtwPath] = dtw3mex(shiftedPv, notePitch, 1, 0);
inducedPv=notePitch(dtwPath(2,:));
plot(1:pvLen, origPv, '.-', 1:pvLen, shiftedPv, '.-', 1:pvLen, inducedPv, '.-');
legend('Original PV', 'Best shifted PV', 'Induced PV', 4);
fprintf('Min. distance = %f\n', minDistance);
fprintf('Hit return to hear the original pitch vector...\n'); pause; pvPlay(origPv, timeStep);
fprintf('Hit return to hear the shifted pitch vector...\n'); pause; pvPlay(shiftedPv, timeStep);
inducedNote=pv2noteStrict(inducedPv, timeStep);
fprintf('Hit return to hear the induced pitch vector...\n'); pause; notePlay(inducedNote);
```

Output message

```
Min. distance = 200.752223
Hit return to hear the original pitch vector...
Error in ==> dtw3inducedPitch01 at 18
```

```
fprintf('Hit return to hear the original pitch vector...\n'); pause; pvPlay(origPv,
timeStep);

Error in ==> goWriteOutputFile at 32
        eval(command);

goWriteOutputFile
1/10: linScaling01
```

Output figure



在上述範例中，由於綠色曲線（平移過的哼唱音高向量）和紅色曲線（由 dtw3 對位所產生的對應音符音高）的吻合程度並不理想，由此可以看出，我們的音調移位出了問題，所以得到的對位效果並不理想，很可能是由使用者哼唱的速度和樂譜的速度並不一致，這一首歌是「生日快樂」，我們可以直接試聽看看相關的檔案：

- 使用者哼唱的錄音
- 對應的歌曲
- Query pitch vector (with rests)
- Query pitch vector (without rests)
- Shifted query pitch vector (without rests)
- Induced notes

若要進行更吻合的對位，我們就必須改善音調移位。我們可以使用一個簡單的線性搜尋法（暴力法）來找到最佳的音高平移量，換句話說，也就是進行 81 次音調移位，平移量則平均分佈於 [-2, 2] 之間，請見下列範例：

Example 3**Input file** mr/dtw3inducedPitch02.m

```
pv=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 47.485736 48.330408 48.917323 49.836778 50.478049 50.807818
50.478049 50.807818 50.478049 49.836778 50.154445 49.836778 50.154445 50.478049 49.524836 0 0
52.930351 52.930351 52.930351 52.558029 52.193545 51.836577 51.836577 51.836577 52.558029
```

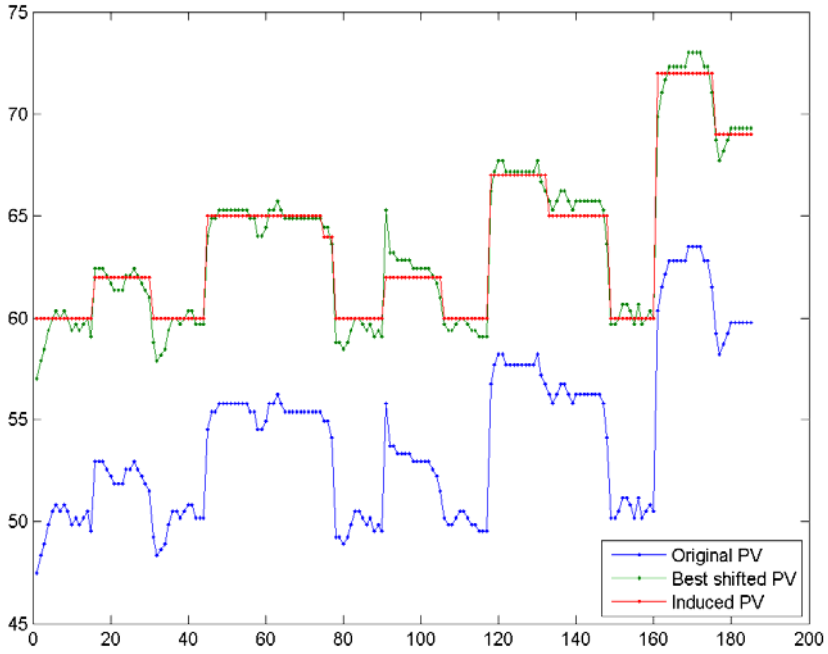
```
52.558029 52.930351 52.558029 52.193545 51.836577 51.486821 49.218415 48.330408 48.621378
48.917323 49.836778 50.478049 50.478049 50.154445 50.478049 50.807818 50.807818 50.154445
50.154445 50.154445 0 0 0 54.505286 55.349958 55.349958 55.788268 55.788268 55.788268 55.788268
55.788268 55.788268 55.788268 55.788268 55.349958 55.349958 54.505286 54.505286 54.922471
55.788268 55.788268 56.237965 55.788268 55.349958 55.349958 55.349958 55.349958 55.349958
55.349958 55.349958 55.349958 55.349958 55.349958 54.922471 54.922471 54.097918 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 49.218415 49.218415 48.917323 49.218415 49.836778 50.478049 50.478049 50.154445
49.836778 50.154445 49.524836 49.836778 49.524836 0 0 55.788268 53.699915 53.699915 53.310858
53.310858 53.310858 53.310858 52.930351 52.930351 52.930351 52.930351 52.930351 52.558029
52.193545 51.486821 50.154445 49.836778 49.836778 50.154445 50.478049 50.478049 50.154445
49.836778 49.836778 49.524836 49.524836 49.524836 0 0 0 0 56.699654 57.661699 58.163541
58.163541 57.661699 57.661699 57.661699 57.661699 57.661699 57.661699 57.661699 57.661699
58.163541 57.173995 56.699654 56.237965 55.788268 56.237965 56.699654 56.699654 56.237965
55.788268 56.237965 56.237965 56.237965 56.237965 56.237965 56.237965 56.237965 55.788268
54.097918 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 50.154445 50.154445 50.478049 51.143991 51.143991
50.807818 50.154445 51.143991 50.154445 50.478049 50.807818 50.478049 0 0 0 60.330408 61.524836
62.154445 62.807818 62.807818 62.807818 62.807818 62.807818 63.486821 63.486821 63.486821
63.486821 62.807818 62.807818 61.524836 59.213095 58.163541 58.680365 59.213095 59.762739
59.762739 59.762739 59.762739 59.762739 59.762739];
pv(pv==0)=[];                                   % Delete rests (刪除休止符)
origPv=pv;
pvLen=length(origPv);
note=[60 29 60 10 62 38 60 38 65 38 64 77 60 29 60 10 62 38 60 38 67 38 65 77 60 29 60 10 72 38 69 38
65 38 64 38 62 77 0 77 70 29 70 10 69 38 65 38 67 38 65 38];
note(2:2:end)=note(2:2:end)/64;                 % The time unit of note duration is 1/64 seconds
timeStep=256/8000;
pv2=note2pv(note, 256/8000);
noteMean=mean(pv2(1:length(pv)));
shiftedPv=pv-mean(pv)+noteMean;                 % Key transposition
notePitch=note(1:2:end);                % Use pitch only (只取音高)
notePitch(notePitch==0)=[];             % Delete rests (刪除休止符)

% Linear search of 81 times within [-2 2] (上下平移 81 次，得到最短距離)
clear minDist dtwPath
shift=linspace(-2, 2, 81);
for i=1:length(shift)
        newPv=shiftedPv+shift(i);
        [minDist(i), dtwPath{i}] = dtw3mex(newPv, notePitch, 1, 0);
end
[minValue, minIndex]=min(minDist);
bestShift=shift(minIndex);
bestShiftedPv=shiftedPv+bestShift;
inducedPv=notePitch(dtwPath{minIndex}(2,:));
plot(1:pvLen, origPv, '.-', 1:pvLen, bestShiftedPv, '.-', 1:pvLen, inducedPv, '.-');
legend('Original PV', 'Best shifted PV', 'Induced PV', 4);
fprintf('Best shift = %f\n', bestShift);
fprintf('Min. distance = %f\n', minValue);
fprintf('Hit return to hear the original pitch vector...\n'); pause; pvPlay(origPv, 256/8000);
```

```
fprintf('Hit return to hear the shifted pitch vector...\n'); pause; pvPlay(bestShiftedPv, timeStep);
inducedNote=pv2noteStrict(inducedPv, 256/8000);
fprintf('Hit return to hear the induced pitch vector...\n'); pause; notePlay(inducedNote);
```

Output message

```
Best shift = 1.250000
Min. distance = 103.142939
Hit return to hear the original pitch vector...
Hit return to hear the shifted pitch vector...
Hit return to hear the induced pitch vector...
```

Output figure



由上述範例可以看出，DTW 對位的效果已經大幅改善，最短距離也大幅降低。

Hint

一般而言，若要進行旋律辨識，是無法進行上述線性搜尋法的音調移位，只能採取計算量較小的計算方法，例如第二節所提到的二元搜尋法。

但是在上述範例中，我們還是可以發現對位的錯誤，例如在第一句「祝你生日快樂」的「樂」這個音符，只有被分配到三個音框，很明顯的過少。若要解決這個問題，有幾個可能的方向：

- 限制每一個音符所能分配的音框個數：例如每個音符所對應的音框總長度，不得小於音符長度的一半，也不得大於音符長度的兩倍。此規則的加入，使我們可以同時用到音符的音高及音長資訊。
- 使用休止符：在使用者哼唱的音高向量中，除了頭尾的休止符不算外，只要遇到一個休止區間，就代表一個舊音符的結束及新音符的開始，此規則可以適用到一般的哼唱選歌輸入。

我們可以修改 DTW 以符合上述規範，以便提高對位的準確度及旋律辨識的辨識率。

若是以音符為單位來進行比對，我們就必須先將輸入向量切成音符。假設音高向量可以表示成 pv(i), i=1~n，那麼最簡單的切音符方法，可以說明如下：

```
for i=1:n
```

```
        if |pv(i)-pv(i-1)|>θ &  音符夠長
                切出一個音符
        else
                將  pv(i)  加入目前的音符
        end
end
```

以下就是一個簡單的範例，來切出音符： Error: D:\users\jang\books\audioSignalProcessing\example\noteSegment01.m does not exist!

16-7 旋律辨識的效能改進

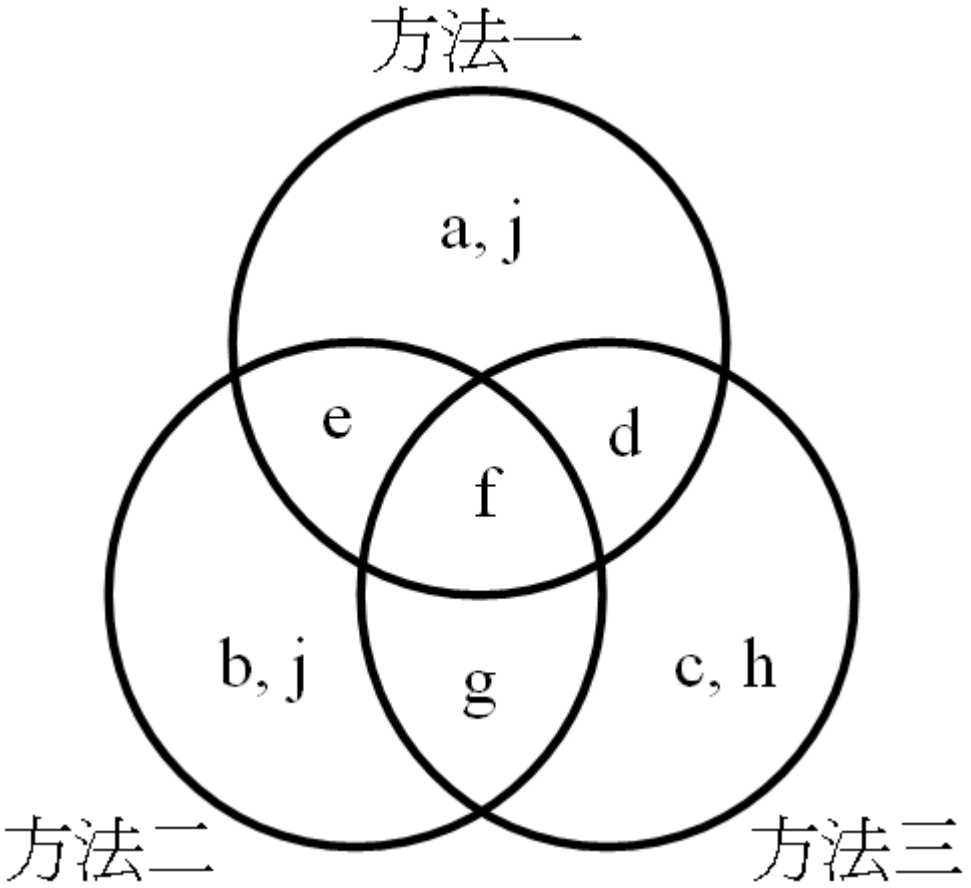
本章提出了很多旋律辨識的比對方法，每一種方法都有其優缺點，但是在實際應用時，最常被提到的評價標準就是辨識率和計算時間。若不考慮其他次要因素，通常我們是希望辨識率越高越好，而計算時間也是越少越好，但是這兩個因素會互相抵觸，套句俗話所說，總不能「又要馬兒好，又要馬而不吃草」吧？因此在方法的選用以及效能的評估，我們都會話出辨識率對計算時間的作圖，同時對於每一種方法，我們也會改變其參數，因此一種方法就會對應一條曲線，若有五種方法，就會出現五條曲線，這時候我們就可以根據應用面的需求來選取一種適合的方法以及相關的參數值。（請見本章作業。）

當資料庫的歌曲越來越多時，比對所花的時間也會越來越慢，但對於一個實際應用的系統而言，等待時間應該以不超過五秒為原則，因此我們必須在有限的時間內，找出各種辨識方法的組合，以便能夠求取辨識率的最大值，這種方法稱為 Progressive Filtering，也就是先用一些粗略的方法來刪除不可能的歌曲，再用比較精密的方法來進行詳細的比對，這樣就能夠大量節省比對時間，也不會大幅降低辨識率。至於如何區分粗略及詳細的方法，以及如何安排每個方法的先後，這些都需要詳細的數學分析，在此不再贅述。

如果不考慮計算時間，只求辨識率的提升，那麼一個簡單的方法，就是使用多個辨識方法來進行投票表決。例如，假設我們有 10 首 wav 檔案待辨識，共使用了三種方法，辨識結果請見下列表單：

wav 檔案	方法一	方法二	方法三	投票表決法
a	0	1	1	1
b	1	1	0	1
c	1	0	1	1
d	0	0	1	0
e	0	1	0	0
f	0	0	0	0
g	1	0	0	0
h	1	0	1	1
i	0	1	1	1
j	1	1	0	1
辨識率：	50%	50%	50%	60%

其中 1 代表辨識正確，0 代表辨識錯誤，換句話說，方法一、二、三的辨識率有是 50%，而進行投票後，辨識率可以提升到 60%。若用集合的方式來表示，可見下圖，其中落於「方法一」的元素代表辨識錯誤之歌曲，餘類推。由於方法一、二、三所辨識錯誤歌曲的重複性不高，因此我們可以使用投票表決法，來達到提升辨識率的目標。



Hint

投票表決法一般而言，都可以提高辨識率，但是也有可能出現反效果，完全看錯誤資料的分佈而定。如果辨識的方法很多，我們也可以使用一套最佳化的方法，讓系統根據辨識率列表，找出三個最有效的方法來進行投票表決，以得到最佳的辨識率。

1. (*) **中位數的證明：** 對於一組已知的純量 $\{x_1, \ldots x_n\}$，請證明此組純量的中位數能夠達到下列目標函數的最小值：

   $$J(u) = \Sigma|x_i-u|$$

2. (*) **平均值的證明：** 對於一組已知的純量 $\{x_1, \ldots x_n\}$，請證明此組純量的平均值能夠達到下列目標函數的最小值：

   $$J(u) = \Sigma(x_i-u)^2$$

3. (**) **切音符的函數：** 請寫一個能夠執行音符切割的函數 myNoteSegment.m，其使用方法如下：

   note = myNoteSegment(pitchVec, timeUnit, pitchTh, minNoteDuration)

   其中

   - pitchVec: 輸入的音高向量
   - timeUnit：每一個音高點所對應的時間
   - pitchTh：音高差異門檻值，用於切出一個音符
   - minNoteDuration：音符的最小長度，以秒為單位。
   - note: 切出的音符，以「音高, 音長, 音高, 音長...」的格式來儲存，其中音高的單位是 Semitone，音長的單位是 1/64 秒。

   除了使用音高差異來切音外，若碰到休止部分（pitchVec 中元素值為零的點），請務必將休止符切出來，因為休止符也是音符的一種。特別要注意的是，休止符的長度可以小於 minNoteDuration。如何測試此函數：

f. 請使用「生日快樂」的音高向量來進行音符切割，你可以聽看看原來的 wave 檔案以及 midi 檔案。請使用 pvPlay.m 來播放原始音高向量，並用 notePlay 來播放切出來的音符，聽看看結果是否正確。
g. 請將你的函數用在最後一題的程式競賽，並只用一次的音高平移，看看辨識率有多高。
h. 請將你的函數用在最後一題的程式競賽，並用五次的音高平移（使用試誤法），看看辨識率有多高。

提示：可以參考 Melody Recognition Toolbox 中的 noteSegment.m，請注意，此函數並未對休止符進行處理。

4. (**) **LinearScaling 的函數：** 請寫一個能夠執行線性伸縮的函數 mylinearScaling.m，其使用方法如下：

[minDist, bestVec1, allDist] = myLinearScaling4mr(pitchVec1, pitchVec2, lowerRatio, upperRatio, resolution, distanceType)

其中

- pitchVec1: 使用者輸入的音高向量，也是被伸縮及平移的向量
- pitchVec2: 資料庫中的音高向量
- lowerRatio: 最低伸縮比例
- upperRatio: 最高伸縮比例
- resolution: 伸縮總次數
- distanceType: 所用的距離函數，1 代表 L_1 norm，2 代表 L_2 norm。
- minDist: 線性伸縮的最短距離
- bestVec1: 對應於最短距離、伸縮及平移後的 pitchVec1
- allDist: 線性伸縮的所有距離

請注意：

- 若 vec1 被拉長後的長度大於 vec2，則不再計算距離。
- 不同的距離計算方式，要進行不同的音高校正。
- 要進行距離的正規化。
- 可以假定輸入的向量不包含休止符。

如何測試你寫的函數：

- 請執行此範例，但將 linScalingMex 改成 myLinearScaling，看看是否得到同樣的效果。
- 請用你的程式碼來測試哼唱選歌的辨識率。

提示：可用 interp1 指令來進行內差，median 指令來計算中位數，mean 指令來計算平均值，norm 指令來計算 L_p norm。若對 interp1.m 的使用有疑問，請見下列範例：

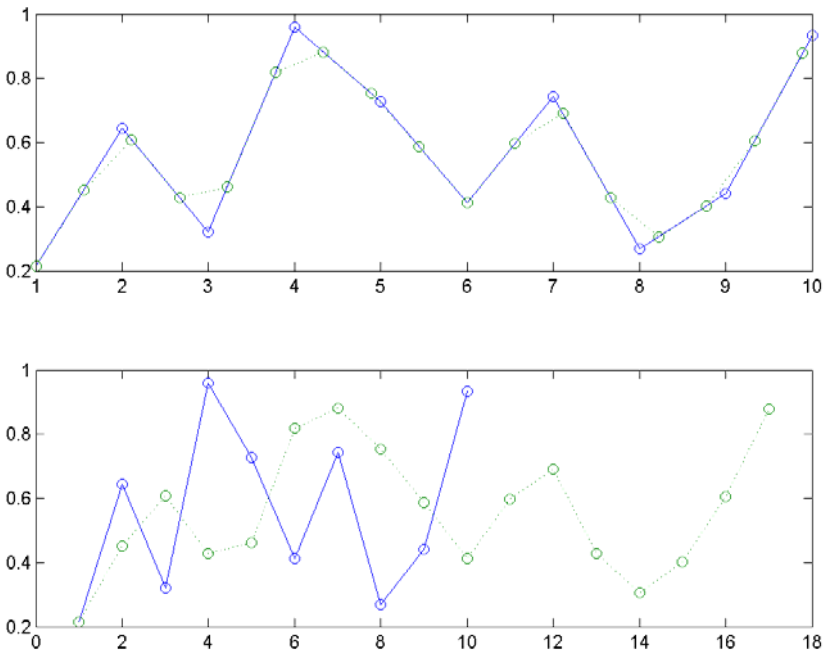
Example 1**Input file** interpDemo01.m

```
n=10;
x=1:n;
y=rand(1,n);                    % Original vector (原始向量)
ratio=1.8;              % The vector interpolation should have a length equal to 1.8 times the original vector
x2=1:1/ratio:n;
y2=interp1(x, y, x2);           % Vector after interpolation
subplot(2,1,1), plot(x, y, 'o-', x2, y2, 'o:');
subplot(2,1,2), plot(1:length(y), y, 'o-', 1:length(y2), y2, 'o:');
fprintf('Desired length ratio = %f\n', ratio);
fprintf('Actual length ratio = %f\n', length(y2)/length(y));
```

Output message

```
Desired length ratio = 1.800000
Actual length ratio = 1.700000
```

Output figure



5. (***) **editDistance 程式碼：** 請寫一個 MATLAB 函數 editDistance.m 來實做 Edit Distance，使用格式如下：

   [minDist, edPath, edTable] = editDistance(str1, str2)

6. (***) **DTW3 程式碼：** 請寫一個 MATLAB 函數 myDtw3.m 來實做 type-3 DTW，使用格式如下：

   [minDist, dtwPath, dtwTable] = myDtw3(vec1, vec2)

   可由下列半成品進行修改：

   原始檔（dtw3skeleton.m）：（灰色區域按兩下即可拷貝）

```
function [minDistance, dtwPath, dtwTable]=dtw3(vec1, vec2, beginCorner, endCorner)
% dtw3: Dynamic time warping with local paths of 0 and 45 degrees
%           Usage: [minDistance, dtwPath, dtwTable]=dtw3(vec1, vec2, beginCorner,
endCorner, plotOpt)
%                       vec1: testing vector, which should be a pitch vector
%                       vec2: reference vector, which should be a vector of note pitch
%                       minDistance: minimun distance of DTW
%                       dtwPath: optimal path of dynamical programming through the DTW
table
%                                   (Its size is 2xk, where k is the path length.)
%                       dtwTable: DTW table

if nargin<3, beginCorner=1; end
if nargin<4, endCorner=1; end
```

```
% If input is vector, make it row vector
if size(vec1,1)==1 | size(vec1,2)==1, vec1 = vec1(:)'; end
if size(vec2,1)==1 | size(vec2,2)==1, vec2 = vec2(:)'; end

size1=length(vec1);
size2=length(vec2);

% ====== Construct DTW table
dtwTable=inf*ones(size1,size2);
% ====== Construct the first element of the DTW table
dtwTable(1,1)=vecDist(vec1(:,1), vec2(:,1));
% ====== Construct the first row of the DTW table (xy view)
for i=2:size1
 dtwTable(i,1)=dtwTable(i-1,1)+vecDist(vec1(:,i), vec2(:,1));
 prevPos(i,1).i=i-1;
 prevPos(i,1).j=1;
end
% ====== Construct the first column of the DTW table (xy view)
if beginCorner==0
 for j=2:size2
            dtwTable(1,j)=vecDist(vec1(:,1), vec2(:,j));
            prevPos(1,j).i=[];
            prevPos(1,j).j=[];
 end
end

% ====== Construct all the other rows of DTW table
for i=2:size1
 for j=2:size2
            pointDist=vecDist(vec1(:,i), vec2(:,j));
            % ====== Check 45-degree predecessor
            ...
            % ====== Check 0-degree predecessor
            ...
 end
end

% ====== Find the overall optimum path
[minDistance, dtwPath]=dtwBackTrack(dtwTable, prevPos, beginCorner, endCorner);

% ========== Sub function ==========
function distance=vecDist(x, y)
distance=sum(abs(x-y));
```

. 請將此範例內之 dtw3mex 改成 myDtw3，並確認所畫出來的圖形和原範例的圖形一樣。

a. 請將你的函數用在最後一題的程式競賽，並只用一次的音高平移，看看辨識率有多高。

b. 請將你的函數用在最後一題的程式競賽，並用五次的音高平移（使用二元搜尋法），看看辨識率有多高。

7. (***) **DTW3 程式碼：改進方案：** 請寫一個 MATLAB 函數來重複上一題，但是必須逐次滿足下列三個條件：

. 限制每一個音符所能分配的音框個數：例如每個音符所對應的音框總長度，不得小於音符長度的一半，也不得大於音符長度的兩倍。此規則的加入，使我們可以同時用到音符的音高及音長資訊。

a. 使用休止符：在使用者哼唱的音高向量中，除了頭尾的休止符不算外，只要遇到一個休止區間，就代表一個舊音符的結束及新音符的開始，此規則可以適用到一般的哼唱選歌輸入。

b. 同時使用上述兩項規則。

請測試此函數：

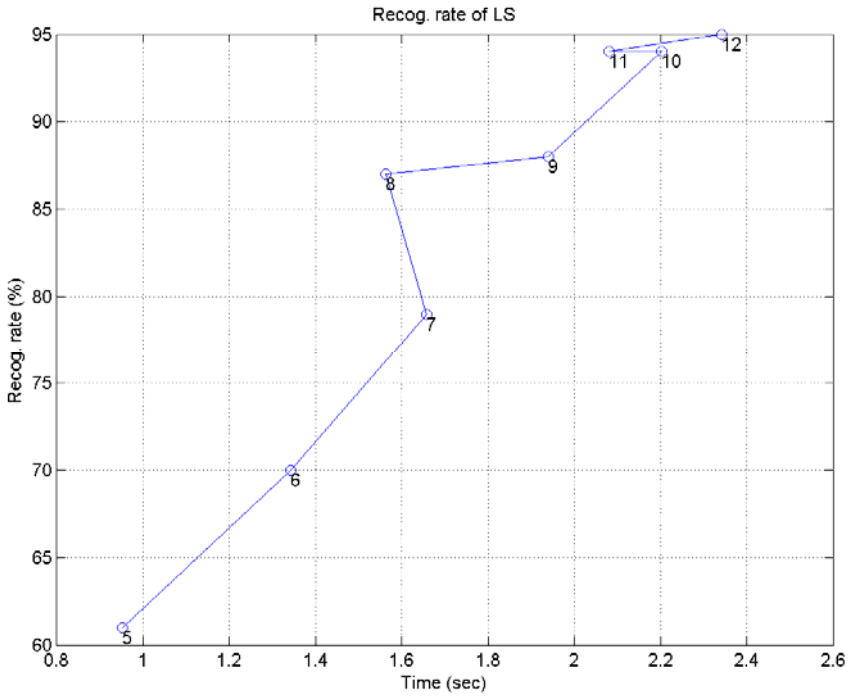
c. 請將你的函數用在最後一題的程式競賽，並只用一次的音高平移，看看辨識率有多高。

d. 請將你的函數用在最後一題的程式競賽，並用五次的音高平移（使用二元搜尋法），看看辨識率有多高。

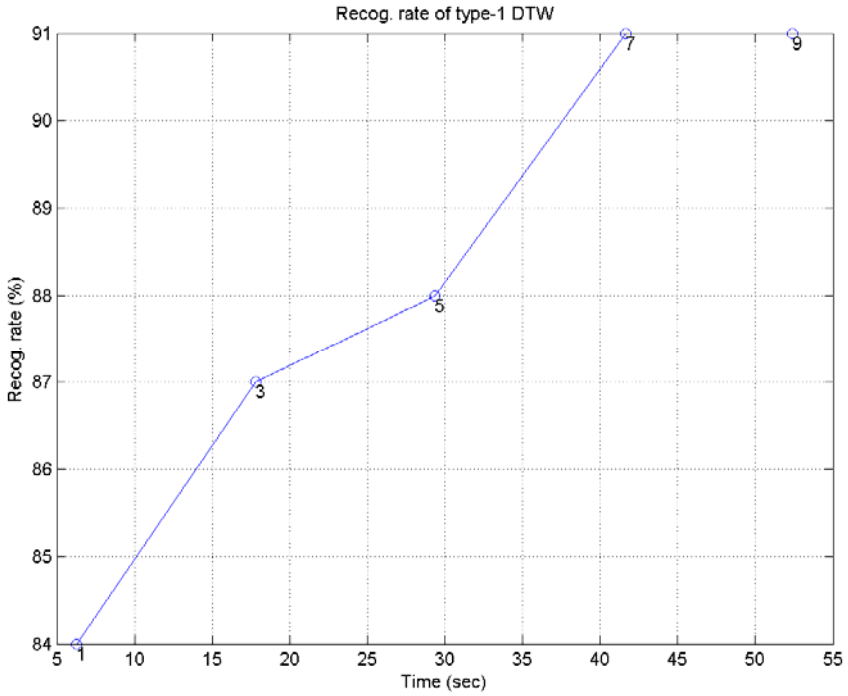
總共有三種 DTW 變形及兩種音調平移的方法，所以你應該共得到六個辨識率。

8. (***) **旋律辨識的效能測試之一：** 請先閱讀本節的最後一個習題，充分瞭解如何改變參數來進行辨識率的測試後，然後進行下列計算及作圖：

. 使用 LS，請畫出辨識率對辨識時間的作圖，並使用 resolution 為可變參數，使其從 5 增加到 12。畫出的圖應該類似下圖：



a. 使用 type-1 DTW，請畫出辨識率對辨識時間的作圖，並使用二元搜尋法的音調移位次數為可變參數（此參數也是每次比對一首 wav 檔案時，所必須執行 DTW 的個數），請使值為 1, 3, 5, 7, 9。畫出的圖應該類似下圖：



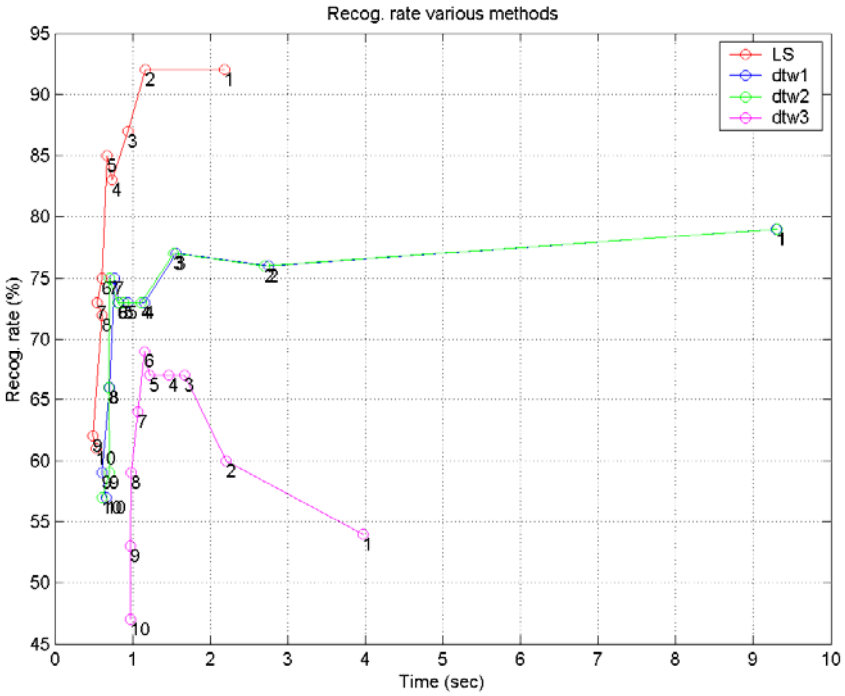
b. 請將上述兩圖同時畫出來，以進行 LS 和 DTW 在辨識效能（含辨識率及辨識時間）的比較。由此圖形，你可以得到什麼結論？

（若計算時間過於冗長，請採用 waveData 的前 100 筆資料來進行本作業即可。）

Hint

請將最後一題範例程式中所提供的辨識率測試主程式改寫為一個函式，才比較容易進行本題的計算和作圖。

9. (***) **旋律辨識的效能測試之二：** 請先閱讀本節的最後一個習題，充分瞭解如何改變參數來進行辨識率的測試後，然後再進行本題。我們標示的音高向量，每一點的持續時間都是 1/31.25 (= 256/8000) 秒，也就是說，每秒會有 31.25 個音高點。如果旋律辨識的計算時間太長，一個改進的方向，就是逐次降低音高向量在時間軸的解析度，換句話說，如果我們在原始的音高向量中，每 pvrr 點中只取用一點，則每秒的音高解析度降低至 31.25/pvrr 點。本題的目的，即是要探討 pvrr (pitch vector reduction ratio) 對辨識率及辨識時間的影響。請使用四種方法（LS, DTW1, DTW2, DTW3），畫出四條辨識率對辨識時間的曲線，並使用 pvrr 為可變參數。畫出的圖應該類似下圖：



（若計算時間過於冗長，請採用 waveData 的前 100 筆資料來進行本作業即可。）

Hint

請將最後一題範例程式中所提供的辨識率測試主程式改寫為一個函式，才比較容易進行本題的計算和作圖。

10. (***) **程式競賽：使用各種方法進行旋律辨識：** 請見此連結。

Chapter 17: HTK

HTK 是 Hidden Markov Model Toolkit 的簡稱，這是一套用於語音訓練與辨識的免費軟體，相關詳細說明，可見下列網址：

http://htk.eng.cam.ac.uk/

HTK 最早是由英國劍橋大學工程系（CUED, Cambridge University Engineering Department）的機器智能實驗室（Machine Intelligence Lab）所開發的軟體，在 1999 年，微軟買下擁有此軟體的 Entropic 公司，並在 2000 年將 HTK 定位為免費軟體，以便作為語音辨識的共同平台，來提升語音辨識等相關技術。因此目前我們可以直接由劍橋大學的網站下載所有 HTK 的原始碼及說明書。

語音辨識牽涉到相當高深的數學，相關程式碼也不容易撰寫，進入門檻很高，複雜度不易掌控。但自從 HTK 從 2000 年變成原始碼公開的免費軟體後，進入門檻大幅降低，快速提昇了語音技術的發展，目前國內外有關於語音技術的實驗和開發，大部分都是以 HTK 為主流。因此若要進行語音技術的研究，HTK 是一個不可或缺的工具。

本章將以幾個範例來說明 HTK 的使用，這些範例只是入門，讓各位讀者可以瞭解 HTK 的功能。若要掌握 HTK 來進行大量語料的訓練和測試，還是要熟讀 HTK 的手冊，並對語音辨識理論有紮實的基礎，才能夠事半功倍。

有關於 CHMM 的實作，我們大部分是採用 HTK (Hidden Markov Model Toolket) 來進行語料的整理和訓練。以下將使用一個簡單的範例，來說明 HTK 的使用。在這個範例中，我們將進行中文數字辨識（從 0 到 9）的工作，包含使用 HTK 進行聲學模型的訓練，並計算相關的辨識率。

本頁使用了幾個特定的附檔名（mlf、scp、template、cfg、pam、init、hmm、net、grammar 等），若要正確顯示這些檔案於瀏覽器，請取消這些附檔名所對應的應用程式，以便直接將檔案內容呈現於瀏覽器。（若不取消這些相關的應用程式，瀏覽器將跳出對話視窗，詢問是否進行下載。）你可以使用下例的 batch 檔案來取消這些附檔名和應用程式的關連：

原始檔（htk/delAssociation.bat）：（灰色區域按兩下即可拷貝）

```
assoc .mlf=
assoc .scp=
assoc .template=
assoc .cfg=
assoc .pam=
assoc .init=
assoc .hmm=
assoc .net=
assoc .grammar=
```

Hint

若要指定一個副檔名的關連應用程式，可用 assoc 命令。例如，若要取消副檔名為 scp 的相關應用程式，可以在 DOS 命令視窗下輸入「assoc .scp=」。

接著，你必須下載下列檔案：

- 下載 HTK 指令：展開後，有兩個目錄 bin 及 bin.win32，裡面包含我們會用到的 HTK 指令和其他資料處理指令。請由「控制台/系統/進階/環境變數/系統變數」，將這兩個目錄加入 path 的環境變數，以便後續由 DOS 視窗呼叫這些指令。
- 下載訓練指令及音訊檔案：展開後，有兩個目錄：
  - training：訓練所用到的程式碼。
  - waveFile：0 到 9 數字錄音的音訊檔案。

  請將上述目錄，放在沒有空格的路徑上。（請勿放在桌面，因為桌面的路徑有包含空格。）

請開啟 DOS 視窗，進入 chineseDigitRecog/training 目錄後，直接在 DOS 視窗下執行 goSyl13.bat，即可進行訓練，並計算辨識率。你也可以在 MATLAB 下，進入此目錄，並執行 goSyl13.m，可以得到相同的結果。

在進行所有的工作之前，我們要手動準備兩個檔案，第一個檔案是 digitSyl.pam，指明如何將每一個數字的拼音（Phonetic Alphabets）拆解成聲學模型（Acoustic Models），我們目前的作法，是將每一個音節看成一個聲學模型，如下：

原始檔（htk/chineseDigitRecog/training/digitSyl.pam）：（灰色區域按兩下即可拷貝）

```
ba      ba
er      er
jiou    jiou
ling    ling
liou    liou
qi      qi
san     san
si      si
sil     sil
wu      wu
i       i
```

以上的拆解方法，是屬於比較簡略的方法，所得到的辨識效果也會比較差。後續會再說明比較精細的方法，例如以 Monophone 或是 Biphone 來做為聲學模型。

Hint

pam 代表 phonetic alphabets to model。

第二個檔案則是 digitSyl.mlf，紀錄每一個音訊檔案所對應的文句內容（以音節為單位），如下：

Example（htk/chineseDigitRecog/training/digitSyl.mlf）：

在上述檔案中，sil 代表靜音，因此當我們在 ling 前後都會加上 sil，就代表 "0" 的發音前後都有靜音。

Hint

- mlf 代表 master label file，用來記錄每個語音檔案的對應內容，可以使用音節（Syllables）或是音素（Phoneme）為單位。
- 由於 HTK 是以檔案名稱來對應至文句內容，所以每一個音訊特徵的檔案名稱，必須是獨一無二的。（音訊檔案之名稱，則不需唯一。）

以下將將針對 MATLAB 指令檔 goSyl13.m 及 Batch 指令檔 goSyl13.bat 來進行說明，這邊共包含了三大項工作：

I. 語音特徵的擷取：計算 MFCC
II. 聲學模型的訓練：使用 EM 來求取最佳參數
III. 聲學模型的效能評估：辨識率的計算

分別說明如下。

I. 語音特徵的擷取：計算 MFCC

1. 產生輸出目錄

我們先產生三個放置輸出檔案的目錄：

- output：放置各種輸出檔案。
- output\feature：放置語音特徵檔案。
- output\hmm：放置訓練過程所產生的 HMM 參數檔案。

MATLAB 指令如下：

```
mkdir('output');
mkdir('output/feature');
mkdir('output/hmm');
```

Batch 指令如下：

```
for %%i in (output output\feature output\hmm) do mkdir %%i > nul 2>&1
```

若目錄已經存在，則 Batch 指令不會印出任何警告訊息。

Hint

上述的 Batch 指令，是位於 goSyl13.bat 的內容。讀者若要將上述指令拷貝至 DOS 視窗執行，請務必將 %%i 改成 %i，以下類推。

2. 產生 digitSyl.mnl 及 digitSylPhone.mlf

先產生 syl2phone.scp 檔案，MATLAB 指令如下：

```
fid=fopen('output\syl2phone.scp', 'w'); fprintf(fid, 'EX'); fclose(fid);
```

3. Batch 指令如下：

```
@echo EX > output\syl2phone.scp
```

所產生的 syl2phone.scp 檔案內容如下：

原始檔（htk/chineseDigitRecog/training/output/syl2phone.scp）：（灰色區域按兩下即可拷貝）

```
EX
```

其中的「EX」是代表 Expand，其意義是「由音節字串展開成 phone 字串」，將被用於 HTK 的 HLEd 指令，詳見下述。

接著，我們可以使用下列 HTK 的 HLEd 指令來產生 digitSyl.mnl 及 digitSylPhone.mlf：

```
HLEd -n output\digitSyl.mnl -d digitSyl.pam -l * -i output\digitSylPhone.mlf
output\syl2phone.scp digitSyl.mlf
```

在上述指令中，我們使用顏色來表示檔案的用途，藍色代表是輸入檔案，紅色代表是輸出檔案。輸出檔案 digitSyl.mnl 列出所有用到的聲學模型，如下：

原始檔（htk/chineseDigitRecog/training/output/digitSyl.mnl）：（灰色區域按兩下即可拷貝）

```
sil
ling
```

```
i
er
san
si
wu
liou
qi
ba
jiou
```

Hint

mnl 代表 model name list。

而 digitSylPhone.mlf 則是將 digitSyl.mlf 的音節資訊轉換成為 phone 資訊的結果，以便用於聲學的訓練，檔案如下：

Example（htk/chineseDigitRecog/training/output/digitSylPhone.mlf）：

Hint

由於我們是使用一個音節來作為一個聲學模型，因此 digitSylPhone.mlf 和原先的 digitSyl.mlf 的內容恰巧一樣。

4. 產生 wav2fea.scp

接著我們要進行語音特徵擷取，也就是要將每一個 wav 檔案轉換成 MFCC，首先我們先產生 wav2fea.scp，MATLAB 指令如下：

```
5.  wavDir='..\waveFile';
6.  waveFiles=recursiveFileList(wavDir, 'wav');
7.  outFile='output\wav2fea.scp';
8.  fid=fopen(outFile, 'w');
9.  for i=1:length(waveFiles)
10.         wavePath=strrep(waveFiles(i).path, '/', '\');
11.         [a,b,c,d]=fileparts(wavePath);
12.         fprintf(fid, '%s\t%s\r\n', wavePath, ['output\feature\', b, '.fea']);
13. end
14. fclose(fid);
```

Batch 指令如下：

```
(for /f "delims=" %%i in ('dir/s/b wave\*.wav') do @echo %%i
output\feature\%%~ni.fea)> output\wav2fea.scp
```

所產生的 wav2fea.scp 用來規範 wav 檔案和所產生的音訊特徵檔案的對應關係，其內容如下：

Example（htk/chineseDigitRecog/training/output/wav2fea.scp）：

由上述檔案可以看出，所有產生的語音特徵檔案，將以 fea 為附檔名，並放置在 output\feature 目錄下。

15. 使用 HCopy.exe 進行語音特徵抽取

接著，我們就可以使用 HTK 的 hcopy.exe 指令來產生 MFCC 檔案：

```
16. HCopy -C mfcc.cfg -S output\wav2fea.scp
```

在上述指令中，mfcc.cfg 是一個參數檔，用來規範產生 MFCC 的參數，其內容如下： Error: D:\users\jang\books\audioSignalProcessing/example/htk/chineseDigitRecog/training/mfcc.cfg does not exist!

在上一節中，我們已經使用中文數字辨識來說明了 HTK 的基本使用方式，在本節及以下各節中，我們將說明如何改變各種參數（包含語音特徵、聲學模型架構等），以便改進辨識率。

首先，我們將基本訓練及測試的程式包成一個函數 htkTrainTest.m，以便能夠反覆呼叫，此函數接收一個結構變數 htkParam，包含訓練所用到的各種參數。
首先，在聲學模型架構不變的情況下，我們可以改變語音特徵，上一節範例所用的語音特徵是 13 維的 MFCC_E（MFCC & Energy），我們可以加到 26 維的 MFCC_E_D 或是 MFCC_E_D_Z，也可以加到 39 維的 MFCC_E_D_A 或 MFCC_E_D_A_Z。其中

- E: Energy
- D: Delta 項目，代表一次差量。
- A: Acceleration 項目，到表二次差量。
- Z: 代表會進行 Cepstrum Mean Subtraction (CMS)。

若使用 26 維的 MFCC_E_D_Z，可見下列範例：
Example 1**Input file** htk/chineseDigitRecog/training/goSyl26.m

```
htkParam=htkParamSet;
htkParam.pamFile='digitSyl.pam';
htkParam.feaCfgFile='mfcc26.cfg';
htkParam.feaType='MFCC_E_D_Z';
htkParam.feaDim=26;
htkParam.streamWidth=[26];
disp(htkParam)
[trainRR, testRR]=htkTrainTest(htkParam);
fprintf('Inside test = %g%%, outside test = %g%%\n', trainRR, testRR);
```

Output message

```
          pamFile: 'digitSyl.pam'
      feaCfgFile: 'mfcc26.cfg'
         waveDir: '..\waveFile'
       sylMlfFile: 'digitSyl.mlf'
    phoneMlfFile: 'digitSylPhone.mlf'
         mnlFile: 'digitSyl.mnl'
     grammarFile: 'digit.grammar'
         feaType: 'MFCC_E_D_Z'
          feaDim: 26
      mixtureNum: 3
        stateNum: 3
     streamWidth: 26

Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Inside test = 91.29%, outside test = 92.86%
```

對應的 batch 檔案，請見 goSyl26.bat。
若使用 39 維的 MFCC_E_D_A_Z，請見下列範例：
Example 2**Input file** htk/chineseDigitRecog/training/goSyl39.m

```
htkParam=htkParamSet;
htkParam.pamFile='digitSyl.pam';
htkParam.feaCfgFile='mfcc39.cfg';
htkParam.feaType='MFCC_E_D_A_Z';
```

```
htkParam.feaDim=39;
htkParam.streamWidth=[39];
disp(htkParam)
[trainRR, testRR]=htkTrainTest(htkParam);
fprintf('Inside test = %g%%, outside test = %g%%\n', trainRR, testRR);
```

Output message

```
          pamFile: 'digitSyl.pam'
       feaCfgFile: 'mfcc39.cfg'
          waveDir: '..\waveFile'
       sylMlfFile: 'digitSyl.mlf'
     phoneMlfFile: 'digitSylPhone.mlf'
          mnlFile: 'digitSyl.mnl'
      grammarFile: 'digit.grammar'
          feaType: 'MFCC_E_D_A_Z'
           feaDim: 39
       mixtureNum: 3
         stateNum: 3
      streamWidth: 39

Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Inside test = 91.07%, outside test = 92.86%
```

對應的 batch 檔案，請見 goSyl39.bat。
在對應的 batch 檔案方面，我們並沒有包成函數來執行，所以其內容看起來會比較繁複。從 goSyl13.bat 演變到 goSyl26.bat，事實上只改變了兩列，讀者可以使用下列指令來比較其差異：

```
fc goSyl13.bat goSyl26.bat
```

利用相同的方法，也可以比較 goSyl26.bat 和 goSyl39.bat 的差異。
在前幾節的說明，我們是以一個中文音節來作為一個語音模型（Acoustic Model），在本節中，我們將每一個音節拆成 Phone，並以此 Phone 為語音模型，此種拆解方式稱為 Monophone，以別於右相關（Right-context dependent, RCD）的 Biphone。這些資訊是記錄在 digitMonophone.pam，如下
原始檔（htk/chineseDigitRecog/training/digitMonophone.pam）：（灰色區域按兩下即可拷貝）

```
ba      b a
er      er
jiou    j i o u
ling    l i ng
liou    l i o u
qi      q i
san     s a n
si      s i
sil     sil
wu      w u
i       i
```

因此只需將前幾節範例中的 digitSyl.pam 改為 digitMonophone.pam，即可進行訓練及辨識率測試，請見下列範例：

Example 1**Input file** htk/chineseDigitRecog/training/goMonophone13.m

```
htkParam=htkParamSet;
htkParam.pamFile='digitMonophone.pam';
htkParam.phoneMlfFile='digitMonophone.mlf';
htkParam.mnlFile='digitMonophone.mnl';
disp(htkParam)
[trainRR, testRR]=htkTrainTest(htkParam);
fprintf('Inside test = %g%%, outside test = %g%%\n', trainRR, testRR);
```

Output message

```
          pamFile: 'digitMonophone.pam'
       feaCfgFile: 'mfcc.cfg'
          waveDir: '..\waveFile'
       sylMlfFile: 'digitSyl.mlf'
     phoneMlfFile: 'digitMonophone.mlf'
          mnlFile: 'digitMonophone.mnl'
      grammarFile: 'digit.grammar'
          feaType: 'MFCC_E'
           feaDim: 13
       mixtureNum: 3
         stateNum: 3
      streamWidth: 13

Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Inside test = 79.24%, outside test = 75.89%
```

此時所產生的 Monophone 列表如下：

原始檔（htk/chineseDigitRecog/training/output/digitMonophone.mnl）：（灰色區域按兩下即可拷貝）

```
sil
l
i
ng
er
s
a
n
w
u
o
q
b
j
```

而對應於 Monophone 的 mlf 檔案如下：
Example（htk/chineseDigitRecog/training/output/digitMonophone.mlf）：
若改用 26 維的 MFCC，可見下列範例：
Example 2**Input file** htk/chineseDigitRecog/training/goMonoPhone26.m

```
htkParam=htkParamSet;
htkParam.pamFile='digitMonophone.pam';
htkParam.phoneMlfFile='digitMonophone.mlf';
htkParam.mnlFile='digitMonophone.mnl';
htkParam.feaCfgFile='mfcc26.cfg';
htkParam.feaType='MFCC_E_D_Z';
htkParam.feaDim=26;
htkParam.streamWidth=[26];
disp(htkParam)
[trainRR, testRR]=htkTrainTest(htkParam);
fprintf('Inside test = %g%%, outside test = %g%%\n', trainRR, testRR);
```

Output message

```
          pamFile: 'digitMonophone.pam'
       feaCfgFile: 'mfcc26.cfg'
          waveDir: '..\waveFile'
       sylMlfFile: 'digitSyl.mlf'
     phoneMlfFile: 'digitMonophone.mlf'
          mnlFile: 'digitMonophone.mnl'
      grammarFile: 'digit.grammar'
          feaType: 'MFCC_E_D_Z'
           feaDim: 26
       mixtureNum: 3
         stateNum: 3
      streamWidth: 26

Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Inside test = 83.71%, outside test = 87.5%
```

若改用 39 維的 MFCC，可見下列範例：
Example 3**Input file** htk/chineseDigitRecog/training/goMonoPhone39.m

```
htkParam=htkParamSet;
htkParam.pamFile='digitMonophone.pam';
htkParam.phoneMlfFile='digitMonophone.mlf';
htkParam.mnlFile='digitMonophone.mnl';
htkParam.feaCfgFile='mfcc39.cfg';
htkParam.feaType='MFCC_E_D_A_Z';
htkParam.feaDim=39;
htkParam.streamWidth=[39];
disp(htkParam)
[trainRR, testRR]=htkTrainTest(htkParam);
```

```
fprintf('Inside test = %g%%, outside test = %g%%\n', trainRR, testRR);
```

Output message

```
          pamFile: 'digitMonophone.pam'
       feaCfgFile: 'mfcc39.cfg'
          waveDir: '..\waveFile'
       sylMlfFile: 'digitSyl.mlf'
     phoneMlfFile: 'digitMonophone.mlf'
          mnlFile: 'digitMonophone.mnl'
      grammarFile: 'digit.grammar'
          feaType: 'MFCC_E_D_A_Z'
           feaDim: 39
       mixtureNum: 3
         stateNum: 3
      streamWidth: 39

Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Inside test = 84.6%, outside test = 89.29%
```

在這一節中，我們將嘗試改變 Mixture 的個數，並同時改變 MFCC 的維度。

首先我們看一個範例，使用 13 維 MFCC，並畫出 Inside & Outside 辨識率對 Mixture 個數的變化，如下：

Example 1**Input file** htk/chineseDigitRecog/training/htkMixture01.m

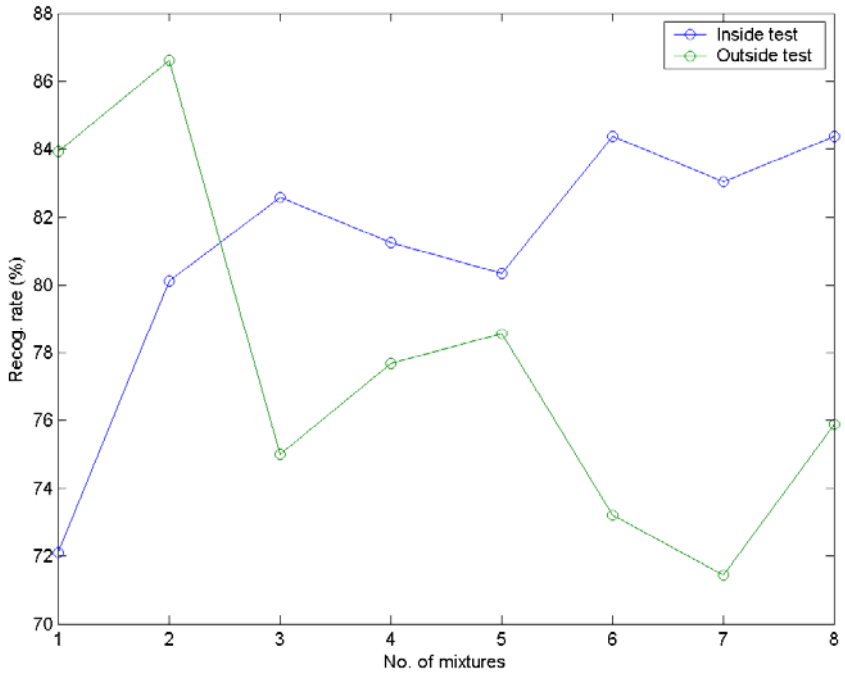
```
htkParam=htkParamSet;
maxMixNum=8;

for i=1:maxMixNum
        htkParam.mixtureNum=i;
        fprintf('====== %d/%d\n', i, maxMixNum);
        [trainRR(i), testRR(i)]=htkTrainTest(htkParam);
end

plot(1:maxMixNum, trainRR, 'o-', 1:maxMixNum, testRR, 'o-');
xlabel('No. of mixtures'); ylabel('Recog. rate (%)');
legend('Inside test', 'Outside test');
```

Output message

```
====== 1/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 2/8
Pruning-Off
Pruning-Off
Pruning-Off
```

```
Pruning-Off
Pruning-Off
====== 3/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 4/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 5/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 6/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 7/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 8/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
```

Output figure



上述範例中，我們把 HTK 的訓練和辨識都包在一個函數 htkTrainTest.m，以方便反覆呼叫。事實上我們可以不必重複 MFCC 的計算，只需要算一次就好了，這樣比較能夠減少計算時間。
我們也可以使用 13/26/39 維 MFCC，並畫出 Inside & Outside 辨識率對 Mixture 個數的變化，如下：

Example 2**Input file** htk/chineseDigitRecog/training/htkMixtureMfcc01.m

```
% Get the RR when feature dim. and mixture no. are changing

htkParam=htkParamSet;
maxMixNum=8;

for i=1:maxMixNum
        htkParam.mixtureNum=i;
        fprintf('====== %d/%d\n', i, maxMixNum);
        [trainRR(i,1), testRR(i,1)]=htkTrainTest(htkParam);
end

htkParam.feaCfgFile='mfcc26.cfg';
htkParam.feaType='MFCC_E_D_Z';
htkParam.feaDim=26;
htkParam.streamWidth=[26];
for i=1:maxMixNum
        htkParam.mixtureNum=i;
        fprintf('====== %d/%d\n', i, maxMixNum);
        [trainRR(i,2), testRR(i,2)]=htkTrainTest(htkParam);
end

htkParam.feaCfgFile='mfcc39.cfg';
htkParam.feaType='MFCC_E_D_A_Z';
htkParam.feaDim=39;
```

```
htkParam.streamWidth=[39];
for i=1:maxMixNum
        htkParam.mixtureNum=i;
        fprintf('====== %d/%d\n', i, maxMixNum);
        [trainRR(i,3), testRR(i,3)]=htkTrainTest(htkParam);
end

plot(   1:maxMixNum, trainRR(:,1), '^-b', 1:maxMixNum, testRR(:,1), 'o-b', ...
        1:maxMixNum, trainRR(:,2), '^-g', 1:maxMixNum, testRR(:,2), 'o-g', ...
        1:maxMixNum, trainRR(:,3), '^-r', 1:maxMixNum, testRR(:,3), 'o-r');
xlabel('No. of mixtures'); ylabel('Recog. rate (%)');
legend('13D, Inside test', '13D, Outside Test', '26D, Inside Test', '26D, Outside test', '39D, Inside test', '39D,
Outside test', 'Location', 'BestOutside');
```

Output message

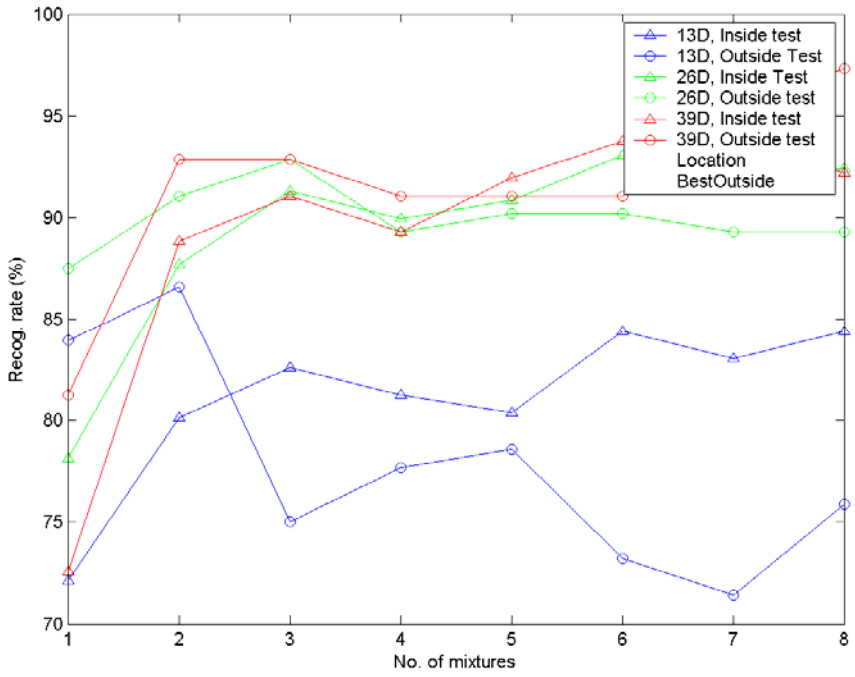
```
====== 1/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 2/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 3/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 4/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 5/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 6/8
Pruning-Off
Pruning-Off
Pruning-Off
```

Pruning-Off
Pruning-Off
====== 7/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 8/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 1/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 2/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 3/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 4/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 5/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 6/8
Pruning-Off
Pruning-Off
Pruning-Off

Pruning-Off
Pruning-Off
====== 7/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 8/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 1/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 2/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 3/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 4/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 5/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 6/8
Pruning-Off
Pruning-Off
Pruning-Off

```
Pruning-Off
Pruning-Off
====== 7/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 8/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
```

Output figure



上述範例的計算時間比較久，請耐心等候。

1. (**) 中文數字辨識之一： 請仿照 goSyl13.bat 或是 goSyl13.m，使用 2003 年的數字錄音為訓練資料，2006 年的數字錄音為測試資料，求取相關的 inside-test 和 outside-test 辨識率。事實上，主要的工作都是在準備相關檔案，因此你的工作就是要寫一個 MATLAB 程式 goGenFile4htk.m，能夠從指定的 wav 檔案資料夾，抓取 wav 檔案資料，並產生下列檔案，以供 HTK 訓練：
    a. 產生 digitSyl.mlf 檔案。
    b. 產生 wav2fea.scp 檔案。
    c. 產生 trainFea.scp 及 testFea.scp 檔案。

    最後再進行訓練與辨識率測試。（建議作法：請先產生上述四個檔案，再修改 goSyl13.bat 或是 goSyl13.m，將即時產生上述檔案之程式碼註解掉，然後執行程式，就可以產生使用上述四個新檔案的結果。）。

請問使用 13, 26, 39 維的 MFCC，inside-test 和 outside-test 的辨識率各是多少？請將 Confusion Matrix 展示給助教看。我得到的結果是：

- 以 goSyl13.bat 為基礎，新資料的結果是：inside test 86.38%, outside test 79.51%。
- 以 goSyl26.bat 為基礎，新資料的結果是：inside test 92.07%, outside test 87.25%。
- 以 goSyl39.bat 為基礎，新資料的結果是：inside test 95.17%, outside test 89.53%。

請注意：

- 特徵檔案的檔名不可使用中文，因為 HTK 不支援。
- 每一個語音特徵檔案的檔名，必須是獨一無二的。因此你必須先將中文目錄轉成數字，再加上原來的檔名，就可以用於特徵檔案的檔名，例如「912508 鄒銘軒\3a_7436_16017.wav ===> 00002-3a_7436_16017.fea」。
- HTK 會分辨大小寫，所以檔名和相關檔案的紀錄必須一致。
- 相關錄音資料會在課堂上由助教提供。（本次 ftp 位置是 ftp://AP2006:ap2006@140.114.88.80:3524。）

提示：可以使用 recursiveFileList.m 來回傳一個目錄下所有的 wav 檔案。

2. (***) **中文數字辨識之二：** 請重複上題，但嘗試各種方法，以求得 outside-test 及 inside-test 平均辨識率之最佳值。請將對應於最佳值的各種相關辨識參數（語音特徵維度、聲學模型、mixture 個數、狀態個數...）記錄於 method.txt，並說明你用了什麼方法。請將 method.txt、你最後產生的 macro 檔案（請改名為 digit.mac）、以及跑辨識率所需的相關資料檔案，一併上傳給助教，讓助教來進行測試，並評比每一個人的辨識率。（本次 ftp 位置是 ftp://AP2006:ap2006@140.114.88.80:3524。）

Chapter 18: 語音辨識前處理

Chapter 18: 語音辨識前處理

18-1 簡介

給一組可辨識的詞彙或文句，在實際進行辨識之前，還有許多工作要進行：

1. 對文句進進行標音：中文是標示注音，英文是標示音標。
2. 產生辨識網路：根據需求，產生辨識網路，以便進行搜尋。

本章將針對這些步驟進行說明。

18-2 文字標音

通常我們將可辨識的語句放在一個文字檔案，以唐詩為三百首例，若希望我們的辨識系統能夠辨識使用者以語音輸入的一句唐詩，那我們共可抽出 3221 句，放在 tangPoem.rt 檔案內，如下：

Example（tangPoem.rt）：

Hint

rt 代表 recognizable text。

對電腦而言，第一個步驟，就是要知道這些文句如何發音。如果 RT 的數量不大，我們當然可以使用手工來標注音，但是對於大量的 RT 而言，就要靠電腦來自動進行標音工作。首先，電腦必須知道每一個國字的注音，於是我們必須有一個 WPA 檔案，中文範例如下：

```
...
邶      bei     ㄅㄟ
采      cai     ㄘㄞ
金      jiN     ㄐㄧㄣ
```

```
長      JaG     ㄓㄤˇ
長      CaG     ㄔㄤˇ
門      mrN     ㄇㄣˊ
阜      fu      ㄈㄨˋ
...
```

Hint

wpa 代表 word to phonetic alphabet，也就是由字到注音或音標的對照表。

在上述範例中，第一欄是國字，第二欄是對應於注音的音節代號，第三欄則是國語注音。由於語音辨識系統目前並沒有用到音調資訊，所以上述的表格並沒有包含音調資訊。此外，此表格也列出了破音字，例如「長」的發音有兩個。英文範例如下：

```
...
eiszner   ay z n er
eitel     ay t ah l
either    iy dh er#ay dh er
eitzen    ay t z ah n
ejaculate ih jh ae k y uw l ey t
...
```

在上述範例中，每一個英文詞彙就對應到一個發音（以音標代號表示，例如 ae 就是蝴蝶音），同樣也會發生破音字，例如 either 就有兩種發音，此時我們使用 # 來分隔這兩種發音。

Hint

通常我們不直接使用注音符號或英文音標，因為文字格式比較難處理，因此我們多用音節代號或音標代號來處理。

有了 WPA 檔案後，我們就可以進行「暴力法」的注音，以唐詩的「朝辭白帝彩雲間」來說，根據破音字發音的各種組合就可以標示成四種發音：

a. 朝（ㄓㄠ ）辭白（ㄅㄞˊ）帝彩雲間
b. 朝（ㄓㄠ ）辭白（ㄅㄛˊ）帝彩雲間
c. 朝（ㄔㄠˊ）辭白（ㄅㄞˊ）帝彩雲間
d. 朝（ㄔㄠˊ）辭白（ㄅㄛˊ）帝彩雲間

而「千里江崚一日還」則可標示成

a. 千里江崚一日還（ㄏㄨㄢˊ）
b. 千里江崚一日還（ㄏㄞˊ）
c. 千里江崚一日還（ㄒㄩㄢˊ）

因此我們可以根據此種暴力法，產生 SYL 檔案，此檔案以音節代號列出每一句可能的發音，範例如下：

```
...
Cau-cy-bai-di-cai-YN-jieN  56
Cau-cy-buo-di-cai-YN-jieN 56
Jau-cy-bai-di-cai-YN-jieN   56
Jau-cy-buo-di-cai-YN-jieN  56
cieN-li-jiaG-lG-i-Ry-hai      57
cieN-li-jiaG-lG-i-Ry-huaN   57
cieN-li-jiaG-lG-i-Ry-sYaN  57
...
```

在上述範例中，前四句是「朝辭白帝彩雲間」的所有可能發音，而後三句則是「千里江崚一日還」的所有可能發音，56 與 57 則是代表這兩句在 rt 檔案出現的位置（列數），以便反查。此種「暴力注音法」的優缺點如下：

- 優點：列出所有可能的發音，不會發生標注音出錯的情況。（這種情況特別適用於標示人名的發音，例如我有位大學同學的大名是「陸重和」，在無法確認使用者會如何發音的情況下，只有列舉所有可能的發音。）
- 缺點：語音搜尋範圍變大，計算時間比較久，而且也產生一些不可能的發音，可能會降低辨識率。

另外一種方法，則是「詞庫標音法」，我們必須先見一個破音字的詞庫，把所有破音字可能出現的詞彙列出來，例如：

```
...
口吃　0073-1252
吃軟不吃硬　0021-3423-2094-0021-2814
吃喝玩樂　0021-0371-3382-0414
吃飯　0021-1284
吃裡扒外　0021-1903-0232-3184
...
```

以「口吃」來說，對應的資料是：

1. 007（ㄎㄡˇ的音碼）3（第三聲）
2. 125（ㄐㄧˊ的音碼）2（第二聲）

有了破音字詞庫後，我們就可以進行「詞庫標音法」，有兩種作法：

1. Maximum matching：
   a. 從頭比對：我喜歡到廟口吃小吃 ===> 我|喜歡|到|廟口|吃|小吃
   b. 從尾比對：我喜歡到廟口吃小吃 ===> 我|喜歡|到|廟|口吃|小吃
2. Dynamic programming：（細節待補充）

「詞庫標音法」的優缺點如下：

- 優點：不會產生不可能的發音。
- 缺點：還是可能會出錯，例如
  - 搶詞：例如「我喜歡到廟口吃小吃」中的「廟口」和「口吃」。
  - 無法由詞庫判斷：「我還你 10 元」、「我還欠你 10 元」中間的「還」的發音，必須靠詞性或語意分析，才能標示出正確發音。另外，一般人名的發音，也很難從詞庫來判斷。

在英文方面，則會遇到新的詞彙，這些詞彙可能不出現於 WPA 檔案之中，例如商標名 Benq 或 google，或是人名 Schwarzenegger 或地名 Rocktop 等，此時就需要人工輔助標示，或是由電腦進行發音的預測。

18-3 辨識網路

一般辨識方法，是針對每一句可辨識語句建立一個 HMM，然後再使用 Viterbi Search 來計算每一個 HMM 的機率值。根據此種方式，我們可以建立一個語句網路（Lexicon Net），來規範 Viterbi Search 計算中，可能產生的辨識語句，主要可以分為三類：

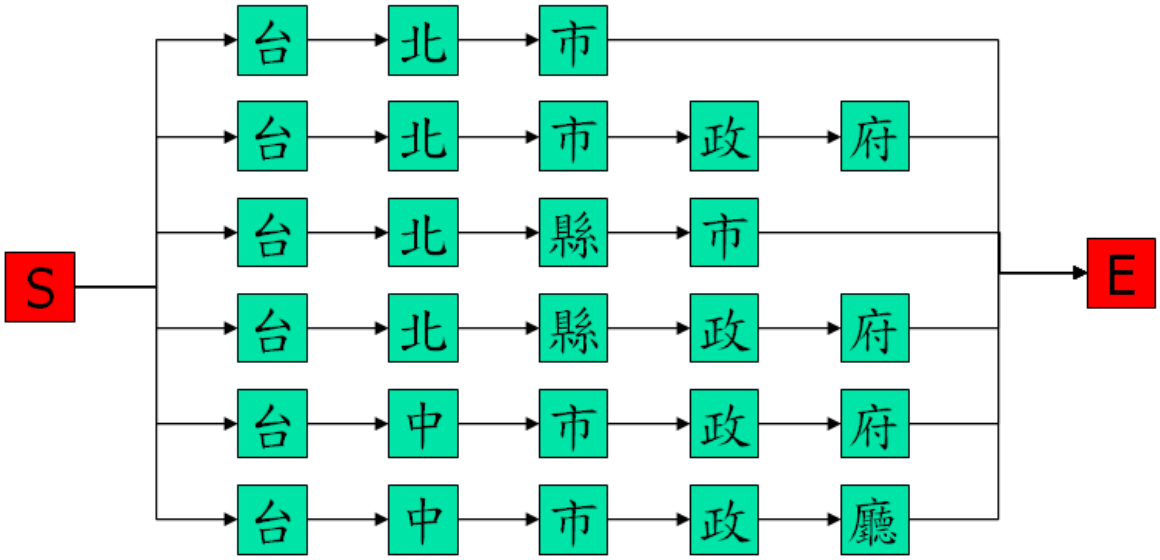
a. Linear Net
b. Tree Net
c. Double-ended Tree Net

以下列可辨識語句為範例：

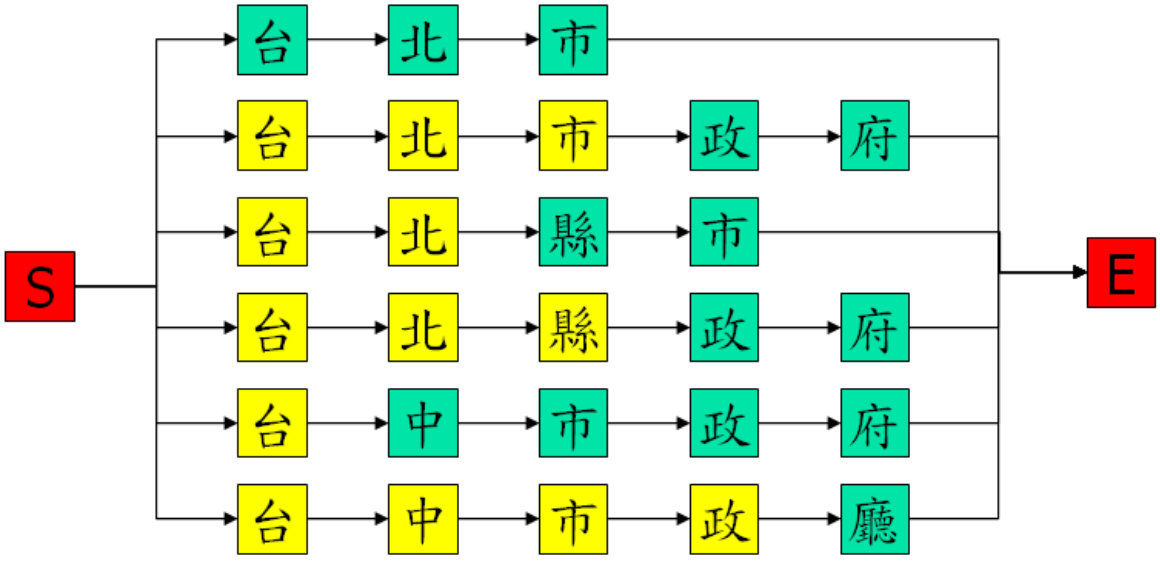
原始檔（政府機關名稱.rt）：（灰色區域按兩下即可拷貝）

```
台北市
台北市政府
台北縣市
台北縣政府
台中市政府
台中市政廳
```

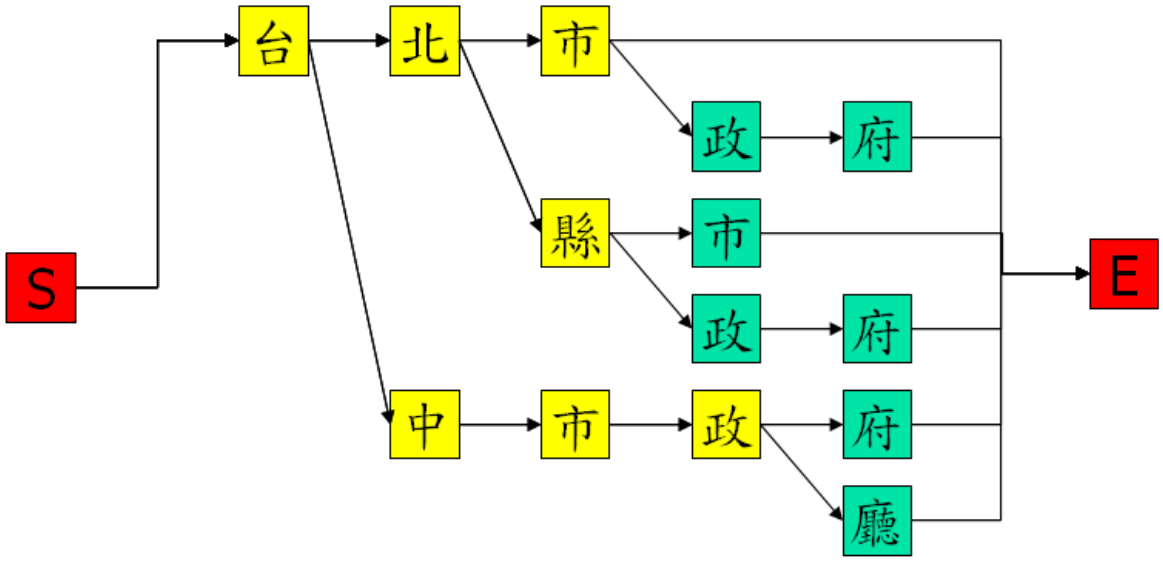
我們可以產生最簡單的 linear net，圖示如下：



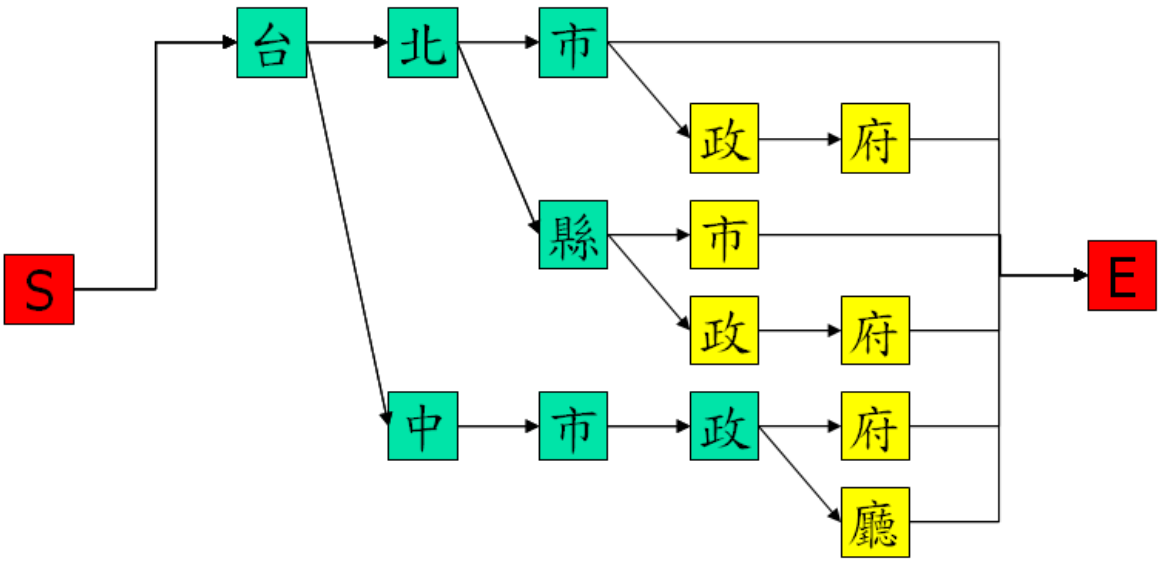
如果將每一條路徑向左對齊，並進行排序，可以找出重複的節點，如下圖之黃色節點：



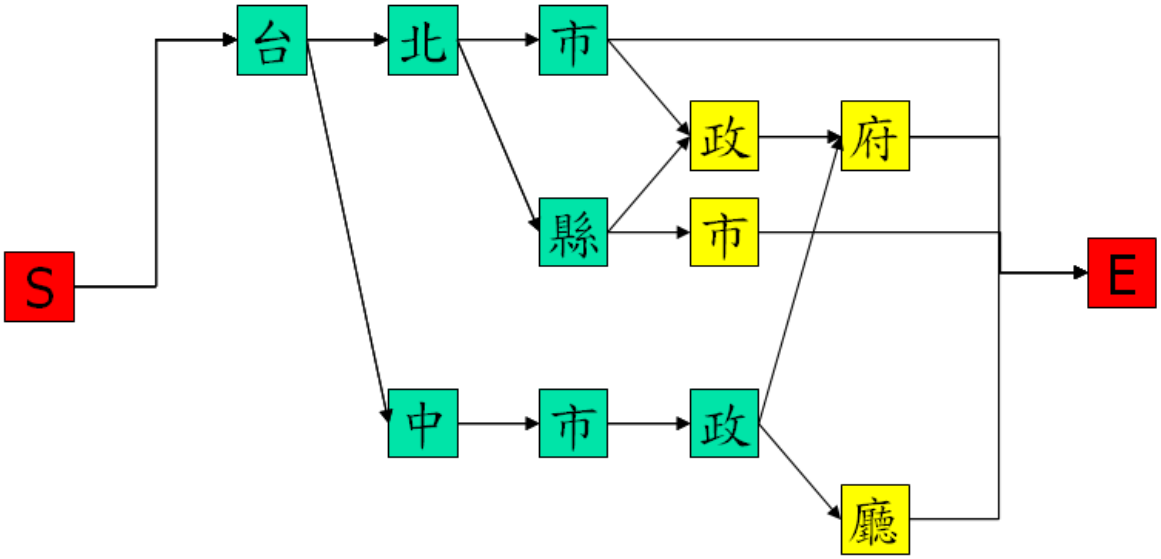
此時我們可以將這些在同一欄且發音相同的黃色節點合併成一個節點，如下：



接著，從每一條路徑的尾端來看，我們可以往回走，找出為分叉之前的節點，如下圖之黃色節點：



若從尾端來合併這些節點，可以得到如下圖的 double-ended tree net：



在上述網路結構的簡化過程中，我們必須把握一個原則：簡化後的網路，其所有可能的路徑應該和原來的網路結構相同。換句話說，無論是 linear net、tree net 或是 double-ended tree net，其所有路徑所成的集合是完全一樣的。

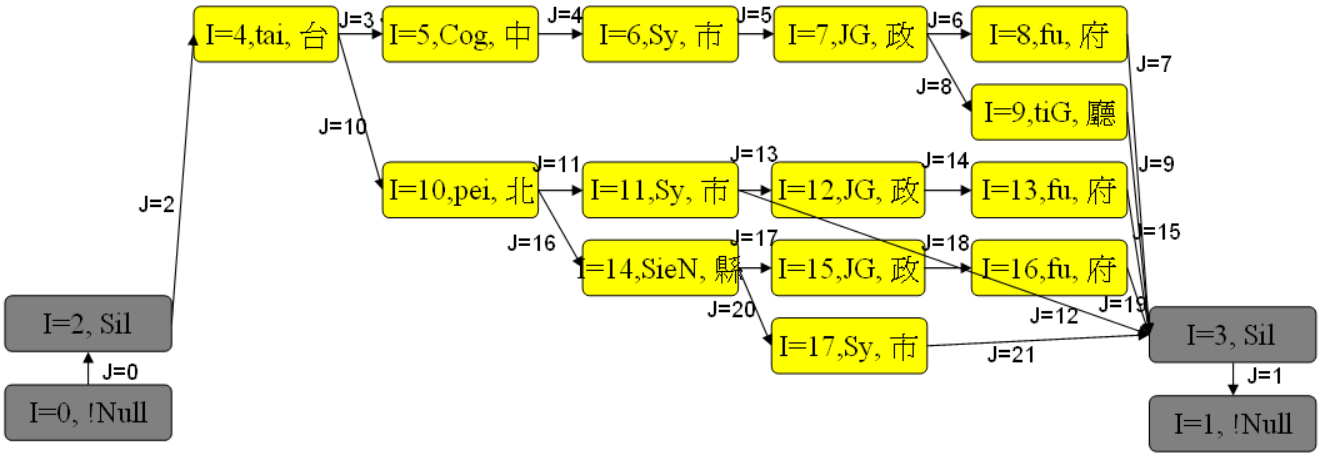
在上述說明中，我們是將 linear net 中的所有路徑向左對齊來進行排序，如果我們改成向右對齊來進行排序，也可以得到另一組 tree net 及 double-end tree net。

至於是否存在一種網路結構的化簡方法，可以在多項式時間內完成計算，並可以保證擁有最少數目的節點，則目前無法得知。（我對演算法並不熟悉，若讀者有相關資訊，歡迎提供。）

根據上述機關名稱所產生的 tree net，可以表示成下列 net 檔案：

Example（政府機關名稱 treeNet.net）：

在上述範例中，「N=18」代表有 18 個節點（Nodes），「L=22」代表有 22 條連結（Links），「I=4 W=tai」則是說明第 4 個節點的發音是 tai，「J=16 S=10 E=14」則是記錄第 16 條連結的開始位置是節點 10，結束位置是節點 14，餘類推。相關的圖示如下：



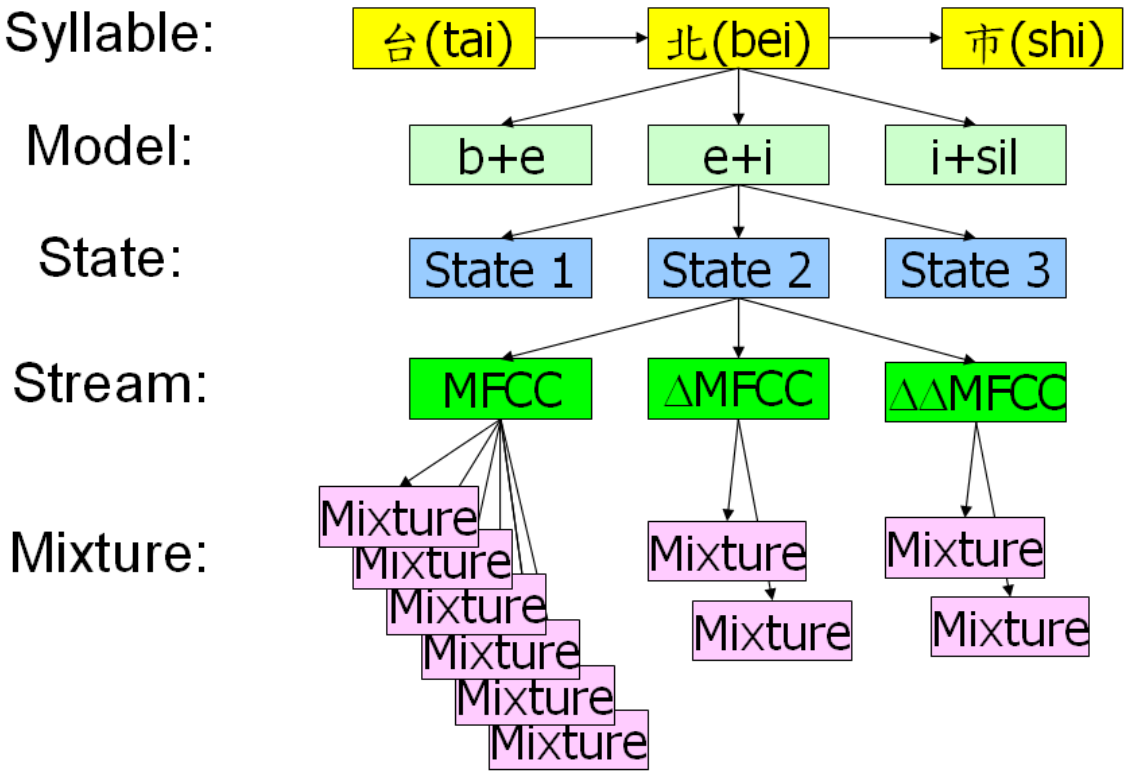
相關投影片請見此連結。

18-4 聲學模型

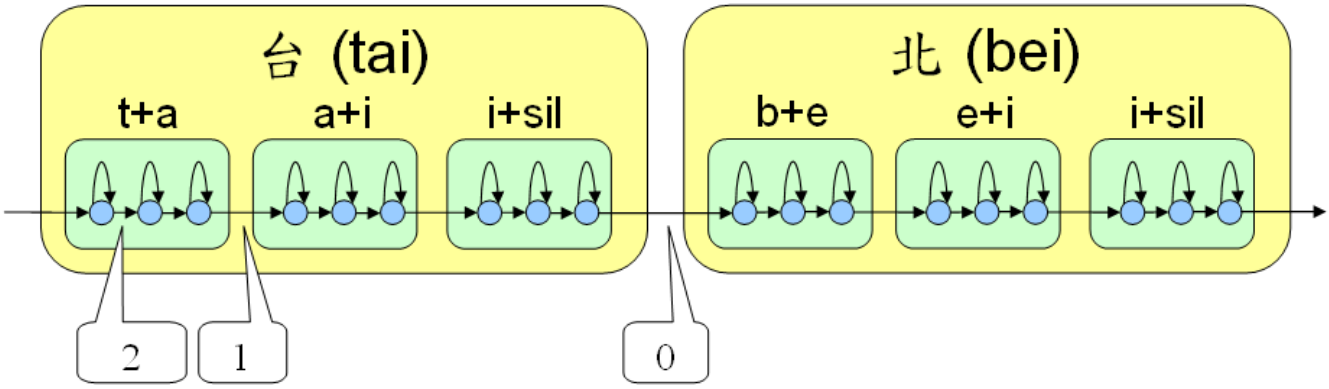
一般語音辨識中，會以聲學模型來作為語音辨識的基本單位，因此要進行語音資料的訓練來求取聲學模型的參數時，就必須要先確認聲學模型的結構。以下先介紹幾個常用的名詞：

- 聲學模型（Acoustic Model，或簡稱 Model）：使用於 HMM 的一個抽象單位，通常一個聲學模型包含數個狀態。我們可以使用音節或是音素作為一個聲學模型。
- 音節（Syllables）：完整發音的單位，以中文來說，一個字元對應一個音節；以英文來說，一個詞彙可以對應到數個音節，例如 tomorrow 有三個音節。
- 音素（Phoneme）：或簡稱 Phone，是發音的最小單位，例如「大」的發音可以拆解成ㄉ和ㄚ兩個音素，但是音素的拆解並非一成不變，例如碰到滑母音，我們通常就會將一個注音符號拆成兩個音素，例如ㄞ、ㄟ、ㄠ、ㄡ等，這幾個母音在發音過程中，都會呈現連續的變化。
- Monophone：以單一音素作為一個聲學模型，例如ㄇ。
- Biphone：以連續兩個音素作為聲學模型，通常是 RCD (Right-context dependent)，例如將ㄇ出現於ㄇ-ㄚ和ㄇ-ㄧ視為兩個不同的聲學模型。
- Triphone：以連續三個音素作為聲學模型，例如將ㄨ在ㄎ+ㄨ-ㄢ及ㄍ+ㄨ-ㄤ視為兩個不同的聲學模型。

以我們常用的語音辨識系統而言，是以 biphone 為聲學模型的單位，根據由音節到 Mixture 的階層架構，我們可以畫出下列示意圖：



在上圖中，每一個 state 又分成三個 stream，分別是 MFCC、ΔMFCC、ΔΔMFCC，由於 MFCC 是最重要的語音特徵，因此我們使用 6 個 mixture 來對 MFCC 建模，至於 ΔMFCC 及 ΔΔMFCC，我們各用兩個 mixture 來建模。
若以辨識網路及 HMM 的觀點來看，示意圖如下：



上圖中特別註明了三種 transition:

- Type 0: Transition between syllable
- Type 1: Transition between model
- Type 2: Transition between state

一旦確認聲學模型的架構，我們就可以使用 HTK 來對大量語料抽取出聲學模型的機率參數，請見下一節的說明。