1-2 Example Programs (如何取得程式碼)

If you want to download the example program used in this book, please follow this link. However, the example programs use extensively several toolboxes, which should be downloaded and added to the search path of MATLAB. These toolboxes is accessible from the following links:

1. Utility Toolbox
2. DCPR Toolbox
3. Audio Procesing Toolbox
4. ASR Toolbox (For speech recognition only)
5. Melody Recognition Toolbox (For melody recognition only)

Once these toolboxes are downloaded, they need to be added to the search path of MATLAB. (Or you can simply edit addMyPath.m under the example programs to include these toolboxes.)

Since the example programs are under constant revisions, you can take the following steps if you find a bug:

1. Download all the toolboxes and try again.
2. If it still does not work, send email to me at jang at cs dot nthu dot edu dot tw.

You are also welcome to notify me if there is any mistakes in the texts/programs.

The texts and programs of this book are provided for learning only. The author cannot be held responsible for any damage or loss directly or indirectly incurred by using the texts or programs.

Finally, hope everyone can enjoy the use of MATLAB for audio signal processing.

1-3 Web Resources (網路資源)

There a numerous websites for speech/audio signal processing and recognition on the internet. Here we have a list of some of the commonly used websites, which can be linked for tutorials, discussions, papers, example code, etc.

- http://www.phys.unsw.edu.au/~jw/dB.html
  Introduction to the definition of Decibels for measuring energy/volume of speech/audio signals.
- http://www.phys.unsw.edu.au/~jw/hearing.html
  Introduction (including interactive demos) to curves of equal loudness.
- http://www.phys.unsw.edu.au/music/
  Homepage for "Music Acoustics".
- http://www.phys.unsw.edu.au/~jw/musFAQ.html
  FAQ for "Music Acoustics".
- http://www.wotsit.org
  File formats for various kinds, including audio and music.
- http://www.speech.cs.cmu.edu/comp.speech/index.html
  FAQ for the newsgroup "Comp.Speech".
- http://www.bdti.com/faq/dsp_faq.htm
  FAQ for the news group "Comp.DSP".
- http://www.harmony-central.com/Effects/effects-explained.html
  Introduction to audio effects, including many examples.

Chapter 2: MATLAB Basics

It is very handy to use MATLAB for audio signal processing. To get started with MATLAB, please read the following tutorials on MATLAB basics directly.

- MATLAB Primer by Sigmon (in English)
- 02-初探 MATLAB.pdf（Examples）(in Chinese)

使用 MATLAB 來處理音訊，非常簡單易學。以下是有關於 MATLAB 的入門介紹說明：

- MATLAB Primer by Sigmon（英文）

- 02-初探 MATLAB.pdf（範例程式）（中文）

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)

3-1 Introduction to Audio Signals (音訊基本介紹)

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。

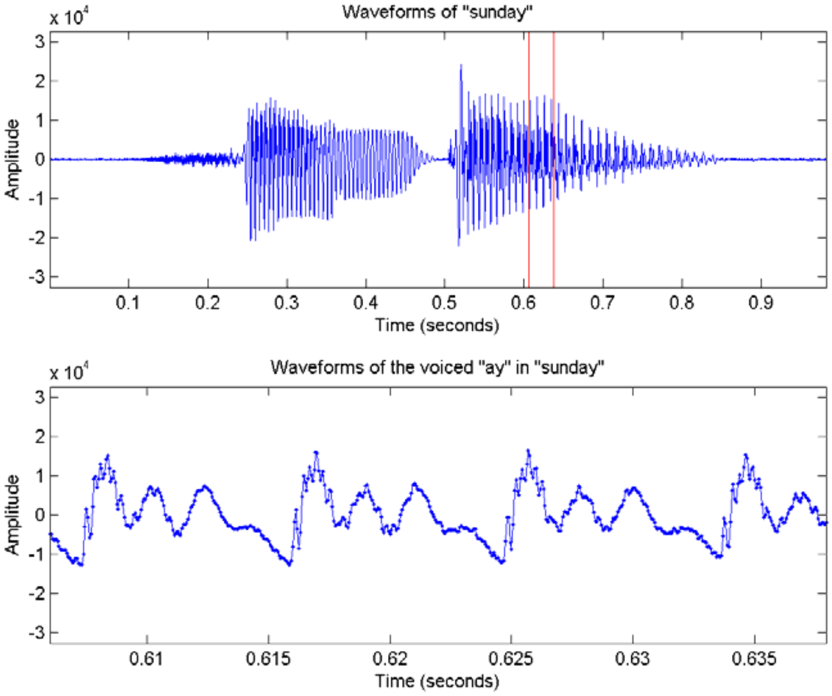
要分辨這兩種聲音，其實很簡單，你只要在發音時，將手按在喉嚨上，若有感到震動，就是 voiced sound，如果沒有感到震動，那就是 unvoiced sound。

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.

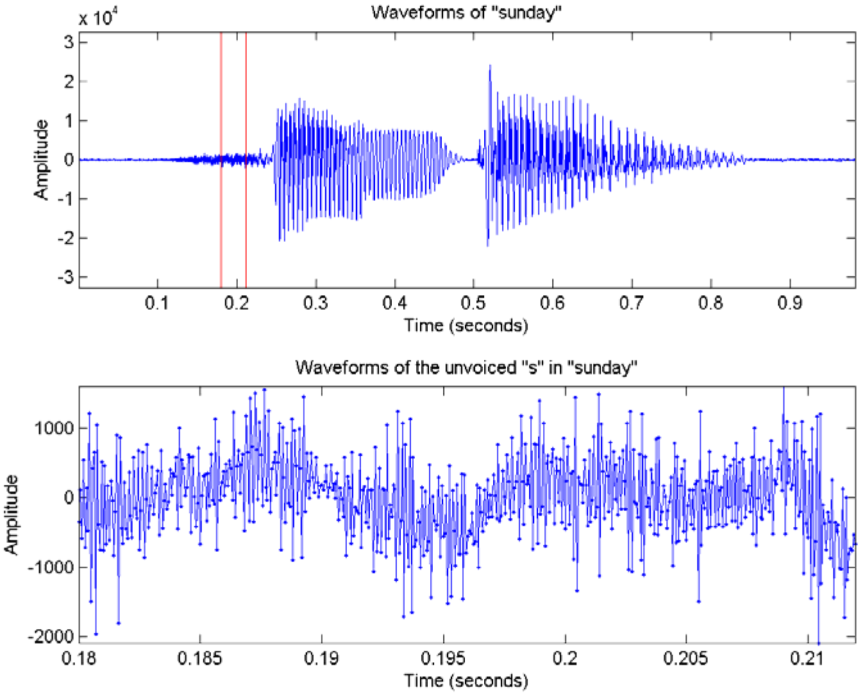
On the other hand, we can also observe the unvoiced sound of "s" in the utterance "sunday", as shown in the following example:

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.

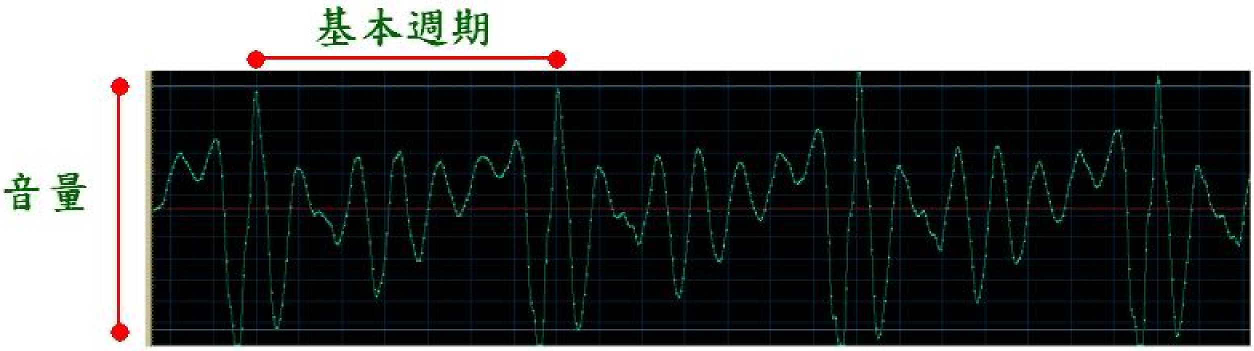
- Volume: This feature represents the loudness of the audio signal, which is correlated to the amplitude of the signals. Sometimes it is also referred to as energy or intensity of audio signals.
- Pitch: This feature represents the vibration rate of audio signals, which can be represented by the fundamental 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:

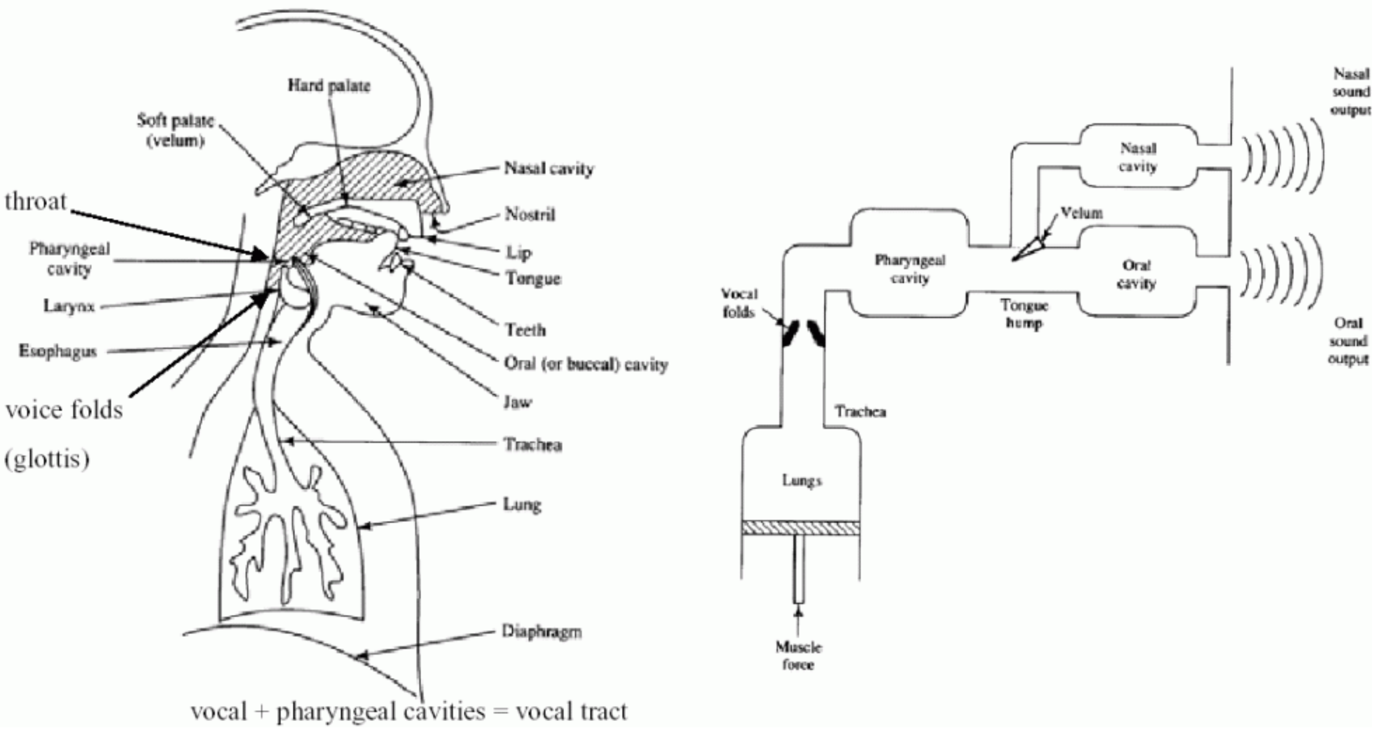
1. Rapid open and close of your vocal cords (or glottis) to generate the vibration in air flow.
2. Resonance of the pharyngeal cavity, nasal cavity, and oral cavity.
3. The vibration of air.
4. The vibration of the ear drum (or tympanum).
5. The reception of the inner ear.
6. The recognition by the brain.

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:

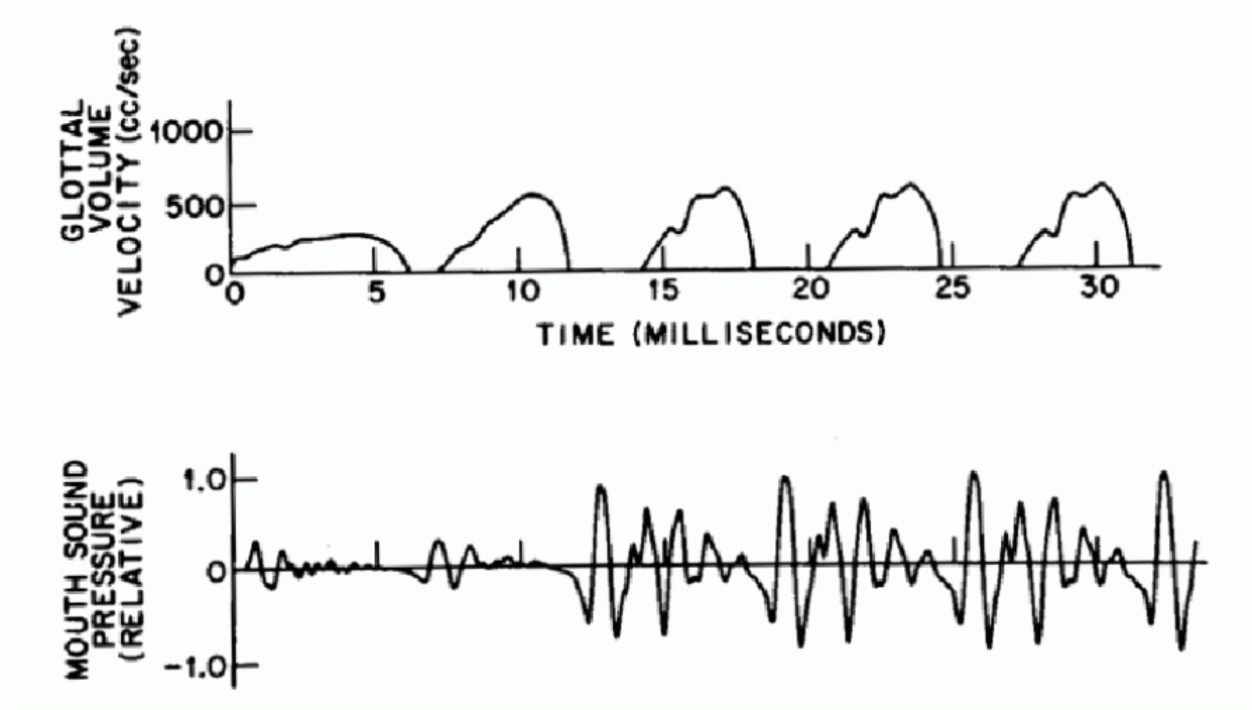
- The vibration frequency of the vocal cords determines the pitch of the voices.
- The positions/shapes of your lips, tongue, and nose determine the timbre.
- The compression from your lungs determine the loudness of the voices.

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）

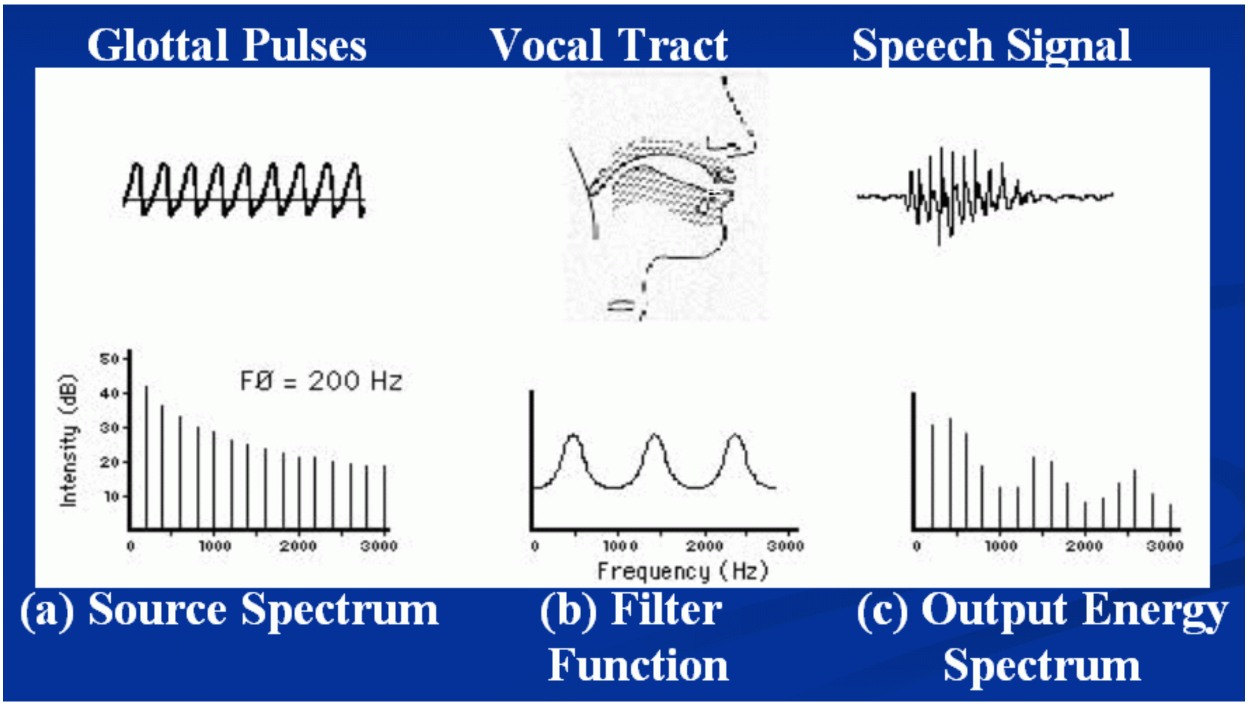
In fact, it is not easy to capture the movements of vocal cords due to its high frequency in movement. So we need to have high-speed cameras for such purpose, for instance:

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：

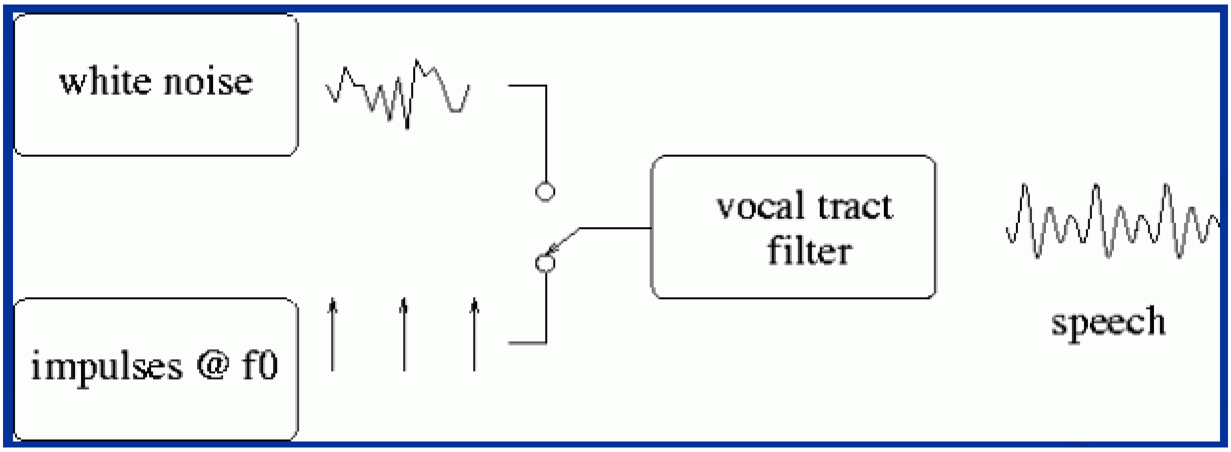


Source-filter model and the corresponding spectra

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：

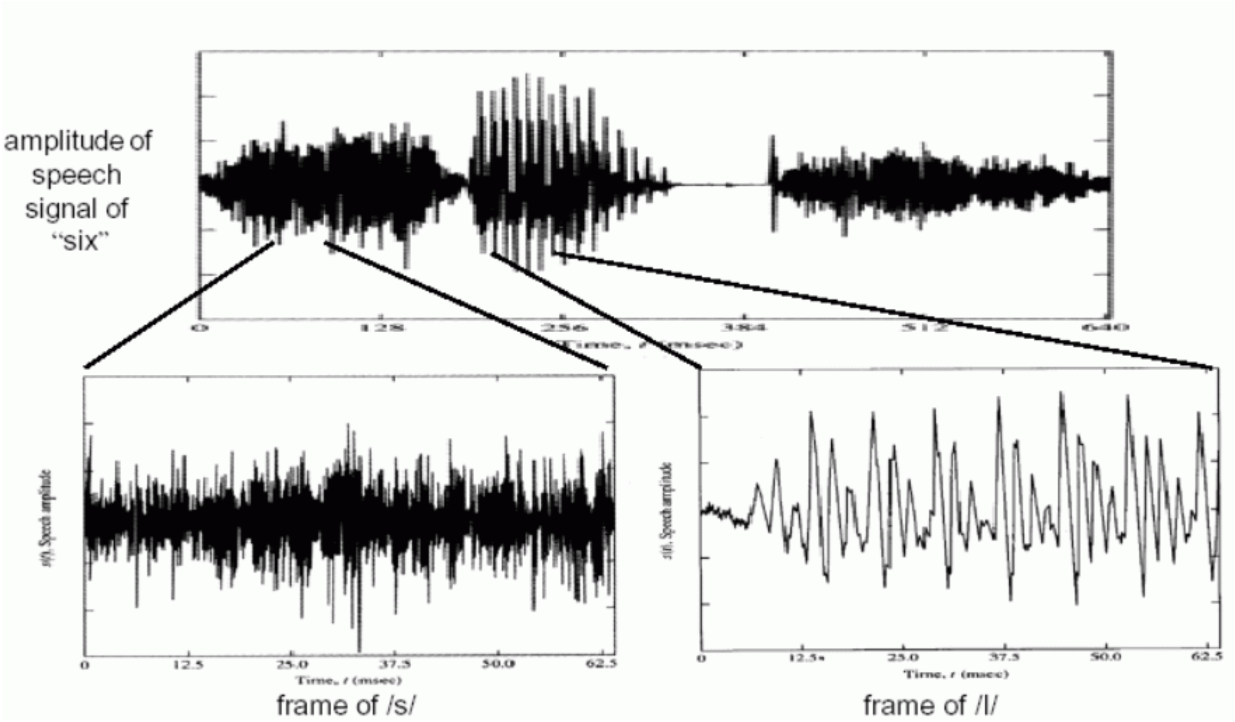


Block diagram representation of source-filter model

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

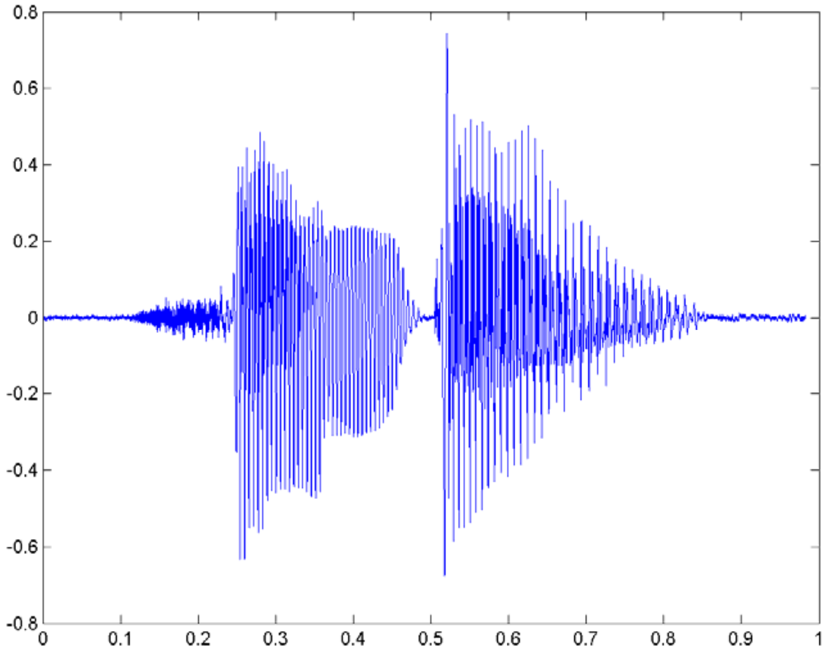
On the Windows platform, the most common file extension for audio signals is "wav". MATLAB can read such wave files via the command "wavread". The following example reads the wave file "sunday.wav" and display its waveform directly.

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
```

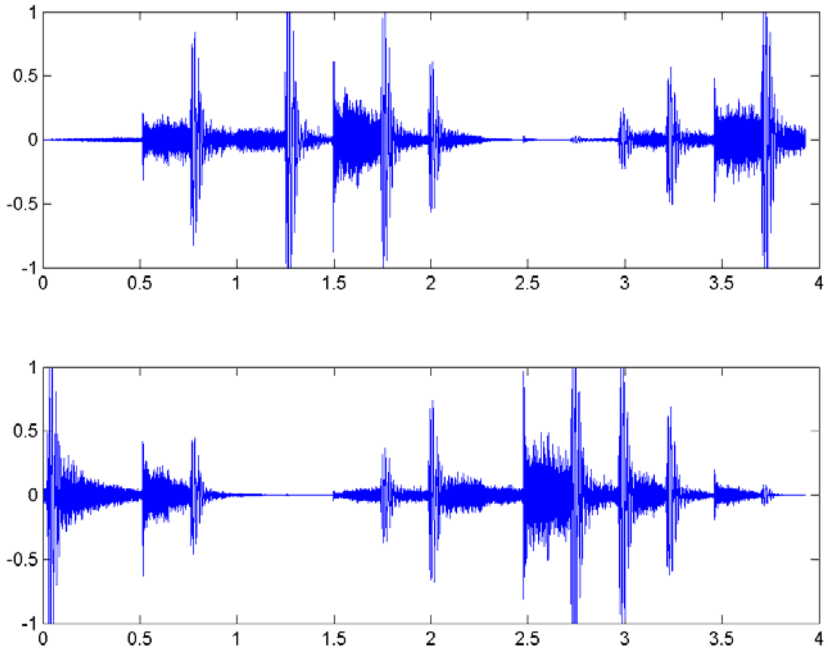
In the above example, the difference is zero, indicating the retrived y0 contains no fractional parts. Moreover, to increase the generality, we use 2^nbits/2 directly instead of 128.

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



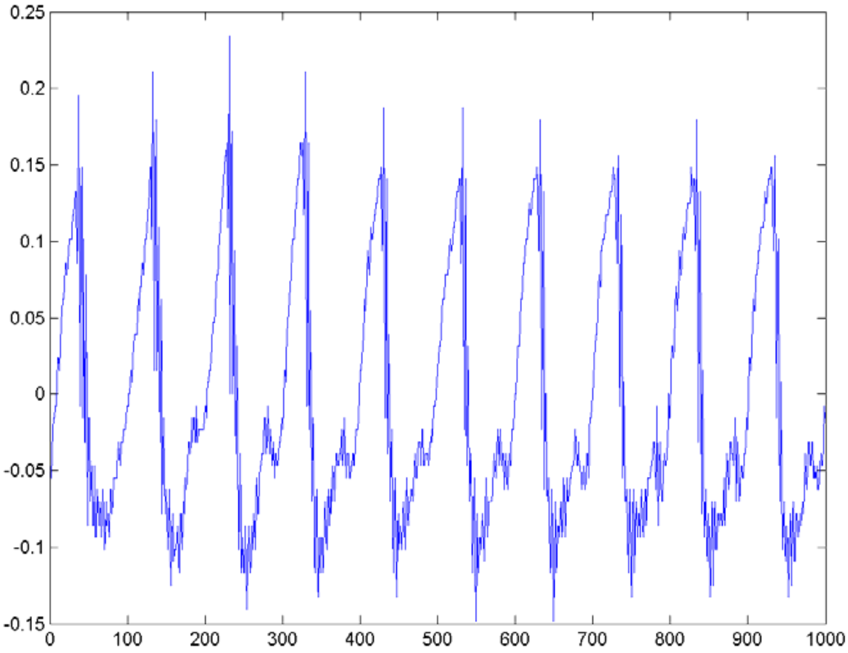
In the above example, MATLAB will read the wave file "flanger.wav", play the stereo sound, and plot two streams of audio signals in two subplots. Because the intnesities of these two channels are more or less complemntary to each other, which let us have an illusion that the sound source is moving back 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.

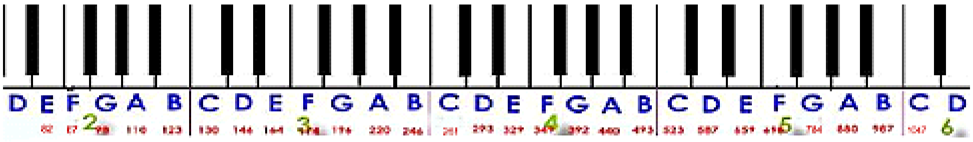
Example 5**Input file** matlab4asp/readWave05.m

```
[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]

4-5 Writing Audio Files

We can write ".wav" audio files by using the MATLAB command "wavwrite". The command format is

wavwrite(y, fs, nbits, waveFile);

where "y" is the audio data, "fs" is the sample rate, "nbits" is the bit resolution, and "waveFile" is the .wav file to be written to. For instance, we can follow the next example to write my recording to a file "test.wav".

Example 1**Input file** matlab4asp/wavWrite01.m

```
fs=11025;               % Sampling rate (取樣頻率)
duration=2;                     % Recording duration (錄音時間)
waveFile='test.wav';    % Wav file to be saved (欲儲存的 wav 檔案)
fprintf('Press any key to start %g seconds of recording...', duration); pause
fprintf('Recording...');
y=wavrecord(duration*fs, fs);
fprintf('Finished recording.\n');
fprintf('Press any key to save the sound data to %s...', waveFile); pause
nbits=8;                        % Bit resolution (每點的解析度為 8-bit)
wavwrite(y, fs, nbits, waveFile);
fprintf('Finished writing %s\n', waveFile);
fprintf('Press any key to play %s...\n', waveFile);
dos(['start ', waveFile]);      % Start the application for .wav file (開啟與 wav 檔案對應的應用程式)
```

(You need to execute the above example in order to experience the recording and the file saving.) In this example, we store the audio data in the data type 'uint8' and write the data to the wave file "test.wav". We then invoke the corresponding application for ".wav" for the playback of the file. Since the variable "y" for the command "wavwrite" should be a double within the range [-1, 1], we need to do some conversion if the recorded data is in other data types, such as 'single', 'int16', or 'uint8'. Here is the table for conversion.

Data types of "y"	How to convert it to 'double' within [-1, 1]
double	No conversion needed
single	y = double(y);
int16	y = double(y)/32768;
uint8	y = (double(y)-128)/128;

MATLAB can also write other audio files, such as '.au', which is the audio files used in NeXT/Sun workstations. The corresponding command is "auwrite". Please type "help auwrite" within the MATLAB command windows for more information on using this command.

Hint

If you want to do audio signal processing with MATLAB exclusively, you can save your audio data using "save" command to save them into .mat files. You can then load these mat files using the command "load" directly.

菴 4 梒釬

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;
```

```
	y2 = y + k*randn(length(y), 1);			% 加入雜訊
	sound(y2, 8000);				% 放音
	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:

$$volume = \Sigma_{i=1}^{n} |s_i|$$

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:

$$volume = 10*\log(\Sigma_{i=1}^{n} s_i^2)$$

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：

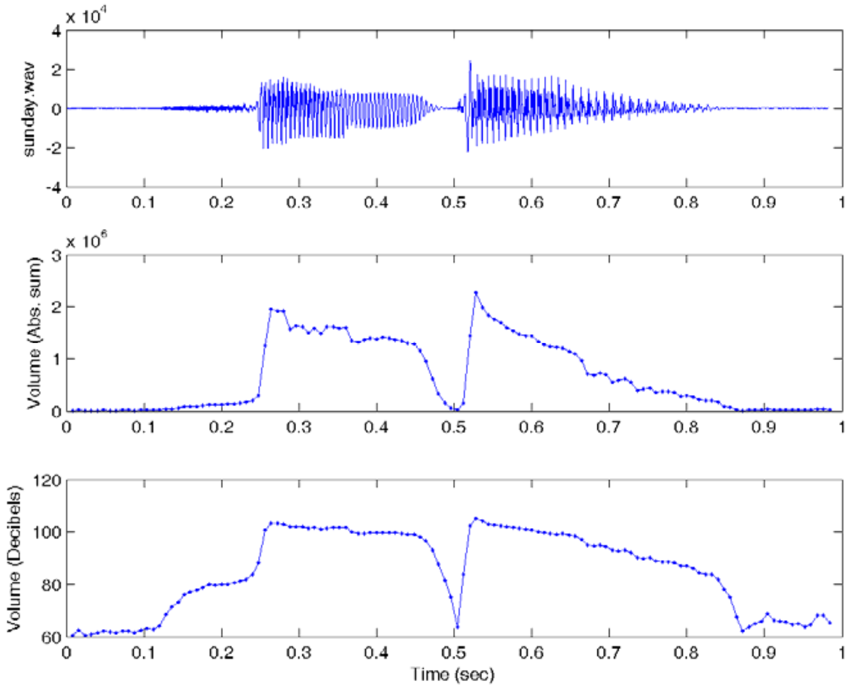
Example 1**Input file** basicFeature/volume01.m

```
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



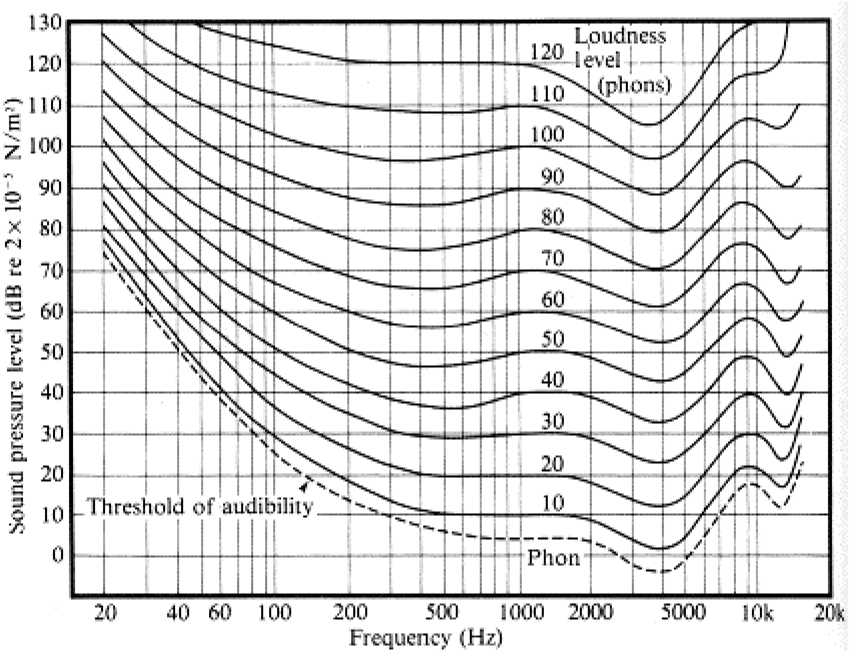
Hint

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。

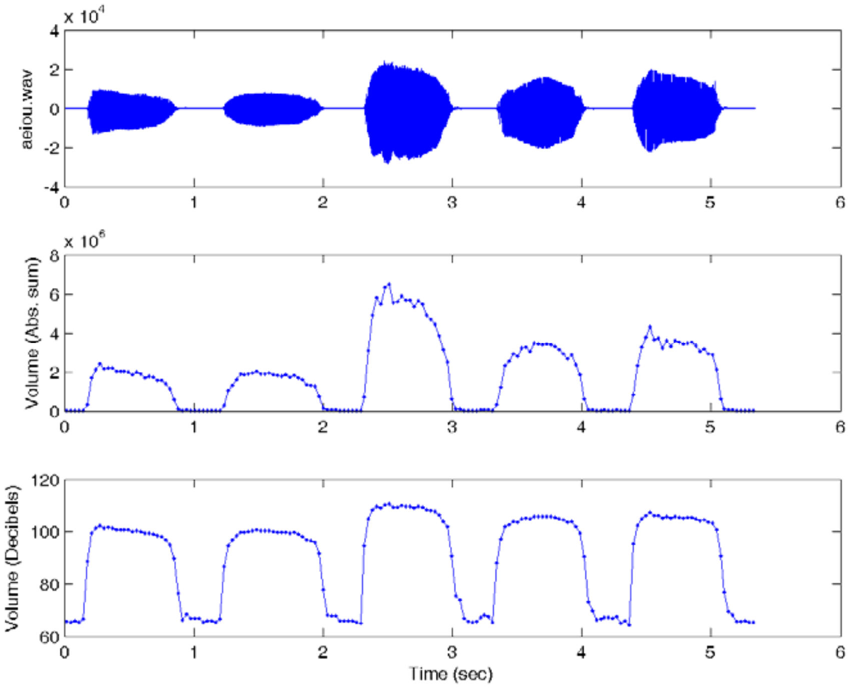
Example 2**Input file** basicFeature/volume02.m

```
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。

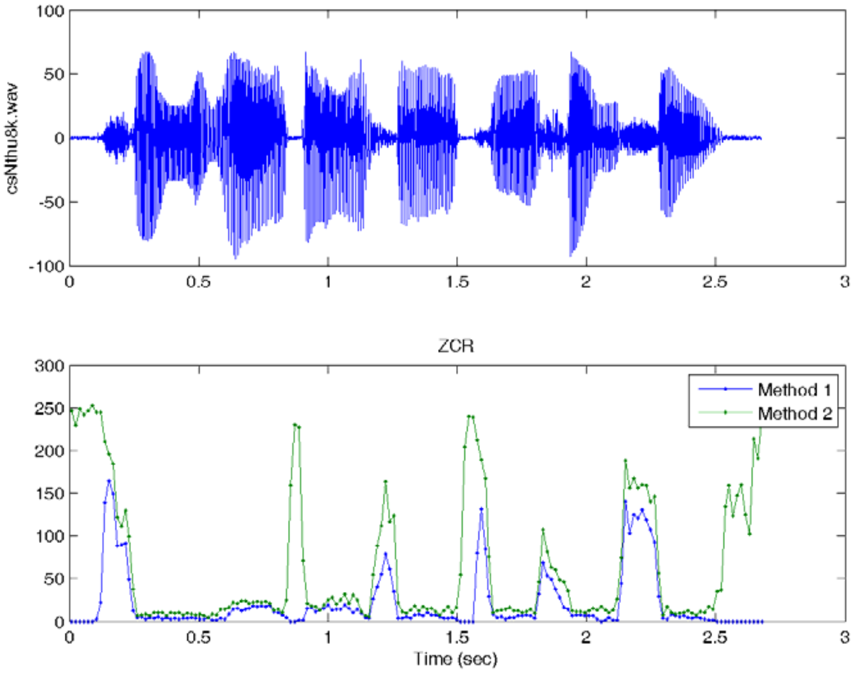
In the following, we use the above-mentioned two methods for ZCR computation of the wave file csNthu8b.wav:

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:

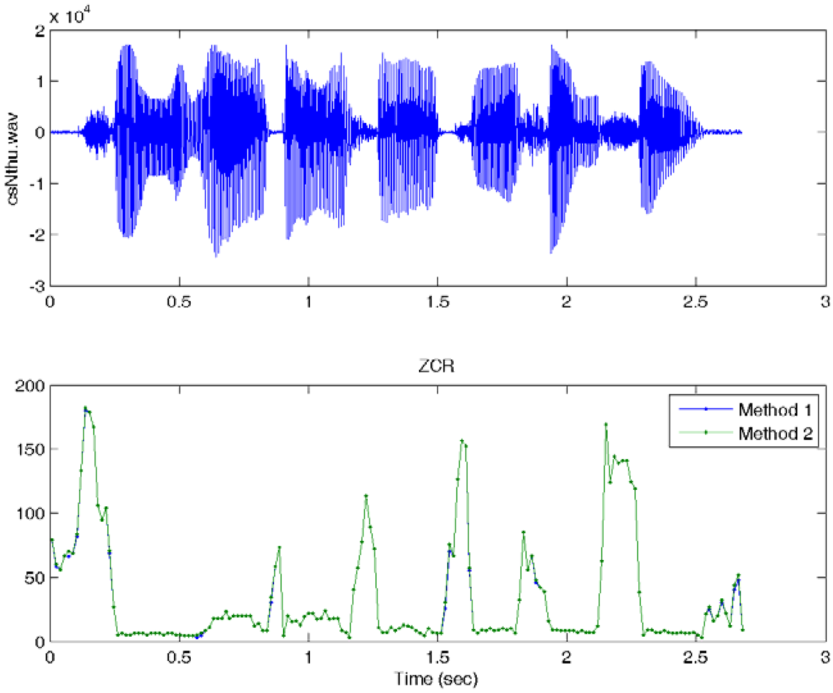
在上述的範例中，我們使用了兩種方式來計算過零率，得到的效果雖然不同，但趨勢是一致的。（另外有一種情況，當錄音環境很安靜時，靜音的訊號值都在零點或零點附近附近跳動時，此時是否計算位於零點的過零率，就會造成很大的差別。）如果取樣頻率提高，得到的結果也會不同：

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



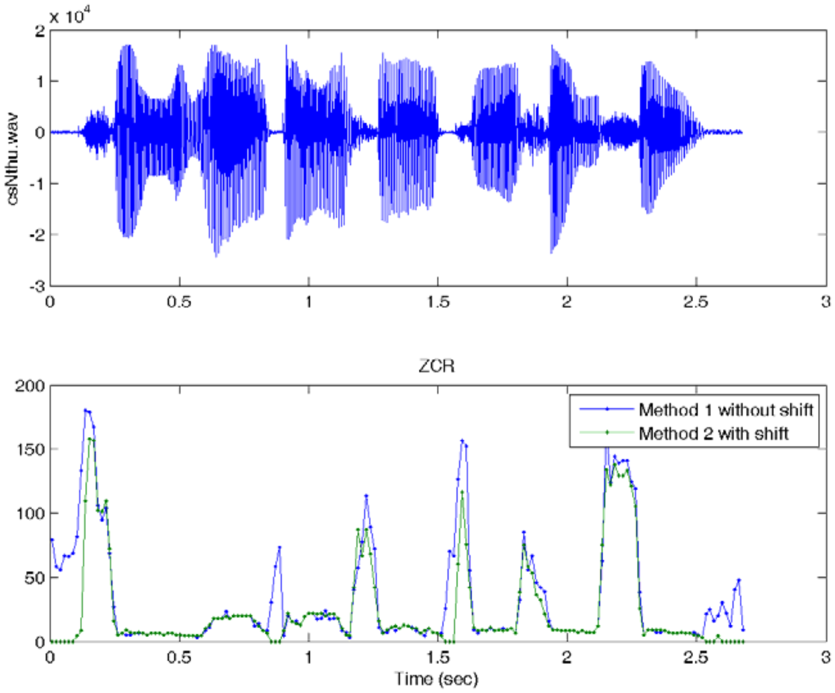
In the above example, methods 1 and 2 return similar ZCR curves. In order to used ZCR to distinguish unvoiced sounds from environment noise, we can shift the waveform before computing ZCR. This is particular useful is the noise is not too big. Example follows:

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：

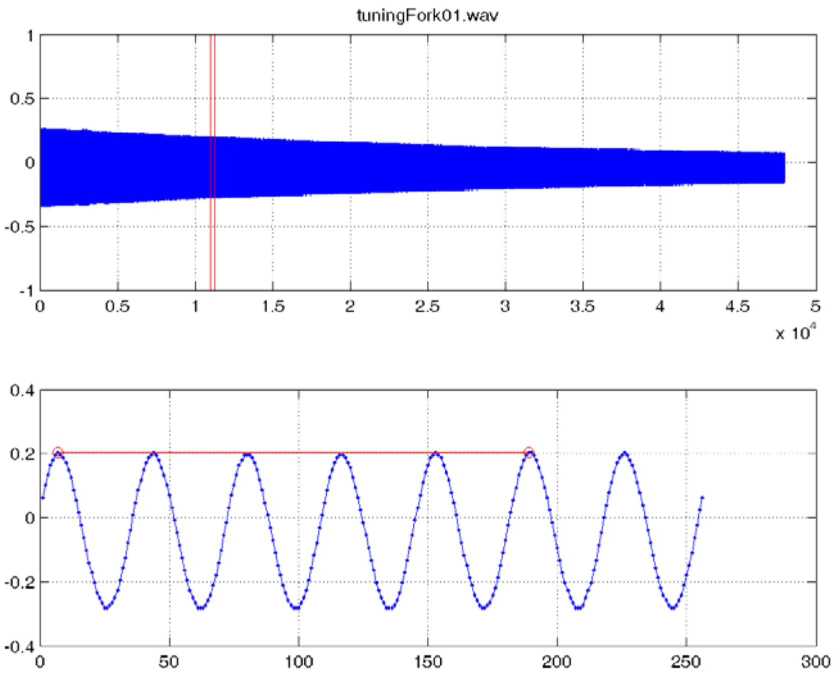
Example 1**Input file** basicFeature/pitchTuningFork01.m

```
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



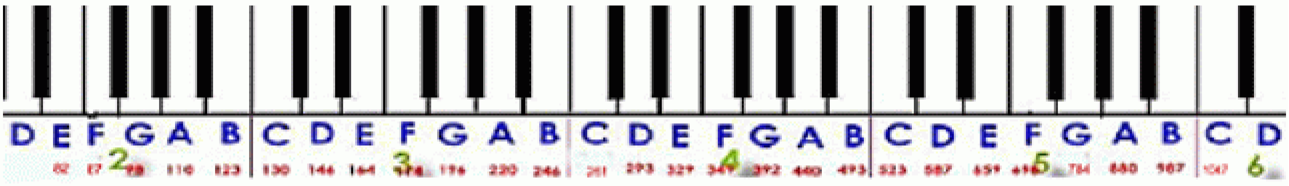
In the above example, the two red lines in the first plot define the start and end of the frame for our analysis. The second plot shows the waveform of the frame as well as two points (identified visually) which cover 5 fundamental periods. Since the distance between these two points is 182 units, the 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：

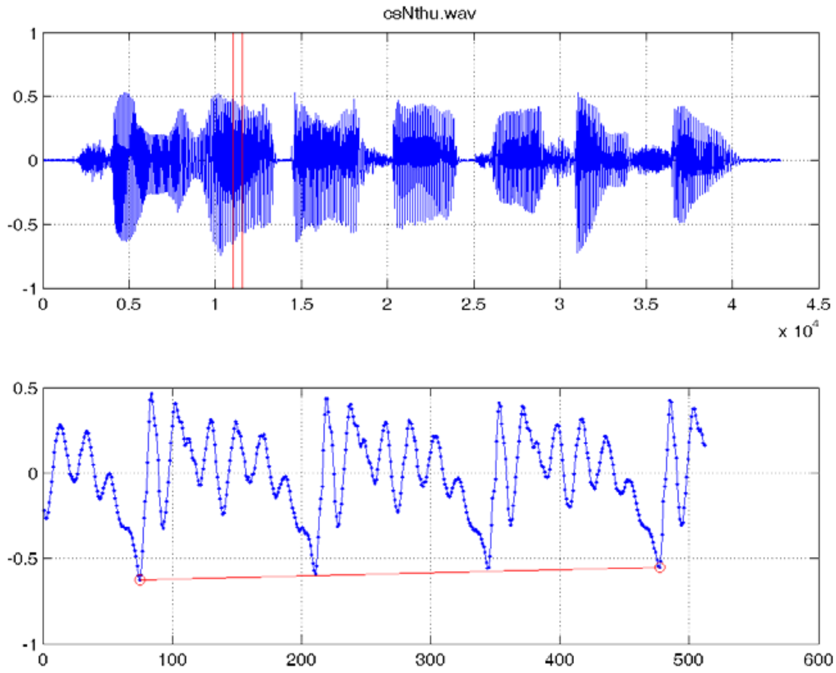
Example 2**Input file** basicFeature/pitchVoice01.m

```
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。

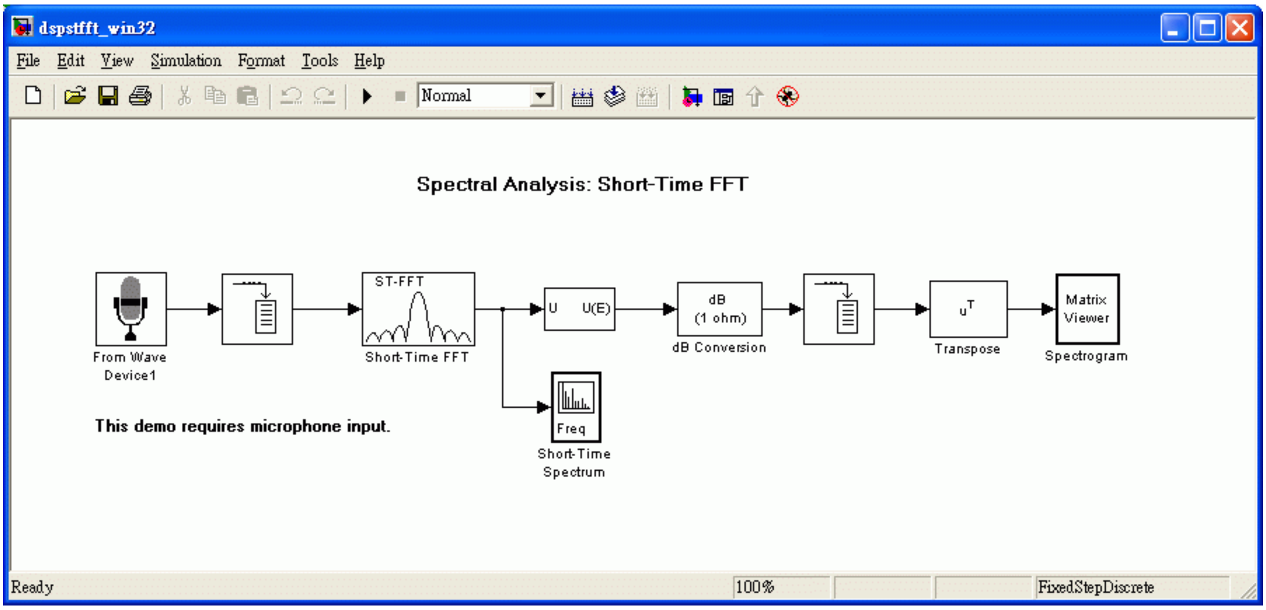
If you want to experience real-time FFT demo, type the following command within the MATLAB command window:

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：

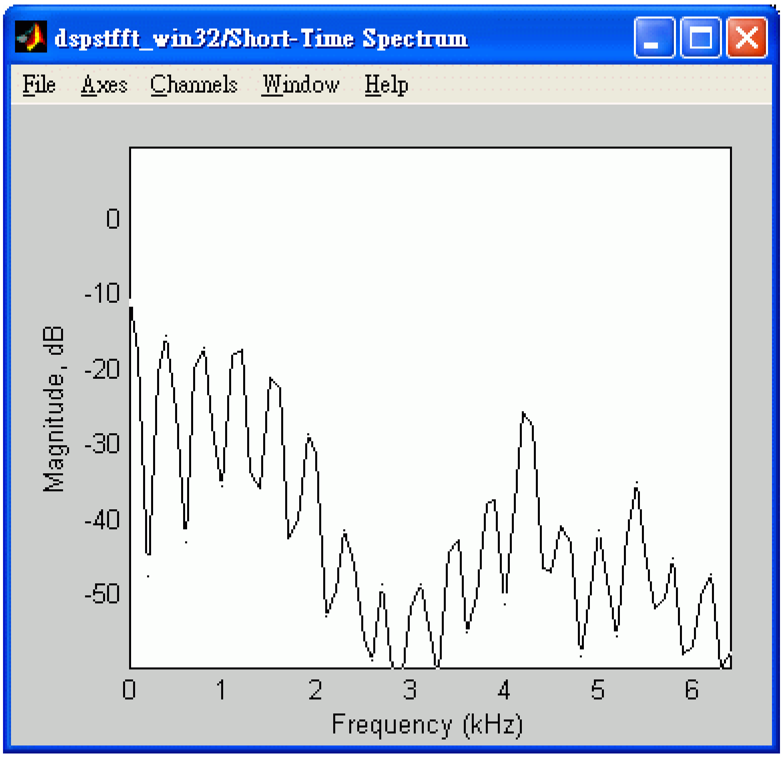


Spectral Analysis: Short-Time FFT

This demo requires microphone input.

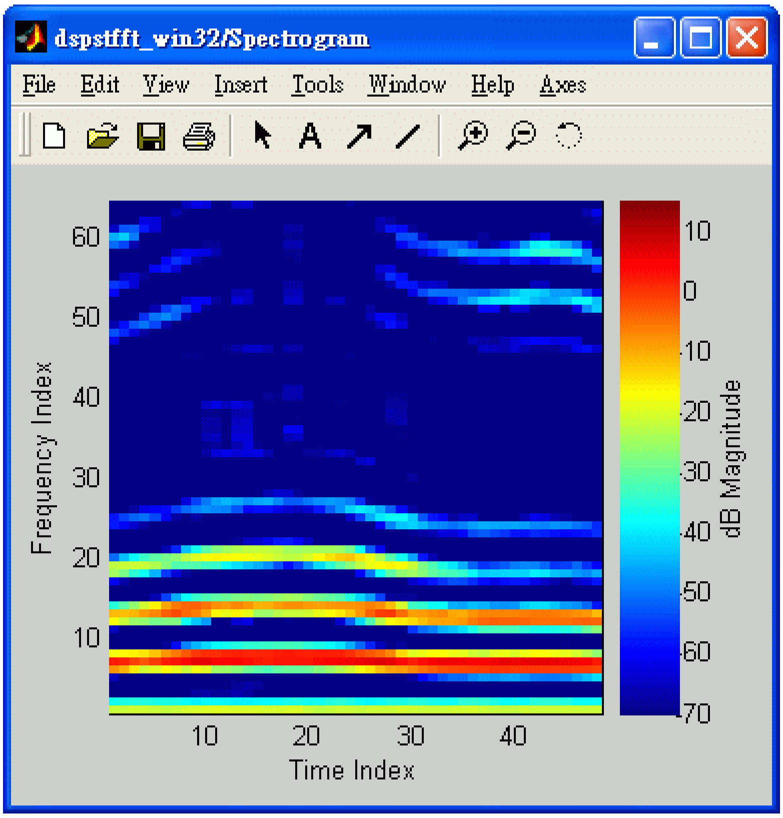
When you start running the system and speak to the microphone, you will able to see the time-varying spectrum:

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

   ```
   freq = 440*2^((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);
   ```

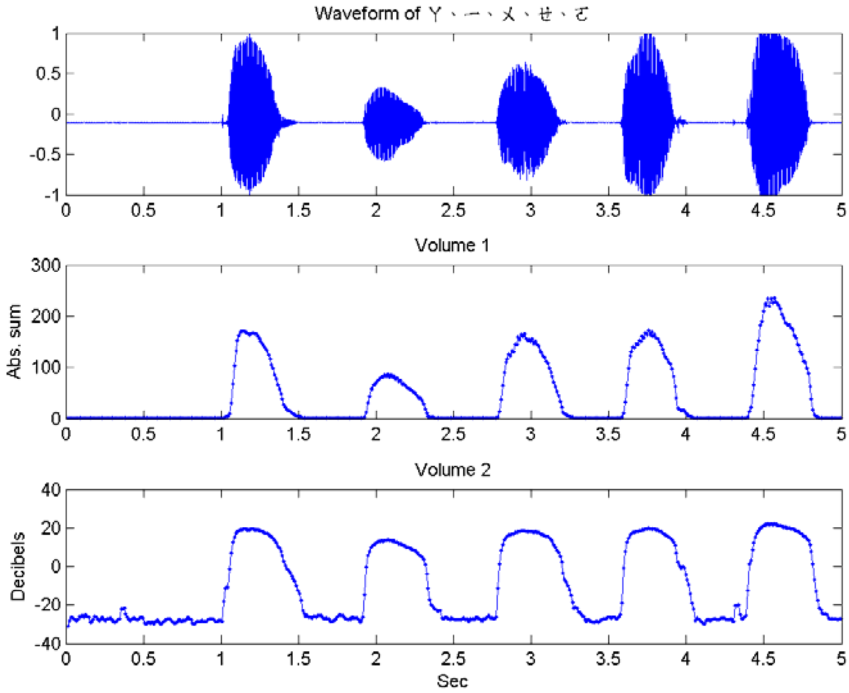
   where "frame" is the input frame and "method" is the method for volume computation (1 for abs. sum; 2 for log square sum), and "volume" is the output volume.

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);
   ```

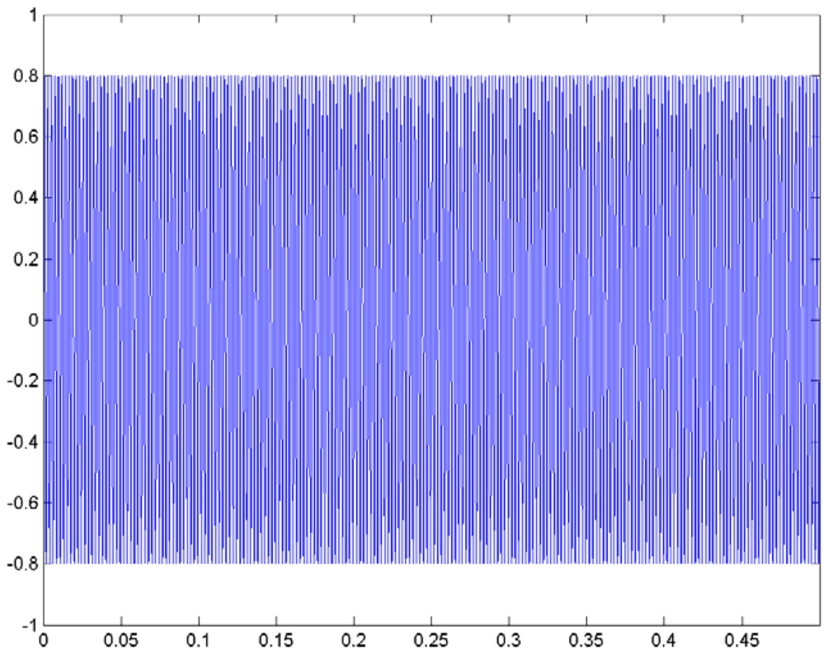
   where "wave" is the input wave signals, "frameSize" is the frame size, "overlap" is the overlap, "method" is the method for computing the volume (1 for abs. sum; 2 for log square sum), and "volume" is the output volume vector.

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:



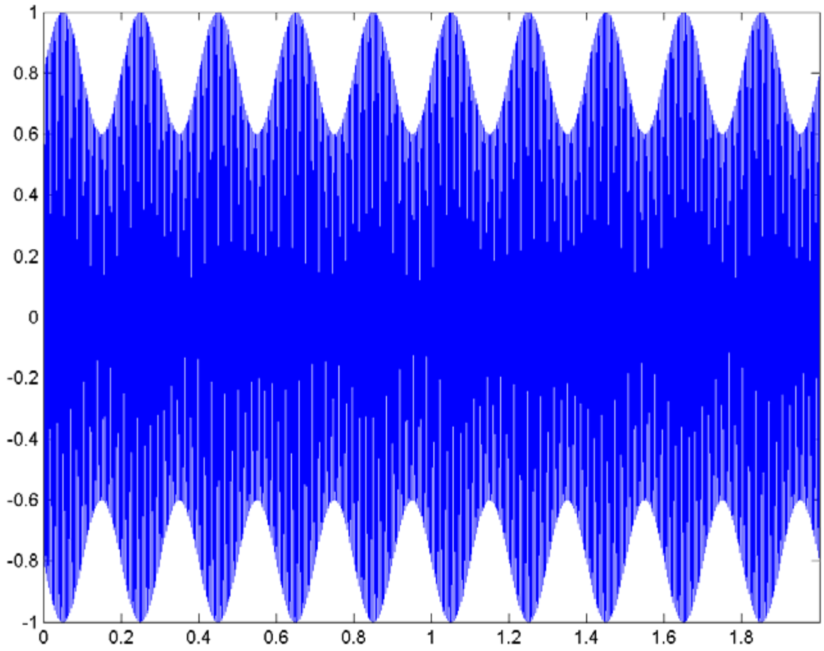
From your plot, can you deduct the relationship between the volume and the shapes of your mouth? Try other vowels to see if your argument still holds.

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:

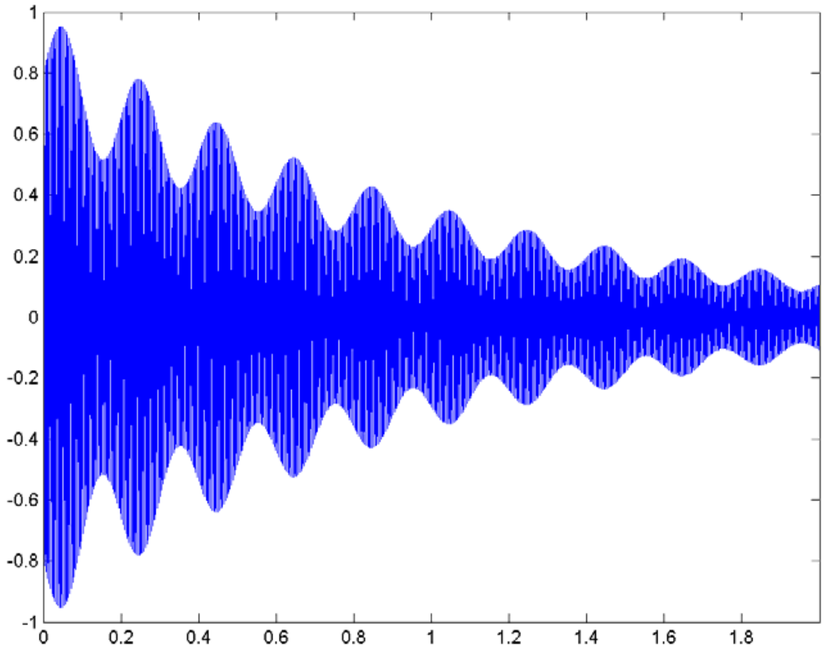


Does the pitch sound the same as the recording of a tuning fork tuningFork01.wav? (Hint: a sinusoid with frequency f can be expressed as y = sin(2*pi*f*t).)

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:

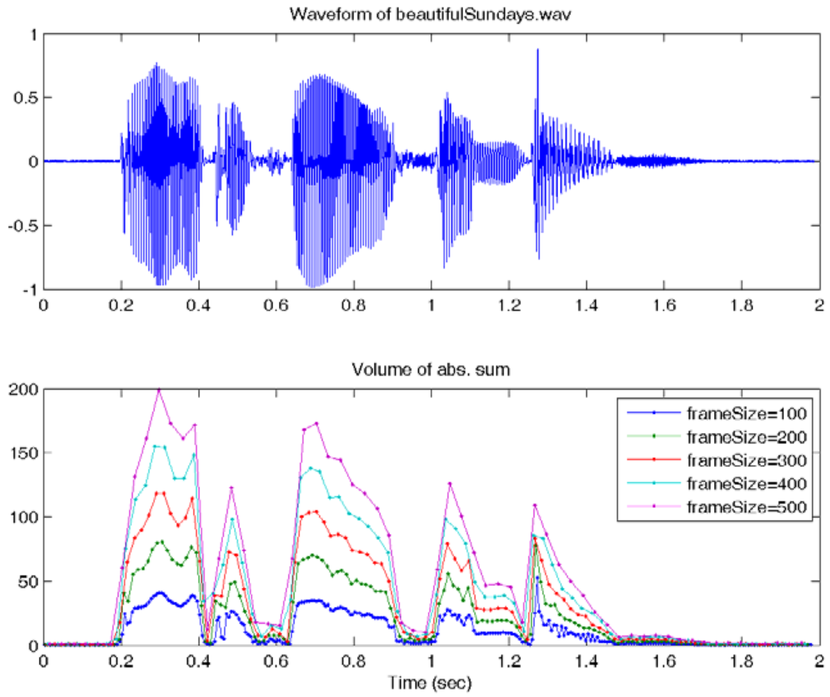


c. Write an m-file script *playSineWave03.m* to repeat the previous sub-problem, but the intensity of the waveform should decrease by following an exponential function exp(-t). Here is an example. Your plot should look like this:

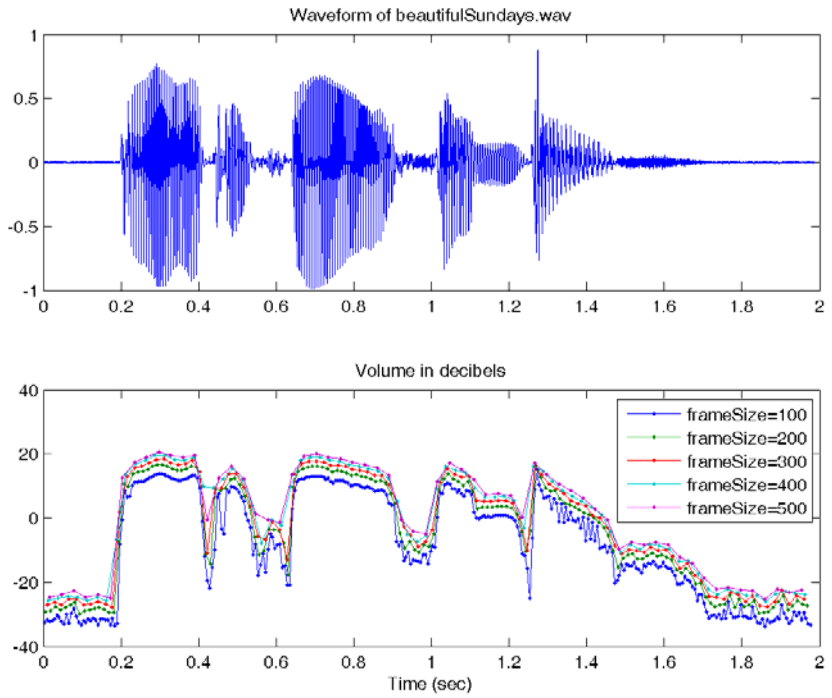


8. (***) **Impact of frame sizes on volume：** First of all, you need to record your utterance of "beautiful sundays" and save it to a wave file of 16 KHz, 16 bits, mono. My example is here. But please use your own recording instead of mine.

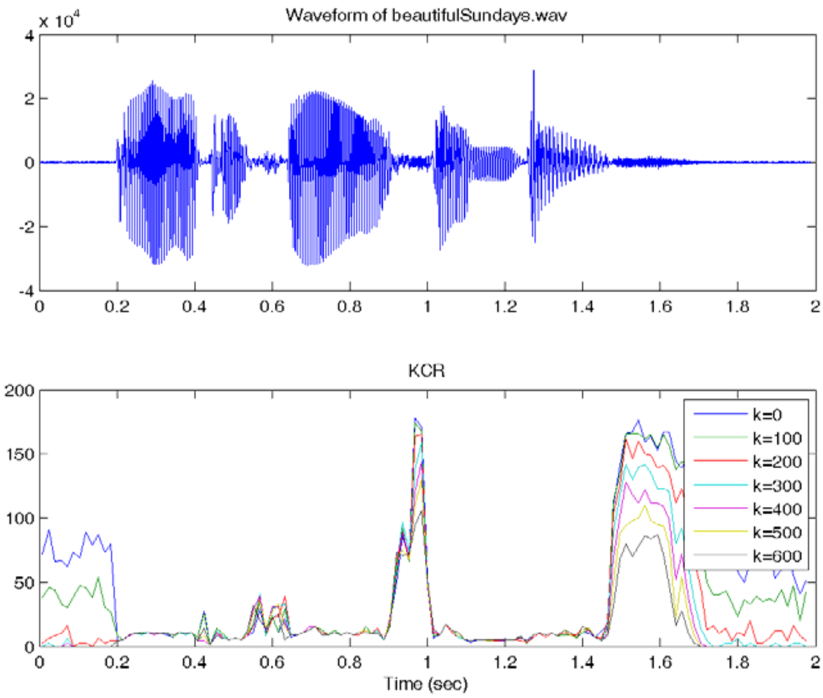
   a. Write an m-file script *plotVolVsFrameSize01.m* that reads the wave file and plot the volumes (as the absolute sum of a frame) with respect to frame sizes of 100 瘉 B200 瘉 B300 瘉 B400 瘉 B500, and overlap of 0. Please use the same time axis for the first plot of the waveform, and the second plots of 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);
```

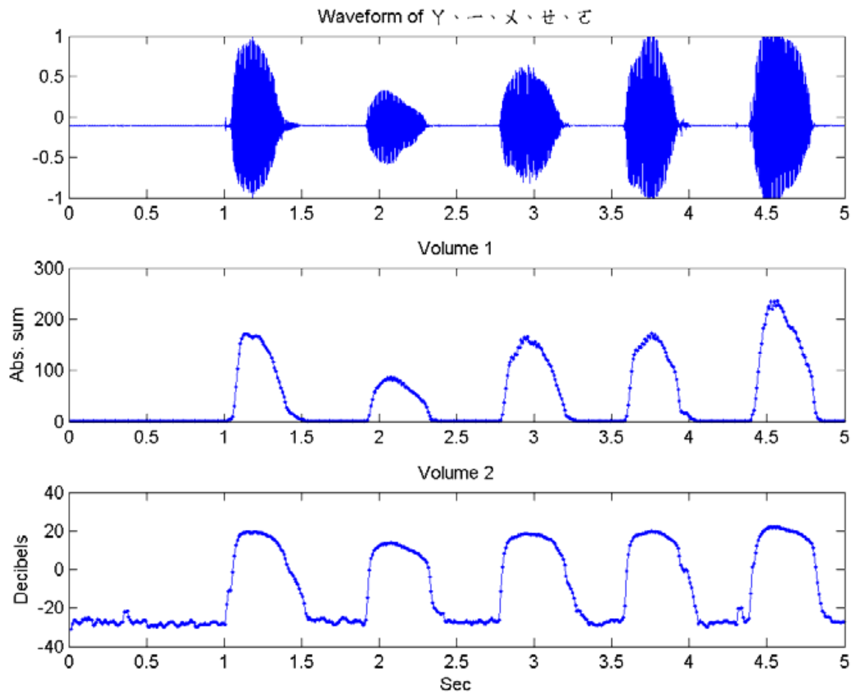
其中 frame 是一個音框向量，method 則是代表使用的方法（'absSum' 代表使用絕對值的總和，'logSquareSum' 代表使用平方和的對數），volume 則是音高。

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。

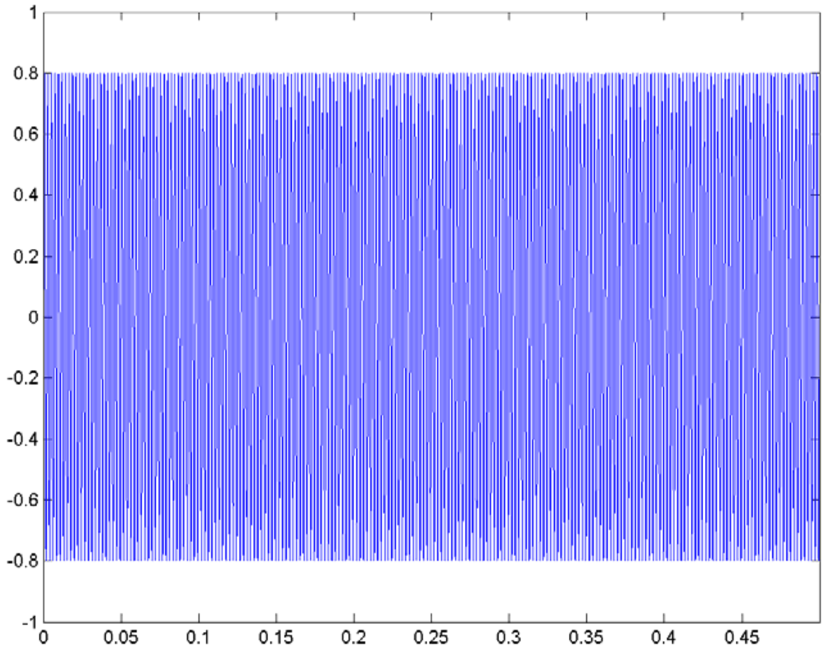
14. (*) **音量與音色的關係：** 請寫一段 MATLAB 程式，先進行錄音，取樣頻率為 16 KHz，解析度為 16 Bits，錄音內容是「ㄚ、ㄧ、ㄨ、ㄝ、ㄛ」。在進行錄音時，盡量用同樣的音量（憑個人的主觀感覺），每個音也要盡量平穩。然後將聲波和兩種音量圖（frame Size = 512，frame Step = 160）同時畫出來，所得到的圖形應該類似下圖：



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。

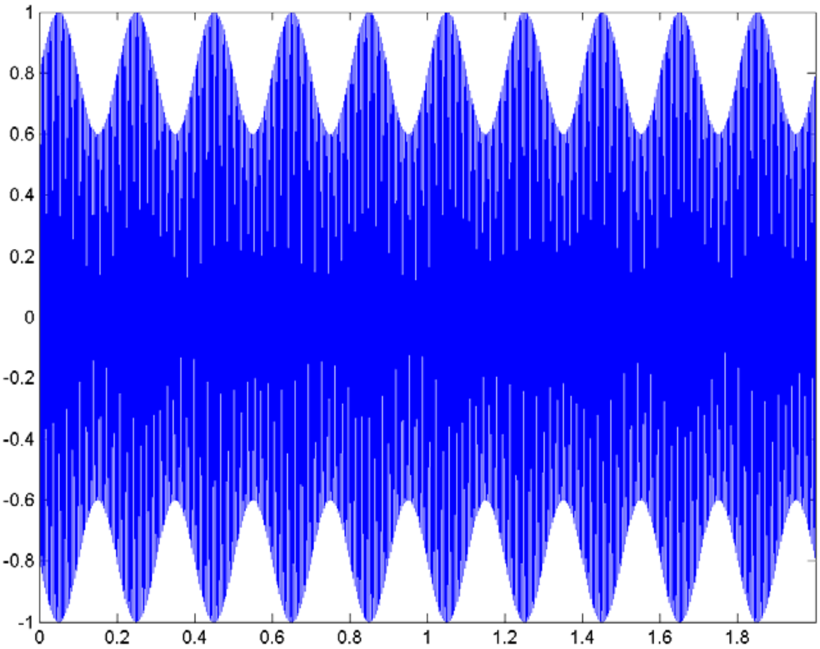
15. (**) **使用正弦波產生特定音高及音量的音訊：** 以下這三小題，都是用正弦波來產生特定音高及音量的音訊。

. 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：

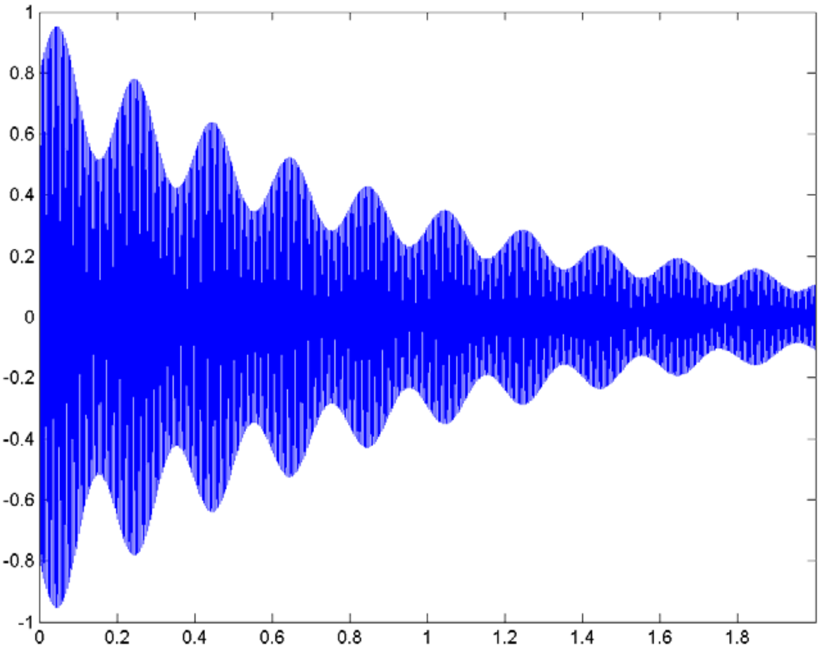


此音訊的音高和音叉的聲音接近嗎？（提示：一個頻率為 f 的正弦波可以表示成 y = sin(2*pi*f*t)。）

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：

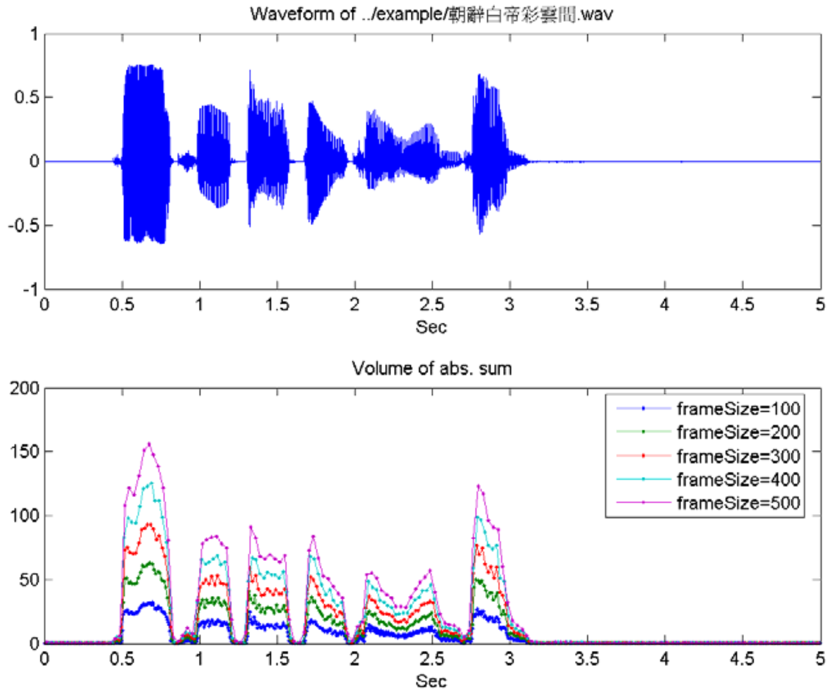


b. 重複上一小題，但是聲波強度隨時間而遞減，遞減函數是 exp(-t)。你的程式碼必須畫出此波形，並同時播放此聲音（偷聽一下）。所得到的圖形應該類似下圖：



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）

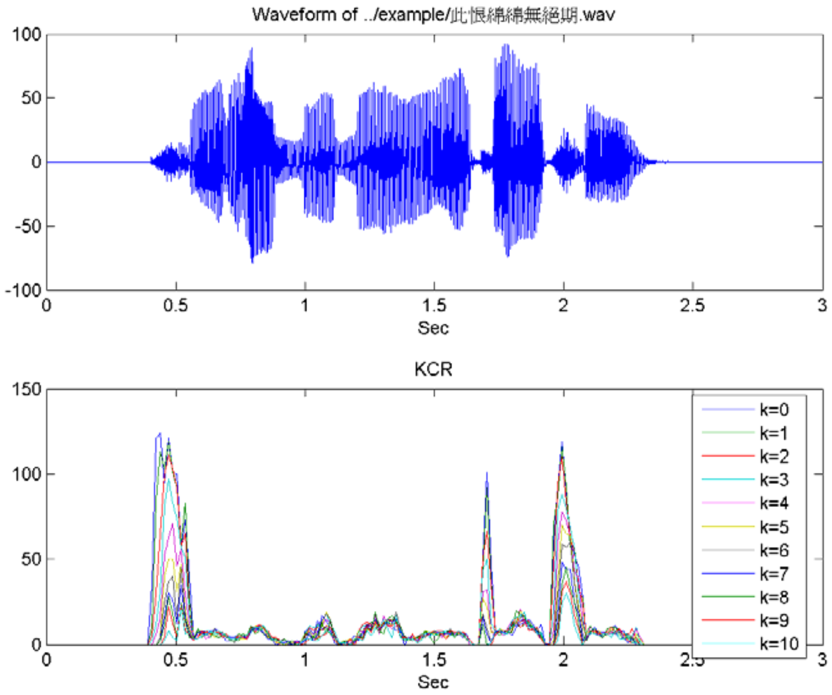
對時間的圖形。請將原聲波和這 5 條音量曲線畫在同一個圖內，所得到的圖形應該類似下圖：



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：

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

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：

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

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.

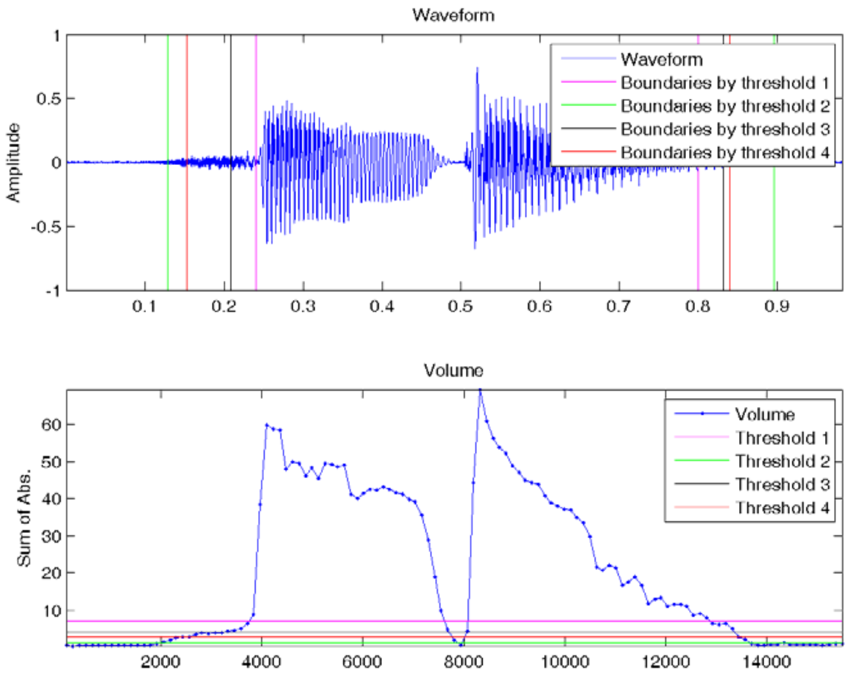
The first method uses volume as the only acoustic feature for EPD. This is the most intuitive method with least computation. We only need to determine a volume threshold and any frame with a volume less than the threshold is regarded as silence. However, how to determine a good volume threshold is not 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.

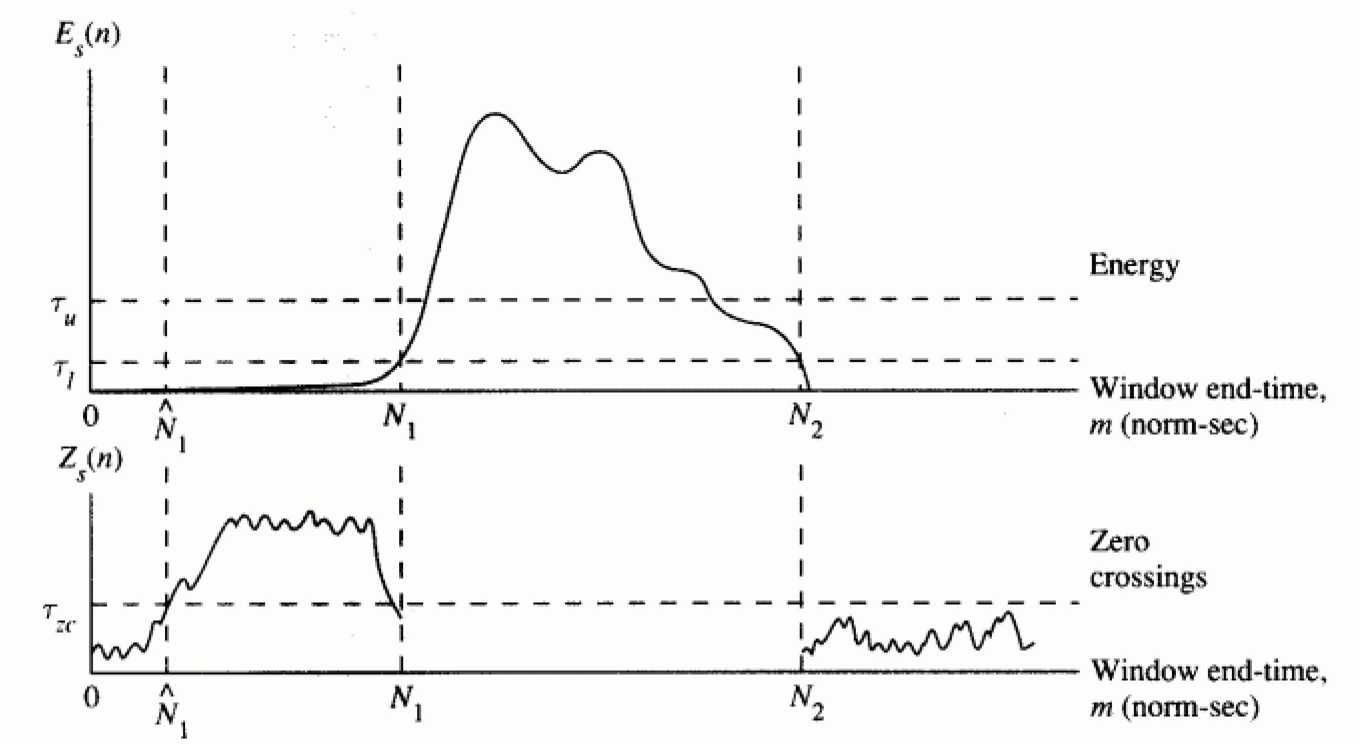
The ratios or constants in the above four methods should be determined through labeled training data. It should be noted wave files of different characteristics (recordings via uni-directional or omni-directional microphones, different sample rates, different bit resolutions, different frame sizes and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

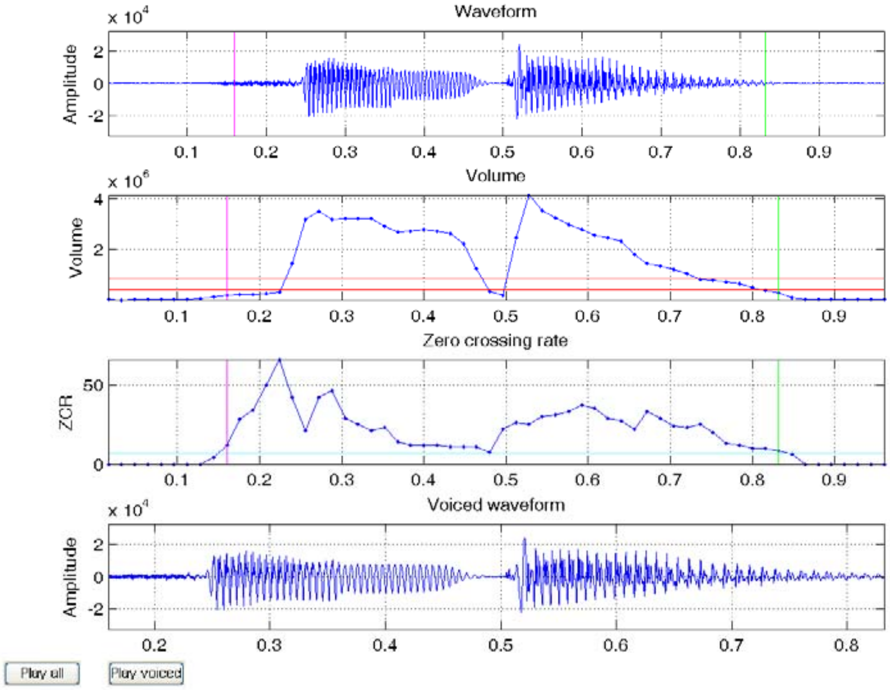
The above method is designed for speech recognition. For melody recognition, we do not need to consider unvoiced sounds since they do not have pitch at all.

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.

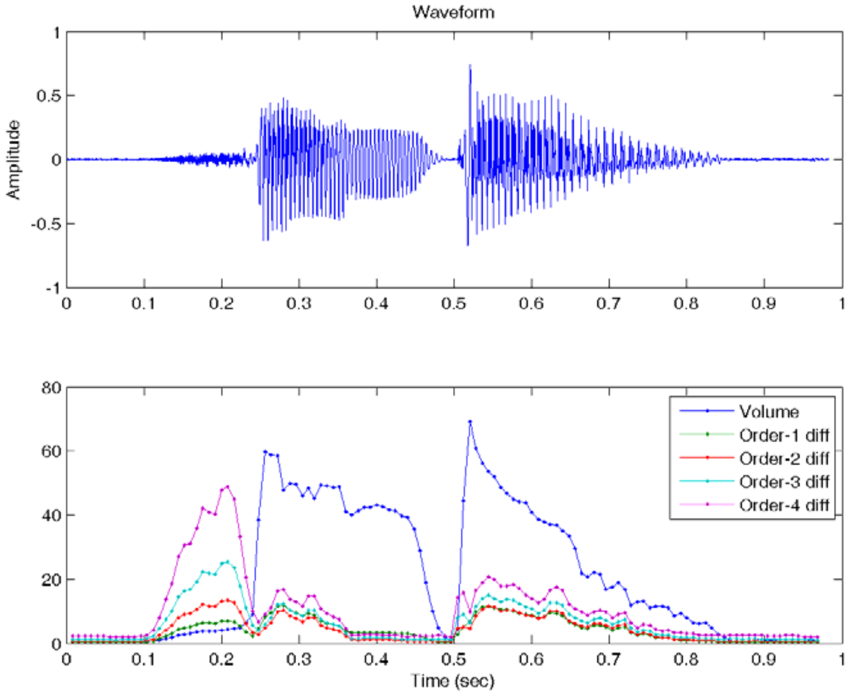
Now it should be obvious that the most difficult part in EPD is to distinguish unvoiced sounds from silence reilably. One way to achieve this goal is to use high-order difference of the waveform as a time-domain features. For instance, in the following example, we use order-1, 2, 3 difference on the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

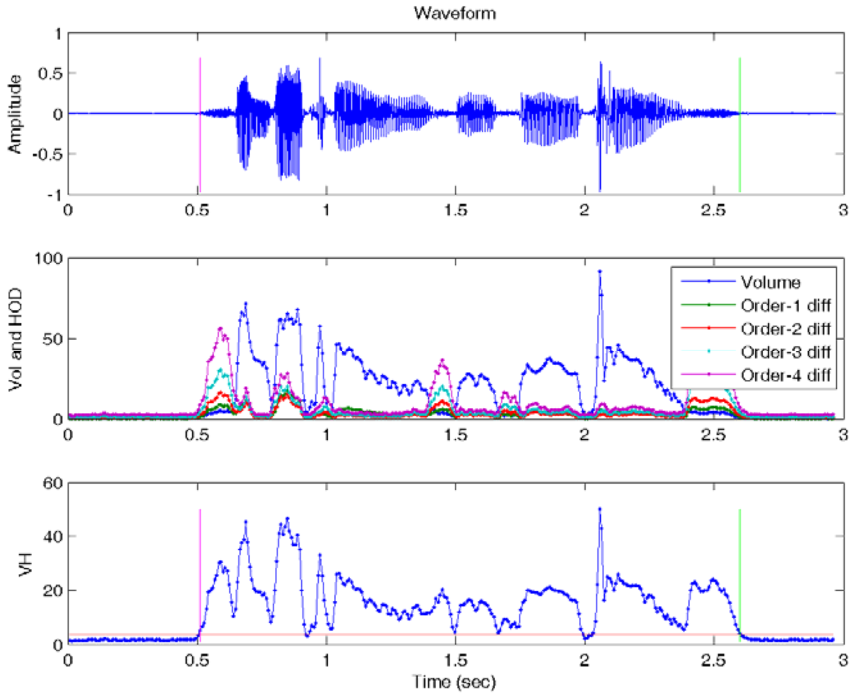
File name	Size (Bytes)	Last modified
096.PDF	231553	2004/8/18 下午 11:41:15
endPointDetectionViaEntropy.ppt	513024	2003/3/27 下午 10:13:18
shenHL98-endpoint.pdf	366756	2004/8/18 下午 11:42:15

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

7-1 Introduction to Pitch Tracking (秞詢祓??賡)

Old Chinese version

From the previous chapter, you should know how to find the pitch by visual inspection and some simple computation. If we want the computer to automatically identify the pitch from audio signals, then we need to have some reliable methods for such a task of "pitch tracking". Once the pitch vector is identified, it can be used for extensive applications in audio signal processing, including:

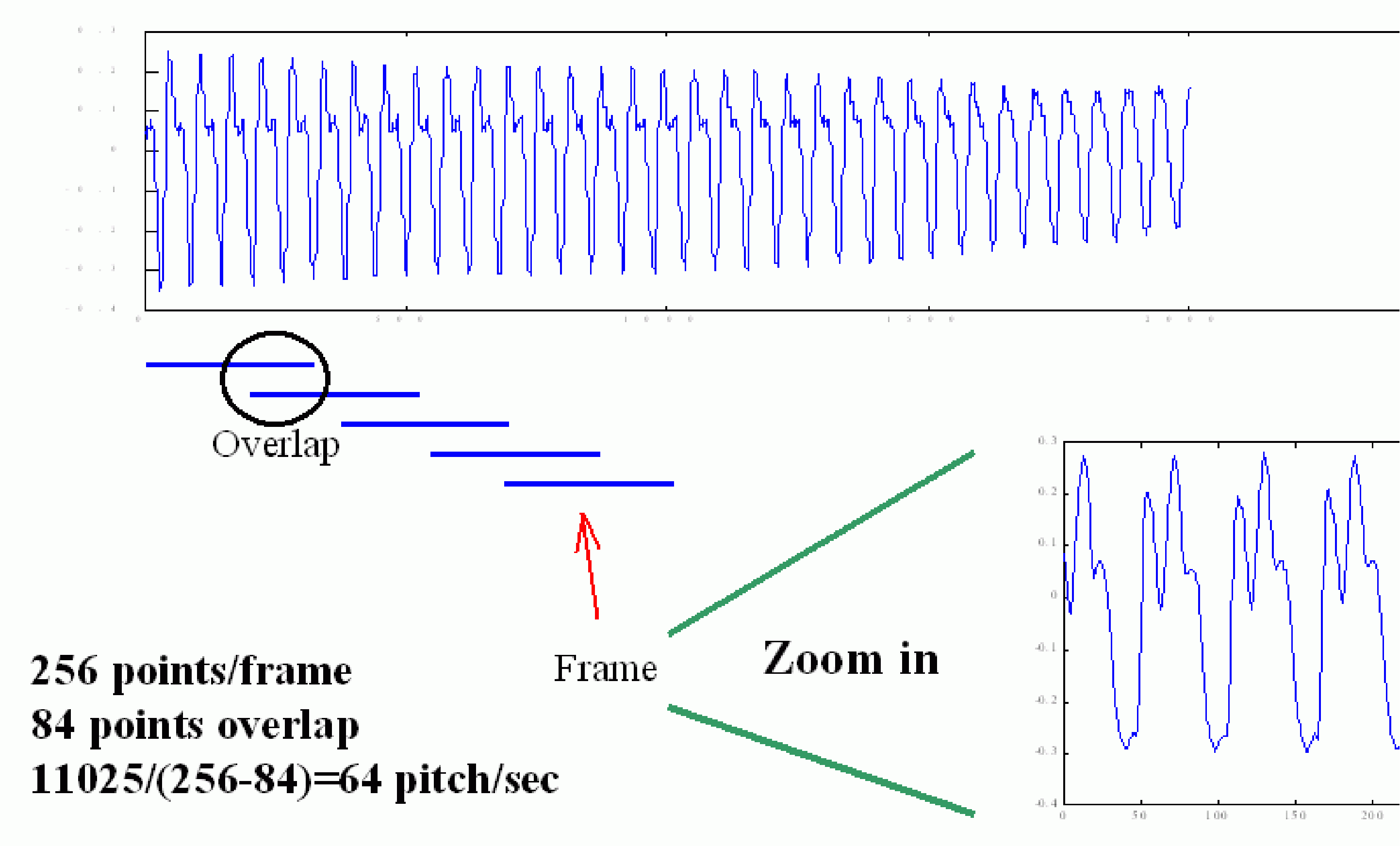
- Melody Recognition: Or QBSH (Query By Singing/Humming), that is, to retrieve a song from the music database that can match a person's singing or humming as close as possible.
- Tone Recognition for tonal language (such as Mandarin): To identify each syllable's tone for computer assisted pronunciation training (CAPT).
- Prosody Analysis for TTS (text-to-speech): To analyze and predict the best F0 curve for TTS applications.
- Intonation Assessment: To compute the similarity between a test and a target utterances for CAPT.
- Speech recognition: Pitch can be used to improve the recognition rates of speech recognition engines.

In summary, pitch tracking is a fundamental step toward other important tasks for audio signal processing. Related research on pitch tracking has been reported for decades, and it still remains an hot topic in the literature. Therefore we need to know the basic concept of pitch tracking as a stepstone for other advanced audio processing techniques.

Pitch tracking follows the general processing of short-term analysis for audio signals, as follows.

1. Chop the audio signals into frames of 20 ms or so. Overlap is allowed between neighboring frames.
2. Compute the pitch of each frame.
3. Eliminate pitch from silence or unvoiced sounds. This can be done by using volume thresholding or pitch range thresholding.
4. Smooth the pitch curve using median filters or other similar methods.

In the processing frame blocking, we allow overlap between neighboring frames to reduce discontinuity between them. We can define "frame rate" as the frames per second for our analysis. For instance, if fs = 11025 Hz, frame size = 256, overlap = 84, then the frame rate is equal to fs/(frameSize-overlap) = 11025/(256-84) = 64. In other words, if we wish to have real-time pitch tracking (for instance, on the platform of micro-controllers), then the computer should be able to handle 64 frames per second. A small overlap will lead to a low frame rate. The process of frame blocking is shown next.



When we choose the frame size and the overlap, we need to consider the following factors.

- The frame size should cover at least two fundamental periods to fully capture the characteristics of the audio signals. Suppose that the pitch range of human voices is between 50 to 1000 Hz, and the sample rate is 16000 Hz, then we can derive the range of the frame size, as follows.
    a. If f = 50 Hz, then the fundamental period = fs/f = 16000/50 = 320 points and the frame size should be 2*320 = 640 points.
    b. If f = 1000 Hz, then the fundamental period = fs/f = 16000/1000 = 16 points and the frame size should be at least 2*16 = 32 points.
- The frame size should not too big, otherwise it cannot capture time-varying characteristics of audio signals. A big frame size also require more computing time to process the frame.
- The overlap is determined by the computing power of your platform. A big overlap leads to a big frame rate and thus requries more computing power. If we do not have enough computing power, we can reduce the overlap or even make the overlap negative.

There are a number of methods to derive a pitch value from a single frame. Generally, these methods can be classified into time-domain and frequency-domain methods, as follows.

- Time-domain methods
    - ACF: Autocorrelation function
    - SMDF: Average magnitude difference function
    - SIFT: Simple inverse filter tracking
- Frequency-domain methods
    - Harmonic product spectrum method
    - Cepstrum method

These methods will be covered in the rest of this chapter.

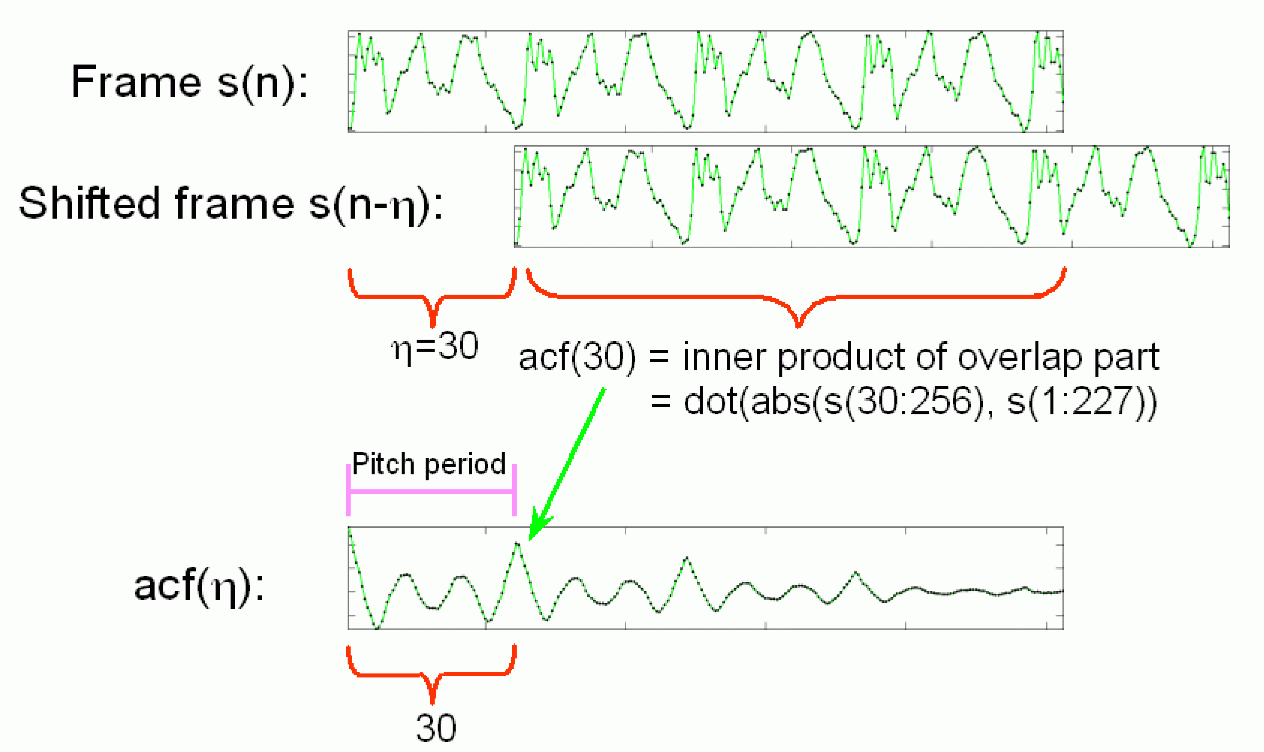
7-2 ACF

Old Chinese version

In this section, we shall introduce the auto-correlation function (ACF) for pitch tracking. This is a time-domain method which estimates the similarity between a frame s(i), i = 0 ~ n-1, and its delayed version via the auto-correlation function:

$$acf(\tau) = \Sigma_{i=0}^{n-1-\tau} s(i)\ s(i+\tau)$$

where τ is the time lag in terms of sample points. The value of τ that maximizes acf(τ) over a specified range is selected as the pitch period in sample points. The following figure demonstrates the operation of ACF:


Frame s(n):
Shifted frame s(n-η):
η=30
acf(30) = inner product of overlap part
= dot(abs(s(30:256), s(1:227))
Pitch period
acf(η):
30

In other words, we shift the delayed version n times and compute the inner product of the overlapped parts to obtain n values of ACF.

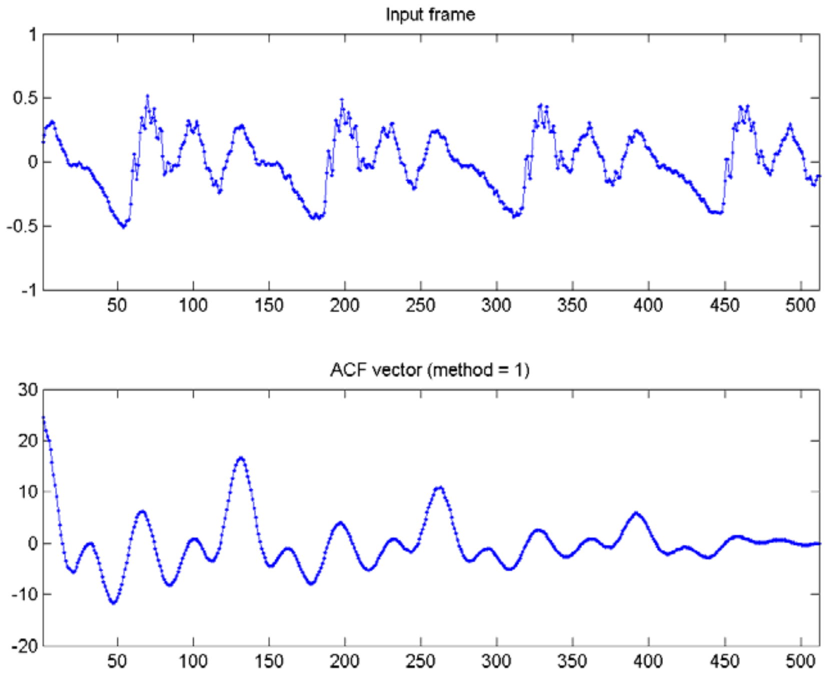
Take my utterance "sunday.wav" for example. If we take a frame of 512 points starting from the 9000th point, which corresponds to the vowel part of "day", the ACF result is the following example:

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



The maximum of ACF occurs at the first point, which is obviously not what we want. If we set the values around the first maximum to be zero, we can identify the second maximum located at index 131 and the corresponding pitch is fs/(131-1) = 16000/130 = 123.08 Hz, or 46.94 semitones.

Hint

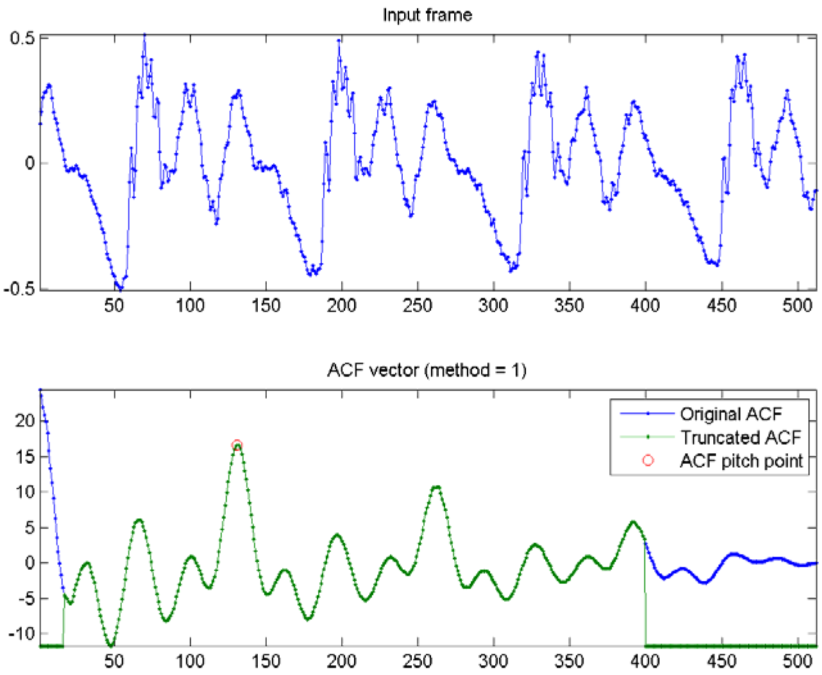
In the above computation, we need to divide fs by 131 minus 1 since 131 is a 1-based index in MATLAB.

The point of ACF at index 131 is referred as the pitch point for ACF. In order to identify the pitch point of ACF automatically, we can simply set the first few points of ACF to be zero and then find the maximal point of ACF. Please refer to the following example.

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



In the above figure, the first plot if the frame, the second plot is the original ACF and the modified ACF. The red circle is the maximal point of the modified ACF, which is also the correct pitch point. The modified ACF is obtained by setting ACF at some unlikely region for the pitch point to be zero. Specifically,

a. Assuming the max. pitch is 1000 Hz. This leads to a fundamental period of fs/1000 = 16000/1000 = 16 points. Therefore we can set the first 16 points of ACF to be zero.
b. Assuming the min. pitch is 40 Hz. This leads to a fundamental period of fs/40 = 16000/40 = 400 points. Therefore we can set all the elements of ACF after index 400 to be zero.

By using the described method, we can perform pitch tracking on a stream of audio signals, as shown in the next example.

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

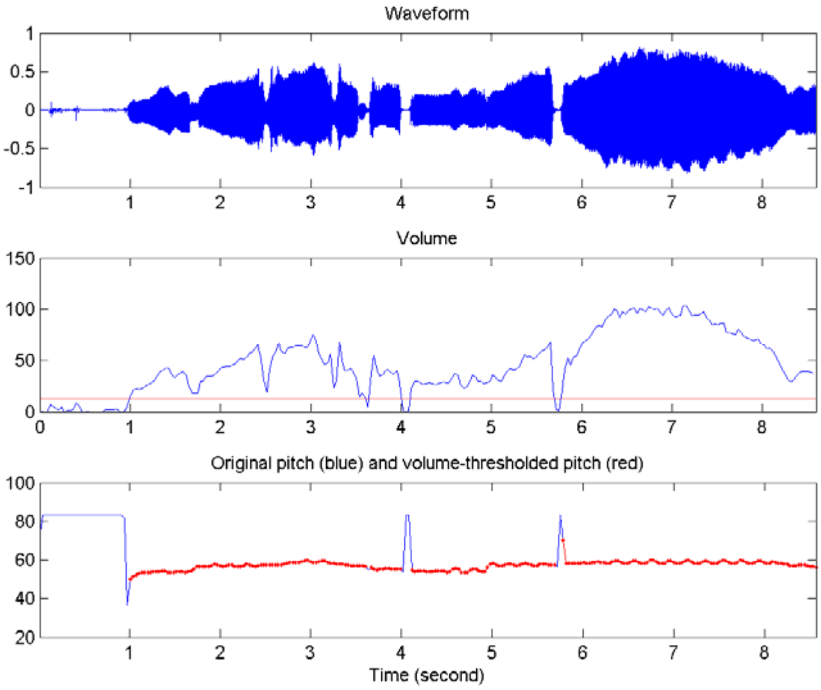
```
waveFile='soo.wav';
[y, fs, nbits]=wavread(waveFile);
y=y-mean(y);
frameDuration=32;     % in ms
```

```
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



In the above example, we use simple volume thresholding with a threshold equal to one eighth of the maximum volume. That is, if a frame has a volume less than 1/8 of the maximum volume, then its pitch is set to zero. From the last plot of the above figure, it is obvious that the values of the pitch curve is mostly correct except for several erroneous points which deviate from the presumably smooth pitch curve. This discontinuous points will cause squeaky sounds during the playback of the pitch curve. We usually apply a smoothing operator (such as median filters) on the pitch curve as a post-process to eliminate erroneous points. Without using the smoothing operator, the results are shown in the following links:

- Original wave file: soo.wav
- Wave file of the identified pitch: sooPitch.wav。

There are several variations of ACF that are also used commonly:

1. The previously defeind ACF has a tapering effect since a larger τ will use a smaller overlap for the calculation. As an alternative, we can compute a new ACF by dividing the inner product by the size of the overlapped region, as shown in the next equation:

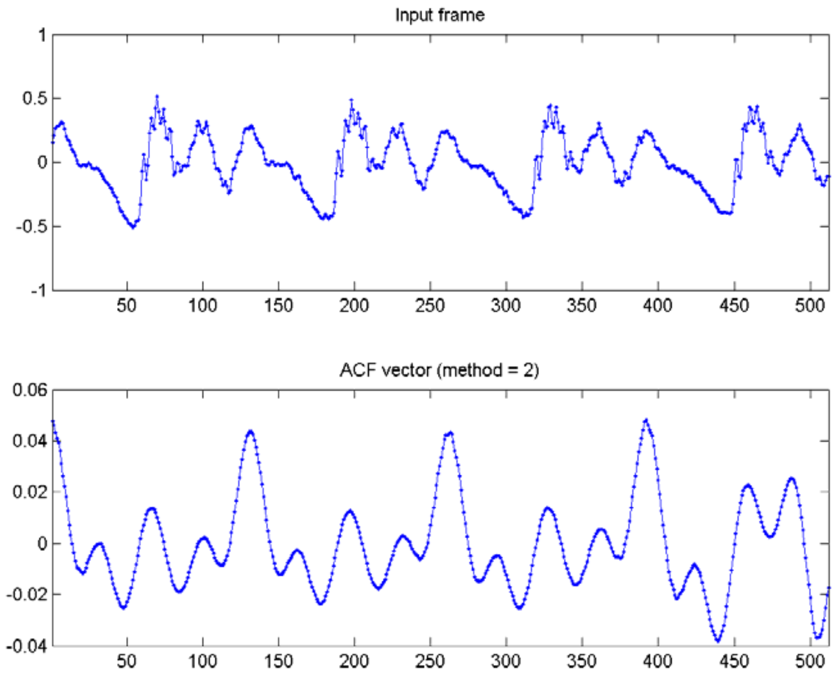
$$acf(\tau) = \Sigma_{i=0}^{n-1-\tau} s(i)\, s(i+\tau)/(n-\tau)$$

Due to the smaller overoap, the last few points of ACF may be unstable, as shown in the following example.

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. Another method to avoid tapering-off is to shift only the first half of a frame, as shown in the next equation:

$$acf(\tau) = \Sigma_{i=0}^{n/2} s(i)\ s(i+\tau)$$

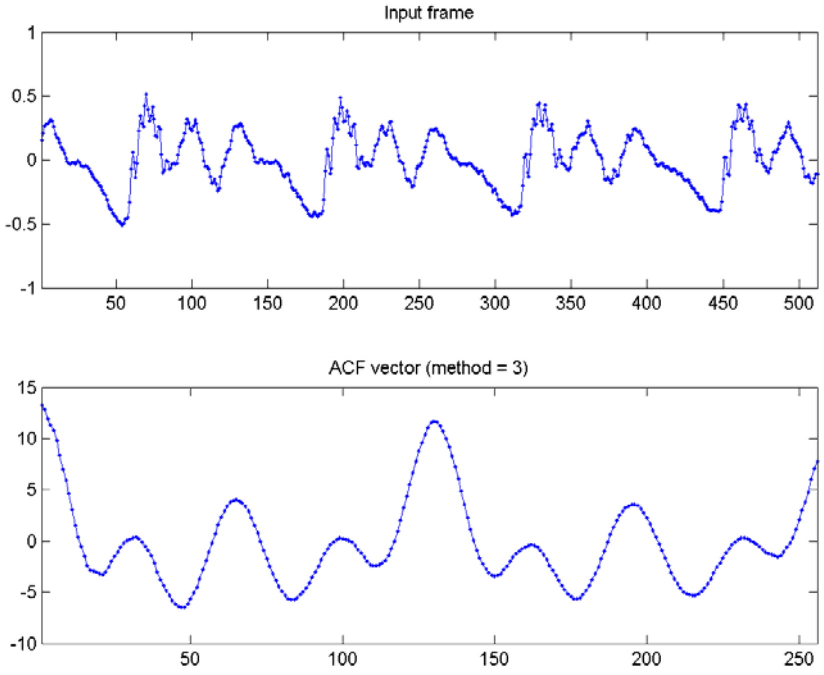
The overlap region will always be half of the frame size and the obtain ACF will not taper off. Please refer to the following example.

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



3. The range of ACF is usually known in advance. We can limit the range of ACF to be [-1, 1] by using the following formula for normalized ACF:

$$acf(\tau) = [2 \Sigma s(i) s(i+\tau)]/[\Sigma s^2(i) + \Sigma s^2(i+\tau)]$$

All the summations in the above equation should have the same lower and upper bounds. The range of the above formula is [-1, 1] due to the following inequality:

$$-(x^2+y^2) <= 2xy <= x^2+y^2$$

If the selected pitch point is $\tau = \tau_0$, then we define the clarity of this frame is

$$\text{clarity} = \text{acf}(\tau_0).$$

A higher clarity indicates the frame is closer to a pure periodic waveform. On the other hand, a lower clarity indicates the frame is less periodic, which is likely to be caused by unvoiced speech or silence.

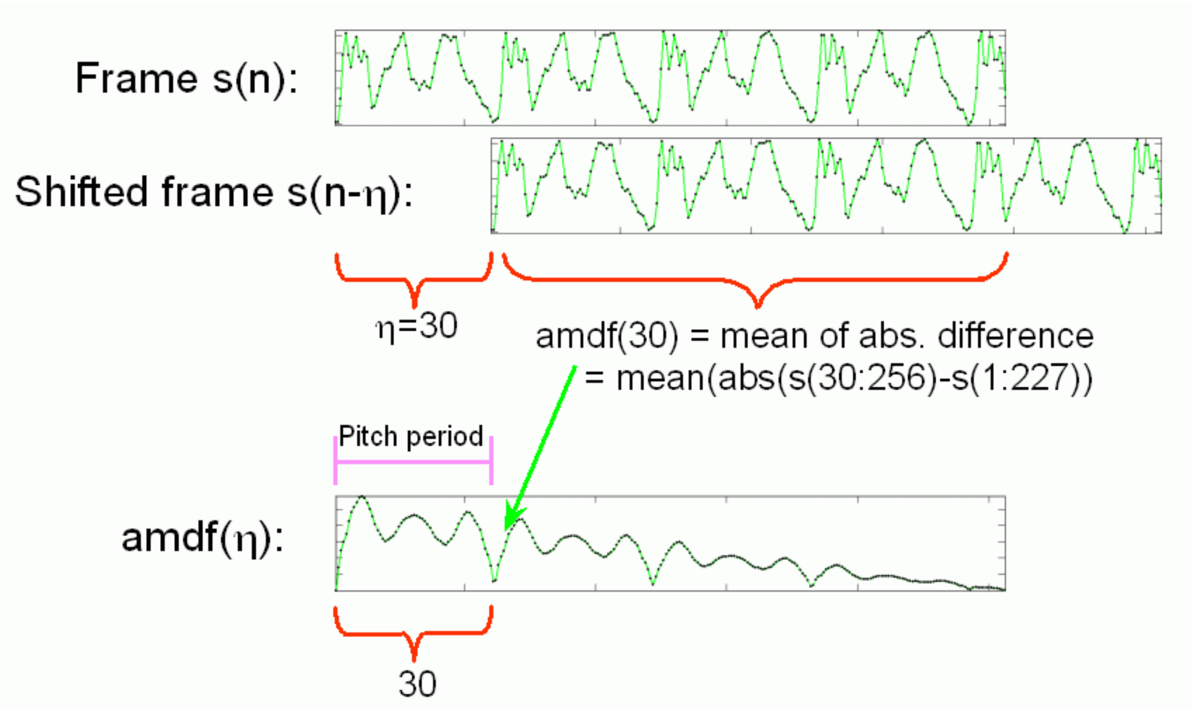
7-3 AMDF

Old Chinese version

The concept of AMDF (average magnitude difference function) is very close to ACF except that it estimates the distance instead of similarity between a frame s(i), i = 0 ~ n-1, and its delayed version via the following formula:

$$\text{amdf}(\tau) = \Sigma_{i=0}^{n-1} | s(i) - s(i-\tau) |$$

where τ is the time lag in terms of sample points. The value of τ that minimizes amdf(τ) over a specified range is selected as the pitch period in sample points. The following figure demonstrates the operation of AMDF:

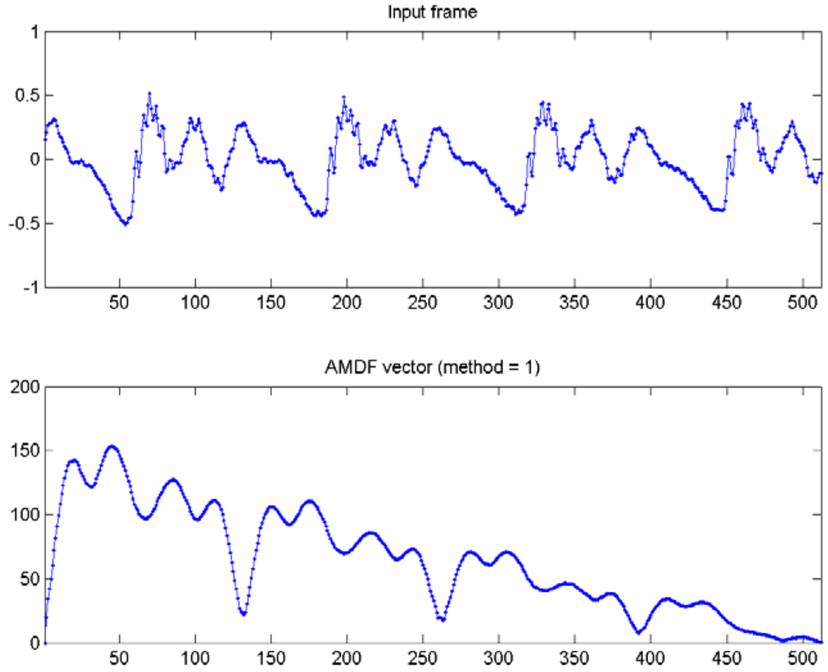

Frame s(n):
Shifted frame s(n-η):
η=30
amdf(30) = mean of abs. difference
= mean(abs(s(30:256)-s(1:227))
Pitch period
amdf(η):
30

In other words, we shift the delayed version n times and compute the absolute sum of the difference in the overlapped parts to obtain n values of AMDF. A typical example of AMDF is shown below.

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



From the above figure, it is obvious that the pitch period point should be the local minimum located at index=132. The corresponding pitch is equal to fs/(132-1) = 16000/131 = 122.14 Hz, or 46.81 semitones. This result is close but not exactly equal to the one obtained via ACF.

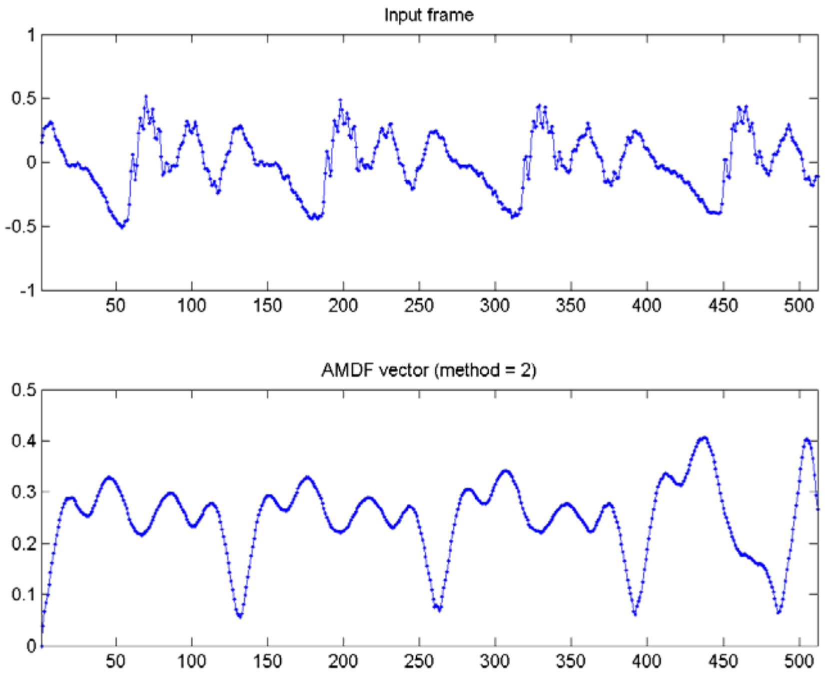
Just like ACF, there are several variations of AMDF, as explained next.

1. Due to less overlap AMDF tapers off with the lag τ. To avoid the taper-off, we can normalize AMDF by dividing it by the length of the overlap. The down sides include more computation and less robustness when τ is small. Here is an example.

   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

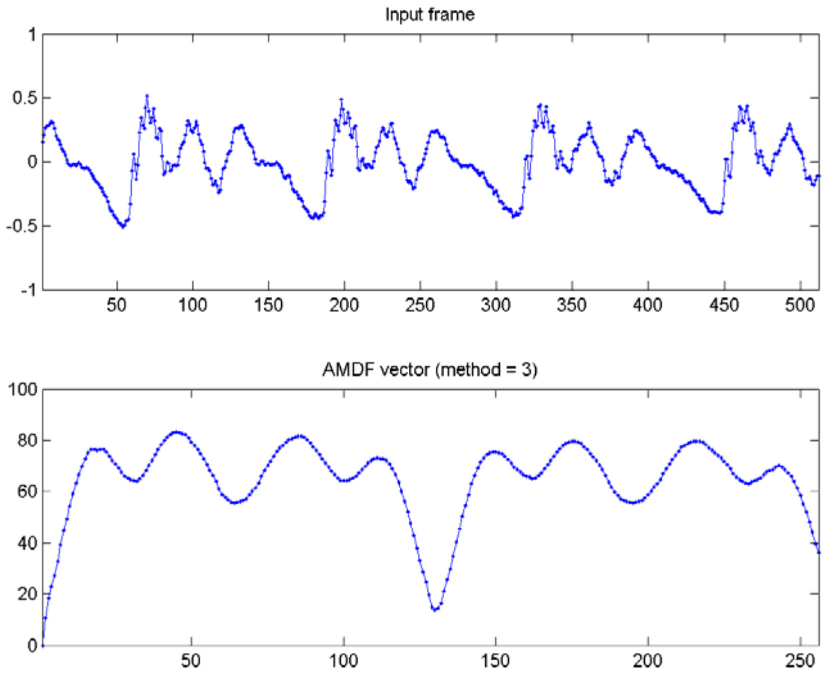


2. Another method to avoid AMDF's taper-off is to shift the first half of the frame only. In the following example, the length of the shifted segment is only 256, leading to an AMDF of 256 points:

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



If the volume of the first half-frame is smaller than that of the second half-frame, it is better to flip the frame first such that a more reliable AMDF curve can be obtained.

Since the computation of AMDF does not require multiplication, it is suitable for low computing platform such as embedded systems or micro-controllers.

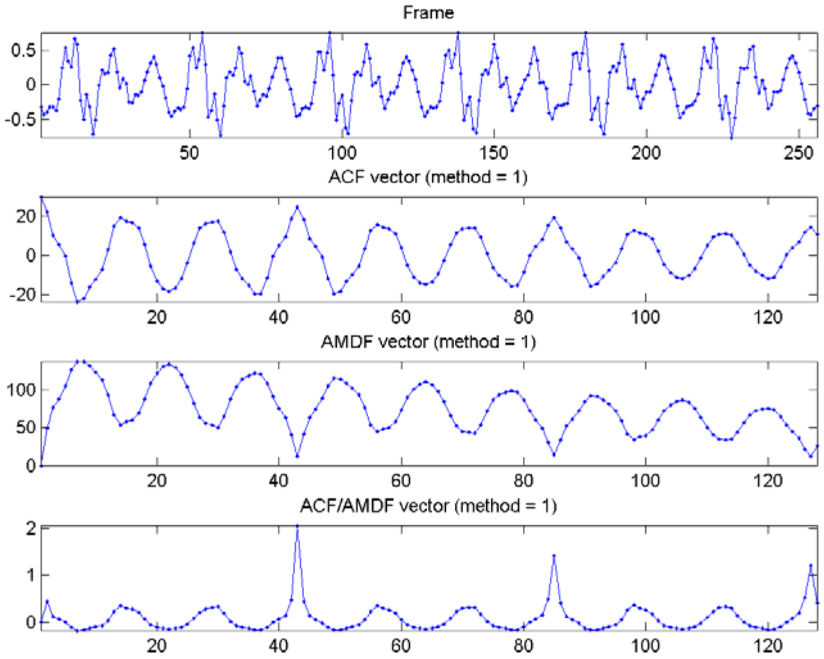
It is possible to combine ACF and AMDF to identify the pitch period point in a more robust manner. For instance, we can divide ACF by AMDF to obtain a curve for easy selection of the pitch point. For example:

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



In the above example, we have used the singing voices from Prof. Soo who was the tenor of the choir at National Taiwan University. In the selected frame, the maxima of ACF or the minima of AMDF is not very obvious. But if we divide ACF by AMDF, the maxima of the final curves are much more obvious than either ACF or AMDF alone.

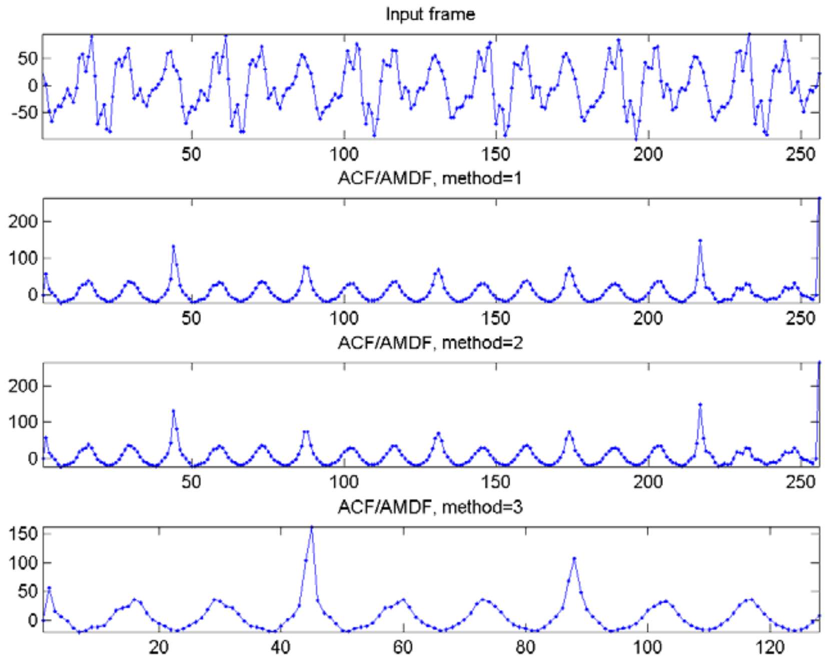
In the next example, we shall use variations of ACF and AMDF for computing ACF/AMDF:

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

```
waveFile='soo.wav';
[y, fs, nbits]=wavReadInt(waveFile);
framedY=buffer(y, 256, 0);
frame=framedY(:, 290);
```

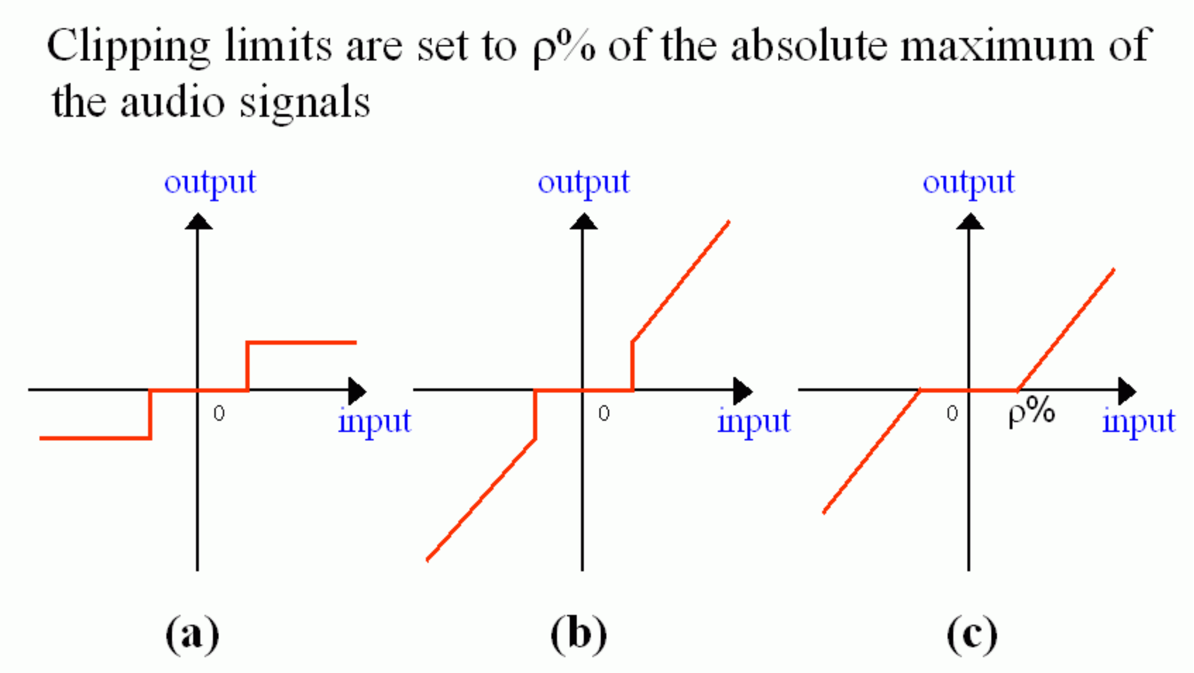
```
subplot(4,1,1);
plot(frame, '.-');
title('Input frame'); axis tight
subplot(4,1,2);
method=1; out=frame2acfOverAmdf(frame, 256, method);
plot(out, '.-'); title('ACF/AMDF, method=1'); axis tight
subplot(4,1,3);
method=2; out=frame2acfOverAmdf(frame, 256, method);
plot(out, '.-'); title('ACF/AMDF, 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



In the above example, method=1 and method=2 lead to the same ACF/AMDF curve.

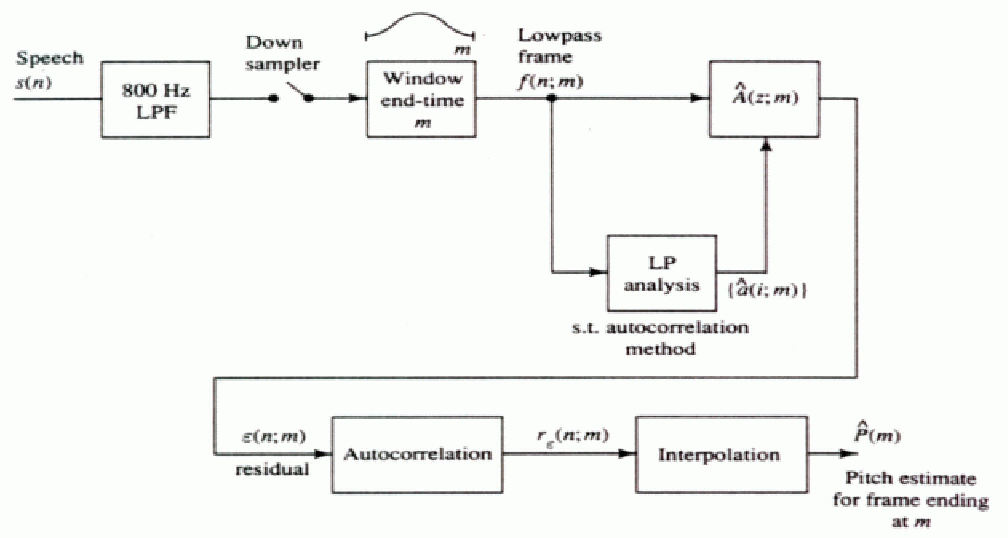
In order to remove noise around zero of the original frame, we can apply center clipping before computing ACF or AMDF. Some of the commonly used techniques for center clipping are display in the following figure:



Clipping limits are set to ρ% of the absolute maximum of the audio signals

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：

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

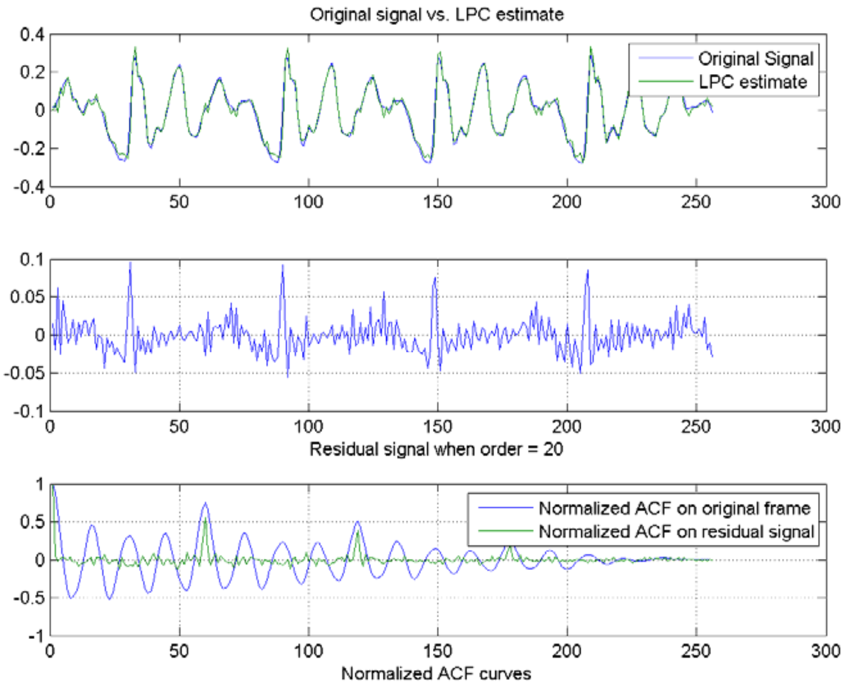
並利用最小平方法來找出最佳的 $\{a_1, a_2, \ldots, a_m\}$，使得 $\Sigma e^2(n)$ 為最小，此 e(n) 又是所謂的 excitation signal （原始激發訊號），再用此 e(n) 來進行 ACF，得到的效果較好。

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。

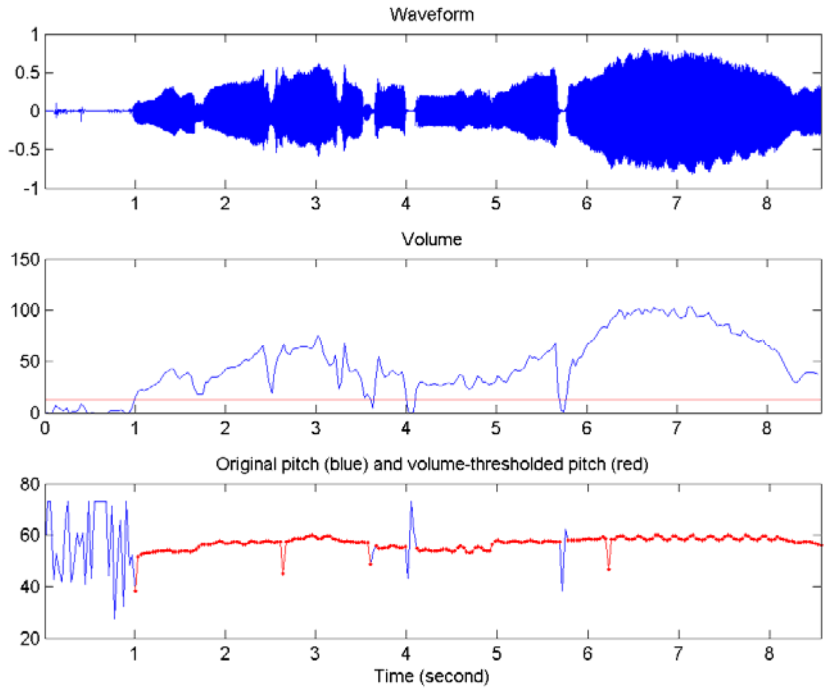
使用 SIFT 加上 ACF 來進行音高追蹤的範例如下：

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：

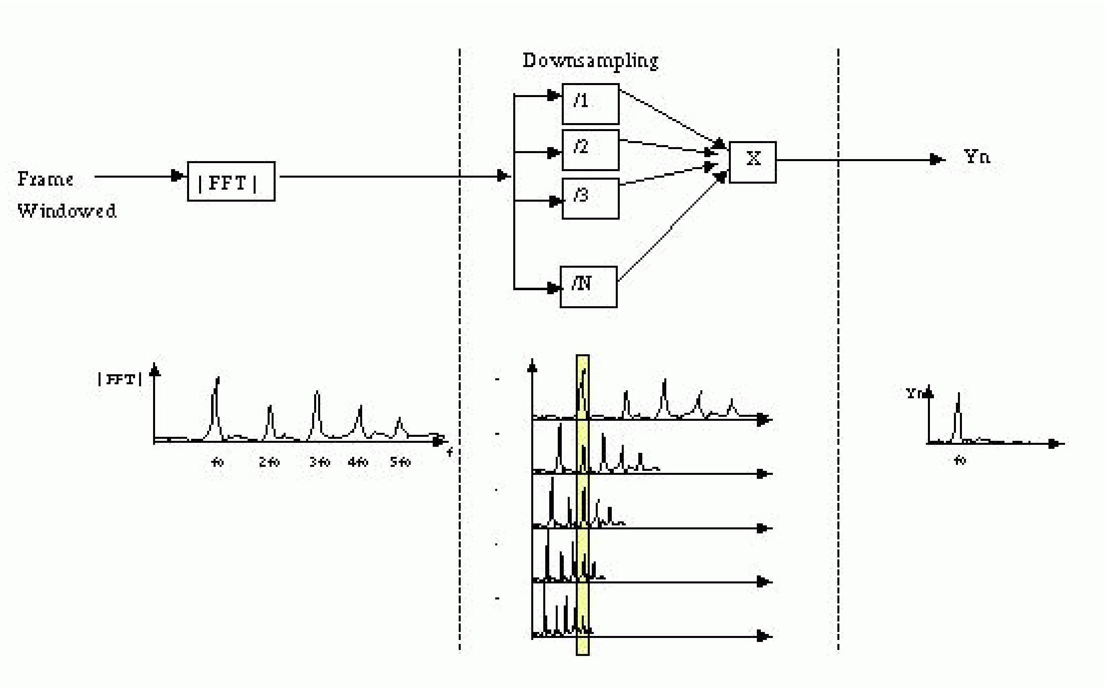
- Pitch Detection Methods Review
- Pitch Detection

7-5 HPS

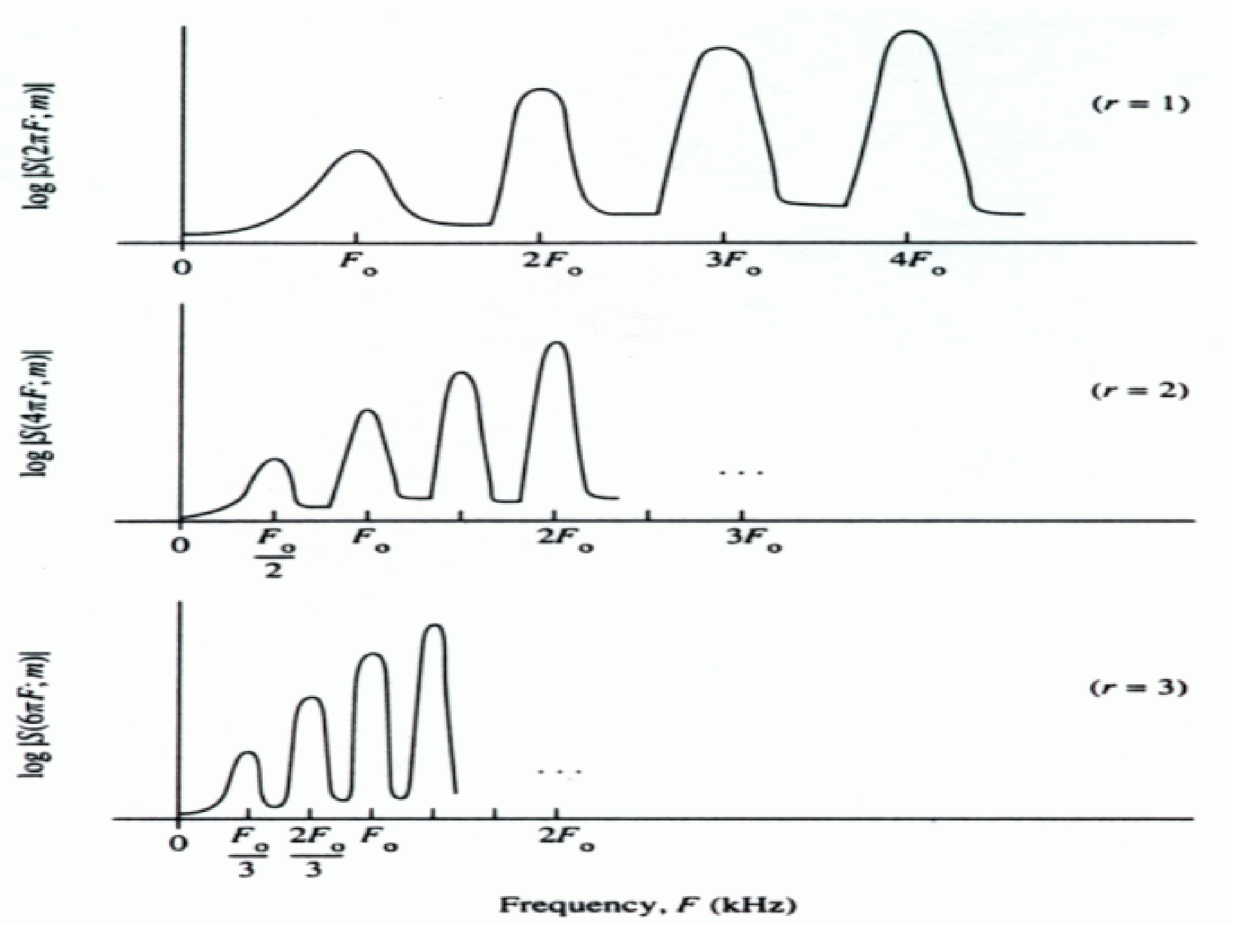
本節介紹在頻域上的音高追蹤的方法，包含

- 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：

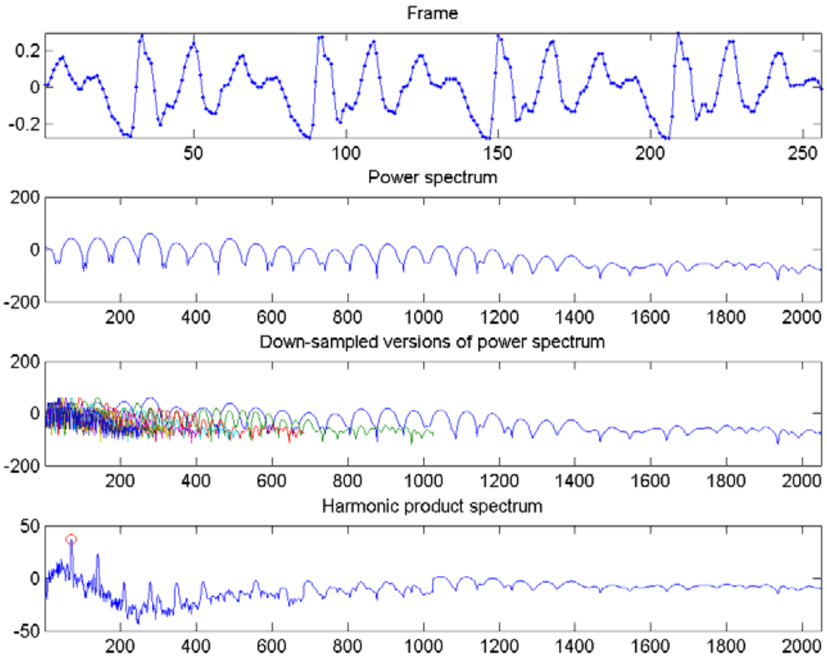
Example 1**Input file** pitchTracking/hps01.m

```
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：

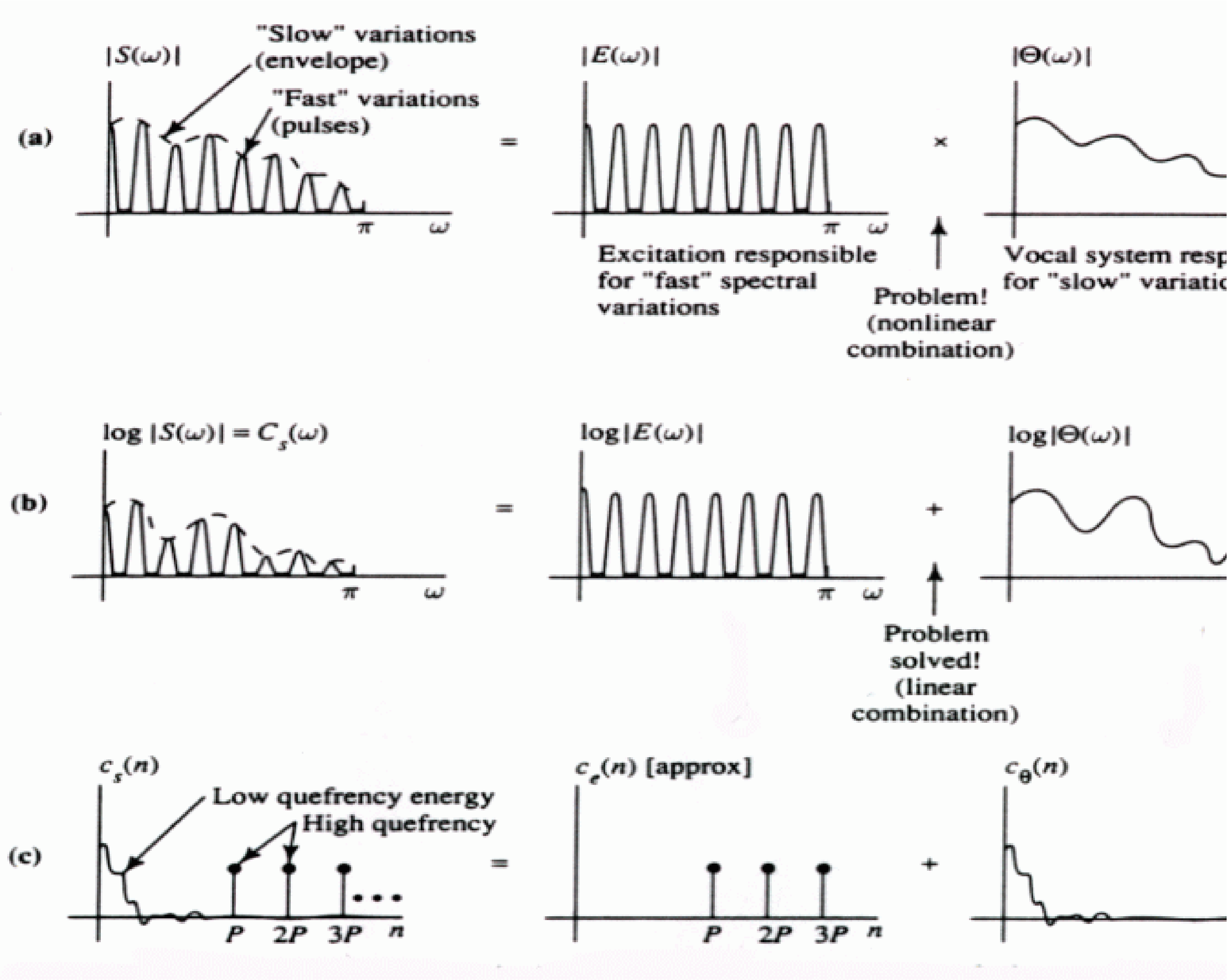
- HPS 所得到的音高解析度並不高，以上述範例而言，若不進行補零（Zero Padding），則無論最高點的位置為何，音高都會是 11025/256 = 43.07 Hz z 的倍數。若需要提高解析度，我們可以將音框進行補零，此時在頻譜就會有較高的解析度，得到的音高解析度也會提高。在上述範例中，我們進行補零直到音框長度為 16*256 = 4096，所得到的頻率解析度是 fs/4096 = 11025/4096 = 2.69 Hz。
- 由 HPS 得到的音高並不很穩定，因為頻譜容易受到共振峰的影響，所以有時候 HPS 得到的最大值並非對應於正確音高。若先拿掉共振峰的影響，再來進行 HPS，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

7-6 Cepstrum

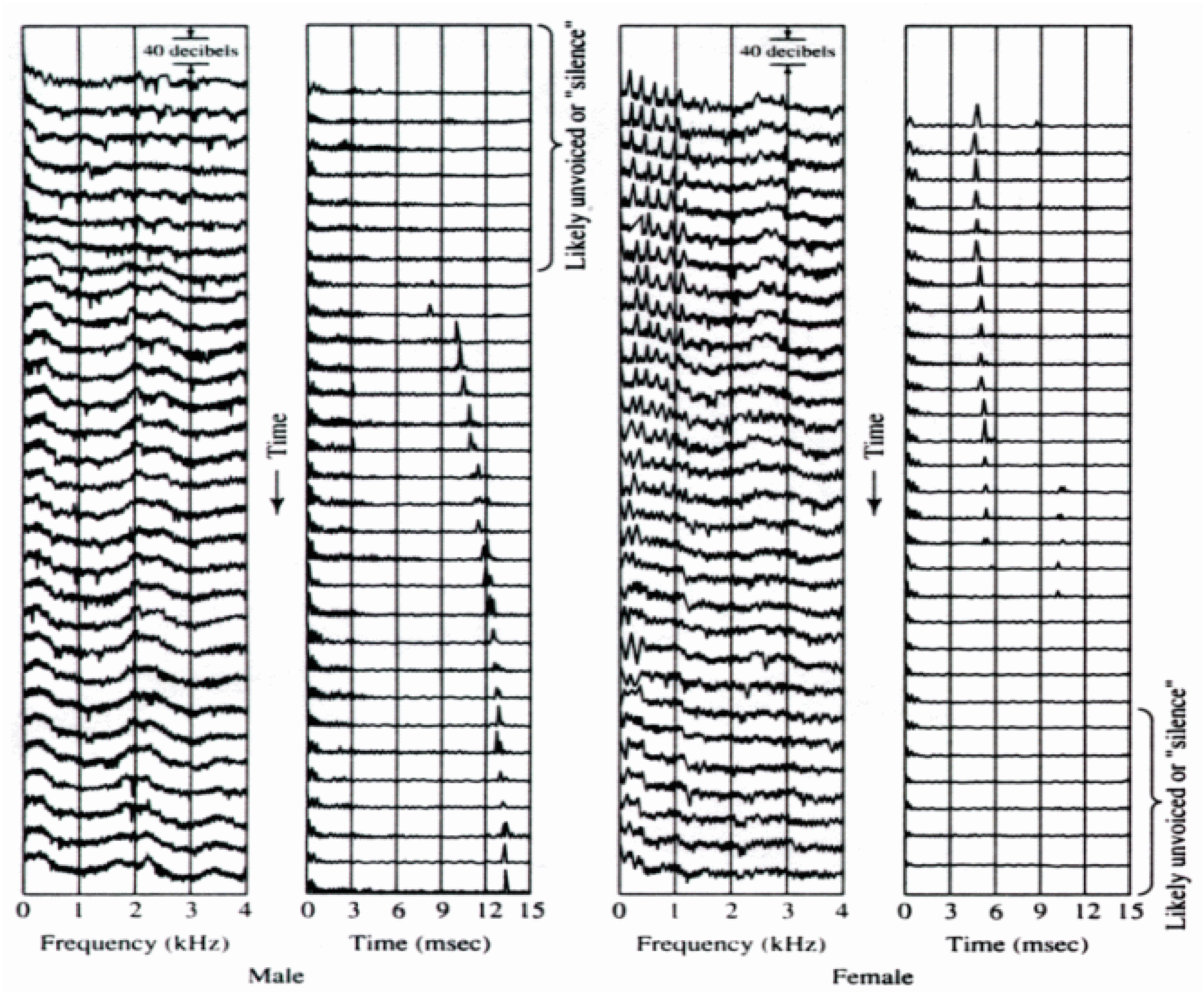
另一個在頻域上的音高追蹤的方法，則是使用「倒頻譜」（Cepstrum），其定義是

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。

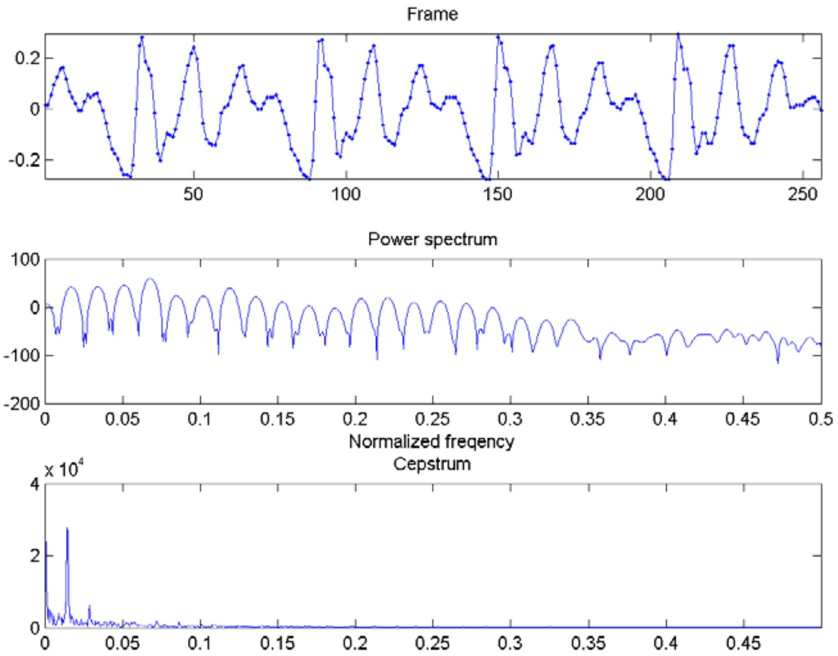
在下面這個範例，我們對一個音框進行倒頻譜的計算：

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：

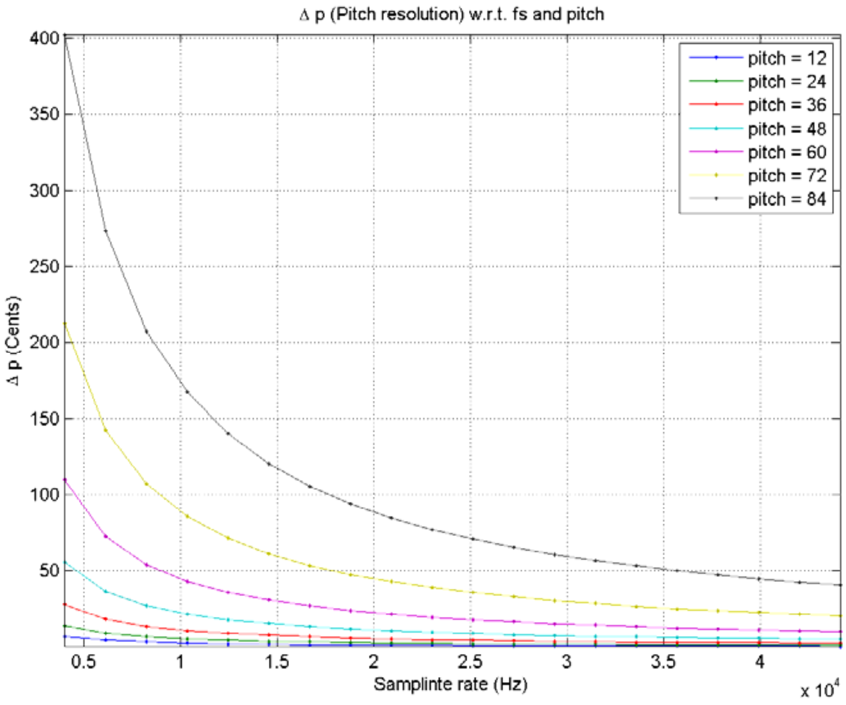
$$\Delta P = (69 + 12*\log_2((fs/L)/440)) - (69 + 12*\log_2((fs/(L+1))/440)) = -12*\log_2(1+1/L) = -12*\log_2(1+f/fs)$$

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

第 7 章作業

Old Chinese version

1. (**) **Frame-to-ACF computation：** Write an m-file function myFrame2acf.m to compute ACF from a given frame, with the following usage:

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

   where frame is the input frame, and acf is the output ACF. The length of acf should be the same as that of frame. (Hint: You can check out the function frame2acf.m in the Audio Processing Toolbox. You can also use xcorr.m in the Signal Processing Toolbox to complete this exercise.)

2. (**) **Frame-to-AMDF computation：** Write an m-file function myFrame2amdf.m to compute AMDF from a given frame, with the following usage:

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

   where frame is the input frame and amdf is the output AMDF. The length of amdf should be the same as that of frame. (Hint: You can check out the function frame2amdf.m in the Audio Processing Toolbox.)

3. (**) **Frame-to-AMDF/ACF computation：** Write an m-file function myFrame2amdfOverAcf to compute AMDF over ACF from a input frame, with the following usage:

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

   Where frame is the input frame, and amdfOverAcf is the output AMDF/ACF. The length of amdfOverAcf should be the same as that of frame. Use this function to plot the curve of AMDF/ACF similar to this example. Does this method better than ACF or AMDF alone? If not, what methods can be used to improve the performance? (Hint: Please refer to frame2acfOverAmdf.m in the Audio Processing Toolbox.)

4. (**) **ACF-to-pitch computation：** Write an m-file function myAcf2pitch.m which computes the pitch from a given vector of ACF, with the usage:

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

where acf is the input ACF vector, fs is the sample rate, pitch is the output pitch value in semitones. If plotOpt is not zero, your function needs to plot ACF and the selected pitch point.

5. (**) **AMDF-to-pitch computation**： Write an m-file function myAmdf2pitch.m which computes the pitch from a given vector of AMDF, with the usage:

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

   where amdf is the input AMDF vector, fs is the sample rate, pitch is the output pitch value in semitone. If plotOpt is not zero, your function needs to plot AMDF and the selected pitch point.

6. (**) **Frame-to-pitch computation**： Write an m-file function myFrame2pitch which computes the pitch from a given frame using various methods, with the usage:

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

   where frame is the vector of input frame, fs is the sample rate, method is the used pitch tracking method ('acf' for ACF, 'amdf' for AMDF, etc), pitch is the output pitch value in semitone. If plotOpt is not zero, your function needs to plot the frame, the ACF or AMDF curve, and the selected pitch point. Moreover, you function should have the capability for self demo. Please refer to frame2acf.m or frame2amdf.m in the Audio Processing Toolbox. (Hint: You will use several functions in the Audio Processing Toolbox, including freq2pitch.m、frame2acf.m、frame2amdf.m, etc. Moreover, you will also use myAcf2pitch.m and myAmdf2pitch.m in the previous exercises.)

7. (**) **Computation, display, and playback of pitch by ACF**： Before trying this exercise, you should fully understand this example since this exercise follow the example closely. In this exercise, you are request to write an m-file script myWave2PitchByAcf01.m which accomplish the following tasks:

- Record your own singing for 8 seconds, with 16 KHz, 16 bits, mono, and save it to a file test.wav. (You can use speech instead of singing, but the pitch tracking will be harder for speech.) Use this file DoReMi.wav unless there is absolutely no way for you to record.
- Read test.wav and do frame blocking with a frame size of 512 and a overlap of 0.
- Compute the volume of each frame and find the volume threshold.
- Compute ACF for each frame.
- Identify the pitch point and compute the pitch in semitone.
- Process the identify pitch vector to make it smooth, and to remove unlikely pitch. Possible methods include:
  a. If a frame has a volume lower than the volume threshold, set the corresponding pitch to zero.
  b. If an element in the pitch vector is out of the range of human voices, set it to zero.
  c. If an element in the pitch vector goes too high or too low compared with its neighbors, set it to the avarage of its neighbors (assuming its neighbors have similar pitch.)
  d. Smooth the pitch vector using median filter. (The corresponding command is median.)
  e. Any other methods that you can think of to do better post-processing on the pitch vector.
- Plot the result with three subplots in a figure:
  a. The first subplot is the original waveform of the audio signals.
  b. The second subplot is the volume.
  c. The third subplot is the identified pitch vector.

  All three subplots should have the same time axis in terms of second.

- Play the identify pitch vector. (Hint: you can use pvPlay.m in the Audio Processing Toolbox.)

You need to fine-tune your program such that the identify pitch should be as close as possible to the original singing. You need to demo the following items to TA:

   9. Playback of test.wav.
   10. Plots mentioned earlier.
   11. Playback of the pitch vector.

8. (***) **Computation, display, and playback of pitch by AMDF：** Write an m-file script myWave2PitchByAmdf01.m to repeat the previous exercise, but use AMDF instead. Compare your result with that of the previous example. (Hint: This is harder than the previous one since there might be several equally good minimum points to select.)
9. (***) **Computation, display, and playback of pitch by ACF/AMDF：** Write an m-file script myWave2PitchByAmdf01.m to repeat the previous exercise, but use ACF/AMDF instead. Compare your result with that of the previous example. (Hint: This is harder than the previous one since there might be several equally good minimum points to select.)
10. (***) **Wave-to-pitch computation：** Write an m-file function myWave2pitch.m which compute a pitch vector from a given stream of audio signals, with the usage:

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

where wave is input audio signals, fs is the sample rate, frameSize is the frame size in samples, overlap is the overlap in samples, and method specifies the method used for pitch tracking: 'acf' for ACF and 'amdf' for AMDF. If there is no pitch in a given frame, the corresponding element in the output pitch vector should be zero. (Hint: you should try the previous exercise before attemping this one.)

11. (*) **Recording task: Children's song recording：** This recording task requires you to do recordings of children's songs and to manually label the pitch of the recordings. Please refer to this page for more details.

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）。

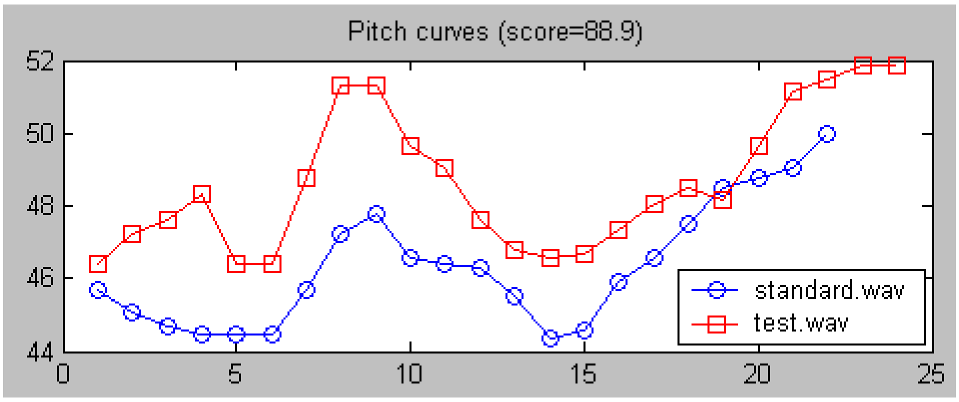
（待完成，請稍候，謝謝！）

8-2 音調評分

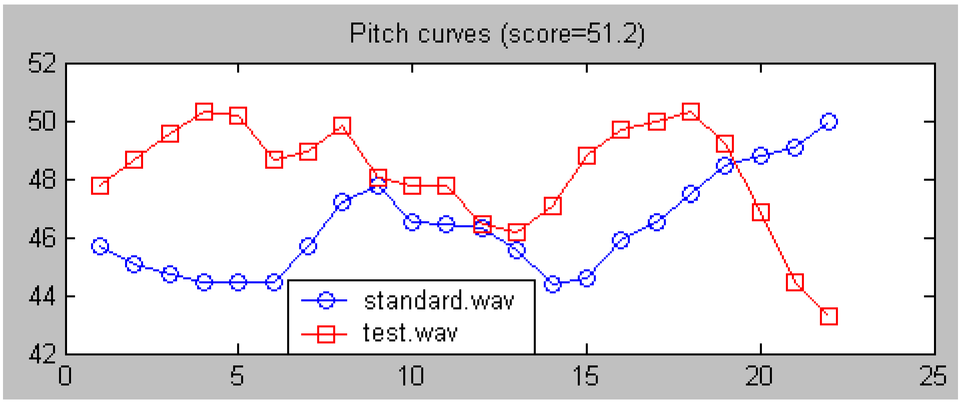
「音調評分」（Intonation Assessment）的目的是要以電腦來自動評斷一個人發音的音調是否標準，並提示相異之處。舉例來說，外國人到了台灣學國語，最難學的就是國語的四聲，所以他們會說「窩腰刀胎杯刊顛鷹」（我要到台北看電影），我們一聽，就知道是外國人在講國語，因為完全沒有音調（全部都是一聲）。同樣地，我們在學英文時，碰到較長的句子，也很難把英文的音調抓準，大部分只能平平地唸，因此老外一聽到老中講音調怪怪的英文，就知道你不是 native speaker。「音調評分」的目的，就是希望以電腦的快速運算能力，來顯示你講的英文語調和老外的標準語調到底差多少，並同時顯示差異之處，以達到電腦輔助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

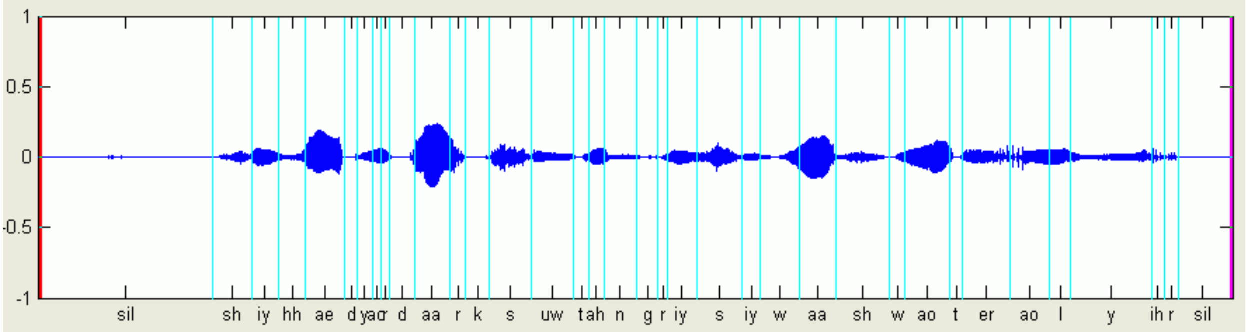
由於個人電腦的普及和運算速度的提高，「電腦輔助語言學習」（CALL, Computer Assisted Language Learning）的研發已經到達可以商業化的階段，其中又以「電腦輔助發音訓練」（CAPT,

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：

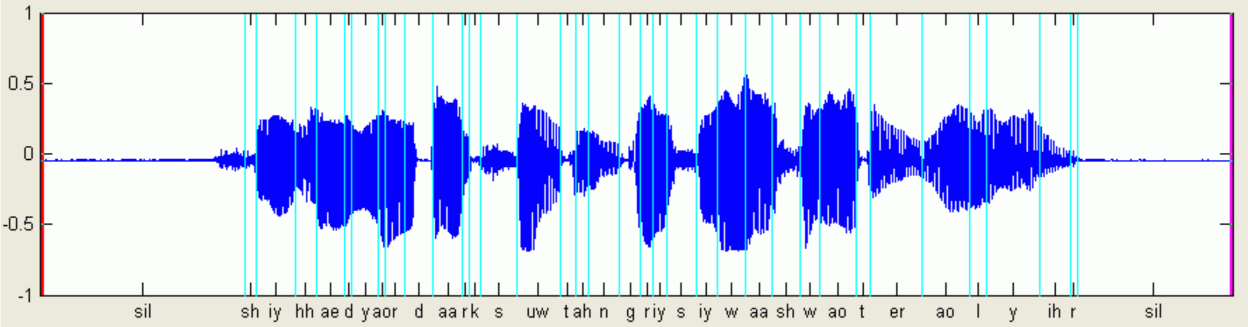
1. 對標準語句及測試語句抽取出語音的特徵參數(通常是 MFCC, Mel-frequency Cepstral Coefficients）。
2. 以 Viterbi Decoding 來進行 Forced Alignment，以便切出來每一個子音及母音。此部分需用到語者無關（Speaker-independent）的英文語音辨識核心。
3. 對每一個子音及母音進行評分因素的擷取，包含音量、音高、音長等，以及之前已經取得的 MFCC。
4. 對每一個評分因素進行個別評分，然後進行加權平均，以得到最後的評分結果。這些評分的因素有：
   a. 音色：語音的內容及發音的準確性，此部份的分數通常是經由計算 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：



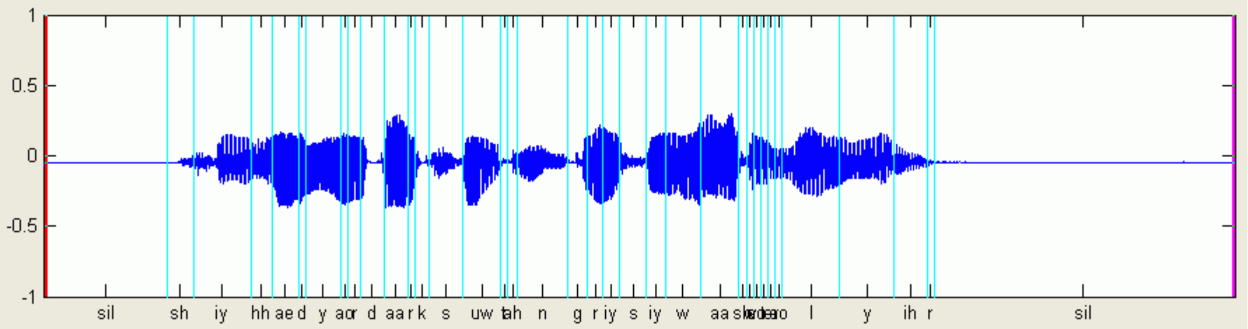
得到的評分結果是：

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

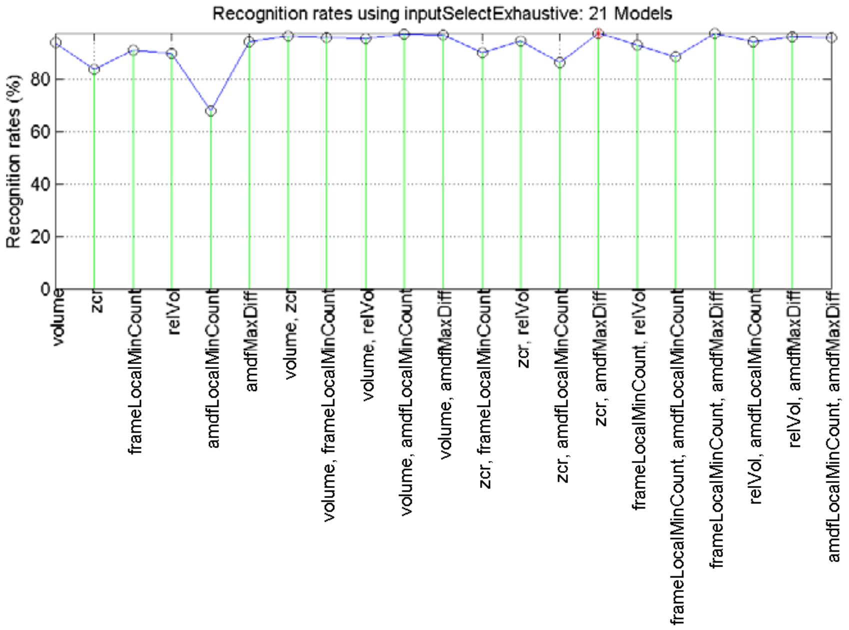
（待完成，請稍候，謝謝！）

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

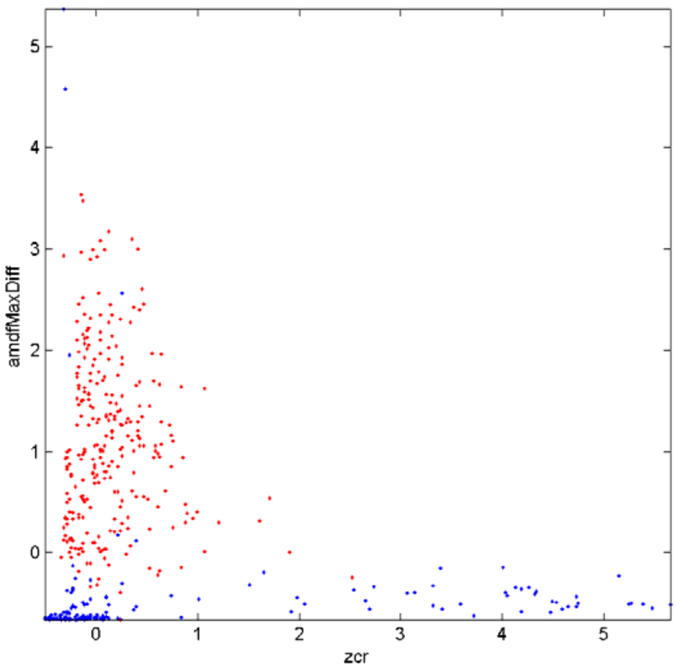
5 個）即可。請寫一個 MATLAB 程式 pitchExistTest01.m，先載入資料 DS.mat，然後對資料集進行正規化後（將每個特徵的數值範圍進行線性調整，使其機率分佈接近於平均值為零、標準差為 1 的高斯機率密度分佈，可用 dataNormalize.m 來對資料正規化）後，來進行下列作業 ．：

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：



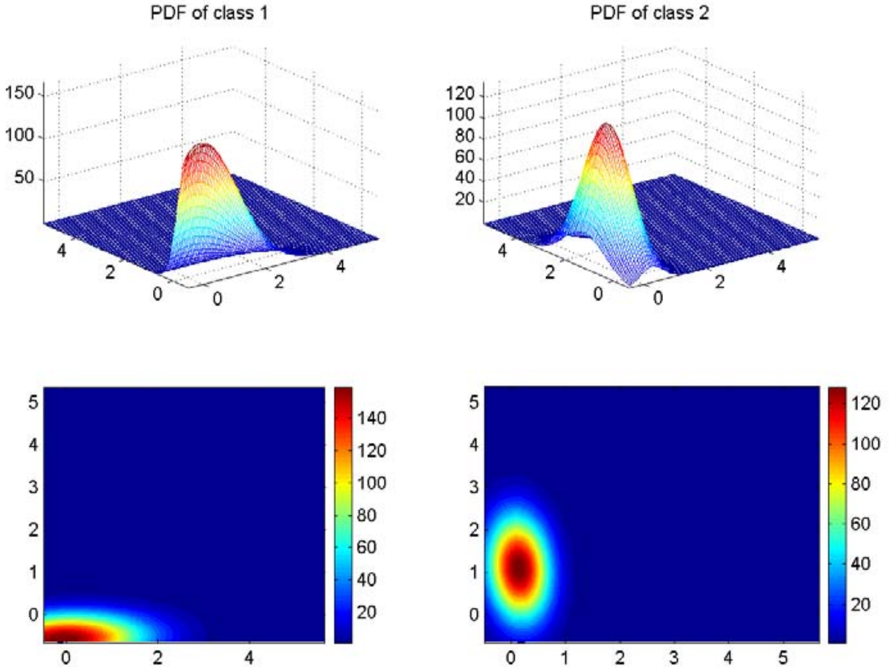
（提示：可用 inputSelectExhaustive.m 來進行輸入特徵的窮舉法選取。）

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：



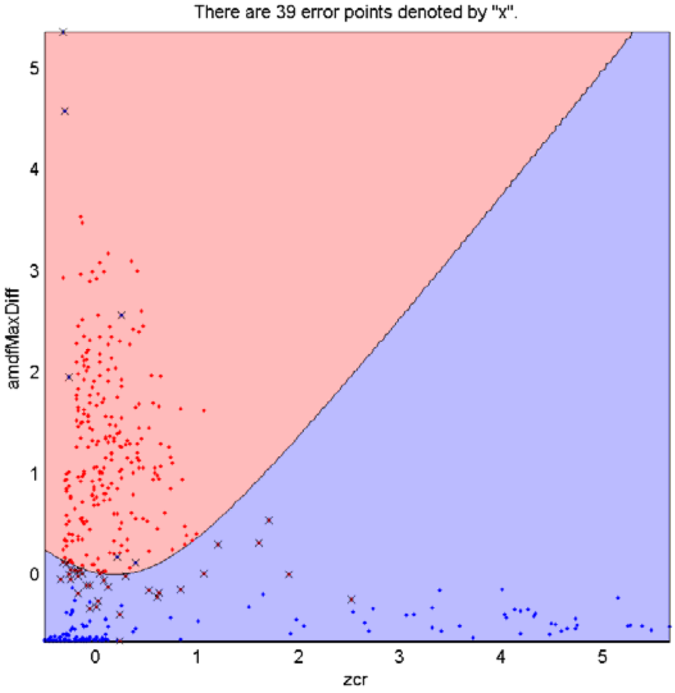
（提示：可用 dcprDataPlot 來畫出資料的分佈圖。）

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)

9-1 Discrete-Time Signals (離散時間訊號)

Old Chinese version

All signals in the real-world are continuous. In contrast, all signals stored in a computer use the unit of byte as the basic storage unit. Therefore these signals are referred to discrete-time signals. In mathematical notations, discrete-time signals can be represented by x(nT), where T is the sampling period (the reciprocal of the sample rate), n is an integer and nT is the elapsed time. For simplicity, we shall use x[n] to represent the discrete-time signal at time nT, with T implicitly defined.

Hint

In fact, signals stored in a computer are discrete in both time and their values. For the discussion for this chapter, we are only concerned with the discretization in time.

In the following, we shall introduce some of the commonly used discrete-time signals:

The unit impulse signal δ[n] is zero everywhere except at n = 0, where its value is 1. Mathematically, we can express the unit impulse signal as the following expression:

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

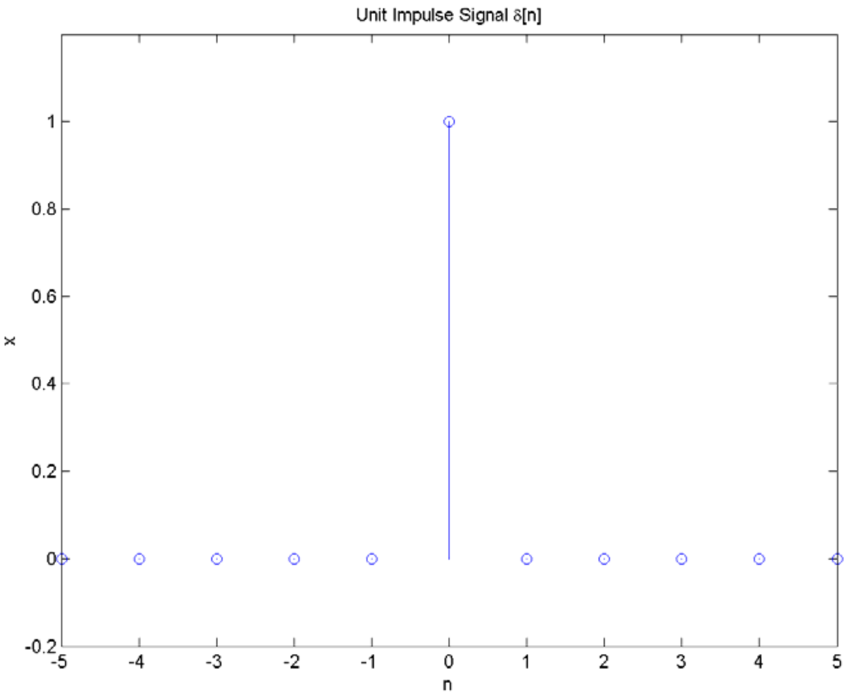
$$\delta[n] = 0, \text{ otherwise}$$

We can plot the unit impluse signal as follows:

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



If we delay the unit impulse signal by k sample points, we have the following equation:

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

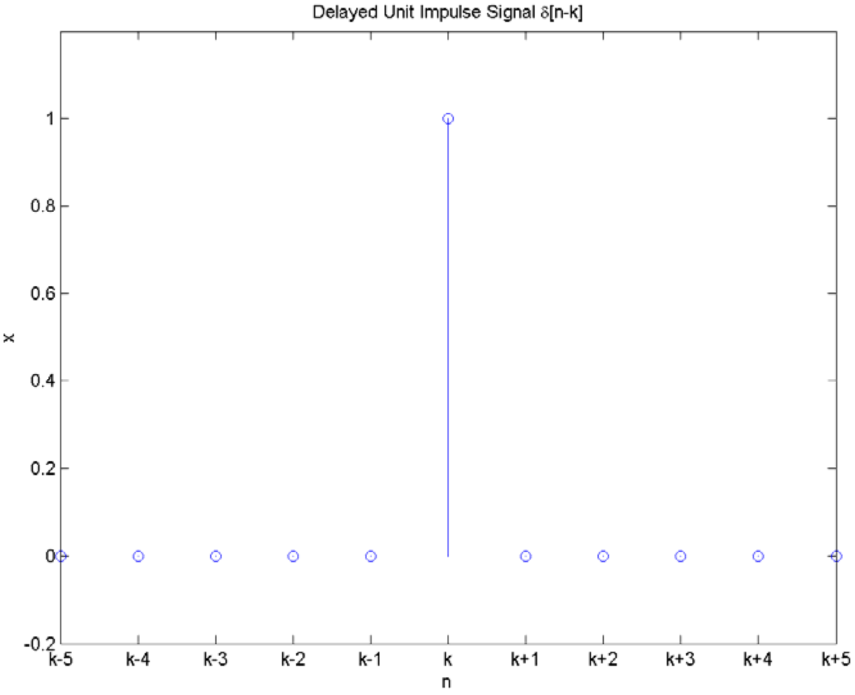
$$\delta[n-k] = 0, \text{ otherwise}$$

The above delayed unit imulse signal can be plotted as follows:

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



The above delayed unit impulse signal have the property of masking out other signals not at n = k. In other words, if we multiply any signals x[n] (n = -∞～∞) with δ[n-k], only the term of x[n-k] is left. This can be expressed as the following equation:

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

By using this property, we can express any discrete-time signals as a linear combination of the unit impulse signals. This is especially handy when we derive the response of a linear time-invariant system. See the following section for more details.

The unit step signal u[n] is 1 when n≧0, and 0 elsewhere. In mathematical notations, we have

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

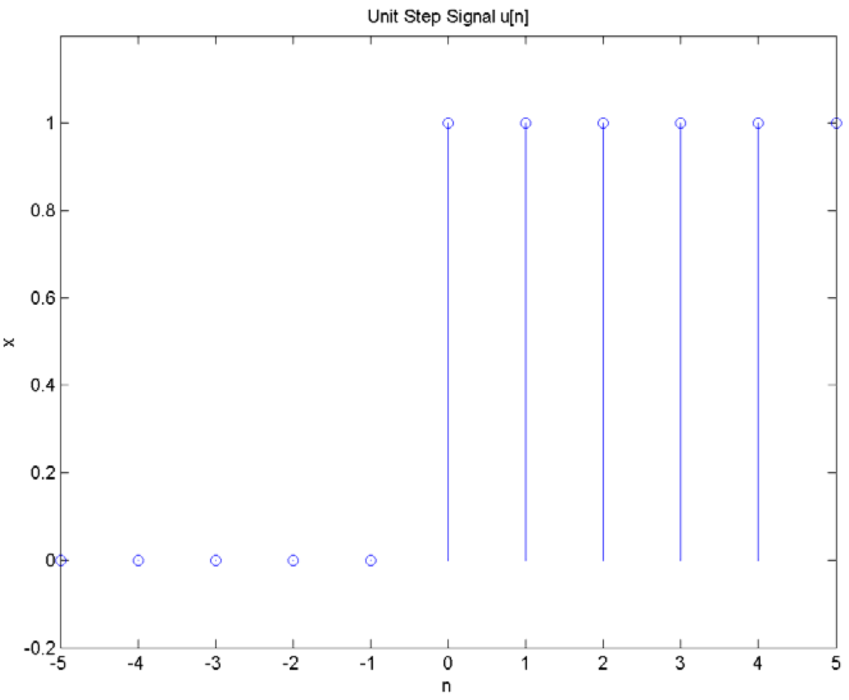
$$u[n] = 0, \text{ otherwise}$$

The unit step signal can be plotted, as shown in the next example:

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



It should be noted that the unit impulse signal is the first-order difference of the unit step signal:

$$\delta[n] = u[n] - u[n-1].$$

The sinusoidal signal is another commonly used discrete-time signals:

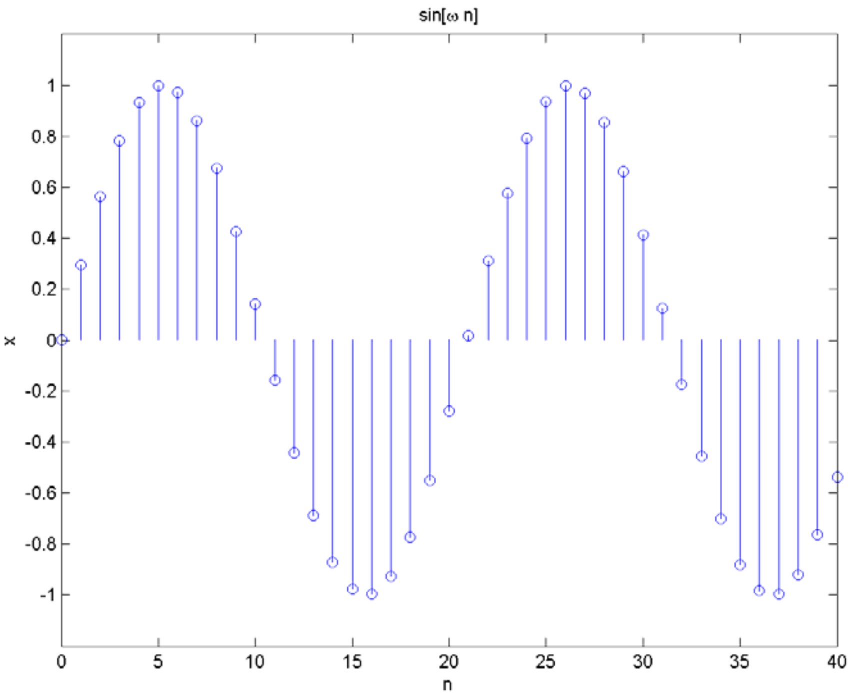
$$\sin[n] = \sin(2\pi f(nT)) = \sin(\omega n))$$

where f is the oscillation frequency of the sinusoidal signal and ω (= $2\pi fT$) is the **normalized angular frequency**. The sinusoidal signal can be plotted as follows:

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



9-2 Linear Time-Invariant Systems (線性非時變系統)

Old Chinese version

For any given discrete-time signal x[n], we can send it to a system to obtain the output signal y[n], with the following mathematical notations:

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

In other words, the input to the sysmte is a function x[n], n = 0～∞, while the output is also a function y[n], n = 0～∞。

If the system, denoted by L{‧}, satisfies the following equations, it is called **linear**:

1. If L{x[n]} = y[n], then L{kx[n]} = ky[n].
2. If L{x_1[n]} = y_1[n] and L{x_2[n]}= y_2[n], then L{x_1[n] + x_2[n]} = y_1[n] + y_2[n].

The above equations can be reduced to a single one:

If L{x_1[n]} = y_1[n] and L{x_2[n]}= y_2[n], then L{ax_1[n] + bx_2[n]} = ay_1[n] + by_2[n], for all constants a and b.

The above equation is referred to as the **superposition principle**. Namely, if a system satifies the superposition principle, then it is a linear system.

If L{‧} satisties the following equation, it is called **time-invariant**:

If L{x[n]} = y[n], then L{x[n-k]} = y[n-k], for all k ≧ 0.

If a system is linear and time-invariant, we call it a **linear time-invariant system**, or **LTI sytem** for short.

Hint

For simplicity, we shall assume the input signal x[n] = 0 when n < 0. In other words, x[n] is activated when n ≧ 0, thus y[n] is nonzero only when n ≧ 0.

For the rest of this book, we should assume all the systems under discussion are LTI systems.

9-3 Convolution (旋積)

Old Chinese version

For any given arbitrary signal x[n], we can express it as a linear combination of the unit impulse signals, as follows:

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

The right-hand side of the above equation can be viewed as the situation where

- k is the index for time.
- x[k] is a fixed function of time index k.
- δ[n-k] is a function parameterized by n.

Similarly, the above equation can be rewritten into another format:

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

The right-hand side of the above equation can be viewed as the situation where

- k is the index for time.

- $\delta[k]$ is a fixed function of time index k.
- $x[n-k]$ is a function parameterized by n.

For a given LTI system L{·}, when the input signal x[n] is decomposed by the first method, the output y[n] can be expressed as follows:

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

where $h(n-k) = L\{\delta(n-k)\}$ is the response of the system with respect to the input of the unit impulse signal at n = k. In other words, the output of an LTI system is determined by the input signal x[n] and the system's **impulse response** h[n]. More specifically, the impulse response of an LTI system exclusively determine the characteristics of the system.

We can use the following plots to demonstrate the operations of the above formula:

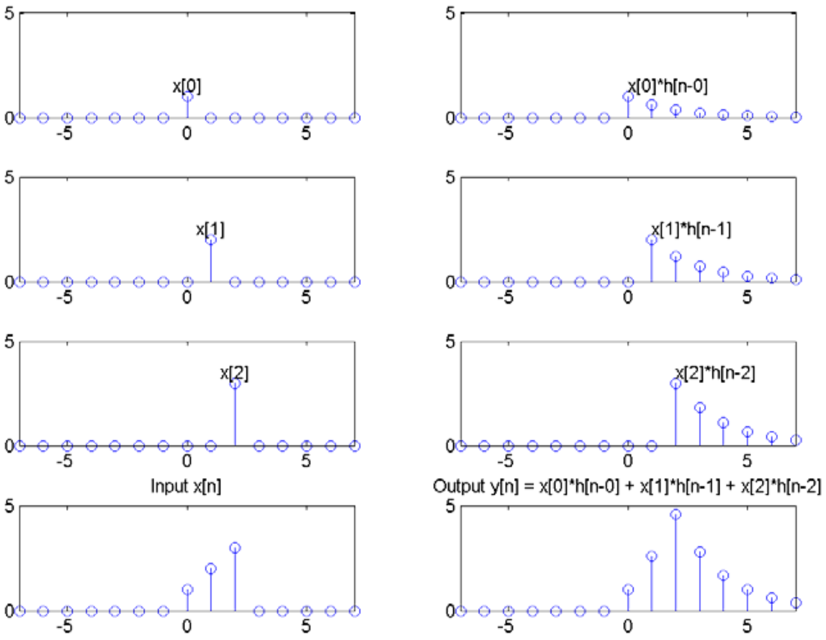
Example 1**Input file** digitalSignalsAndSystems/convolution01.m

```
% 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



If we choose to use the second method for decomposing x[n], then y[n] can be expressed as follows:

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

Since the computation of y[n] is used frequently, we shall define the **convolution** of two signals x[n] and h[n] as follows:

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

The convolution operator has the following characteristics:

1. Commutative law:

$$x[n] \ast y[n] = y[n] \ast x[n]$$

2. Associative law:

$$(x[n] \ast y[n]) \ast z[n] = x[n] \ast (y[n] \ast z[n])$$

3. Distributive law:

$$x[n] \ast (y[n]+z[n]) = x[n] \ast y[n]+x[n] \ast z[n]$$

4. Shift property:

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

5. Convolving with unit impulse:

$$x[n] \ast \delta[n] = x[n]$$

6. Width:

If duration(x[n])=N_1 and duration(y[n])=N_2, then duration(x[n]＊y[n])=N_1+N_2-1.

9-4 Eigen Functions (固有函數)

Old Chinese version

If the output of a system has the same type as its input signal, then the input signal is referred to as the **eigen function** of the system. In this section, we shall demonstrate that the exponential function (including the sinusoidal function as a special case) is an eigen function of the LTI system.

Suppose that the input signal to our LTI system is an exponential function:

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

Then the output can be derived based on convolution:

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

In other words, the output signal y[n] is equal to the original input signal multiplied by a constant $H(e^p)$（= $\Sigma_k e^{-pk}$ h[k], which is not a function of time）. Hence it can be established that the exponential function is the eigen function of an LTI system.

Based on the above fact, for any given input signal that can be decomposed as a linear combination of exponential functions, we can then apply the superposition principle to find the responses of these exponential functions first, and then linearly combine these responses to obtain the final output. In mathematical notations, we have:

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

Since the exponential function and the sinusoidal function are related by the famous **Euler identity**:

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

Therefore we can express the sinusoidal function as a linear combination of the exponential function:

$$\cos(\theta) = (e^{j\theta} + e^{-j\theta})/2 = \mathrm{Re}\{e^{j\theta}\}$$

$$\sin(\theta) = (e^{j\theta} - e^{-j\theta})/(2j) = \mathrm{Im}\{e^{j\theta}\}$$

As a result, when the input signal is:

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

The corresponding response is:

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

From the above derivation, we can see that the output is still a sinusoidal function, except that the amplitude is multiplied by $|H(e^{j\omega})|$, and the phase is shifted by $\theta = \angle H(e^{j\omega})$.

Some terminologies are explained next.

- The multiplier $|H(e^{j\omega})|$ represents how the system amplifies or compresses the input signal depending on the angular frequency ω. Hence $|H(e^{j\omega})|$ is called **magnitude frequency response**.
- $\angle H(e^{j\omega})$ is called **phase frequency response**, which is not used often since human's aural perception is not sensitive to phase shift.
- $H(e^{j\omega})$ is called the system's **frequency response**.

Similarly, when the input signal is

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

The output can be derived as follows:

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

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

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

In summary, we have the following important conclusion. For an LTI system, if the input signal is a single sinusoidal function, the output is also a same-frequency sinusoidal function, with the amplitude and phase modified by the system's frequency response $H(e^{j\omega})$.

In general, we can use an complex exponential function to subsume the input of sine and cosine functions:

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

The corresponding output is

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

Alternatively, $H(e^{j\omega})$ is also referred to as the the **discrete-time Fourier transform** of h[n], which can be expressed as the following equation:

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

In the real world, we cannot have the complex exponential function as the input to a system. However, the derivation based on complex numbers does hold mathematically and it indeed makes our derivation more concise.

Chapter 10: Fourier Transform (傅立葉轉換)

10-1 Discrete-Time Fourier Transform (離散時間傅立葉轉換)

Old Chinese version

In the previous chapter, we have covered several important concepts:

- The frequency response of an LTI system is the discrete-time Fourier transform of the system's impulse response.
- The frequency response of an LTI system specifies how the system changes the amplitude and phase of an input sinusoidal function with a specific angular frequency ω.

In particular, for a given discrete-time signal x[n], its discrete-time Fourier transform (DTFT for short) and the corresponding inverse transform can be expressed as follows:

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

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

We can derive the second expression from the first expression easily. Since the period of $X(e^{j\omega})$ is 2π, we can use any interval of length 2π for integration.

From the second expression, we can decompose x[n] into an infinite linear combination of $e^{j\omega n}$, where the amplitude of each basis function is specified by $X(e^{j\omega})$. In other words, $X(e^{j\omega})$ is actually the amplitude of the component of x[n] at the continuous angle frequency ω. This can be futher elaborated, as follows.

From elementary calculus, we know that the integration of f(x) can be approximated by summation:

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

Therefore if x[n] is real, we can approximate it by using summation:

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

From the derivation above, x[n] has been decomposed into the linear combination N cosine functions with the angular frequency ω from 0 to $2\pi(N-1)/N$, and the corresponding amplitude $|X(e^{j\omega})|/N$.

From the discussion in this section and the previous section, we know that there are two important meanings of DTFT：

1. If h[n] is the impulse response of an LTI system, then $H(e^{j\omega})=\Sigma_k h[k]e^{-j\omega k}$ represents the gain ($|H(e^{j\omega})|$) and phase shift ($\angle H(e^{j\omega})$) when the input signal's angular frequency is ω.
2. If x[n] is an arbitrary signal, then $X(e^{j\omega})=\Sigma_k x[k]e^{-j\omega k}$ represents the amplitude ($|H(e^{j\omega})|$) and the phase shift ($\angle H(e^{j\omega})$) of the component at ω of x[n].

Some important properties of DTFT are listed next.

1. Linearity:

$$z[n] = a\ x[n] + b\ y[n] \longleftrightarrow Z(e^{j\omega}) = a\ X(e^{j\omega}) + b\ Y(e^{j\omega})$$

2. Periodicity:

$$X(e^{j(\omega + 2\ k\ \pi)}) = X(e^{j\omega})$$

   Namely, the period of DTFT is 2π。

3. Delay in time:

$$y[n] = x[n-k] \longleftrightarrow Y(e^{j\omega}) = e^{j\omega k}\ X(e^{j\omega})$$

4. Convolution:

$$y[n] = x[n]*h[n] \longleftrightarrow Z(e^{j\omega}) = X(e^{j\omega})Y(e^{j\omega})$$

   That is, convolution in time domain corresponds to multiplication in frequency domain.

If x[n] is a real sequence, we can separate its DTFT $X(e^{j\omega})$ into the real and imaginary parts:

$$\begin{aligned} X(e^{j\omega}) &= \Sigma_k e^{-j\omega k}\ x[k] \\ &= \Sigma_k x[k]\ \cos(\omega k) - j\ \Sigma_k x[k]\ \sin(\omega k) \\ &= X_R(e^{j\omega}) + j\ X_R(e^{j\omega}) \end{aligned}$$

其中

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

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

This functions have the following properties:

- Conjugate symmetric: $X^*(e^{j\omega}) = X(e^{-j\omega})$
- $X_R(e^{j\omega})$ is an even function.
- $X_I(e^{j\omega})$ is an odd function.
- $|X(e^{j\omega})|$ is an even function.
- $\angle X(e^{j\omega})$ is an odd function.

10-2 Discrete Fourier Transform (離散傅立葉轉換)

Old Chinese version

In the previous section, we have introducde the discrete-time Fourier transform (DTFT) that transforms a discrete-time signal into a continuous function of its frequency components. Since DTFT is continuous in frequency, it is not suitable for digital processing by computers. In this section, we shall introduce **discrete Fourier transform** (**DFT** for short) that converts a discrete-time signals into its discrete-frequency components, which are suitable for further computer processing.

Suppose that the discrete-time signal can be expressed as x[n], n = 0~N-1. Then the formula for DFT is:

$$X[k]=(1/N)\ \Sigma_{n=0}^{N-1}\ x[n]\ e^{-j2\pi nk/N},\ k=0,\ ...,\ N-1$$

The coefficient X[k] represents the amplitude and phase of the component of x[n], n = 0~N-1 at a specific frequency (to be detailed later). Therefore X[k], k = 0~N-1, are usually referred to as **spectrum**.

From X[k], we can also compute the original signal x[n], as follows:

$$x[n]=\Sigma_{k=0}^{N-1}\ X[k]e^{j2\pi nk/N},\ n=0,\ ...,\ N-1$$

Hint

DTFT and DFT are very similar; both are used to convert a discrete-time signal into its components at different frequencies. However, the former generates a continuous-frequency spectrum while the later produces a discrete-frequency spectrum.

Here we have several important facts about X[k]:

- If the length of the original signal x[n] is N, then the length of X[k] is also N.
- In general, X[k] is a complex number with a magnitude of |(X[k])| （abs(X[k]) in MATLAB） and a phase of ∠X[k] (angle(X[k]) or atan(imag(X[k])/real(X[k]) in MATLAB).
- If the time-domain signal x[n] is real, then X[k] and X[N-k] are complex conjugate staisfying |(X[k])| = |(X[N-k])| and ∠X[k] = -∠X[N-k].

If x[n] is real, we can express it as follows:

$$\begin{gathered} x[n] = X[0] \\ + X[1]e^{j2\pi n/N} + X[N-1]e^{j2\pi n(N-1)/N} \\ + X[2]e^{j2\pi n2/N} + X[N-2]e^{j2\pi n(N-2)/N} \\ + X[3]e^{j2\pi n3/N} + X[N-3]e^{j2\pi n(N-3)/N} \\ + \ldots \end{gathered}$$

For the terms involving k and N-k, we have

$$\begin{gathered} X[k]e^{j2\pi nk/N} + X[N-k]e^{j2\pi n(N-k)/N} \\ = X[k]e^{j2\pi nk/N} + X[N-k]e^{j2\pi n}e^{-j2\pi nk/N} \\ = X[k]e^{j2\pi nk/N} + X[N-k]e^{-j2\pi nk/N} \\ = 2\ \mathrm{Re}\{X[k]e^{j2\pi nk/N}\} \end{gathered}$$

If we use m_k and p_k to represent the magnitude and phase of X[k], respectively, then the above equation can be simplified as follows:

$$\begin{gathered} 2\ \mathrm{Re}\{m_k\exp(jp_k)\exp(j2\pi nk/N)\} \\ = 2\ \mathrm{Re}\{m_k\exp(j(2\pi nk/N + p_k)\} \\ = 2m_k\cos(2\pi nk/N + p_k) \end{gathered}$$

In general, N is even and we can express x[n] as

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

The above equation states that we can decompose x[n] into a DC component X[0] plus the summation of N/2 sinusoidal functions, where the amplitude and phase are determined by those of X[k], respectively. Therefore for a given real sequence x[n], we only need to have to observe X[k]，k = 0 ~ N/2 for spectrum analysis. This "one-side spectrum" consists of 1+N/2 sinusoidal functions, with frequencies located at 0, f_s/N, 2f_s/N, 3f_s/N, ..., N/2f_s/N (=f_s/2), with f_s as the sample rate. These sinusoidal functions are referred as the fundamental sinusoids.

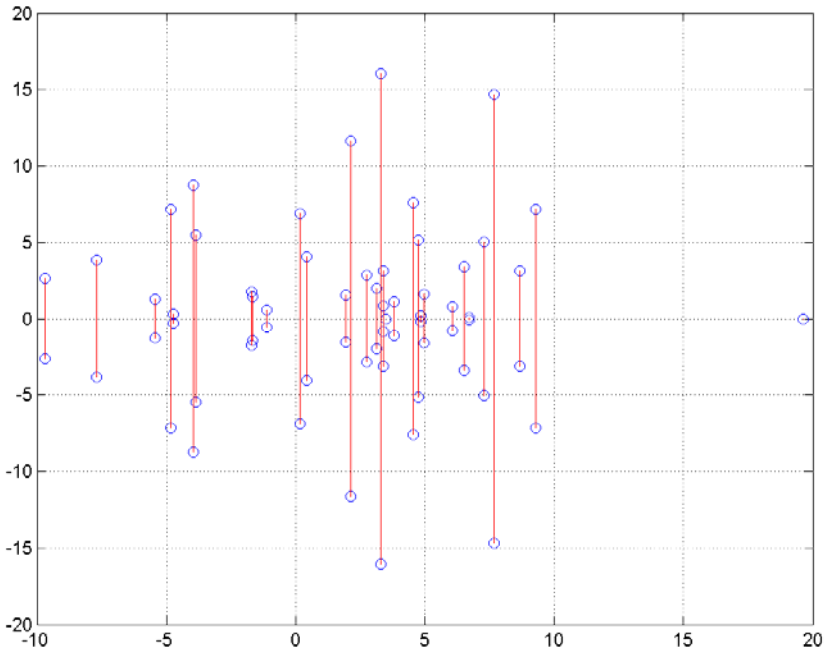
If we employ the above direct method for computing DFT, the time complexity if O(n^2). In 1965, a smart method based on divide-and-conquer was proposed to compute DFT with a time complexity of O(n log n). The method is called **fast Fourier transform** (**FFT** for short). In other words, FFT is an efficient method for computing DFT.

First of all, let us use the MATLAB command fft to verify the conjugate property of the DFT coefficients:

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



From the above plot, it can be seen that the DFT coefficients appear as complex conjugate pairs, which have the x-axis as the symmetric axis.

Hint

For real sequence x[n], n = 0~N-1:

- If N is even, then X[0] and X[N/2] are real and all the other terms are complex conjugate pairs.
- When N is odd, only X[0] is real and all the other terms are complex conjugate pairs.

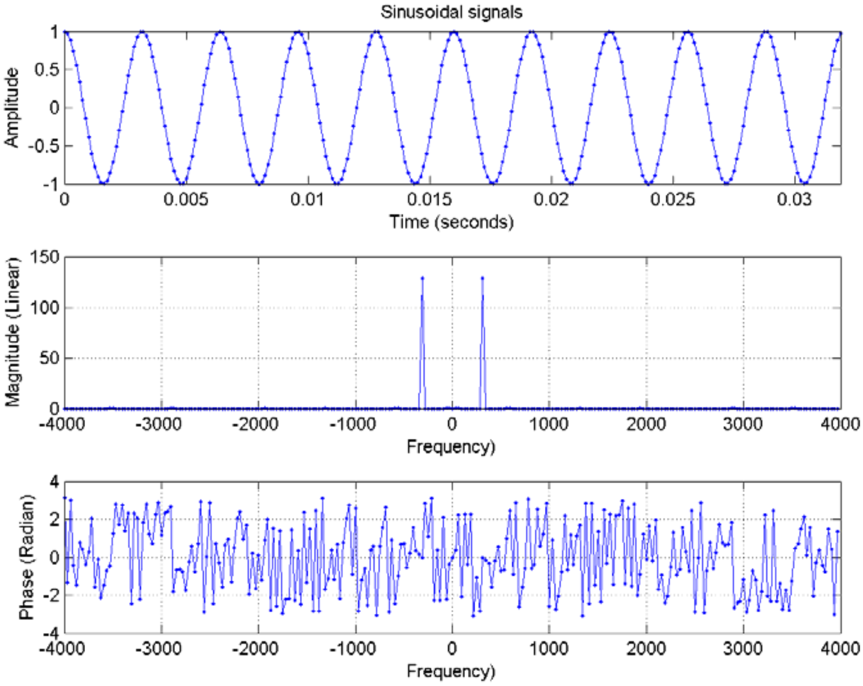
If x[n] happens to be one of the fundamental sinusoids, then the DFT coefficients should have only two non-zero terms, as shown in the following example:

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 (頻域的頻率的解析度)
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



We can observe that:

- In theory, the magnitude spectrum should have only two nonzero points. In practice, those zero points are not zero exactly, but very close to zero due to truncation or round-off errors.
- The phase spectrum appears to be random. This is simply because of the fact that most of the spectrum is zero, and hence the phase does not bear any significant meanings at all.

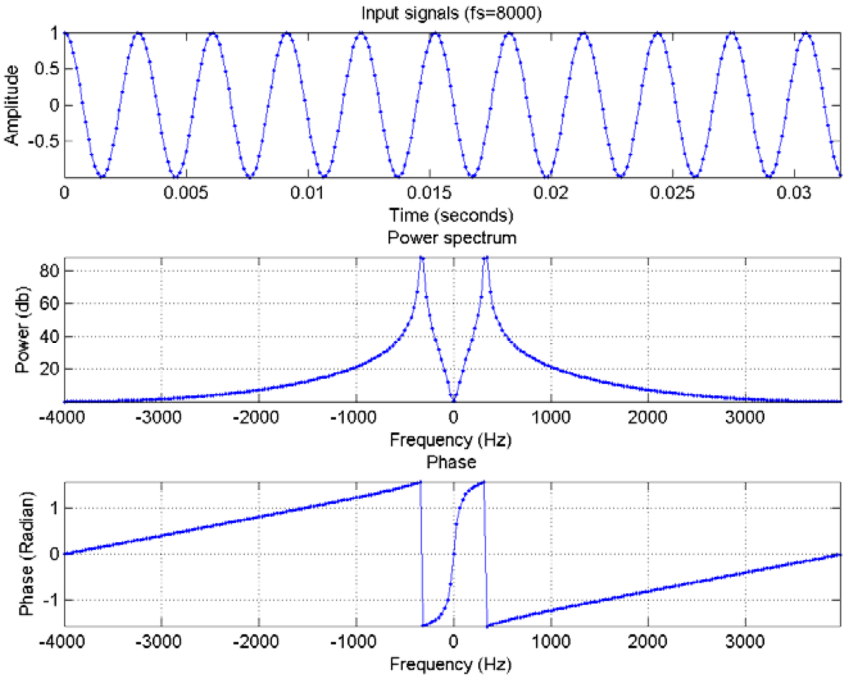
If x[n] is not one of the fundamental sinusoids, DFT will still decompose x[n] into the combination of fundamental sinusoids with spreaded magnitude spectrum:

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



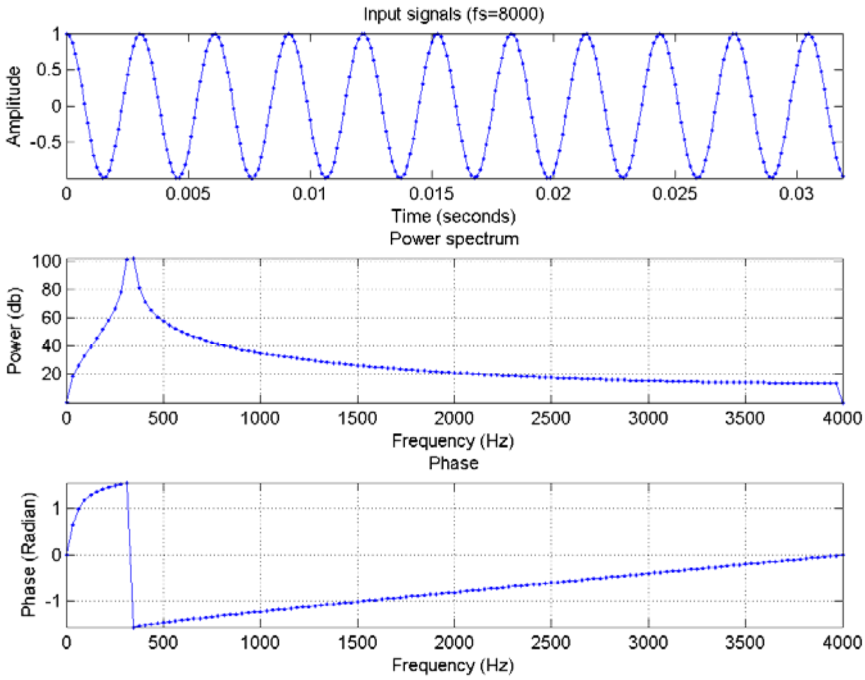
In the above example, we have used the function fftTwoSide.m (in the Audio Processing Toolbox) for the computation of two-side spectrum, and then plot the time-domain signal, magnitude spectrum and phase spectrum.

If x[n] is real, the magnitude spectrum is symmetric while the phase spectrum is anti-symmetric. Since we are dealing with audio signals which are all real, so we only need to have one-side spectrum for our analysis. In the following example, we shall use the function fftOneSide.m (in the Audio Processing Toolbox) to plot the time-domain signal and one-side magnitude/phase spectrum:

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



Hint

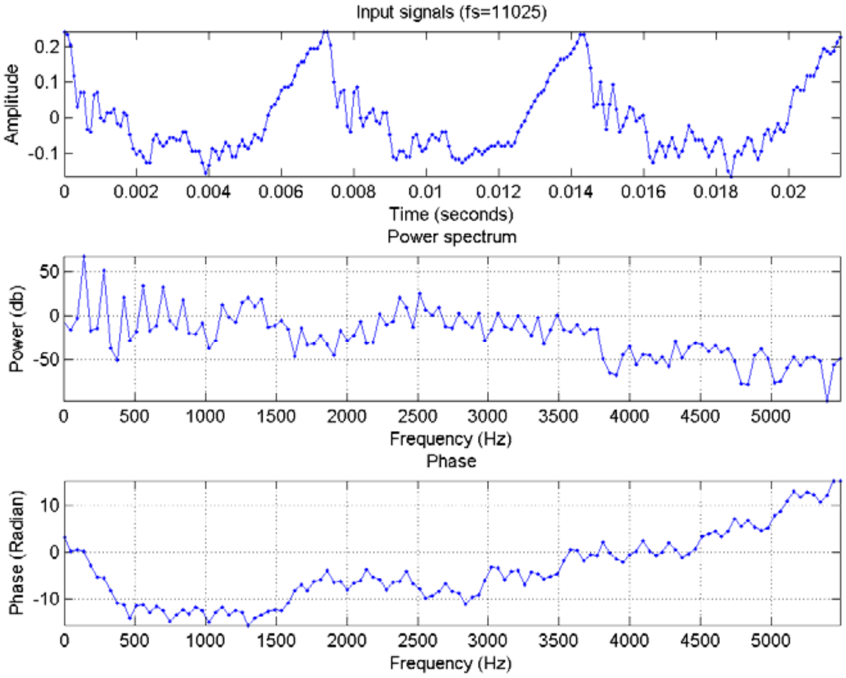
Signals from the real world are all real, so we only need to use fftOneSide.m to obtain one-side spectrum for spectrum analysis.

In the following example, we shall demonstrate the one-side spectrum for a frame of an audio signal:

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



In the above example, we can observe that there are **harmonics** in the magnitude spectrum. This is the result due to the fact that the time-domain signal is quasi-periodic. (To make the harmonics more obvious, we have carefully selected a time-domain signal containing three fundamental periods.)

When the value of k is getting bigger, we will have the high-frequency components which are more likely to be noises in the original time-domain signal. We can actually delete high-frequency components and use only low-frequency components to approximate and synthesize the original signal, to achieve the following goals:

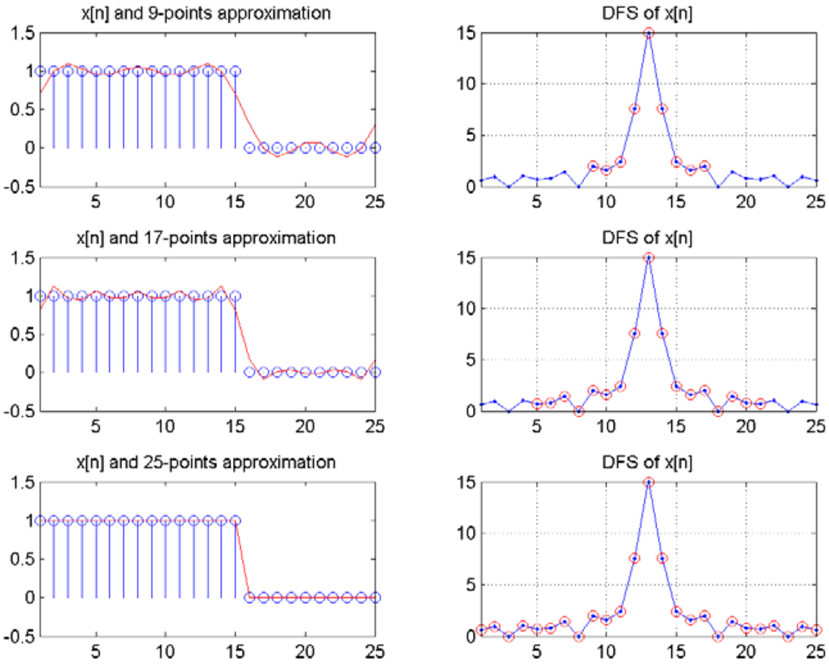
- Data compression
- Low-pass filter

In the next example, we shall demonstrate how to synthesize a square wave using sinusolids:

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



It is obvious that when we use more DFT coefficients for the synthesis, the reconstructed waveform is closer to the original one.

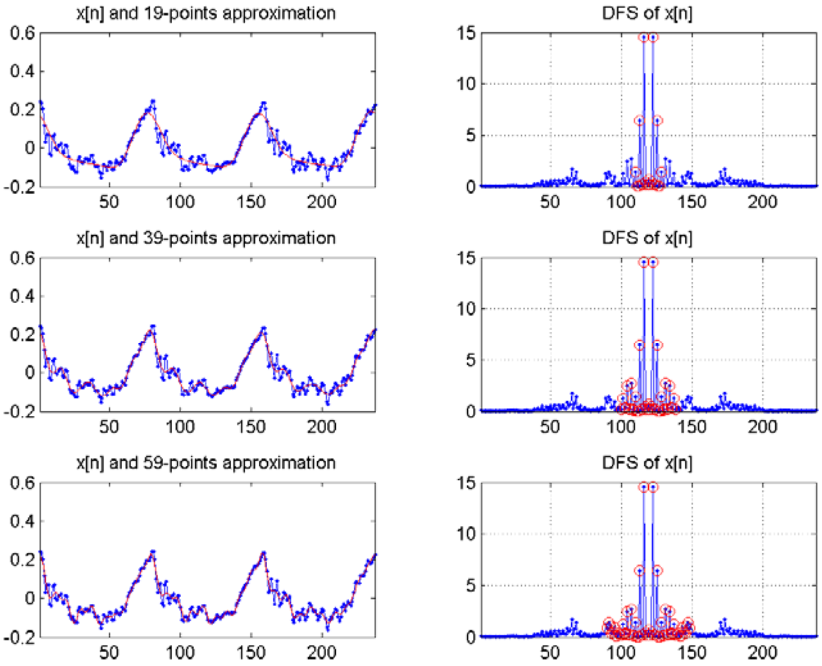
The following example uses sinusoids to approximate the waveform of an utterance:

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



From the above example, it is known that we only need a few DFT coefficents to synthesize the original signal with satisfactory quality. This fact demonstrate that the high-frequency components are not important in reconstructing the original signal.

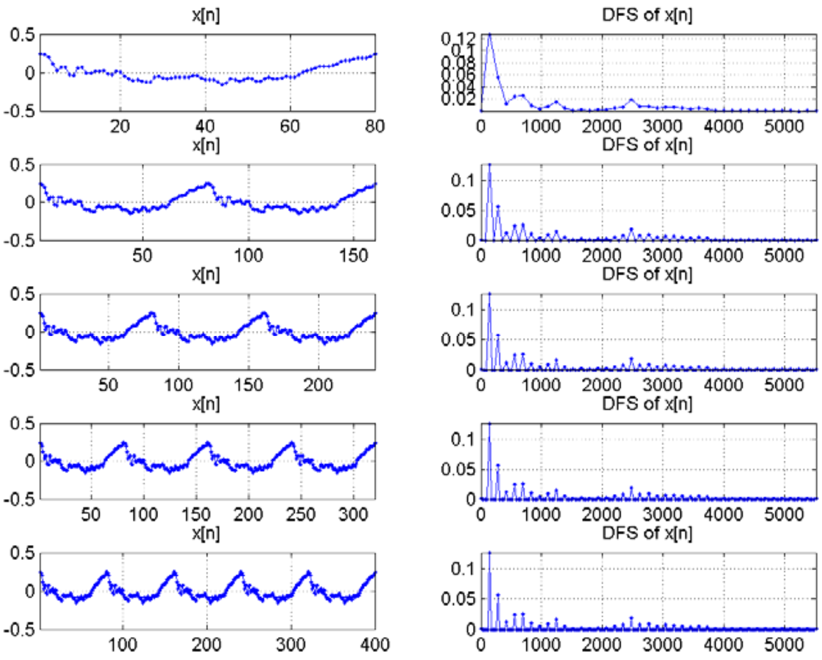
For perfectly periodic signals, DFT will insert zeros inbetween, as follows:

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



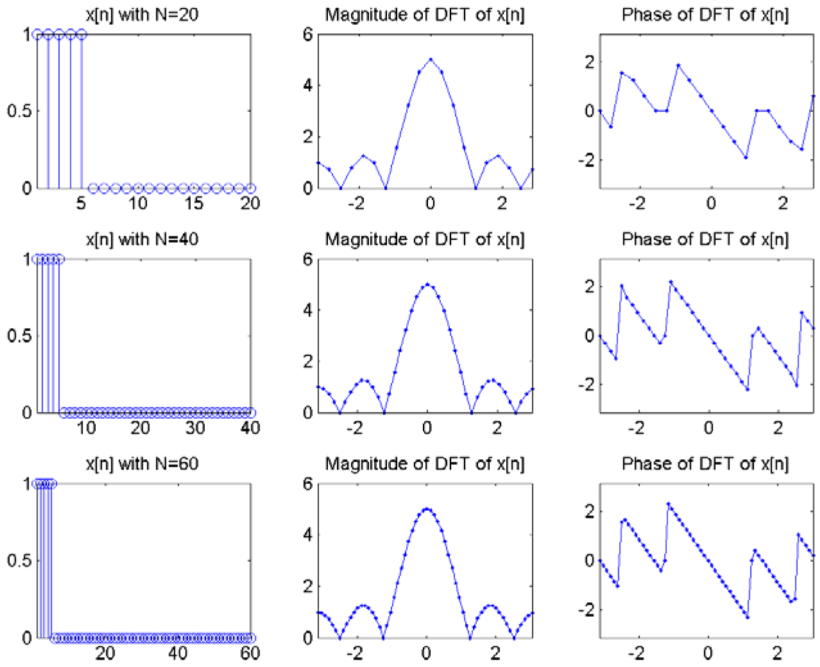
Why is it so? Without rigorous analysis, can you explain this phonomenon by intuition?

If we pad the signal x[n] with zeros, the corresponding effect in frequency domain is interpolation, as shown in the next example:

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



In other words, zero padding does not add new information to x[n]. However, DFT need to have the same length and the net result is the interpolation to have a dense sampling in the frequency domain.

Hint

For common audio signal processing, if the frame size is not 2^n, then we need to use zero padding until the frame size is 2^n. This is also easier for DFT since a fast method based on 2-radix FFT can be invoked directly.

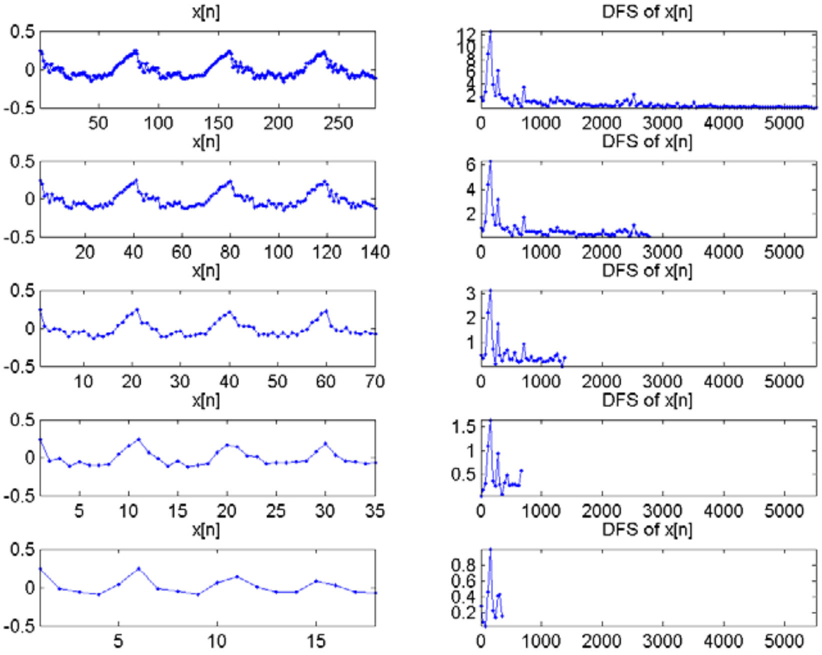
If we down sample the signal in the time domain, then the result in the spectrum is shown next:

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



The more times we do down sampling, the smoother the time-domain signal will be. Therefore the high-frequency components in the spectrum are missing gradually.

Moreover, we can express the time-domain signal as a linear combination of the fundamental sinusoids and then apply the least-square estimate to find the best coefficients of these sinusoids. It turns out that the coefficients identified by the least-square method are the same as those by fft, as shown next:

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
```

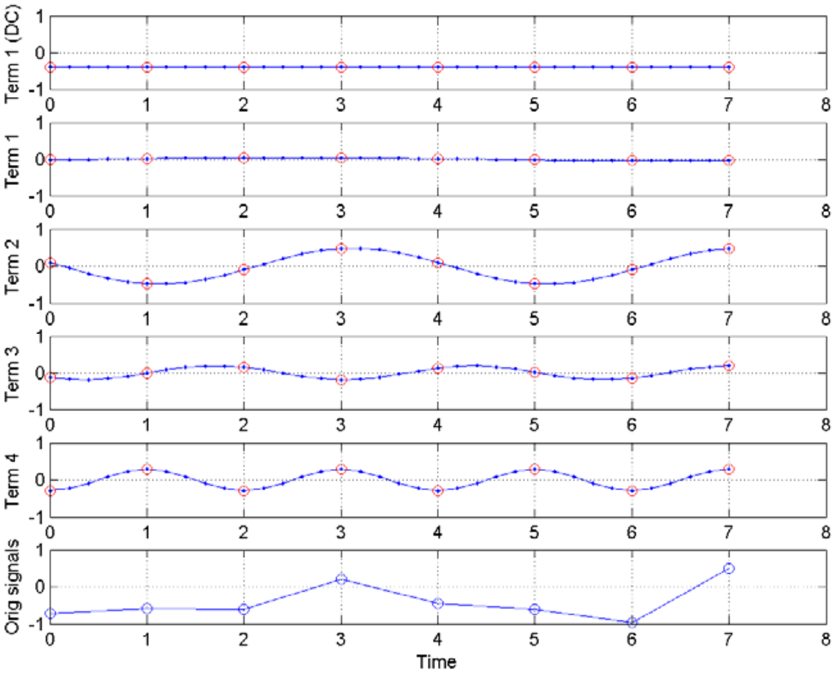
```
th=A\x;

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

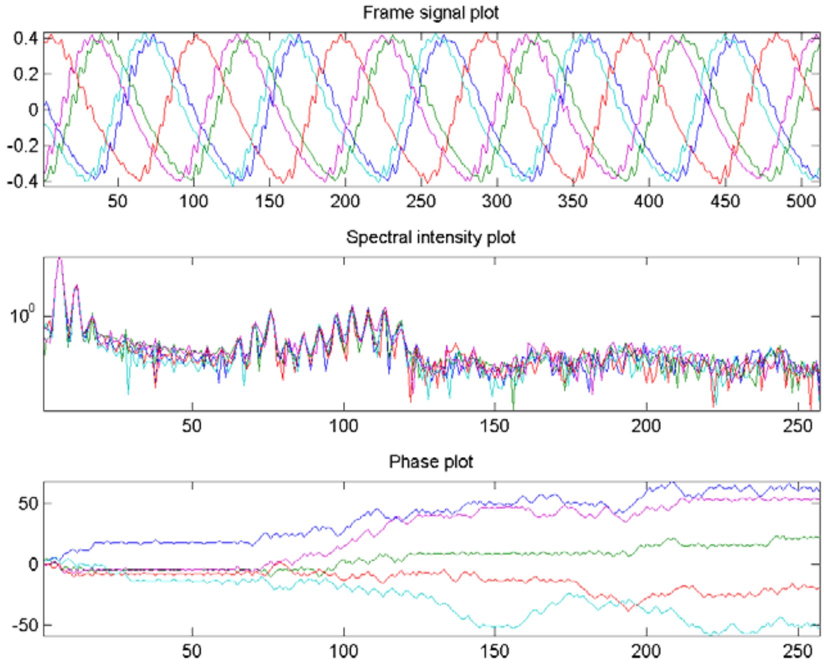


第 10 章作業

Old Chinese version

1. (**) **Spectrum of vowel of constant pitch：** Write an m-file script to accomplish the following tasks:
   a. Record your utterance of the English letter "e" for 3 seconds, with 16KHz/16Bits/Mono. (Please try to maintain a stable pitch and volume.)
   b. Use buffer2.m and frame2volume.m to do frame blocking and volume computation, respectively, with frameSize=32ms, overlap=0ms. Please identify the frame with the maximum volume, and the preceding/succeeding 2 frames, to have 5 frames in total.
   c. Use fft to compute the spectrum of these 5 frames and have the following plots:
      1. subplot(3,1,1): time-domain signals of these 5 frames.
      2. subplot(3,1,2): one-side magnitude spectra of these 5 frames.
      3. subplot(3,1,3): one-side phase spectra of these 5 frames.

   You plots should be similar to the following:



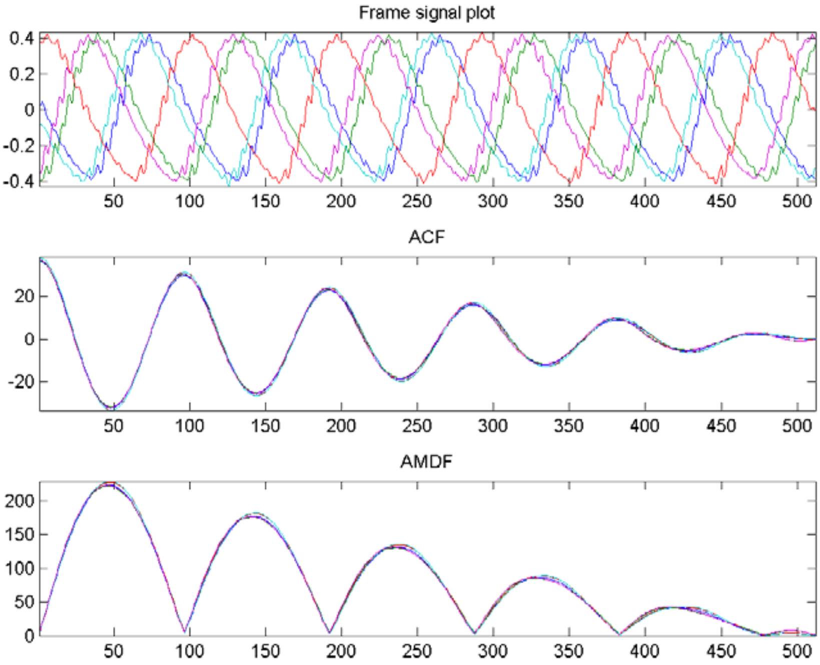
Hints:

- Before performing FFT on each frame, you should multiply the frame with the Hamming window to make the harmonics more obvious. The command is "frame=frame.*hamming(length(frame))".
- When you plot the magnitude spectrum, you can use log scale on y-axis. To achieve this, you can simply issue "set(gca, 'yscale', 'log')" after the second plot.
- In order to make the phase a continuous curve, you can use the function unwrap.

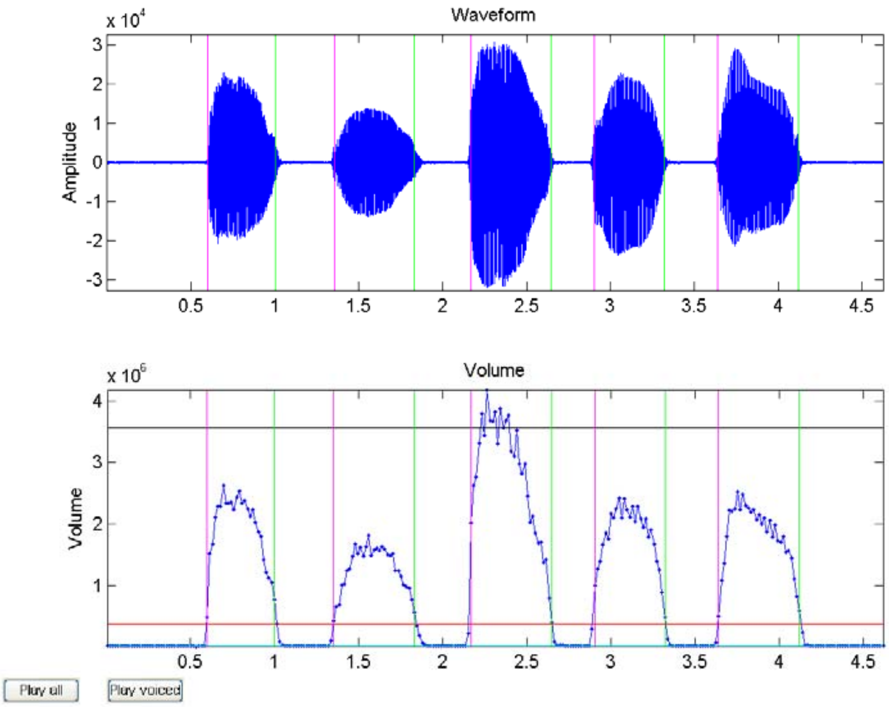
d. Observe your plots and find which plots are more consistent within 5 frames? (Hint: Since our pronunciation of these 5 frames are the same, the features which are more invariant can be used for speech recognition.)

2. (**) **Spectrum of vowel of varying pitch：** Write an m-file script to repeat the previous exercise, but change the recording to Mandarin of「一ˊ」. This exercise can be used to observe the variation of spectrum under varying pitch.

3. (**) **ACF/AMDF of vowel of constant pitch：** Repeat the first exercise using ACF & AMDF as the features. You plots should be similar to the following:

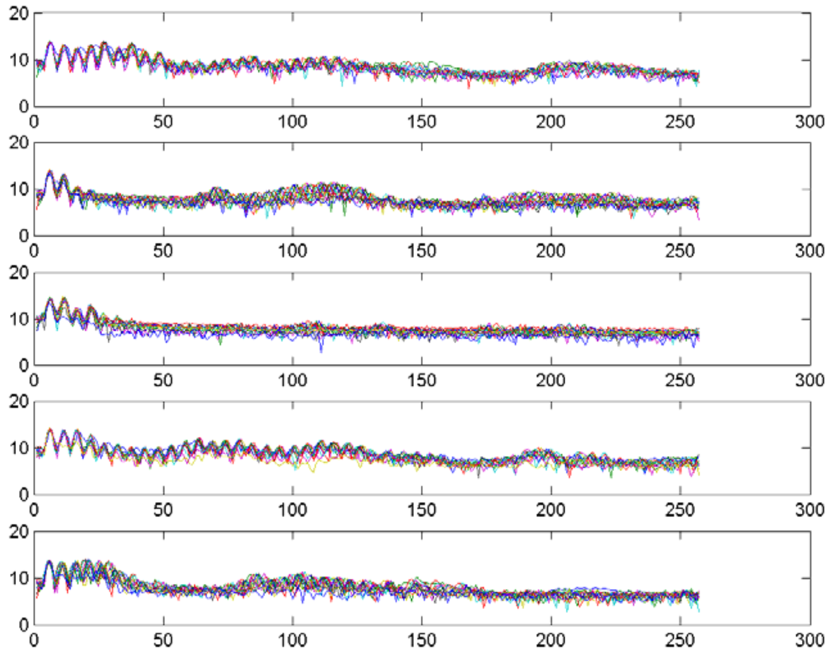


4. (***) **Use spectrum for classifying vowels**：Write an m-file script to do the following tasks:
   a. Record a 5-second clips of the Chinese vowel 「ㄚ、ㄧ、ㄨ、ㄝ、ㄛ」 (or the English vowels "a, e, i, o, u") with 16KHz/16Bits/Mono. (Please try to maintain a stable pitch and volume, and keep a short pause between 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:

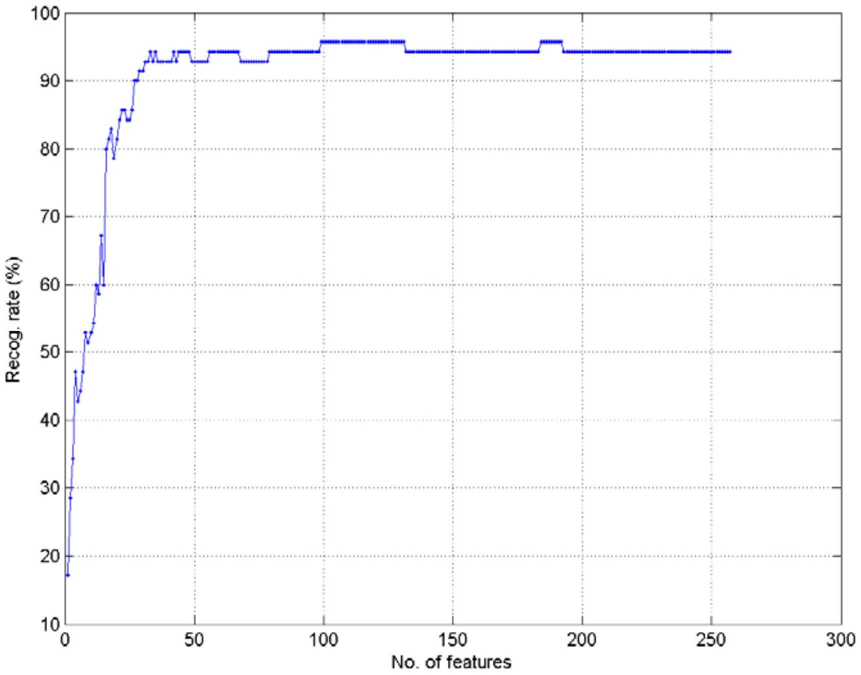


If the segmentation is not correct, you should adjust the parameters to `epdByVol.m` until you get the correct segmentation.

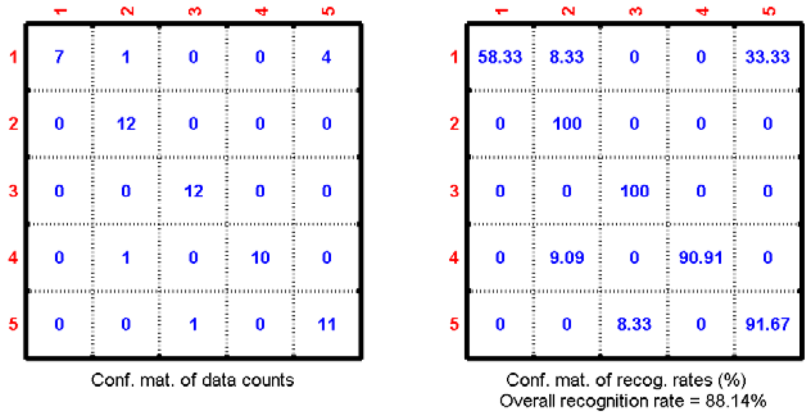
c. Use buffer2.m to do frame blocking on these 5 vowels, with frameSize=32ms and overlap=0ms. Please plot 5 plots of one-sided magnitude spectra (use `fftOneSide.m`) corresponding to each vowel. Each plot should contains as many curves of magnitude spectra as the number of frames in this vowel. Your 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 the one-side magnitude spectrum to classify each frame into 5 classes of different vowels. In particular, we need to change the dimension of the feature from 1 to 257 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:



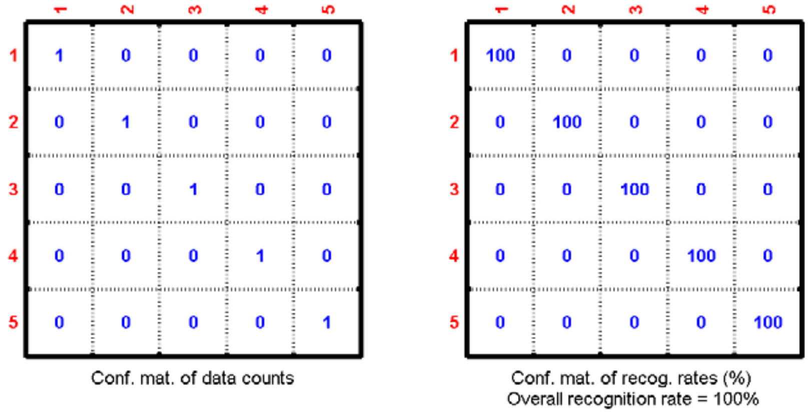
	1	2	3	4	5
1	7	1	0	0	4
2	0	12	0	0	0
3	0	0	12	0	0
4	0	1	0	10	0
5	0	0	1	0	11

Conf. mat. of data counts

	1	2	3	4	5
1	58.33	8.33	0	0	33.33
2	0	100	0	0	0
3	0	0	100	0	0
4	0	9.09	0	90.91	0
5	0	0	8.33	0	91.67

Conf. mat. of recog. rates (%)
Overall recognition rate = 88.14%

f. What is the vowel-based recognition rate? Plot the confusion matrix, which should be similar to the following figure:



	1	2	3	4	5
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4	0	0	0	1	0
5	0	0	0	0	1

Conf. mat. of data counts

	1	2	3	4	5
1	100	0	0	0	0
2	0	100	0	0	0
3	0	0	100	0	0
4	0	0	0	100	0
5	0	0	0	0	100

Conf. mat. of recog. rates (%)
Overall recognition rate = 100%

(Hint: The vowel-based recognition result is the voting of frame-based results. You can use mode to compute the result of voting.)

5. (***) **Use ACF for classifying vowels**: Repeat the previous exercise by using ACF as the acoustic features. Does it perform better than magnitude spectrum? (Hint: Some sort of ACF normalization will make the recognition better.)
6. (***) **Use AMDF for classifying vowels**: Repeat the previous exercise by using AMDF as the acoustic features. Does it perform better than magnitude spectrum? (Hint: Some sort of AMDF normalization will make the recognition better.)

Chapter 11: Digital Filters

11-1 Filter Applications (濾波器應用)

Old Chinese version

In this section, we shall introduce the concept of **digital filters** and their applications. Most students who have taken DSP (digital signal processing) are more or less intimidated by the complexity of the underlying mathematics of digital filters. However, the focus of this section is on the applications of digital filters using MATLAB and the treatment should be pretty accessible for most engineering-majored students.

To put it simply, a digital filter can be represented by two parameter vectors **a** and **b**, where the lenghts of **a** and **b** are p and q, respectively, and the first element of **a** is always 1, as follows:

$$\mathbf{a} = [1, a_2, \ldots a_p]$$

$$\mathbf{b} = [b_1, b_2, \ldots b_q]$$

If we apply a digital filter with parameters **a** and **b** to a stream of discrete-time signal x[n], the output y[n] should satisfy the following equation:

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

Or equivalently, we can express y[n] explicitly:

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

The preceding equation seems a little bit complicated. We shall give some more specific examples to make it more easily understood.

First of all, if we have a filter with the parameters:

$$\mathbf{a} = [1]$$

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

Then the output of the filter is

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

This is a simple digital filter that sets y[n] as the average of the preceding five points of the input signals. In fact, this is a a so-called **low-pass filter** since after the averaging operator, the high-frequency component is averaged out while the low-frequency component is more or less retained. The effect of a low-pass filter is like putting a paper cup on the mouth while speaking, generating a murmuring-like indistinct sound.

Hint

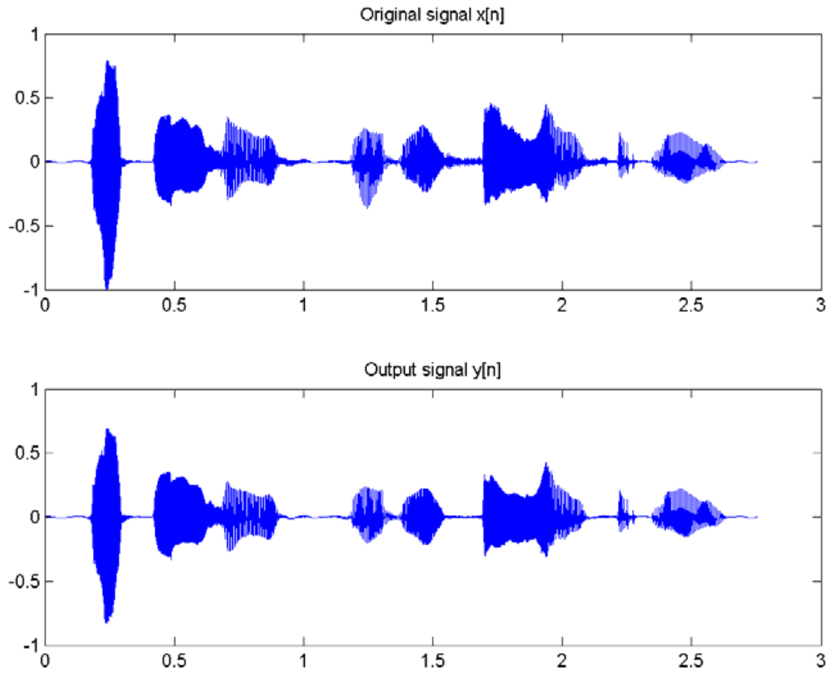
Why putting a paper cup on the mouth makes it a low-pass filter? Can you think of an intuitive reason?

The next example uses a low-pass filter on a speech input.

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



In the above example, "y = filter(b, a, x)" is the MATLAB command to generate the output signal y from the original signal x and the filter's parameters a and b. From the waveform plots, we can also observe that the high-frequency components (especially the unvoiced sounds, such as "s" in "movies" and "seen", and "c" in "recently") have a smaller amplitude after filtering. This is a typical effect of low-pass filters. We can also hear the original and the output signals:

- Original signal x[n]： example/filter/whatMovies.wav
- Output signal y[n]： example/filter/lpf01.wav

Let us take a look at another filter with the parameters:

$$\mathbf{a} = [1]$$

$$\mathbf{b} = [1, -1]$$

The output of the fitler is

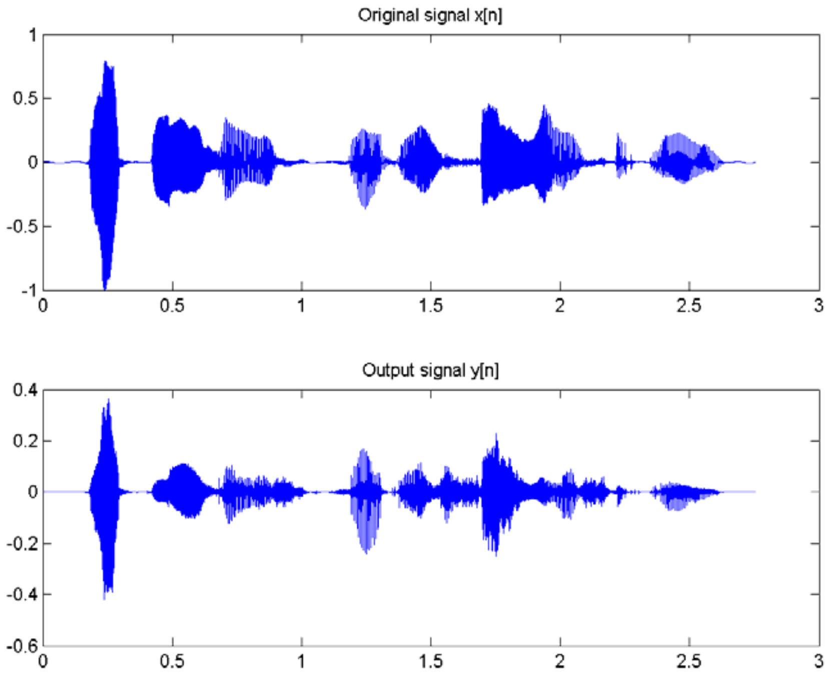
$$y[n] = x[n] - x[n-1]$$

In other words, the output of the filter y[n] is equal to the difference of the preceding two points of x[n]. As the result, low-frequency components (with slow variations) will have low values while high-frequency components (with fast variations) will have high values. So this is a typical **high-pass filter** which amplifies high-frequency and suppresses low-frequency components. See the following example.

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



From the waveform plots, we can observe that the amplitudes of the unvoiced sounds (high-frequency components) are relatively larger that of the voiced sounds (low-frequency components). We can also hear the sound clips directly:

- Original signal x[n]： example/filter/whatMovies.wav
- Output signal y[n]： example/filter/hpf01.wav

After the high-pass filter, the output signal is like the output from a small radio set with noise-like creaky sounds.

Besides low-pass and high-pass, digital filters can also produce some other familiar sound effects. For instance, if a filter has the following parameters:

a = [1]

b = [1, 0, 0, 0, ..., 0, 0.8] (That is, 3199 zeros between 1 and 0.8.)

Then the output of the filter is:

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

In other words, this filter can produce "one-fold echo" for the given input signal. If the sample frequency fs is 16kHz, then the time difference between the input and the echo is 3200/fs = 3200/16000 = 0.2 second. Please see the following example.

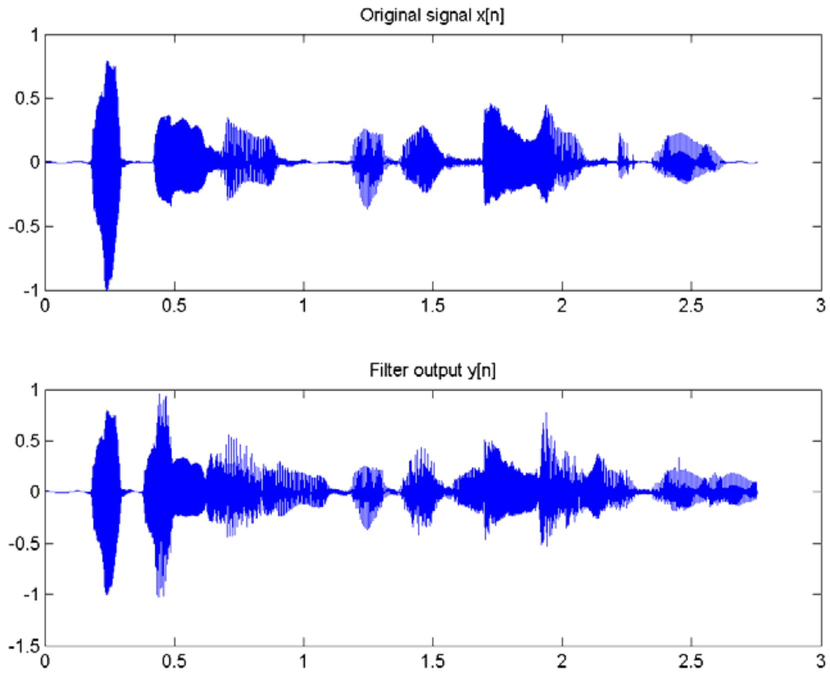
Example 3**Input file** filter/echo01.m

```
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



In the above example, gain = 0.8 is the decay ratio of the echo, and delay = 0.2 is the delay time of the echo. We can hear the sound clips:

- Original signal x[n]： example/filter/whatMovies.wav
- Output signal y[n]： example/filter/echo01.wav

The above filter can only produce one-fold echo. If we want to have a more realistic multiple-fold echo, we can use the following parameters:

a = [1, 0, 0, 0, ..., 0, -0.8] (That is, 3199 zeros between 1 and -0.8.)

b = [1]

The output of the filter is

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

The filter can produce a more realistic multiple-pass echo. If the sample frequency fs = 16kHz, then the time delay between echos is 3200/fs = 3200/16000 = 0.2 秒. See the following example.

Example 4**Input file** filter/echo02.m

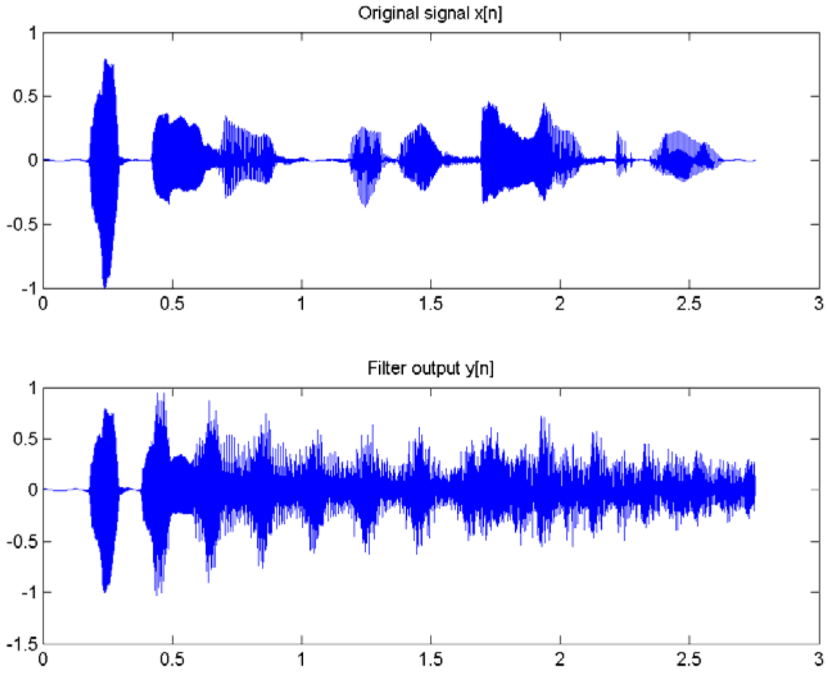
```
waveFile='whatMovies.wav';
[x, fs, nbits]=wavread(waveFile);
```

```
% 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



We can hear the sound clips:

- Original signal x[n]：example/filter/whatMovies.wav
- Output signal y[n]：example/filter/echo02.wav

11-2 Filter Design (濾波器設計)

Old Chinese version

In the previous section, we have seen some basic filters and their applications. In this section, we shall cover some basic approaches to filter design using MATLAB.

We can use the MATLAB command "butter" to design a Butterworth low-pass filter, with the following format:

[b, a] = butter(order, wn, function)

The input arguments to butter can be explained next:

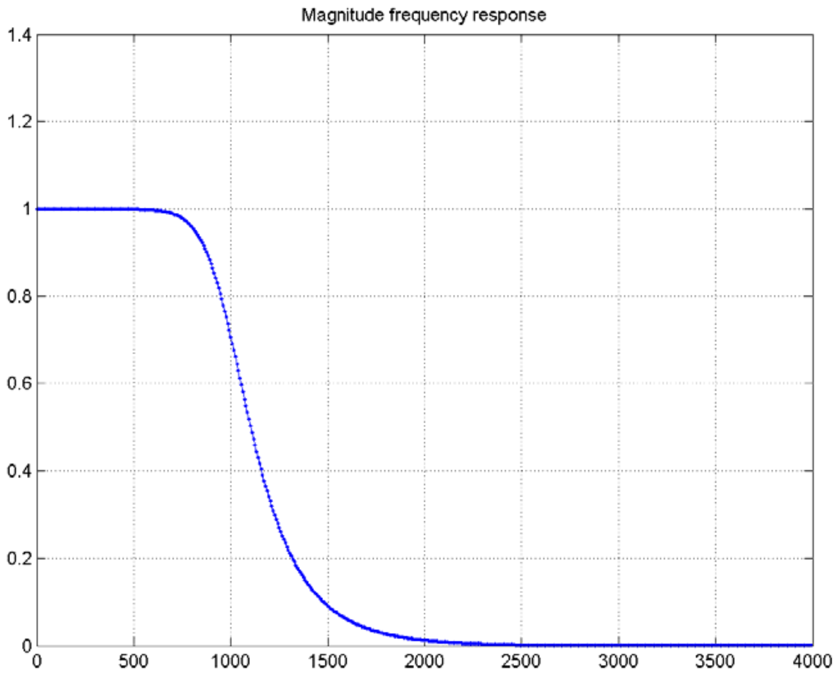
- order: order of the filte. A larger order leads to a better filtering effect. However, the required computing time is also proportional to the order since the the length of the parameters a and b is equal to order+1。
- wn: normalized cutoff frequency within the range of 0 and 1. When the sample frequency is fs, the maximum allowable frequency in frequency-domain processing is fs/2. Therefore the normalized cutoff frequency wn is equal to the real cutoff frequency divided by fs/2.
- function: a string representing the filter function, which could be 'low' or 'high', representing low-pass and high-pass, respectively.

In the next example, we use the command "butter" to design a Butterworth low-pass filter.

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



In the above example, we have designed a Butterworth filter with a cutoff frequency of 1000 Hz. The plot is the magnitude frequency response of the filter.

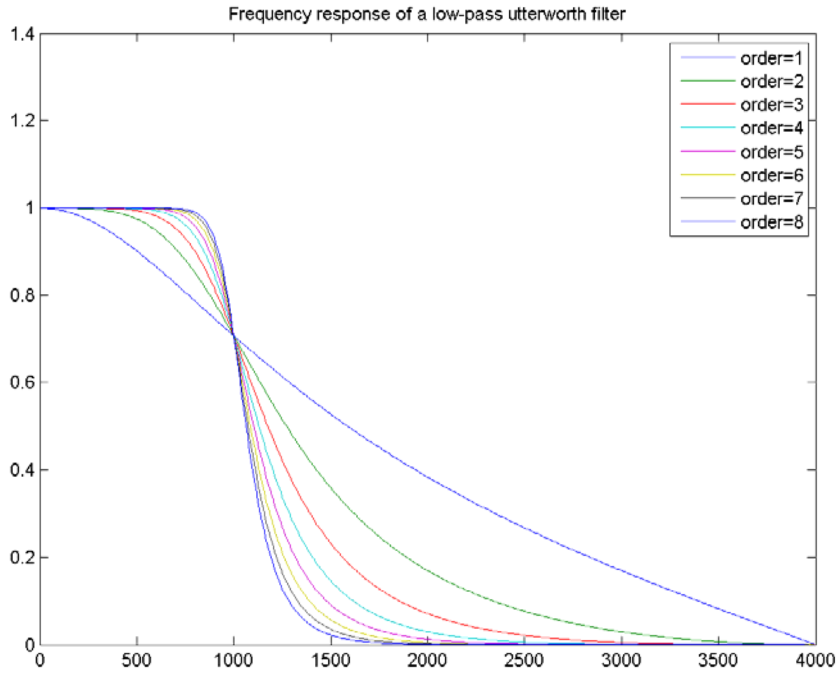
When the order of the filter is bigger, the filtering effect is better at the cost of longer computation time. On the other hand, a smaller order leads to a shorter computation time and less desirable filtering effect. The following example demonstrates the magnitude frequency response as a function of the order of the Butterworth filter.

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



As it is obvious in the above example, when the order is increased from 1 to 8, the magnitude frequency response is becoming sharper at the cutoff frequency of 1000 Hz.

We can apply the above filter to a clip of a pop song to see if the high-frequency components can be removed. See the next example.

Example 3**Input file** filter/butter03.m

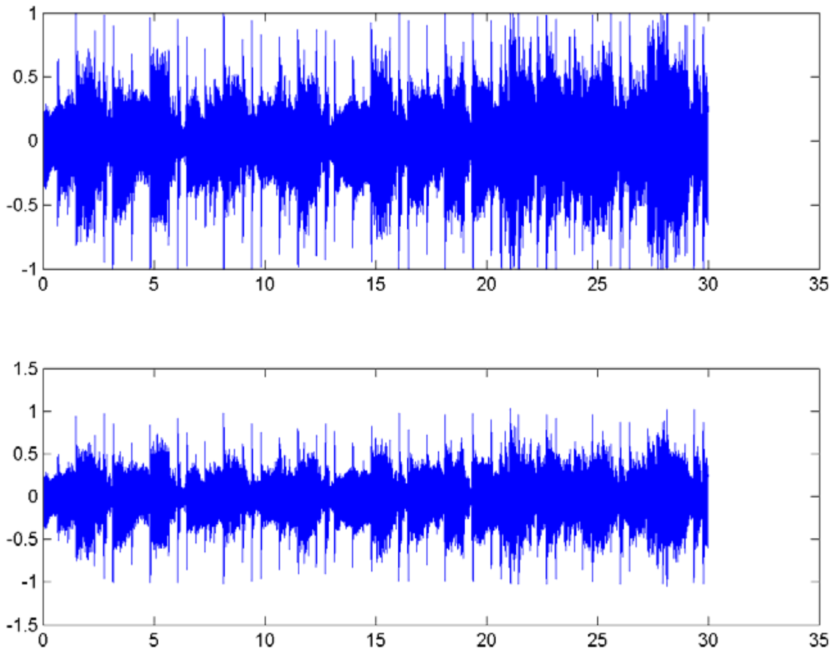
```
cutOffFreq=1000;      % Cutoff frequency
filterOrder=5;                        % Order of filter
[x, fs, nbits]=wavRead('wubai_solicitude.wav');
[b, a]=butter(filterOrder, cutOffFreq/(fs/2), 'low');
x=x(60*fs:90*fs);      % 30-second signal
y=filter(b, a, x);
```

```
% ====== 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



We can hear the original and the output clips:

- Original signal x[n]：example/filter/wubai_solicitude_orig.wav
- Output signal y[n]：example/filter/wubai_solicitude_1000.wav

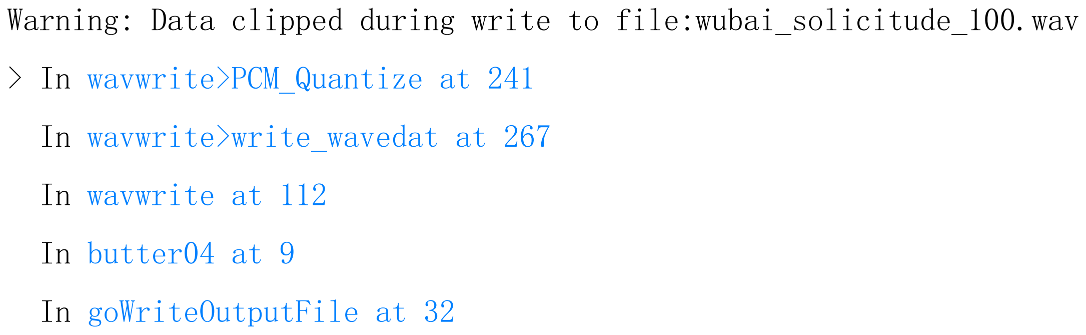
The playback of the output signal demonstrates that the high-frequency components are eliminated from the original signal.

If we set up the cutoff frequency to 100 Hz, then the output signal is almost inaudible unless we have a speaker with a subwoofer. See the next example.

Example 4**Input file** filter/butter04.m

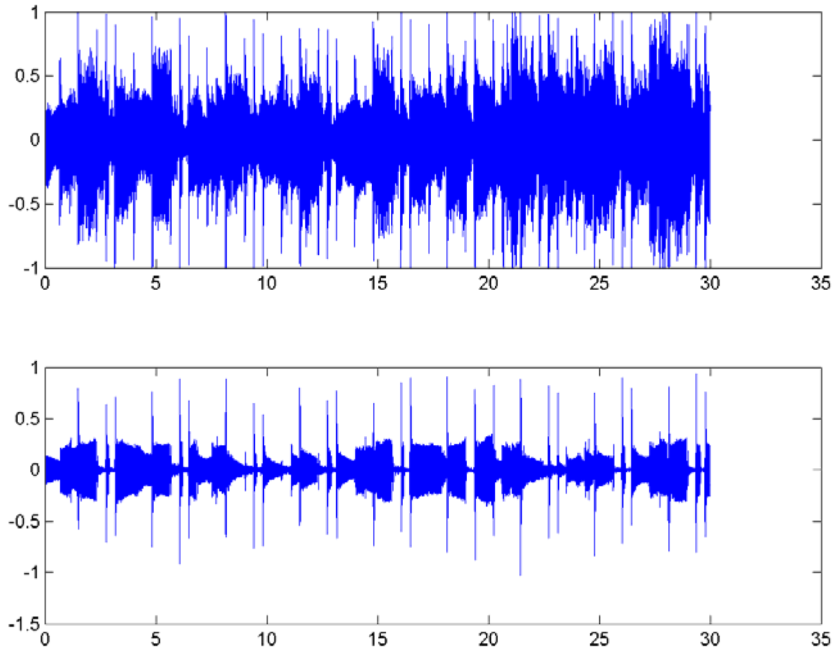
```
cutOffFreq=100;                    % Cutoff freq (截止頻率)
filterOrder=5;                     % Order of filter (濾波器的階數)
[x, fs, nbits]=wavRead('wubai_solicitude.wav');
[b, a]=butter(filterOrder, cutOffFreq/(fs/2), 'low');
x=x(60*fs:90*fs);       % 30 seconds of singing (30 秒歌聲)
y=filter(b, a, x);
% ====== 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



- Original signal x[n]：example/filter/wubai_solicitude_orig.wav
- Output signal y[n]：example/filter/wubai_solicitude_100.wav

Obviously, after the low-pass filter at a cutoff frequency of 100 Hz, most of the sounds are removed except for those from the bass drum.

Hint

In fact, the periodical sounds of the bass drum can help us tracking the beat of the music. This problem of beat tracking is an active research topic in the literature of music information retrieval.

If you cannot identify the sounds of the bass drum, you can hear the playback of the following clips one by one. (If you use CoolEdit for playback, the progress bar can help you identify where the sounds of the bass drum are.)

- Original signal：example/filter/wubai_solicitude_orig.wav
- Cutoff frequency = 1000 Hz：example/filter/wubai_solicitude_1000.wav
- Cutoff frequency = 500 Hz：example/filter/wubai_solicitude_500.wav
- Cutoff frequency = 400 Hz：example/filter/wubai_solicitude_400.wav
- Cutoff frequency = 300 Hz：example/filter/wubai_solicitude_300.wav

- Cutoff frequency = 200 Hz：example/filter/wubai_solicitude_200.wav
- Cutoff frequency = 100 Hz：example/filter/wubai_solicitude_100.wav

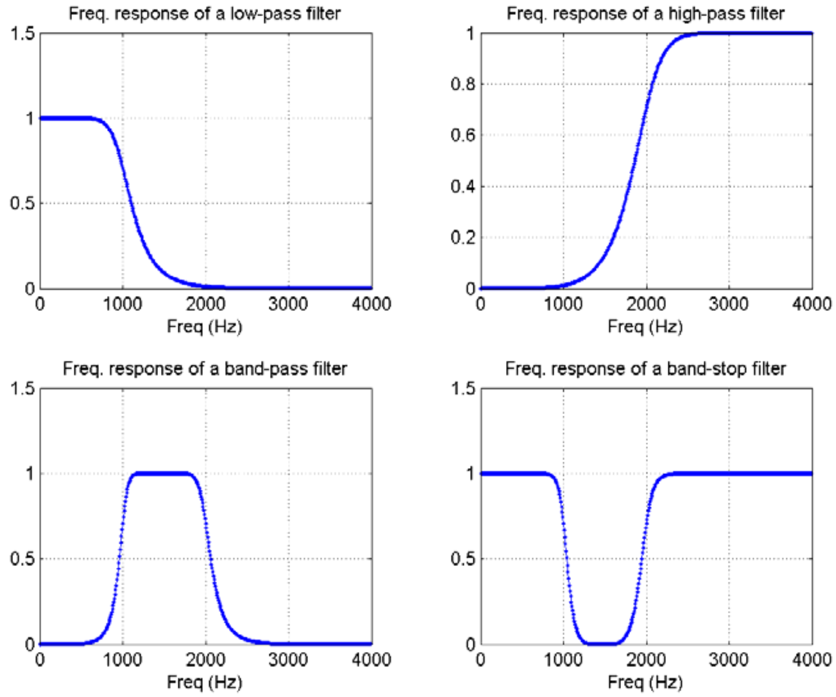
In fact, by supplying appropriates input parameters, we can use the command "butter" to design four types of filters, including low-pass, high-pass, band-pass, band-stop filters. The following example plots typical frequency responses of these filters.

Example 5**Input file** filter/butter05.m

```
fs=8000;                % Sampling rate
filterOrder=5;                  % Order of filter

% ====== low-pass filter
cutOffFreq=1000;
[b, a]=butter(filterOrder, cutOffFreq/(fs/2), 'low');
[h, w]=freqz(b, a);
subplot(2,2,1);
plot(w/pi*fs/2, abs(h), '.-');
xlabel('Freq (Hz)'); title('Freq. response of a low-pass filter'); grid on

% ====== high-pass filter
cutOffFreq=2000;
[b, a]=butter(filterOrder, cutOffFreq/(fs/2), 'high');
[h, w]=freqz(b, a);
subplot(2,2,2);
plot(w/pi*fs/2, abs(h), '.-');
xlabel('Freq (Hz)'); title('Freq. response of a high-pass filter'); grid on

% ====== band-pass filter
passBand=[1000, 2000];
[b, a]=butter(filterOrder, passBand/(fs/2));
[h, w]=freqz(b, a);
subplot(2,2,3);
plot(w/pi*fs/2, abs(h), '.-');
xlabel('Freq (Hz)'); title('Freq. response of a band-pass filter'); grid on

% ====== band-stop filter
stopBand=[1000, 2000];
```

```
[b, a]=butter(filterOrder, stopBand/(fs/2), 'stop');
[h, w]=freqz(b, a);
subplot(2,2,4);
plot(w/pi*fs/2, abs(h), '.-');
xlabel('Freq (Hz)'); title('Freq. response of a band-stop filter'); grid on
```

Output figure



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!

12-2 MFCC

Old Chinese version

For speech/speaker recognition, the most commonly used acoustic features are **mel-scale frequency cepstral coefficient** (**MFCC** for short). MFCC takes human perception sensitivity with respect to frequencies into consideration, and therefore are best for speech/speaker recognition. We shall explain the stey-by-step computation of MFCC in this section.

1. **Pre-emphasis**: The speech signal s(n) is sent to a high-pass filter:

$$s_2(n) = s(n) - a*s(n-1)$$

where $s_2(n)$ is the output signal and the value of a is usually between 0.9 and 1.0. The z-transform of the filter is

$$H(z)=1-a*z^{-1}$$

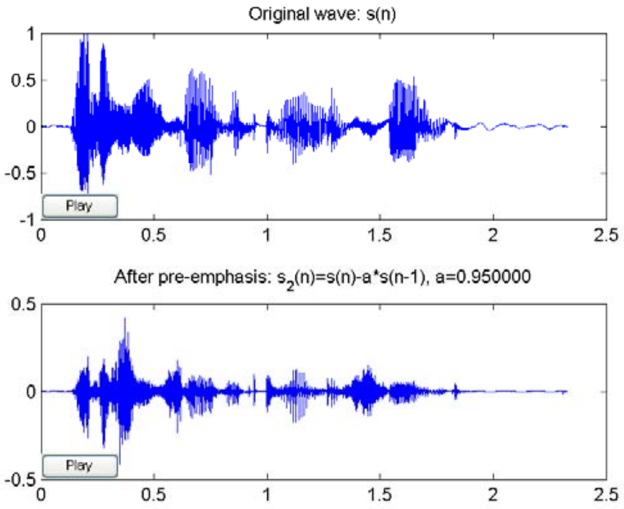
The goal of pre-emphasis is to compensate the high-frequency part that was suppressed during the sound production mechanism of humans. Moreover, it can also amplify the importance of high-frequency formants. The next example demonstrates the effect of pre-emphasis.

Example 1**Input file** speechFeature/preEmphasis01.m

```
waveFile='whatFood.wav';
[y, fs, nbits]=wavread(waveFile);
a=0.95;
y2 = filter([1, -a], 1, y);
time=(1:length(y))/fs;
wavwrite(y2, fs, nbits, 'whatFood_preEmphasis.wav');

subplot(2,1,1);
plot(time, y);
title('Original wave: s(n)');
subplot(2,1,2);
plot(time, y2);
title(sprintf('After pre-emphasis: s_2(n)=s(n)-a*s(n-1), a=%f', a));

subplot(2,1,1);
```

```
set(gca, 'unit', 'pixel');
axisPos=get(gca, 'position');
uicontrol('string', 'Play', 'position', [axisPos(1:2), 60, 20], 'callback', 'sound(y, fs)');
subplot(2,1,2);
set(gca, 'unit', 'pixel');
axisPos=get(gca, 'position');
uicontrol('string', 'Play', 'position', [axisPos(1:2), 60, 20], 'callback', 'sound(y2, fs)');
```

Output figure



In the above example, the speech after pre-emphasis sounds sharper with a smaller volume:

- Original: whatFood.wav
- After pre-emphasis: whatFood_preEmphasis.wav

2. **Frame blocking**: The input speech signal is segmented into frames of 20~30 ms with optional overlap of 1/3~1/2 of the frame size. Usually the frame size (in terms of sample points) is equal power of two in order to facilitate the use of FFT. If this is not the case, we need to do zero padding to the nearest length of power of two. If the sample rate is 16 kHz and the frame size is 320 sample points, then the frame duration is 320/16000 = 0.02 sec = 20 ms. Additional, if the overlap is 160 points, then the frame rate is 16000/(320-160) = 100 frames per second.
3. **Hamming windowing**: Each frame has to be multiplied with a hamming window in order to keep the continuity of the first and the last points in the frame (to be detailed in the next step). If the signal in a frame is denoted by s(n), n = 0,…N-1, then the signal after Hamming windowing is s(n)*w(n), where w(n) is the Hamming window defined by:

$$w(n, \alpha) = (1 - \alpha) - \alpha \cos(2\pi n/(N-1)),\ 0 \leqq n \leqq N-1$$

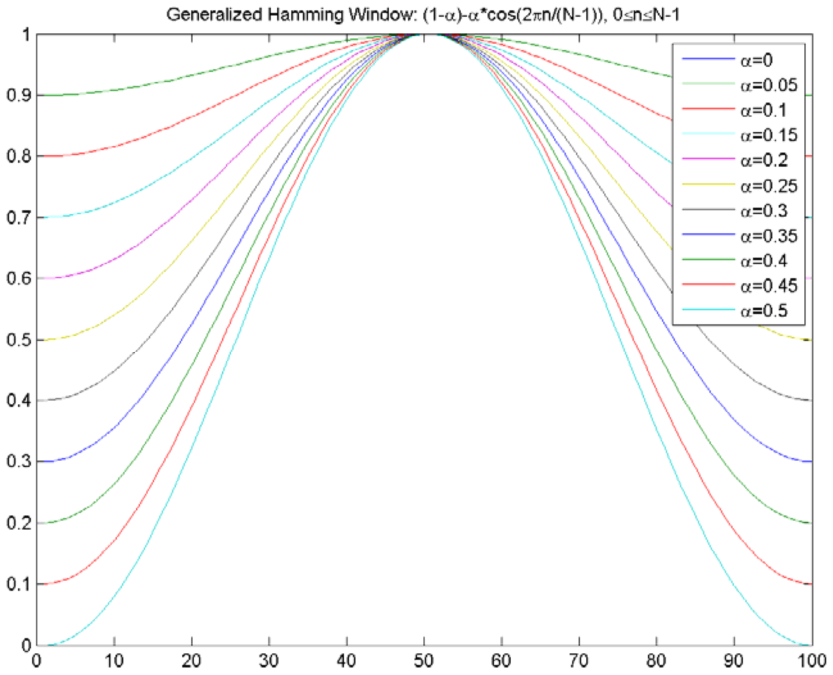
Different values of α corresponds to different curves for the Hamming windows shown next:

Example 2**Input file** speechFeature/hammingWindow01.m

```
% Plot of generalized Hamming windows
N=100;
n=(0:N-1)';
alpha=linspace(0,0.5,11)';
h=[];
for i=1:length(alpha),
        h = [h, (1-alpha(i))-alpha(i)*cos(2*pi*n/(N-1))];
end
plot(h);
title('Generalized Hamming Window: (1-\alpha)-\alpha*cos(2\pin/(N-1)), 0\leqn\leqN-1');

legendStr={};
for i=1:length(alpha),
        legendStr={legendStr{:}, ['\alpha=', num2str(alpha(i))]};
end
```

```
legend(legendStr);
```

Output figure



In practice, the value of α is set to 0.46. MATLAB also provides the command `hamming` for generating the curve of a Hamming window.

4. **Fast Fourier Transform** or **FFT**: Spectral analysis shows that different timbres in speech signals corresponds to different energy distribution over frequencies. Therefore we usually perform FFT to obtain the magnitude frequency response of each frame.

   When we perform FFT on a frame, we assume that the signal within a frame is periodic, and continuous when wrapping around. If this is not the case, we can still perform FFT but the incontinuity at the frame's first and last points is likely to introduce undesirable effects in the frequency response. To deal with this problem, we have two strategies:

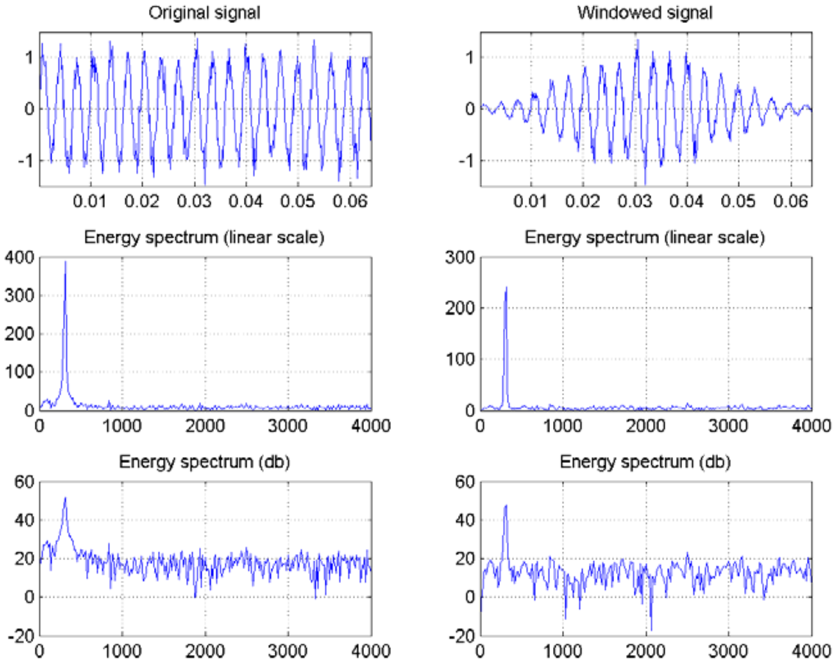
a. Multiply each frame by a Hamming window to increase its continuity at the first and last points.
b. Take a frame of a variable size such that it always contains a integer multiple number of the fundamental periods of the speech signal.

The second strategy encounters difficulty in practice since the identification of the fundamental period is not a trivial problem. Moreover, unvoiced sounds do not have a fundamental period at all. Consequently, we usually adopt the first strategy to mutiply the frame by a Hamming window before performing FFT. The following example shows the effect of multiplying a Hamming window.

Example 3**Input file** speechFeature/windowing01.m

```
fs=8000;
t=(1:512)'/fs;
f=306.396;

original=sin(2*pi*f*t)+0.2*randn(length(t),1);
windowed=original.*hamming(length(t));
[mag1, phase1, freq1]=fftOneSide(original, fs);
[mag2, phase2, freq2]=fftOneSide(windowed, fs);

subplot(3,2,1); plot(t, original); grid on; axis([-inf inf -1.5 1.5]); title('Original signal');
subplot(3,2,2); plot(t, windowed); grid on; axis([-inf inf -1.5 1.5]); title('Windowed signal');
subplot(3,2,3); plot(freq1, mag1); grid on; title('Energy spectrum (linear scale)');
subplot(3,2,4); plot(freq2, mag2); grid on; title('Energy spectrum (linear scale)');
subplot(3,2,5); plot(freq1, 20*log10(mag1)); grid on; axis([-inf inf -20 60]); title('Energy spectrum (db)');
subplot(3,2,6); plot(freq2, 20*log10(mag2)); grid on; axis([-inf inf -20 60]); title('Energy spectrum (db)');
```

Output figure



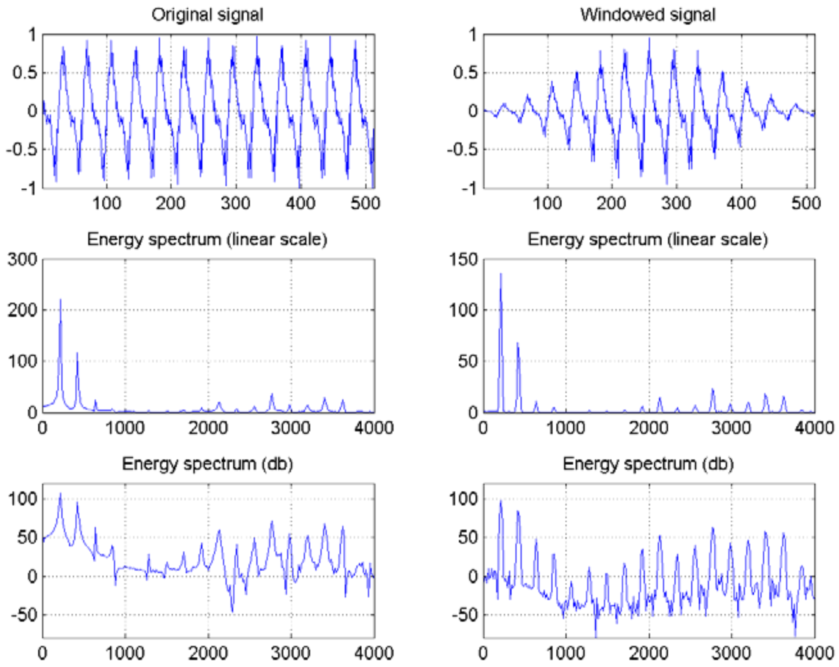
In the above example, the singal is a sinusoidal function plus some noise. Without the use of a Hamming window, the discontinuity at the frame's first and last points will make the peak in the frequency response wider and less obvious. With the use of a Hamming, the peak is sharper and more distinct in the frequency response. The following example uses a speech signal for the same test.

Example 4**Input file** speechFeature/windowing02.m

```
waveFile='littleStar.wav';
[y, fs]=wavread(waveFile);

n=512;
t=(1:n)'/fs;
startIndex=30418;
endIndex=startIndex+n-1;

original=y(startIndex:endIndex);
windowed=original.*hamming(n);
[mag1, phase1, freq1]=fftOneSide(original, fs);
```

```
[mag2, phase2, freq2]=fftOneSide(windowed, fs);


subplot(3,2,1); plot(original); grid on; axis([-inf inf -1 1]); title('Original signal');
subplot(3,2,2); plot(windowed); grid on; axis([-inf inf -1 1]); title('Windowed signal');
subplot(3,2,3); plot(freq1, mag1); grid on; title('Energy spectrum (linear scale)');
subplot(3,2,4); plot(freq2, mag2); grid on; title('Energy spectrum (linear scale)');
subplot(3,2,5); plot(freq1, 20*log(mag1)); grid on; axis([-inf inf -80 120]); title('Energy spectrum (db)');
subplot(3,2,6); plot(freq2, 20*log(mag2)); grid on; axis([-inf inf -80 120]); title('Energy spectrum (db)');
```

Output figure



In the above example, we use a frame from a clip of singing voice for the same test. With the use of a Hamming window, the harmonics in the frequency response are much sharper.

Remember that if the input frame consists of 3 identical fundamental periods, then the magnitude frequency response will be inserted 2 zeros between every two neighboring points of the frequency response of a single fundamental periods. (See the chapter on Fourier transform for more details.) In other words, the harmonics of the frequency response is generally caused by the repeating fundamental periods in the frame. However, we are more interested in the **envelope** of the frequency response instead of the frequency response itself. To extract an envelop-like features, we use the triangular bandpass filters, as explained in the next step.

5. **Triangular Bandpass Filters**: We multiple the magnitude frequency response by a set of 20 triangular bandpass filters to get the log energy of each triangular bandpass filter. The positions of these filters are equally spaced along the Mel frequency, which is related to the common linear frequency f by the following equation:

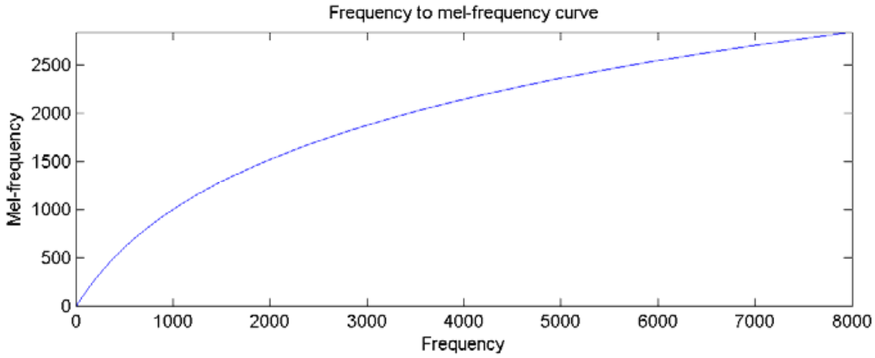
$$mel(f)=1125*ln(1+f/700)$$

Mel-frequency is proportional to the logarithm of the linear frequency, reflecting similar effects in the human's subjective aural perception. The following example plots the relationship between the mel and the linear frequencies:

Example 5**Input file** speechFeature/showMelFreq01.m

```
linFreq=0:8000;
melFreq=lin2melFreq(linFreq);
plot(linFreq, melFreq);
xlabel('Frequency');
ylabel('Mel-frequency');
title('Frequency to mel-frequency curve');
axis equal tight
```

Output figure

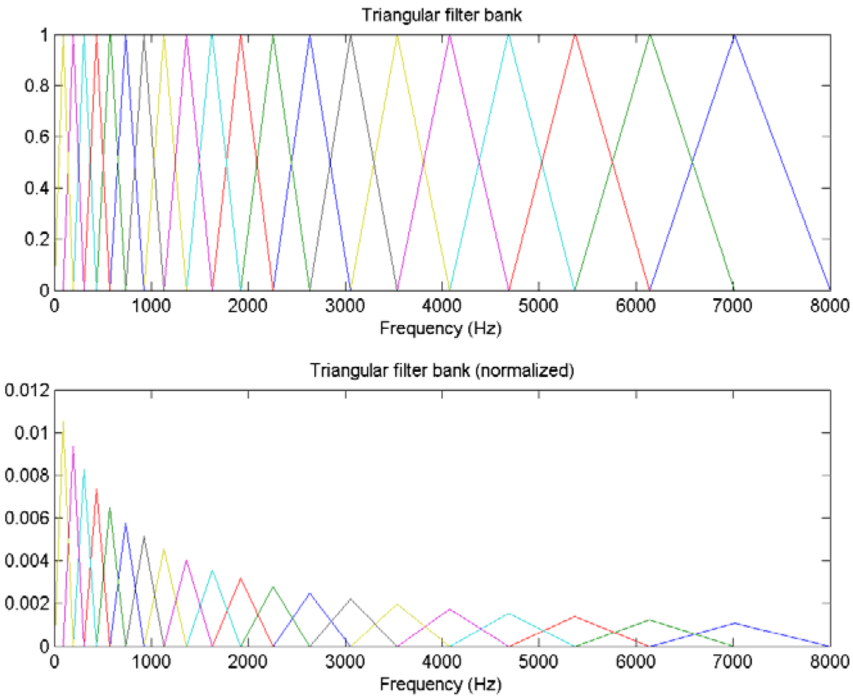


In practice, we have two choices for the triangular bandpass filters, as shown in the next example:

Example 6**Input file** speechFeature/showTriFilterBank01.m

```
fs=16000;
filterNum=20;
plotOpt=1;
getTriFilterBankParam(fs, filterNum, plotOpt);
```

Output figure



The reasons for using triangular bandpass filters are two fold:

- Smooth the magnitude spectrum such that the harmonics are flattened in order to obtain the envelop of the spectrum with harmonics. This indicates that the pitch of a speech signal is generally not presented in MFCC. As a result, a speech recognition system will behave more or less the same when the input utterances are of the same timbre but with different tones/pitch.
- Reduce the size of the features involved.

6. **Discrete cosine transform** or **DCT**: In this step, we apply DCT on the 20 log energy E_k obtained from the triangular bandpass filters to have L mel-scale cepstral coefficients. The formula for DCT is shown next.

$$C_m=\Sigma_{k=1}^{N}\cos[m*(k-0.5)*\pi/N]*E_k,\ m=1,2,\ ...,\ L$$

where N is the number of triangular bandpass filters, L is the number of mel-scale cepstral coefficients. Usually we set N=20 and L=12. Since we have performed FFT, DCT transforms the frequency domain into a time-like domain called quefrency domain. The obtained features are similar to cepstrum, thus it is referred to as the mel-scale cepstral coefficients, or MFCC. MFCC alone can be used as the feature for speech recognition. For better performance, we can add the log energy and perform delta operation, as explained in the next two steps.

7. **Log energy**: The energy within a frame is also an important feature that can be easily obtained. Hence we usually add the log energy as the 13rd feature to MFCC. If necessary, we can add some other features at this step, including pitch, zero cross rate, high-order spectrum momentum, and so on.
8. **Delta cepstrum**: It is also advantagous to have the time derivatives of (energy+MFCC) as new features, which shows the velocity and acceleration of (energy+MFCC). The equations to compute these features are:

$$\triangle C_m(t) = [\Sigma_{\tau=-M}{}^{M} C_m(t+\tau)\tau] / [\Sigma_{\tau=-M}{}^{M}\tau^2]$$

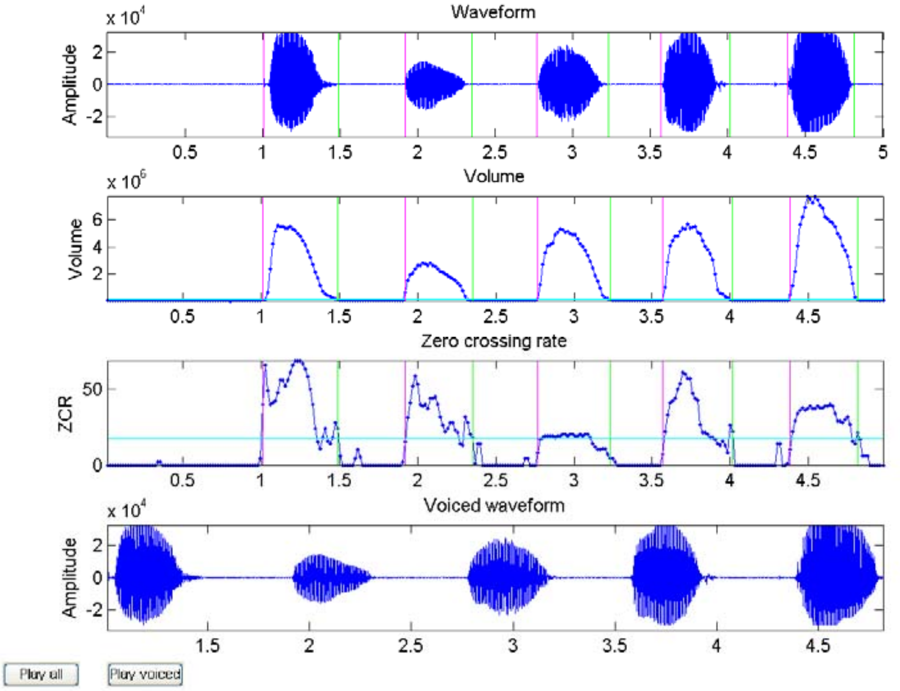
The value of M is usually set to 2. If we add the velocity, the feature dimension is 26. If we add both the velocity and the acceleration, the feature dimension is 39. Most of the speech recognition systems on PC use these 39-dimensional features for recognition.

第 12 章作業

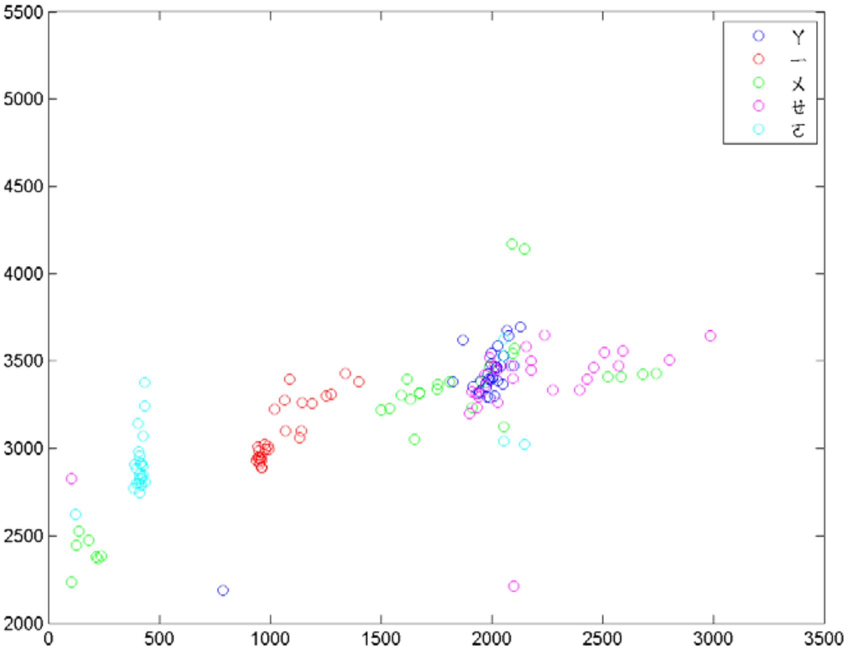
Old Chinese version

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。

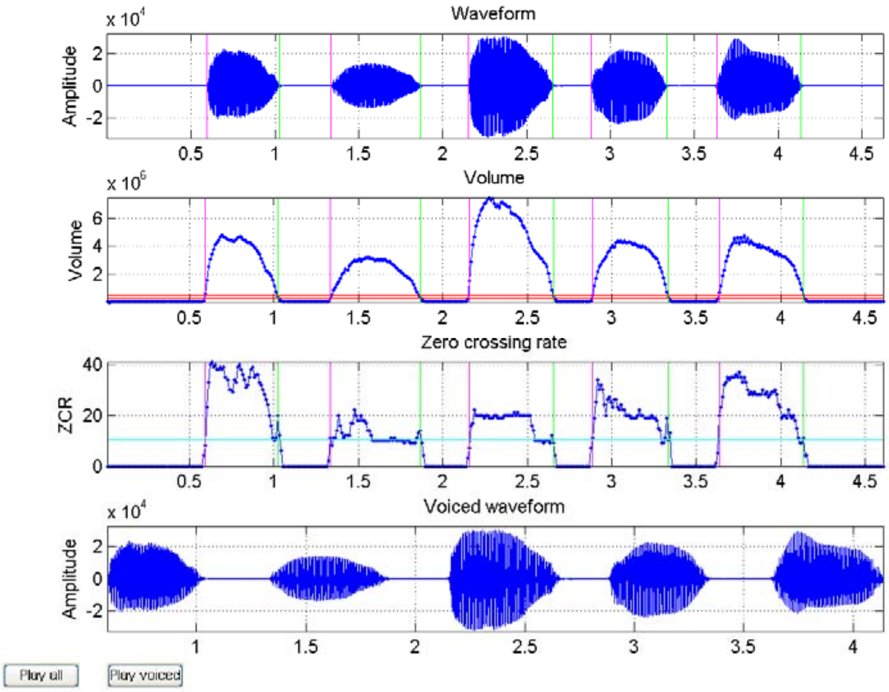
2. (**) **Frame to MFCC conversion**： Write a MATLAB function frame2mfcc.m that can compute 12-dimensional MFCC from a given speech/audio frame. Please follow the steps in the text.

   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:

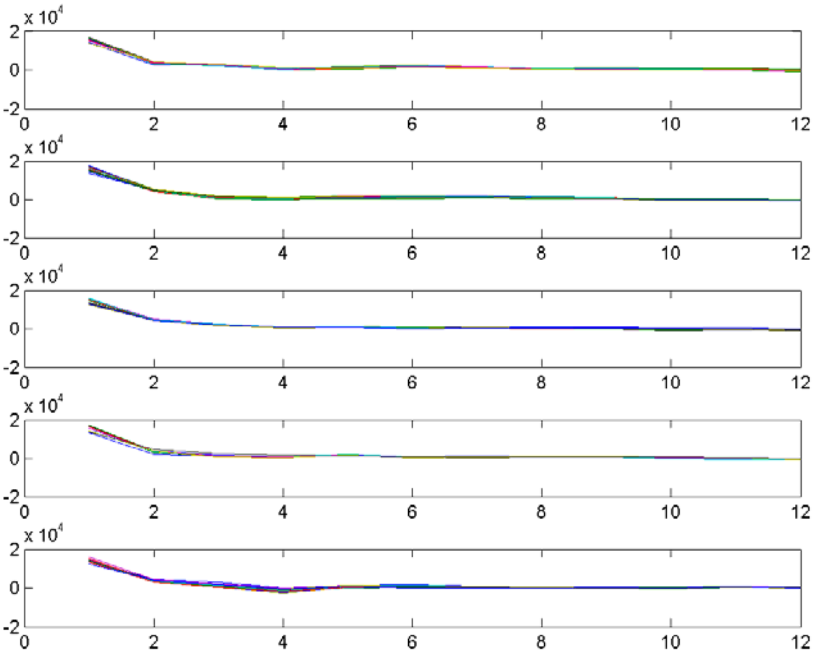
   a. Record a 5-second clips of the Chinese vowel 「ㄚ、ㄧ、ㄨ、ㄝ、ㄛ」 (or the English vowels "a, e, i, o, u") with 16KHz/16Bits/Mono. (Please try to maintain a stable pitch and volume, and keep a short pause between vowels to facilitate automatic vowel segmentation. Here is a sample file for your reference.)

   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:

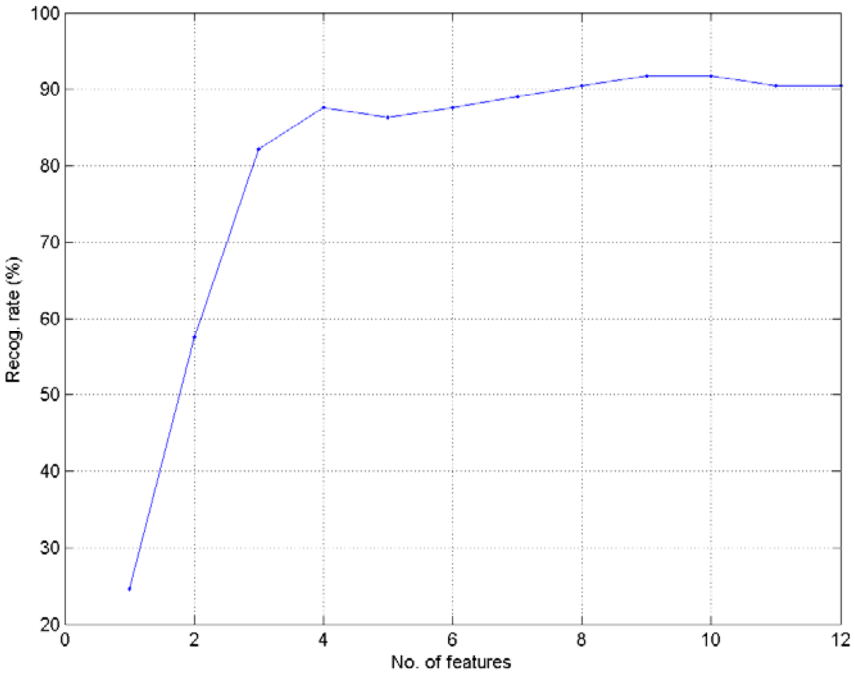


If the segmentation is not correct, you should adjust the parameters to epdByVol.m until you get the correct segmentation.

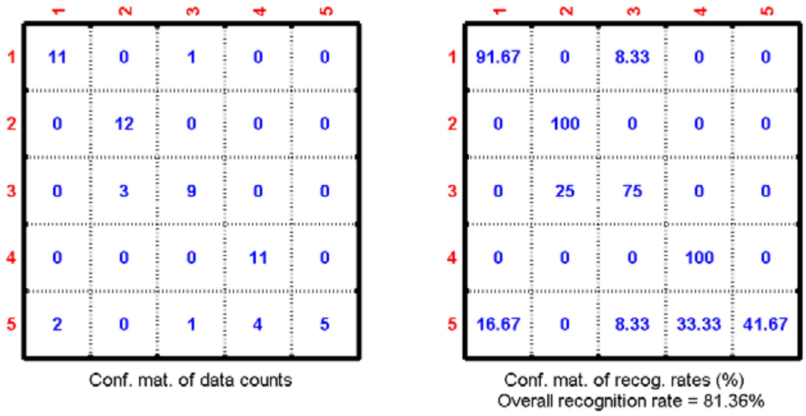
c. Use buffer2.m to do frame blocking on these 5 vowels, with frameSize=32ms and overlap=0ms. Please generate 5 plots of MFCC (use frame2mfcc.m or wave2mfcc.m) corresponding to each vowel. Each plot should contains as many curves of MFCC vectors as the number of frames in this vowel. Your plots should 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:



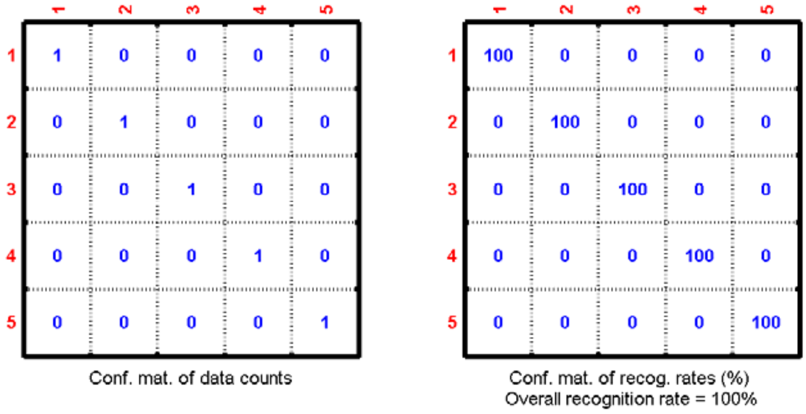
	1	2	3	4	5
1	11	0	1	0	0
2	0	12	0	0	0
3	0	3	9	0	0
4	0	0	0	11	0
5	2	0	1	4	5

Conf. mat. of data counts

	1	2	3	4	5
1	91.67	0	8.33	0	0
2	0	100	0	0	0
3	0	25	75	0	0
4	0	0	0	100	0
5	16.67	0	8.33	33.33	41.67

Conf. mat. of recog. rates (%)
Overall recognition rate = 81.36%

f. What is the vowel-based recognition rate? Plot the confusion matrix, which should be similar to the following figure:



	1	2	3	4	5
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4	0	0	0	1	0
5	0	0	0	0	1

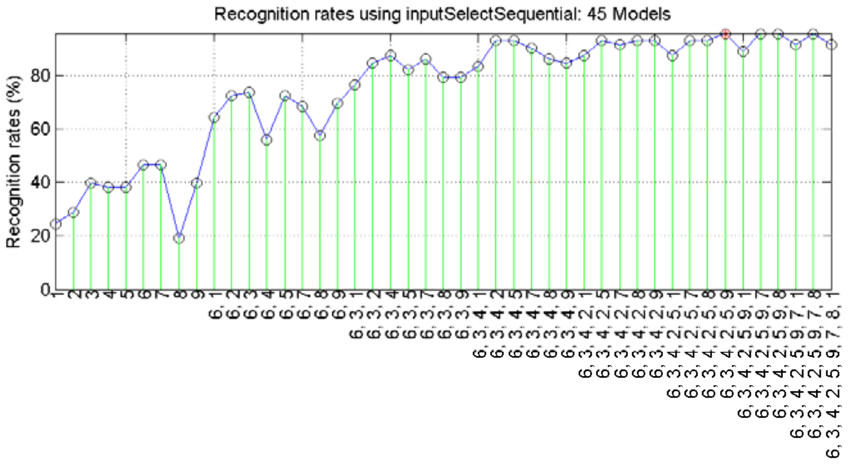
Conf. mat. of data counts

	1	2	3	4	5
1	100	0	0	0	0
2	0	100	0	0	0
3	0	0	100	0	0
4	0	0	0	100	0
5	0	0	0	0	100

Conf. mat. of recog. rates (%)
Overall recognition rate = 100%

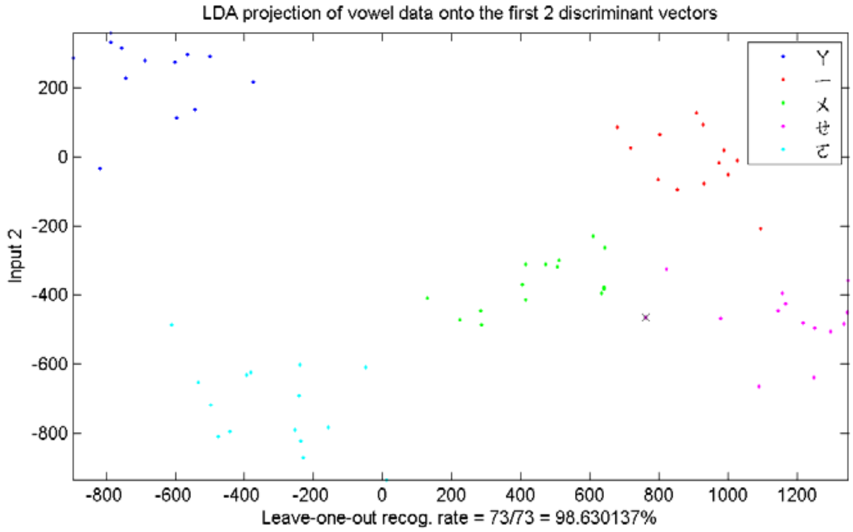
(Hint: The vowel-based recognition result is the voting of frame-based results. You can use `mode` to compute the result of voting.)

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)

Old Chinese version

The goal of this programming contest is to let students get familiar with the use of GMM (Gaussian Mixture Model) for speaker recognition. The students are required to tune a set of parameters to improve the recognition rates.

1. Data to download:
   - exampleProgram.rar: All example programs
   - Wave files of Tang Poems to be use for this contest. TA will give the ftp address in the class.
2. How to execute the example program:
   - Change addMyPath.m to point to the correct paths of the required toolboxes.
   - Set the variable waveDir in goExtractFeature.m to the path containing all the wave files.
   - Run the main program by typing "go" within MATLAB. The contents of go.m is shown next:
   - `% This is the main file for speaker recognition`
   -
   - `goExtractFeature;`
   - `goTrainGmm;`
   - `goUtteranceRr;`

   In other words, go.m invokes three other m-file scripts, with the following functions:

   a. goExtractFeature.m: Feature extraction.
   b. goTrainGmm.m: GMM training.
   c. goUtteranceRr.m: Evaluation of utterance-based recognition rates

   From the parameter settings at goExtractFeature.m, the script uses 10 utterances from each speaker, where the five odd-indexed utterances are used for training and the other ones are used for test. The inside-test recognition rate is 100% while the outside-test one is 99.00%. (Note that these are all utterance-based recognition rates.) The confusion matrix will also be given, as follows:

3. What you need to demonstrate during the class:
    - Plot the frame-based recognition rates (both inside and outside tests) as a function of the number of mixtures which takes the values of 2, 4, 8, 16, 32. (You might need to have more sentences for training if the number of mixture is large.)
    - Use the number of mixtures at which the outside-test recognition rate is at its maximum. Plot the segment-based recognition rates (both inside and outside tests) as a function of the segment length (in terms of number of frames).
    - You need to combine the above two to have this plot of the recognition rate w.r.t. the segment length. You need to show 10 curves, corresponding to the inside and outside tests for mixture numbers = 2, 4, 8, 16, 32.

    (Hint: Be sure to save the speakerData for further processing, since we do not change the feature set.)

4. How to modify the program to get better utterance-based recognition rates (please refer to "Robust Text-Independent Speaker Identification using Gaussian Mixture Speaker Models"):
    - Our example program only take 10 utterances from each speaker. If you take all utterances from a speaker, the computing time will be much longer. To get around, you should try your best to find a high-speed computer for this contest.
    - For end-point detection, you should try to get rid of silence as well as unvoiced sounds.
    - For GMM model parameters, you can try the following items:
        - Change the number of mixtures in GMM. Choose a number that can have the best outside-test recognition rate.
        - Use different methods for the initialization of k-means. A better k-means can improve the performance of GMM.

        - Choose a VQ method by center-splitting. The function is vqLbg.m in DCPR toolbox.
        - Increase the iteration count to see if we can get a higher log probability.
    - For feature extraction, you can try the following items:
        - Use the feature extraction function by HTK (htkWave2mfccMex.dll).
        - Use MFCC only, which does not contain the log energy.
        - Try cepstral mean normalization
    - For reducing the computing time:
        - You can replace vqKmeans.m with vqKmeansMex.dll for speeding up.
        - If you think gmmTrain.m is too slow, you can write a C-callable mex file for speeding up.

5. Performance evaluation: our TA will carry out both inside and outside tests to compute utterance-based recognition rates based on all the utterances (odd-indexed utterances as the training and even-indexed as the test set), and to have their average as the final performance index.
6. The files for uploading:
    - gmmParam.mat: Parameters for GMM
    - method.txt: Description of your methods
    - Any files that are necessary for running your main program.

第 13 章作業

1. (***) **Programming contest: use GMM for speaker identification**： Please see the detailed descriptions。

Chapter 14: Dynamic Programming

14-1 Introduction to Dynamic Programming (動態規劃)

Old Chinese version

Dynamic programming (DP for short) is an effective method to find a optimum solution to a problem, as long as the solution can be defined recursively. DP roots in the **principle of optimality** proposed by Richard Bellman:

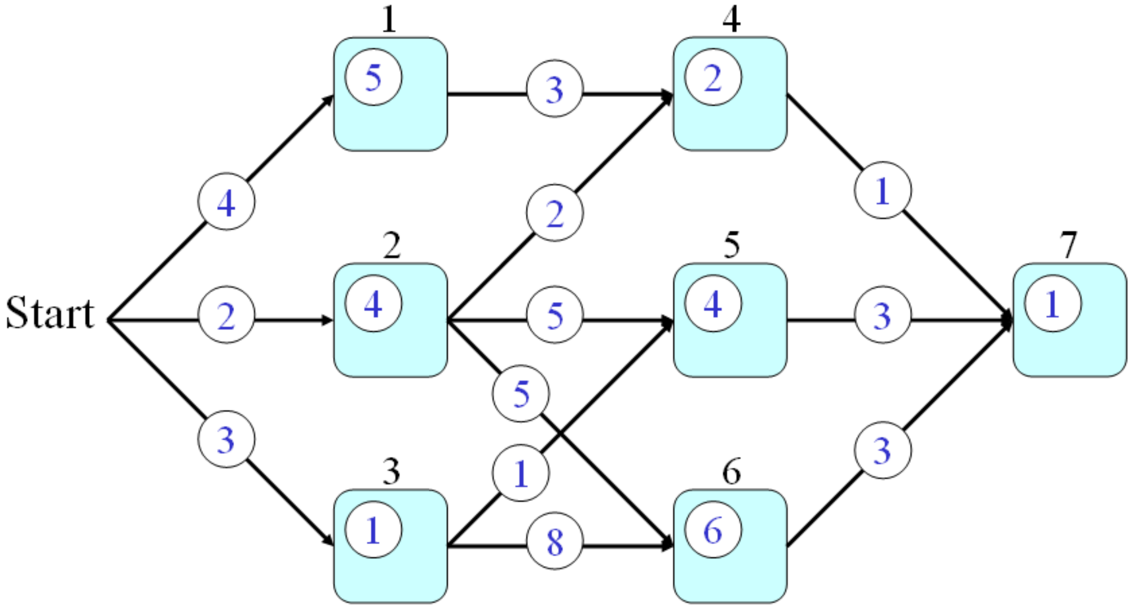
An optimal policy has the property that whatever the initial state and the initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

In terms of path finding problems, the principle of optimality states that any partial path of the optimum path should be itself an optimum path given the starting and ending nodes. This is an obvious principle that can be proved by contradiction. In practice, a great number of seemingly unrelated applications can be effectively solved by DP.

To employ the procedure of DP to solve a problem, usually we need to specify the following items:

I. Define the optimum-value function.
II. Derive the recursive formula for the optimum-value function, together with its initial condition.
III. Specify the answer of the problem in term of the optimum-value function.

We shall give an example of DP by using the following graph:



In the above graph, we can assume that:

- Every node is a city and we use q(a) to represent the time required to go through the city represented by node a.
- Every link is a route connecting two city. We use p(a, b) to represent the time required to go from node a to b.

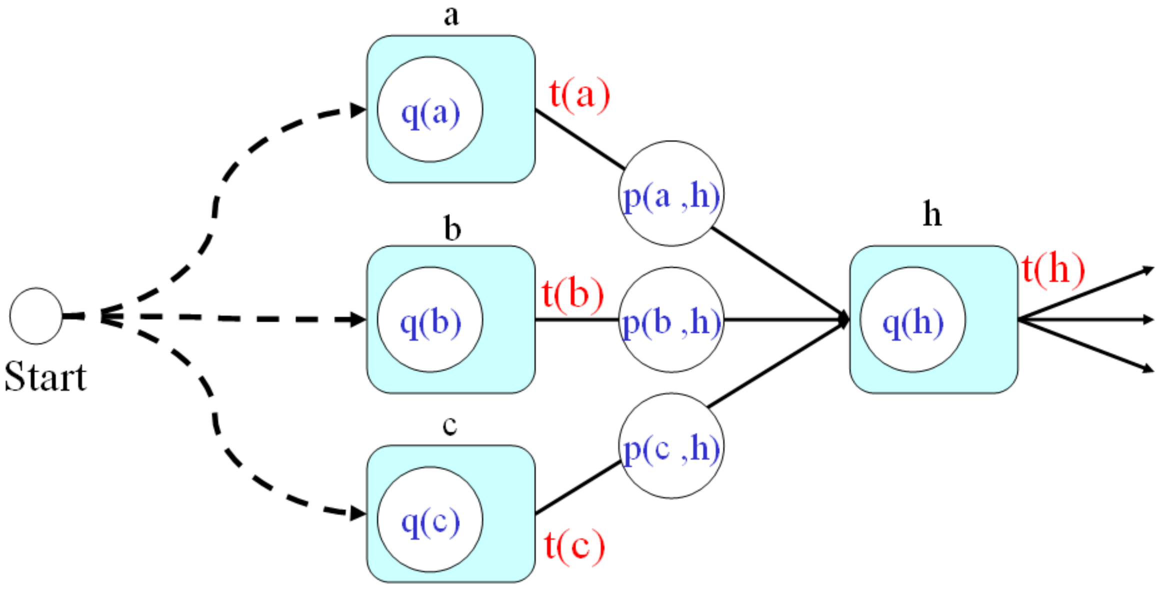
From the start point, how can we find a path the requires the minimum time to end at node 7? This is a typical problem of DP which can be solved systematically and efficiently by the following three steps:

I. First of all, we can define the optimum-value function t(h) as the minimum time from the start point to node h (including the time for passing the node h).

II. Secondly, the optimum-value function should satisfy the following recursive formula:

$$t(h) = \min\{t(a)+p(a,h),\ t(b)+p(b,h),\ t(c)+p(c,h)\} + q(h)$$

In the above equation, we assume the fan-ins of node h are a, b, and c. Please refer to the next figure.



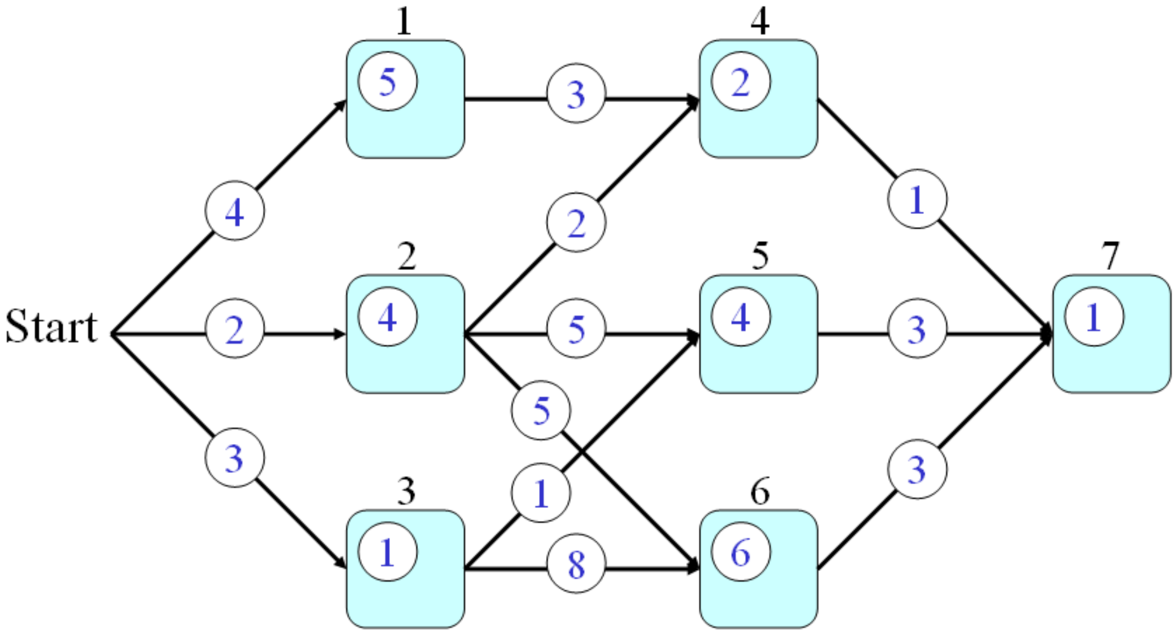
And the initial condition is t(0)=0 where 0 is the start node.

III. And finally, the answer to the original problem should be t(7). By using the recursion, we can have the following steps for computing the time required by the optimum path:

1. t(0) = 0
2. t(1) = t(0)+4+5 = 9
3. t(2) = t(0)+2+4 = 6
4. t(3) = t(0)+3+1 = 4

5. t(4) = min(9+3, 6+2)+2 = 10
6. t(5) = min(6+5, 4+1)+4 = 9
7. t(6) = min(6+5, 4+8)+6 = 17
8. t(7) = min(10+1, 9+3, 17+3)+1 = 12

The value of the optimum-value function is shown as the red number in each node in the following figure:



Hint

You can open a new figure by clicking the above figure, and then click the newly opened figure to see each step of the recursion by DP.

The above formulation of DP are usually referred to as the **forward DP**. On the other hand, we can defined the formula for **backward DP** in a similar manner:

I. Firstly, the optimum-value function s(h) is defined as the minimum time from a node h to the end node (including the time for passing the end node but excluding the time for passing node h.)

II. Secondly, the optimum-value function should satisfy the following recursive formula:

s(h) = q(h) + min{p(h,a)+s(a), p(h,b)+s(b), p(h,c)+s(c)}

where a, b, c are the fan-out of node h. The initial condition is s(7) = q(7).

III. Finally, the answer to the original problem is s(0). By using the recusion, we can have the following steps for computing the time required by the optimum path:

1. s(7) = 1
2. s(6) = q(6)+3+s(7) = 6+3+1 = 10
3. s(5) = q(5)+3+s(7) = 4+3+1 = 8
4. s(4) = q(4)+1+s(7) = 2+1+1 = 4
5. s(3) = q(3)+min{1+s(5), 8+s(6)} = 1 + min{9, 18} = 10
6. s(2) = q(2)+min{2+s(4), 5+s(5), 5+s(6)} = 4 + min{6, 13, 15} = 10
7. s(1) = q(1)+3+s(4) = 5 + 3 + 4 = 12
8. s(0) = min{4+s(1), 2+s(2), 3+s(3)} = min(16, 12, 13} = 12

The answer obtained from backword DP is the same as that of the forward DP.

The efficiency of DP is based on the fact that we can use some previous results iteratively. Some common characteristics of DP are summarized as follows:

- After finding the optimum fan-in node of a given node, it would be better to keep this information in the data structure for further use. This will help us to back track the optimum path at the ending node.
- DP usually only gives the optimum path. If we want to find the second-best path, we need to invoke another method of top-n path finding, which is beyond our scope for the time being.

14-2 Longest Common Subsequence

Old Chinese version

Given a sequence, we can delete any elements to form a subsequence of the original sequence. For instance, given a string **s** = uvwxyz, we can delete v and x to get a subsequence uwyz. For any given two sequences **a** and **b**, the similarity between them can be defined as the length of the **longest common subsequence** (**LCS** for short) of these two sequences, which can be computed efficiently by DP.

Let us define the optimum-value function LCS(**a**, **b**) as the length of the longest common subsequence between **a** 和 **b**. Then the recursive formula for LCS can is shown next.

1. LCS(**a**x, **b**y) = LCS(**a**, **b**)+1 if x = y.
2. LCS(**a**x, **b**y) = max(LCS(**a**x, **b**), LCS(**a**, **b**y)) if x ≠ y.

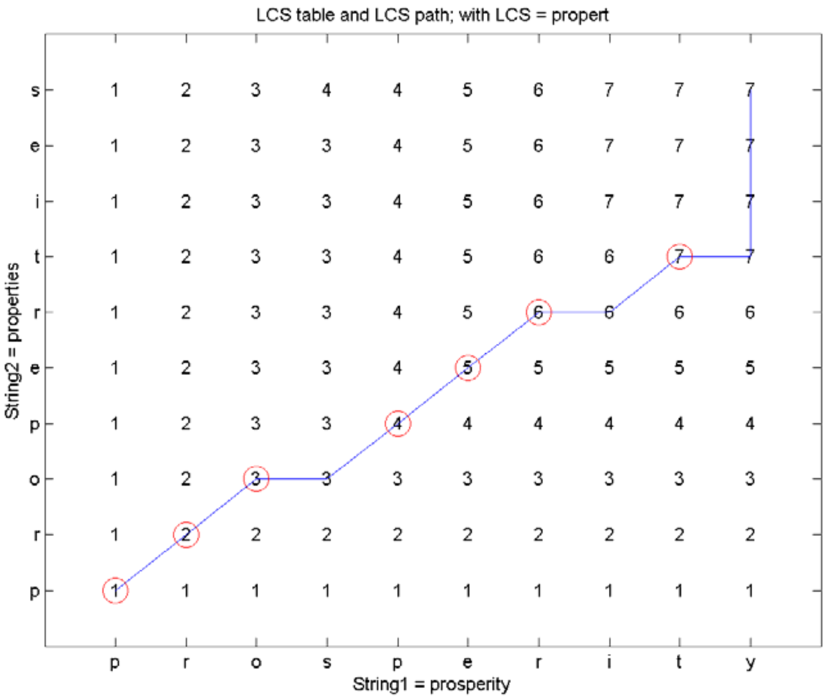
The boundary conditions are LCS(**a**, []) = 0, LCS([], **b**) = 0。

The following example is the result of a typical LCS result:

Example 1**Input file** dp/lcs01.m

```
str1 = 'prosperity';
str2 = 'properties';
plotOpt = 1;
[lcscount, lcsPath, lcsStr, lcsTable] = lcs(str1, str2, plotOpt);
```

Output figure


LCS table and LCS path; with LCS = propert

	p	r	o	s	p	e	r	i	t	y
s	1	2	3	4	4	5	6	7	7	7
e	1	2	3	3	4	5	6	7	7	7
i	1	2	3	3	4	5	6	7	7	7
t	1	2	3	3	4	5	6	6	7	7
r	1	2	3	3	4	5	6	6	6	6
e	1	2	3	3	4	5	5	5	5	5
p	1	2	3	3	4	4	4	4	4	4
o	1	2	3	3	3	3	3	3	3	3
r	1	2	2	2	2	2	2	2	2	2
p	1	1	1	1	1	1	1	1	1	1

14-3 Edit Distance

Old Chinese version

We can use three basic operations of "delete", "insert", and "substitute" to convert a string into another one. The **edit distance** between two strings is defined as the minimum number of the basic operations that are required to converting a string into another.

We can also use the concept of DP to compute the edit distance between two strings. Let the optimum-value function ED(**a**, **b**) defined as the edit distance between strings **a** and **b**. Then the recursive formula for ED is shown next.

1. ED(**a**x, **b**y) = ED(**a**, **b**) if x = y.
2. ED(**a**x, **b**y) = min(ED(**a**, **b**), ED(**a**x, **b**), ED(**a**, **b**y)) if x ≠ y.

The boundary conditions are ED(**a**, []) = len(**a**), ED([], **b**) = len(**b**)。

Hint

The edit distance is also used in the command "fc" (under DOS) or "diff" (under UNIX) for compariing the contents of two text files.

14-4 Dynamic Time Warping

Old Chinese version

The distance between two point **x** = [x_1, x_2, ..., x_n] and **y** = [y_1, y_2, ..., y_n] in a n-dimensional space can be computed via the Euclidean distance:

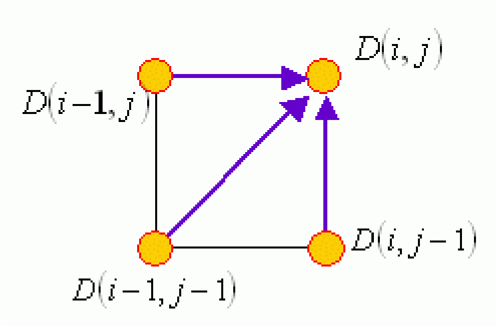
$$\text{dist}(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}| = [\ (x_1-y_1)^2 + (x_2-y_2)^2 + \dots + (x_n-y_n)^2\ ]^{1/2}$$

However, if the length of **x** is different from **y**, then we cannot use the above formula to compute the distance. Instead, we need a more flexible method that can find the best mapping from elements in **x** to those in **y** in order to compute the distance.

The goal of **dynamic time warping** (**DTW** for short) is to find the best mapping with the minimum distance by the use of DP. The method is called "time warping" since both **x** and **y** are usually vectors of time series and we need to compress or expand in time in order to find the best mapping. We shall give the formula for DTW in this section.

Let **t** and **r** be two vectors of lengths m and n, respectively. The goal of DTW is to find a mapping path {(p_1, q_1), (p_2, q_2), ..., (p_k, q_k)} such that the distance on this mapping path $\Sigma_{i=1}^{k}$ ｜ $t(p_i)$ - $r(q_i)$｜ is minimized, with the following constraints:

- Boundary conditions: (p_1, q_1) = (1, 1), (p_k, q_k) = (m, n). This is a typical example of "anchored beginning" and "anchored end".
- Local constraint: For any given node (i, j) in the path, the possible fan-in nodes are restricted to (i-1, j), (i, j-1), (i-1, j-1). This local constraint guarantees that the mapping path is monotonically non-decreasing in its first and second arguments. Moreover, for any given element in **t**, we should be able to find at least one corresponding element in **r**, and vice versa.



How can we find the optimum mapping path in DTW? An obvious choice is forward DP, which can be summarized in the following three steps:

1. Optimum-value function: Define D(i, j) as the DTW distance between t(1:i) and r(1:j), with the mapping path from (1, 1) to (i, j).
2. Recursion: D(i, j) = ︱t(i) - r(j)︱ + min{D(i-1, j), D(i-1, j-1), D(i, j-1)}, with the initial condition D(1, 1) = ︱t(1) - r(1)︱.
3. Final answer: D(m, n)

In practice, we need to construct a matrix D of dimensions m×n first and fill in the value of D(1, 1) by using the initial condition. Then by using the recursive formula, we fill the whole matrix one element at a time, by following a column-by-column or row-by-row order. The final answer will be available as D(m, n), with a computational complexity of O(mn).

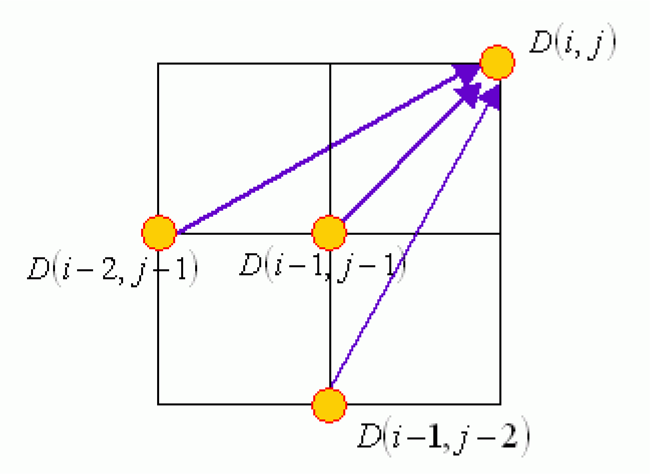
If we want to know the optimum mapping path in addition to the minimum distance, we may want to keep the optimum fan-in of each node. Then at the end of DP, we can quickly back track to find the optimum mapping path between the two input vectors.

We can also have the backward DP for DTW, as follows:

1. Optimum-value function: Define D(i, j) as the DTW distance between t(i:m) and r(j:n), with the mapping path from (i, j) to (m, n).
2. Recusion: D(i, j) = ︱t(i) - r(j)︱ + min{D(i+1, j), D(i+1, j+1), D(i, j+1)}, with the initial condition D(m, n) = ︱t(m) - r(n)︱
3. Final answer: D(1, 1)

The answer obtain by the backward DP should be that same as that obtained by the forward DP.

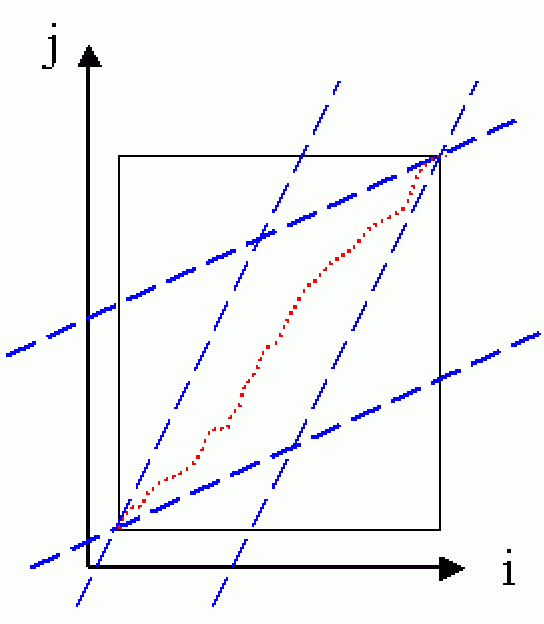
Another commonly used local path constraint is to set the fan-in of 27°-45°-63° only, as shown in the following figure:



The use of this local path constraint leads to the following potential results:

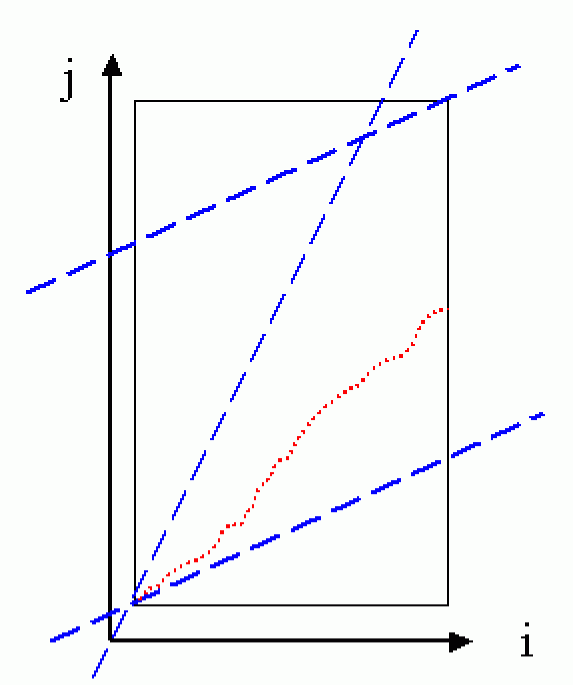
- Some points in the input vectors could be skipped in the mapping path. This is advantageous if the input vectors contains sporadic outliners.
- Since the optimum-value function is based on the total distance, sometimes the mapping path will take the local paths of 27° or 63° instead of 45° in attempting to minimize the total distance.

If the required mapping path is "anchored beginning, anchored end", then the local path constraints can induce the global path constraints, as shown next:



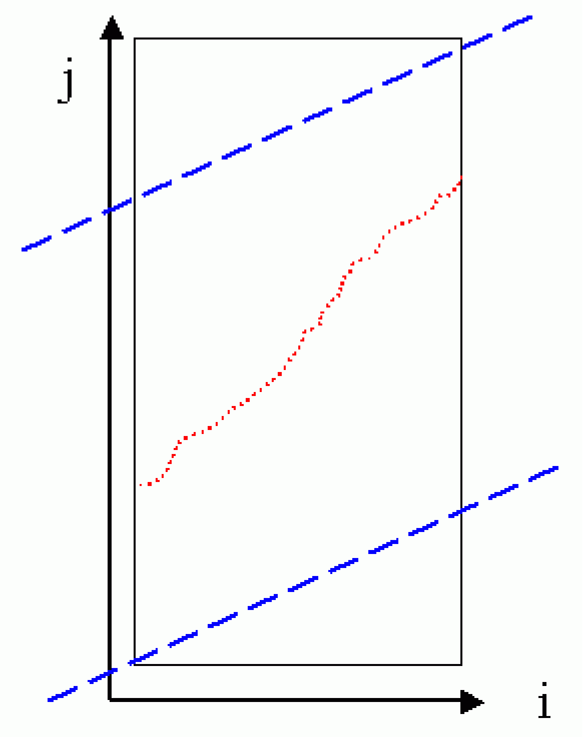
In the above figure, the feasible region for the mapping path is shown as a parallelogram. In filling up the matrix, we can save computation by skipping the elements outside the parallelogram. In addition, a feasible mapping exists only when the ratio of the input vectors' lengths are between 0.5 and 2.

If the required mapping path is "anchored beginning and free end" (which is used for query by singing or humming), then the induced global path constraints are shown next:



The feasible region for the mapping path is shown as a trapezoid in the above figure. Compared with the case of "anchored beginning and anchored end", we can save less computation by using this set constraints.

If the required mapping path is "free beginning and free end" (which is used for speaker-dependent keyword spotting), then the induced global path constraints are shown next:



The feasible region for the mapping path is shown as a parallelogram in the above figure. Compared with the case of "anchored beginning and free end", we can save less computation by using this set constraints.

It should be noted that the global path constraints are induced as a result of the local path constraints of 27°-45°-63°. However, when we are using the local path constraints of 0°-45°-90°，sometimes we still apply the global path constratins in order to limit the mapping path to a reasonable shape. In summary, the use of global path constraints serve two purposes:

- Reduce computation load.
- Limit the mapping path to be a reasonable path.

In the following, we shall give some examples of DTW. For simplicity, we shall distinguish two types of DTW:

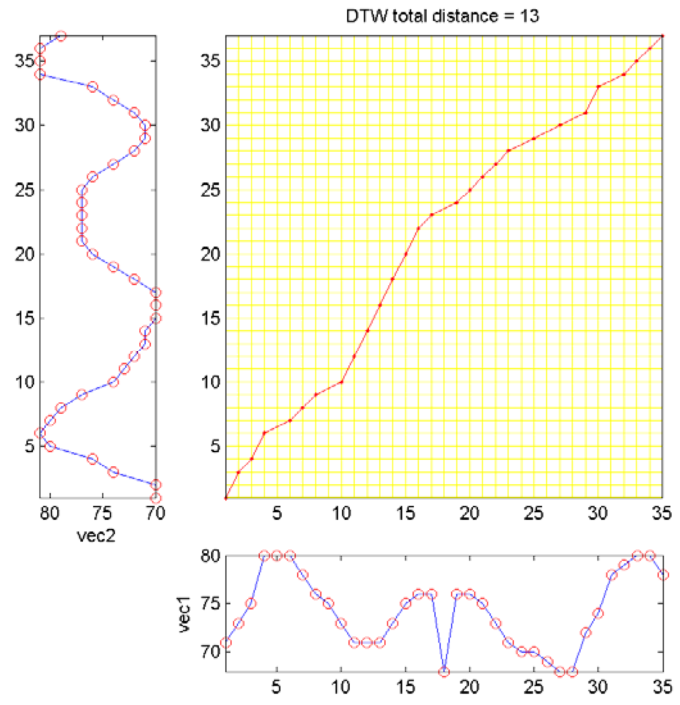
- Type-1 is the DTW with 27°-45°-63° local path constraints.
- Type-2 is the DTW with 0°-45°-90° local path constraints.

First, we can use dtw1.m and dtwPlot.m to plot the mapping path of type-1 DTW in the following example:

Example 1**Input file** dp/dtw1Plot01.m

```
vec1=[71 73 75 80 80 80 78 76 75 73 71 71 71 73 75 76 76 68 76 76 75 73 71 70 70 69 68 68 72 74 78 79 80 80 78];
vec2=[69 69 73 75 79 80 79 78 76 73 72 71 70 70 69 69 69 71 73 75 76 76 76 76 76 75 73 71 70 70 71 73 75 80 80 80 78];
[minDist, dtwPath, dtwTable] = dtw1(vec1, vec2);
dtwplot(vec1, vec2, dtwPath);
```

Output figure



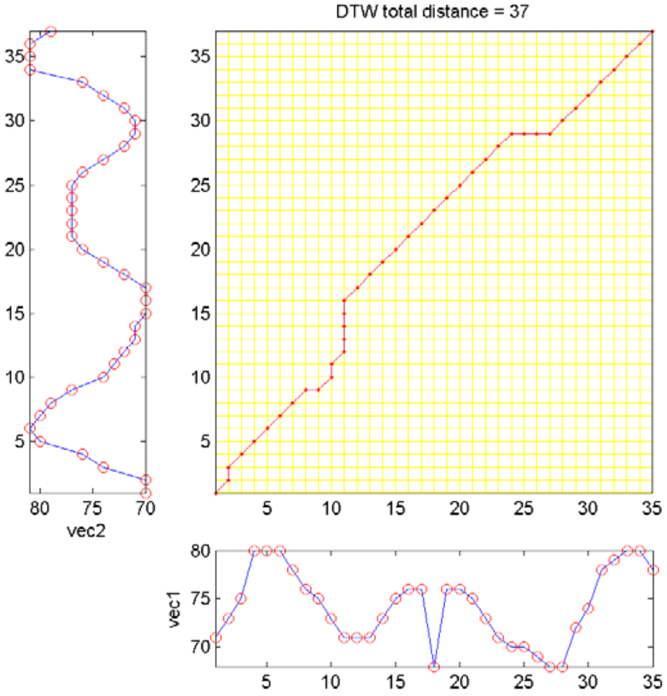
In the above example, we deliberately put an outliner in vec1. Due to the local path constraints of type-1 DTW, this outliner is skipped in the optimum mapping path.

Similarly, we can use dtw2.m and dtwPlot.m to plot the mapping path of type-2 DTW:

Example 2**Input file** dp/dtw2Plot01.m

```
vec1=[71 73 75 80 80 80 78 76 75 73 71 71 71 73 75 76 76 68 76 76 75 73 71 70 70 69 68 68 72 74 78 79 80 80 78];
vec2=[69 69 73 75 79 80 79 78 76 73 72 71 70 70 69 69 69 71 73 75 76 76 76 76 76 75 73 71 70 70 71 73 75 80 80 80 78];
[minDist, dtwPath, dtwTable] = dtw2(vec1, vec2);
dtwplot(vec1, vec2, dtwPath);
```

Output figure



The outliner cannot be skipped since the local path constraints require each point in a vector has at least one correspondence in the other vector.

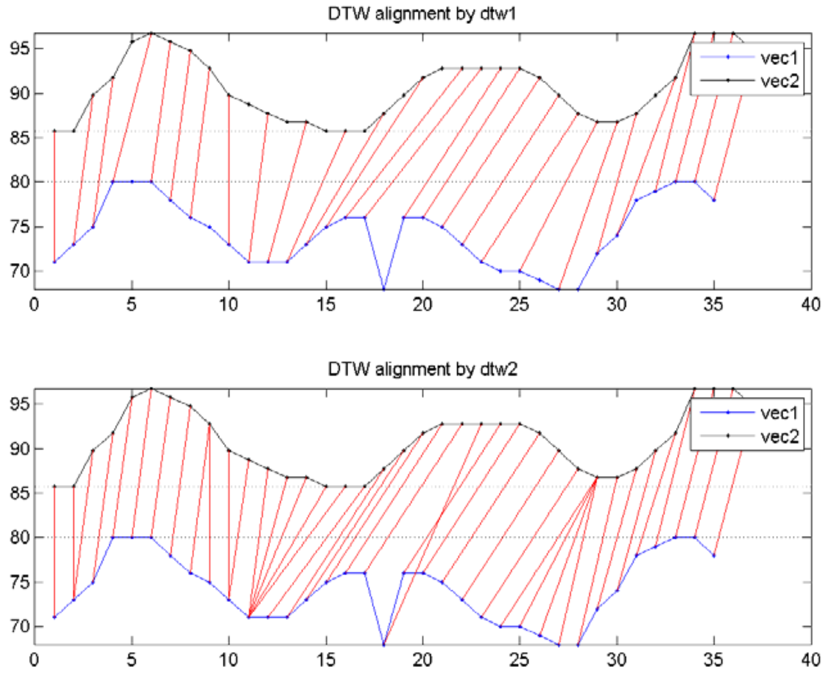
By using dtwPlot2.m, we can use red segments to connect corresponding points in two vectors, as follows:

Example 3**Input file** dp/dtwPlot02.m

```
vec1=[71 73 75 80 80 80 78 76 75 73 71 71 71 73 75 76 76 68 76 76 75 73 71 70 70 69 68 68 72 74 78 79 80 80 78];
vec2=[69 69 73 75 79 80 79 78 76 73 72 71 70 70 69 69 69 71 73 75 76 76 76 76 76 75 73 71 70 70 71 73 75 80 80 80 78];
[minDist1, dtwPath1, dtwTable1] = dtw1(vec1, vec2);
```

```
[minDist2, dtwPath2, dtwTable2] = dtw2(vec1, vec2);
subplot(2,1,1); dtwplot2(vec1, vec2, dtwPath1); title('DTW alignment by dtw1');
subplot(2,1,2); dtwplot2(vec1, vec2, dtwPath2); title('DTW alignment by dtw2');
```

Output figure

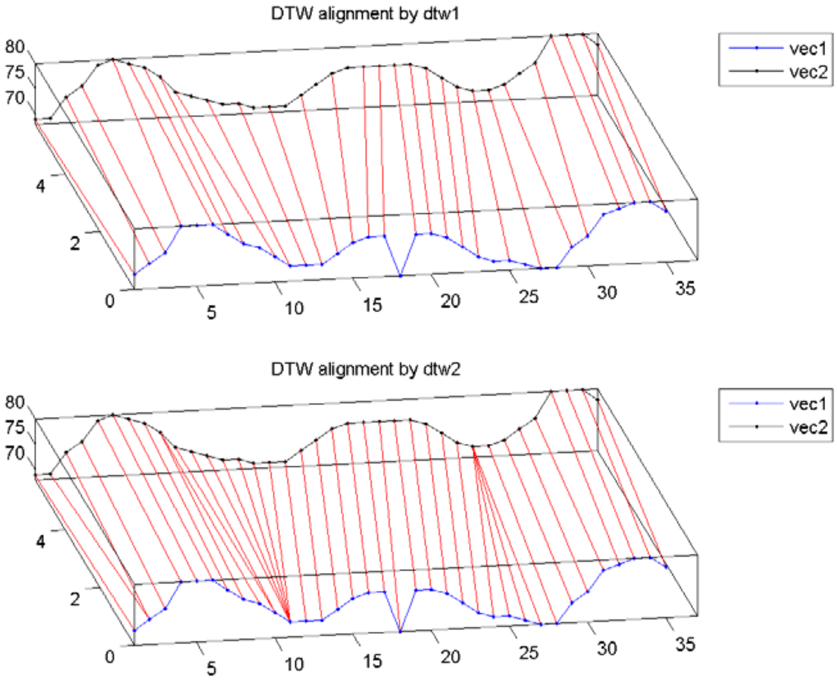


It becomes more obvious that type-1 DTW can have empty correspondence while type-2 DTW can have multiple correspondences. Moreover, we can use dtwPlot3.m to put the two vectors in 3d space to have a more interesting representation:

Example 4**Input file** dp/dtwPlot03.m

```
vec1=[71 73 75 80 80 80 78 76 75 73 71 71 71 73 75 76 76 68 76 76 75 73 71 70 70 69 68 68 72 74 78 79 80 80 78];
vec2=[69 69 73 75 79 80 79 78 76 73 72 71 70 70 69 69 69 71 73 75 76 76 76 76 76 75 73 71 70 70 71 73 75 80 80 80 78];
[minDist1, dtwPath1, dtwTable1] = dtw1(vec1, vec2);
[minDist2, dtwPath2, dtwTable2] = dtw2(vec1, vec2);
subplot(2,1,1); dtwplot3(vec1, vec2, dtwPath1); title('DTW alignment by dtw1'); view(-10, 70);
subplot(2,1,2); dtwplot3(vec1, vec2, dtwPath2); title('DTW alignment by dtw2'); view(-10, 70);
```

Output figure



Here are some slides for your reference:

- DTW for melody recognition
- DTW for speech recognition

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:

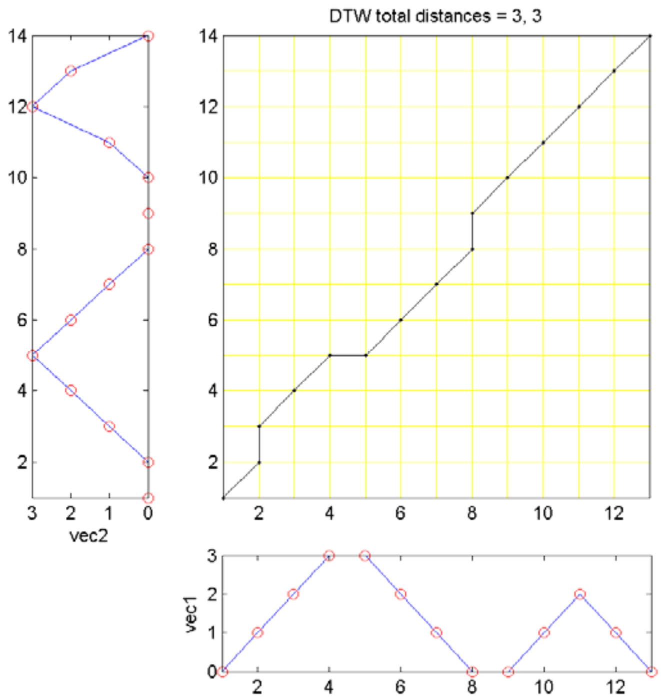
Example 1**Input file** dp/dtw2test01.m

```
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:

```
  ii.    one
 iii.    two
  iv.    three
   v.    five
  vi.    ...
```

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)

15-1 Introduction (簡介)

Old Chinese version

In the previous chapter, we have introduced DTW which is ideal for speaker-dependent ASR applications. This is suitable for the retrieval of utterances from the same speaker. A typical example of such applications is name dialing on mobile phones where you need to record your voices first. On the other hand, if we want to have speaker-independent ASR, we need to resort HMM (Hidden Markov Models) which is a statistic model that requires a massive amount of training data for reliable recognition. Based on the used statistical models, HMM can be classified into two categories:

- Discrete HMM (DHMM for short): This type of HMM evaluates probabilities based on discrete data counting.

- Continuous HMM (CHMM for short): This type of HMM evaluates probabilities based on continuous probability density functions such as GMM.

We shall introduce these two types of HMMs for speech recognition in the following sections.

15-2 Discrete HMM

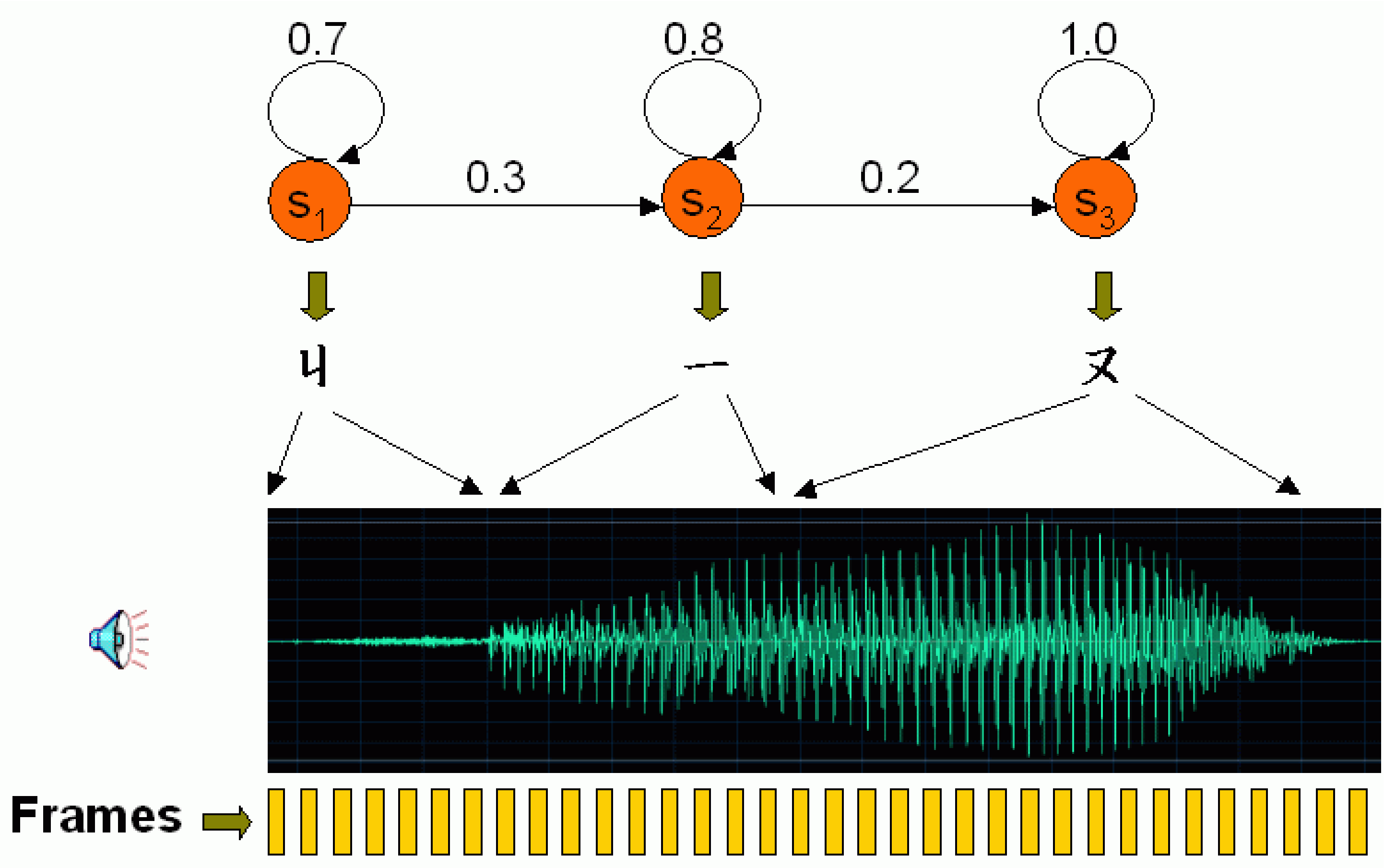
Old Chinese version

The simplest ASR application is voice command recognition. To construct a voice command recognizer, we need to collect training data for each voice command. This data is used for constructing an HMM for each voice command. For a given utterance, we need to send the corresponding acoustic features (MFCC, for example) to each HMM for evaluation the log probability. The HMM with the highest log probability represents the predicted voice command for the given utterance.

For simplicity in the following discussion, we shall assume our goal is to construct a speaker-independent digit recognizer that can recognize the utterance of 0 ~ 9. There are two steps involved in the design of such recognition system:

1. Corpus collection: To achieve speaker independence, we need to have a wide collection of speech corpus, including:
    - Subjects: We need to have utterances from various subjects, including different genders, different age groups, different accent (from different regions), etc.
    - Recording devices: We need to use different microphones and different sound cards, etc.
    - Environment: We need to have a recording environment that is close to that of the recognizer at application. To increase the robustness, we should do the recording in lab/office as well as road sides, etc.
2. Recognizer design: We need to design a recognizer based on HMM and test its performance. This will be detailed in the rest of this section.

First of all, we need to construct a DHMM for each digit. Take the digit 「九」for example, the corresponding DHMM is shown next:



In other words, we can segment the utterance of 「九」into 3 states, each representing the pronunciation of ㄐ，ㄧ，ㄡ. Note that each state covers a few frames, and the degree to which a frame belongs to a state is specified by a state **probability**. Moreover, for a given frame, it can stay in the current state with a **self-transition probability**, or transit to the next state with a **next-transition probability**.

Before using DHMM, each frame is converted into a feature vector (such as MFCC), and each vector is transformed into a symbol. More specifically, feature vectors of the same state are partitioned into clusters using vector quantization. We then use the index of a cluster to represent the symbol of a frame. That is, if the feature vector of frame i belongs to cluster k, then the symbol of frame i is k, as shown in the following expression:

$$k = O(i).$$

In order to simplify the discussion, we define some variables shown next.

- frameNum: The number of frames for a given utterance
- dim: The dimension of the acoustic features used in our recognizer. (For instance, the basic MFCC has 12 dimensions.)
- stateNum: the number of states in a DHMM
- symbolNum: The number of symbols, or equivalently, the number of clusters after vector quantization.

Usually we use the state and transition probabilities to characterize a given DHMM, as follows.

- We use a stateNum×stateNum matrix A to denote the state probability, in which A(i, j) denotes the probability from state i to j. For instance, the probability of transition from state 1 to 2 is 0.3, hence A(1, 2) = 0.3. In general, A(i, j) satisfies the following conditions:
    - State transitions are allowed only between neighboring states. Hence A(i, j) = 0 if j≠i and j≠i+1。
    - For a given state, the total of the self-state and next-state transition probabilities is equal to 1. Namely, A(i, i) + A(i, i+1) = 1, for all i.
- We use a symbolNum×stateNum matrix B to denote the state probability, in which B(k, j) denotes the probability to which symbol k belongs to state j. Therefore to compute the probability of frame i, we need to find the corresponding symbol k=O(i), and then retrieve the probability from B(k, j).

Assume that we already have matrices A and B for the DHMM of 「九」. Then for a given utterance, we can use **Viterbi decoding** to evaluate the probability of the utterance being generated by the DHMM. Viterbi decoding is based on the concept of dynamic program to assign each frame to a state such the accumulated probability can be maximized, as follows.

1. The optimum-value function D(i, j) is defined as the maximum probability between t(1:i) and r(1:j), where t(1:i) the feature vectors of frame 1 to i, r(1:j) is the DHMM formed by state 1 to j. The optimum mapping path is from (1, 1) to (i, j).
2. The recursion of D(i, j) can be expressed as follows:

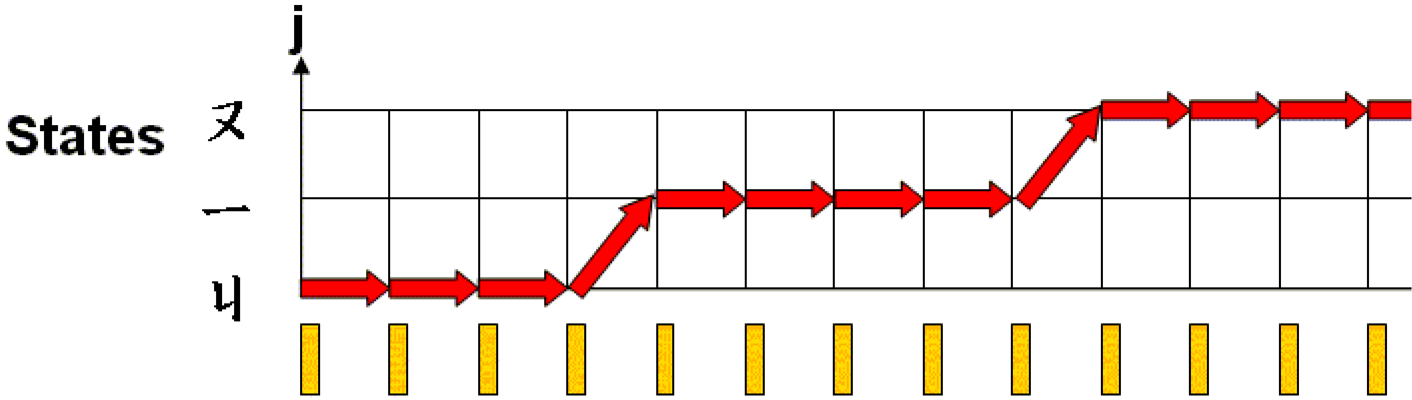
$$D(i, j) = B(O(i), j) + \max\{D(i-1, j)+A(j, j), D(i-1, j-1)+A(j-1, j)\}$$

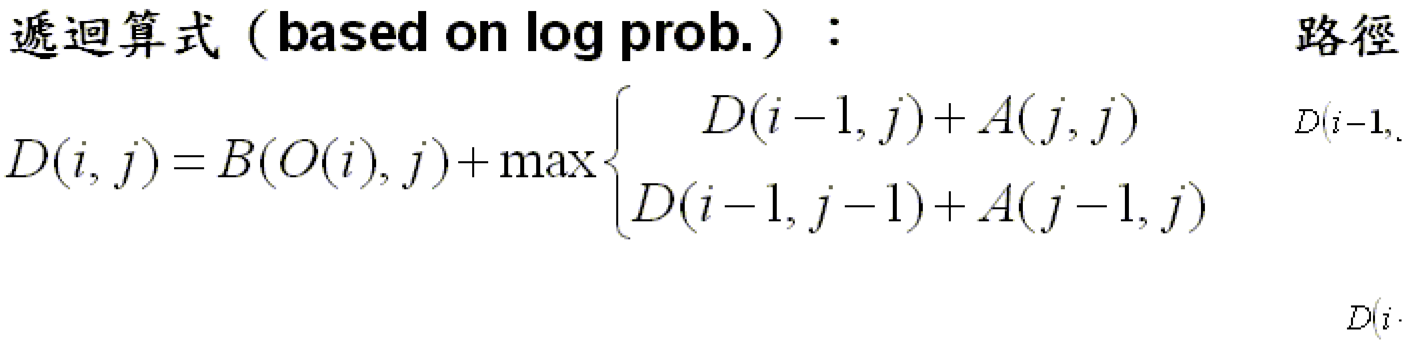
The boundary conditions is

$$D(1, j) = \pi(1, j) + B(O(1), j), j = 1 \sim \text{stateNum}$$

3. The final answer is D(m, n).

The schematic diagram is shown next:





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

$D(i-1,$

$D(i$

Note that in the above expression, we are using log probabilities for the following reasons:

- Reduce the round-off errors due to repeated multiplications.
- Use addition instead of multiplication to reduce computation load.

Based on Viterbi decoding, we can find the optimum assignment of frames to states as well as the accumulated log probability of a given utterance to a DHMM. A larger log probability indicates the utterance is likely to be the voice command of the corresponding DHMM.

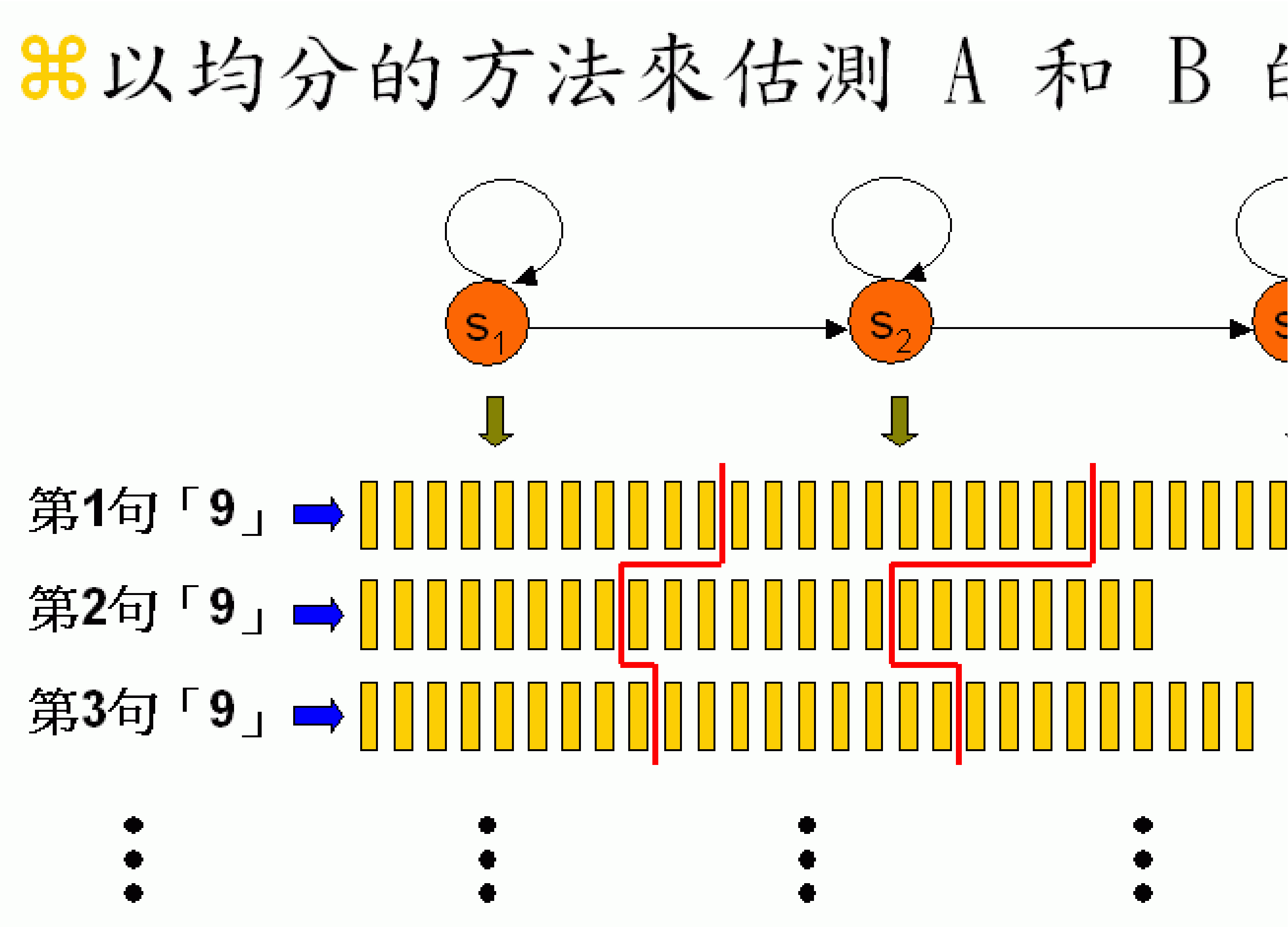
To evaluate the probability of frame i to state j, there are two different methods:

- For DHMM, we need to find the corresponding symbol O(i) and then look up the value of B(O(i), j) from matrix B.
- For CHMM, B(O(i),j) is defined by a continuous probability density function. Please refer to the next section for detail.

For a given DHMM, how can we define the optimum A and B? In general, the optimum values of of A and B should generate the maximum total log probability of all training utterance. As a result, we need to apply the concept of MLE to find the values of A and B.

The procedure to find the optimum values of A and B is called **re-estimation**. The basic concept is close to that of k-means (Forgy's method) which identify parameters by Picard iteration. That is, we can guess the values of A and B first, then perform Viterbi decoding, and then identify A and B again based on the given optimum path, until the values of A and B converge. We can prove that during the iterations, the total probability will increase monotonically. However, we cannot guarantee the identify values of A and B is the global optimum values. The steps of re-estimation are explained next.

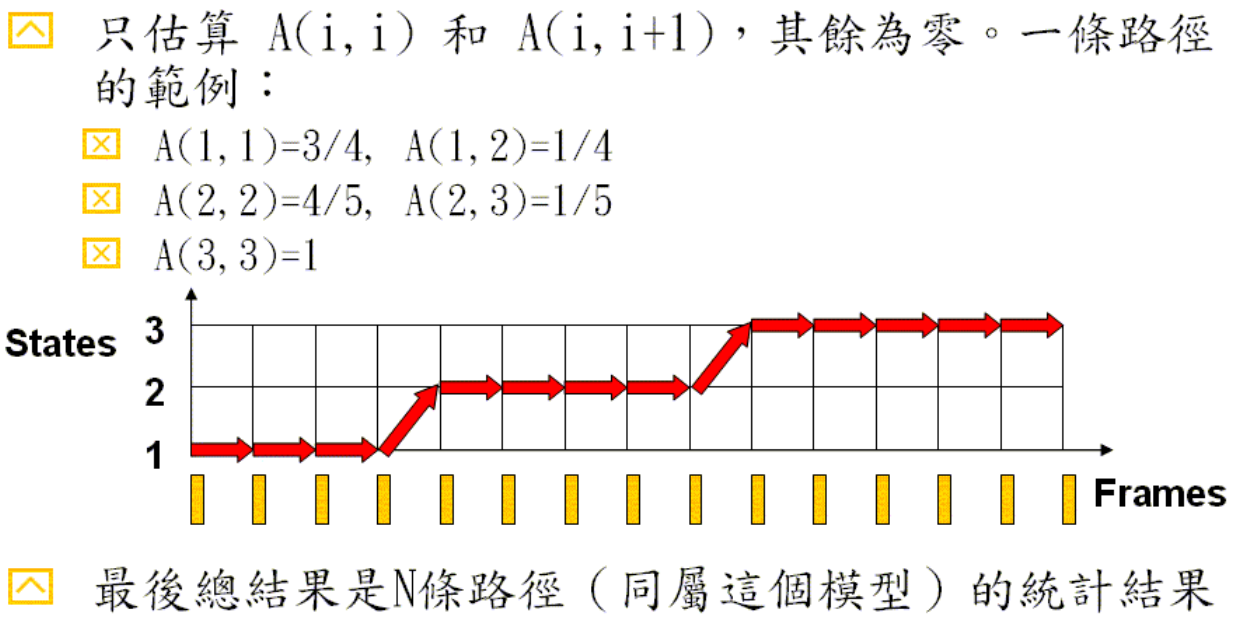
1. Convert all utterances into acoustic features, say, 39-dimensional MFCC.
2. Perform vector quantization on all feature vectors and find the symbol (index of a cluster center) of each feature vector.
3. Guess the initial values of A and B. If there is no manual transcription, we can adopt the simple strategy of "equal division" shown next.



以均分的方法來估測 A 和 B

4. Iterate the following steps until the values of A and B converge.
   - Viterbi decoding: Given A and B of an DHMM, find the optimum mapping paths of all the corresponding utterances of this DHMM.
   - Re-estimation: Use the optimum mapping paths to estimate the values of A and B.

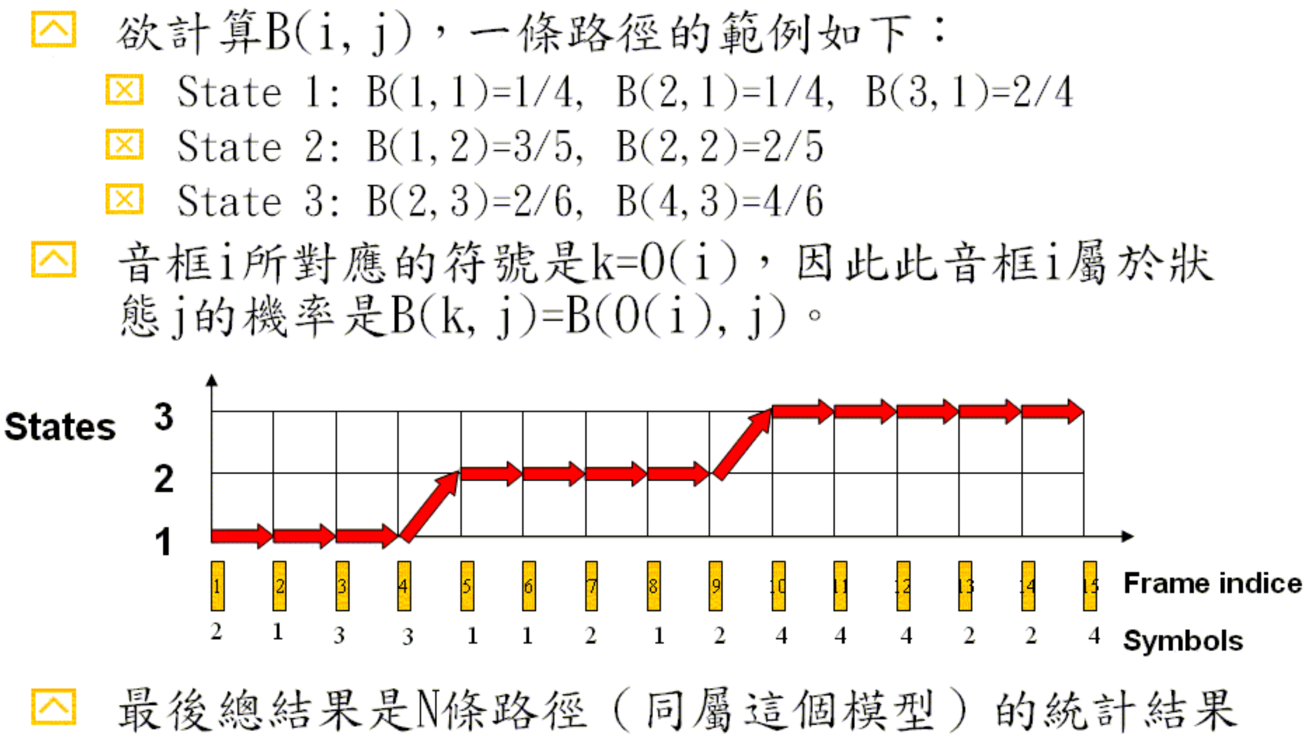
An example of estimating A is shown next:



- 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

An example of estimating B is shown next:



- 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

If we use D(i,j) to denote the probability to which frame i belongs to state j, then D(i,j) = B(O(i),j). Base on the path in the previous example, we can obtain the following values for D(i, j):

- 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

If we use P(A, B, Path) as our objective function, then the above re-estimation can guarantee the monotonic increasing of the objective function, based on the following two facts.

1. When A and B is given, Viterbi decoding can find the optimum mapping path to maximize P(A, B, Path).
2. When the mapping path (Path) is given, re-estimation can find the optimum A and B to maximize P(A, B, Path).

The first fact is obvious since the goal of Viterbi Decoding is to maximize the probability of each path, hence their total P(A, B, Path) will also be maximized. The second fact is also obvious, to be explained next.

Take the above example for instance, the probability of the given path can be decomposed into three components based on each state:

- 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) = $A(1,1)^3$A(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) = $A(2,2)^4$A(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) = $A(3,3)^5$D(10,3)D(11,3)D(12,3)D(13,3)D(14,3)D(15,3)

Since we have N paths for N utterances, the total probability is the product of these N paths. When we are identifying the optimum values of A and B, we need to take these N paths into consideration. We shall decompose the total probability over N path into the probability associated with each state for finding the optimum values of A and B, as follows.

First of all, we shall derive the optimum value of matrix A. For any given state, we can define the following quantities:

- a: self-transition count (known)
- b: next-transition count (known)

- p: self-transition prob. (unknown)
- q: next-transition prob. (unknown)

Since we want to optimize the total transition probability associated with this state, we can form the following optimization problem:

$$\text{Max } J(p, q) = p^a q^b \text{ s.t. } p \geqq 0, q \geqq, \text{ and } p + q = 1$$

We need to apply the theorem that the arithmetic mean is greater than or equal to the geometric mean:

$$x_1 + x_2 + \dots + x_n \geqq n (x_1 x_2 \dots x_n)^{1/n}$$

The equality holds only when $x_1 = x_2 = \dots = x_n$. Based on the above inequality, we have

$$p/a + p/a + \dots + q/b + q/b \dots \geqq (a + b) [(p/a)^a(q/b)^b]^{1/(a+b)}$$

$$1 \geqq (a + b) [(p/a)^a(q/b)^b]^{1/(a+b)}$$

The maximum value of $J(p, q) = p^a q^b$ is $a^a b^b/(a+b)^{a+b}$, which happens at $p/a = q/b$, or equivalently, $p=a/(a+b)$, $q=b/(a+b)$.

Secondly, we shall derive the optimum value of B. For any given state, assume that frames of this state belongs to 3 clusters. We can define the following quantities:

- Symbol count: a, b, c (known)
- State prob: p, q, r (unknown)

Since we want to optimize the total state probability associated with this state, we can form the following optimization problem:

$$J(p, q, r) = p^a q^b r^c$$

where p, q, r must satisfy $p + q + r = 1$. Based on the theorm (the arithmetic mean is greater than or equal to the geometric mean), we can obtain the optimum parameters $p=a/(a+b+c)$, $q=b/(a+b+c)$, $r=c/(a+b+c)$.

Therefore by decomposing the total probability of N paths into the probability associated with each state, we can identify the optimum values of the transition and state probabilities of this state. This complete the coverage of re-estimation which can guarantee the monotonic increasing of the total log probability.

- Slide for 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(·,·) 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(‧,‧) 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 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

16-1 Introduction (簡介)

Old Chinese version

This chapter introduces methods for melody recognition. In fact, there are many different types of melody recognition. But in this chapter, we shall focus on melody recognition for query by singing/humming (QBSH), which is an intuitive interface for karaokes when one cannot recall the title or artist of a song to sing. Other applications include intelligent toys, on-the-fly scoring for karaokes, edutainment tools for singing and language learning, and so on.

A QBSH system consists of three essential components:

1. Input data: The system can take two types of inputs:
    a. Acoustic input: This includes singing, humming, or whisling from the users. For such an input, the system needs to compute the corresponding pitch contour, and convert them into note vector (optionally) for further comparison.
    b. Symbolic input: This includes music note representation from the user. Usually this is not a part of QBSH since no singing or humming is involved, but the computation procedure is quite the same, as detailed later.
2. Song database: This is the song database which hosts a collection of songs to be compared with. For simplicity, most of the song database contains symbolic music whose music scores can be extracted reliable. Typical exmaples of symbolic music are MIDI files, which can be classified into two types:
    a. Monophonic music: For a given time, only a single voice of an instrument can be heard.
    b. Polyphonic music: Multiple voices can be heard at the same time. Most pop music is of this type.

    Most of the time, we are using monophonic symbolic music for melody recognition systems.

    Theoretically it is possible to use polyphonic audio music (such as MP3) for constructing the song database. However, it remains a tough and challenging problem to extract the major pitch (from vocals, for instance, for pop songs) from polyphonic audio music reliably. (This can be explained from the analogy of identifying swimming objects on a pond by the observed waveforms coming from two channels.)

3. Methods for comparison: There are at least 3 types of methods for comparing the input pitch representation with songs in the database:
    a. Note-based methods: The input is converted into note vector and compared with each song in the database. This method is efficient since a note vector is much shorter when compared with the corresponding pitch vector. However, note segmentation itself may introduce errors, leading to the degradation in performance. Typical methods of this type include edit distance for music note vectors.
    b. Frame-based methods: The input and database songs are compared in the format of frame-based pitch vectors, where the pitch rate (or frame rate) can be varied from 8 to 64 points per second. The major advantage of this method is effectiveness, at the cost of more computation. Typical methods include linear scaling (LS) and dynamic time warping (DTW, including type-1 and 2).
    c. Hybrid methods: The input is frame-based while the database song is note-based. Typical methods include type-3 DTW and HMM (hidden markov models).

We shall introduces these methods in the subsequent sections.

16-2 Key Transposition (音調移位)

Old Chinese version

Since the query input may have a different key than those used in the song database, we need to perform key transposition before comparison.

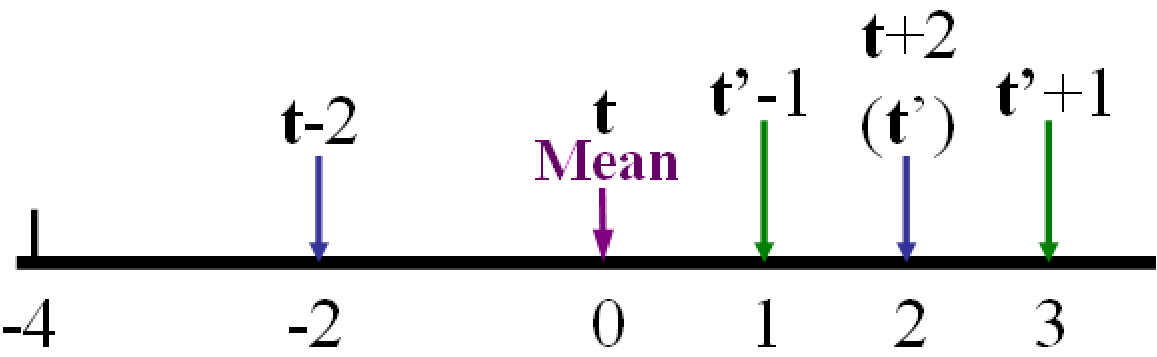
Depending on the methods for comparison, we have different schemes for key transposition, as follows:

- One-shot method: If we are using linear scaling, we can simple shift the pitch vector to the mean or median of the database entry to guarantee the distance is minimized. This is very efficient and requires less computation.

- Trial-and-error method: If we are using type-1 or type-2 DTW, we will not be able to know the best amount for key transposition in an efficient manner. As a result, we need to resort to a certain trial-and-error method to find the best pitch shift amount for key transposition.

If we are using trial-and-error methods, we can actually adopt any methods for one-dimensional function optimization for such a purpose. Two commonly used methods are explained next.

- Linear search (exhaustive search):
    1. Shift the mean of the input pitch vector **t** to the same value of the mean of the song in the database. (If the length of the input vector is m, the mean of the database song should be computed based on the same length too.)
    2. Add each element of [-2.0, -1.8, -1.6, ... 1.6, 1.8, 2.0] to **t** and compute the minimum distance between **t** and **r**.
- Binary-like search:
    1. Shift the mean of the input pitch vector **t** to the same value of the mean of the song in the database. (If the length of the input vector is m, the mean of the database song should be computed based on the same length too.)
    2. Set s = 2.
    3. Shift **t** to each element of [-s, 0, s] and compute the distances to **r** to find the minimum distance. Shift **t** according to the minimum distance.
    4. Set s = s/2 and go back to the previous step until it converges. See the following schematic plot for the binary-like search:



We can use the function binarySearch4min.m in the Utility Toolbox for binary-like search of the minimum of a single-input function. For instance, in the following example, we want to find the minimum of the humps function within [0, 1], and restrict the number of function evaluations to be 9:

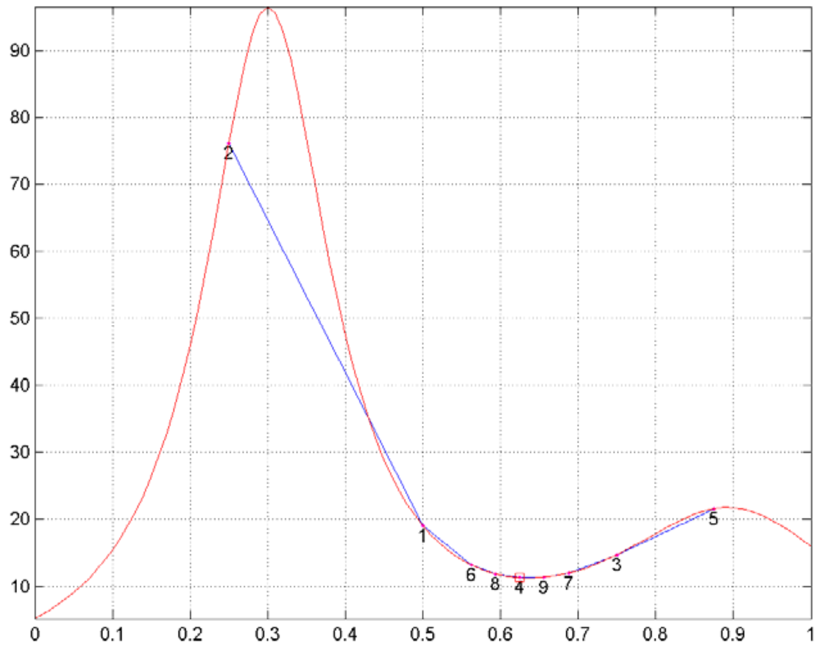
Example 1**Input file** mr/binarySearch4min01.m

```
objFunction='humps';                    % Objective function
optimParam.searchRange=[0.0, 1.0];
optimParam.evalCount=9;
optimParam.plotOpt=1;
[minPair, allPairs]=binarySearch4min(objFunction, optimParam);
fprintf('minX=%f, minY=%f\n', minPair(1), minPair(2));
x=linspace(optimParam.searchRange(1), optimParam.searchRange(2), 101);
y=humps(x);
line(x, y, 'color', 'r'); grid on
set(gca, 'ylim', [min(y), max(y)]);
```

Output message

```
minX=0.625000, minY=11.297297
```

Output figure



In the above plot, each red point indicate a function evaluation. The number beside each red dot indicates the sequence in function evaluation. The identified minimum is indicated by a red square.

In practice, for different applications, the user can adjust these parameters, including the search range and the number of function evaluation.

16-3 Linear Scaling (線性伸縮)

Old Chinese version

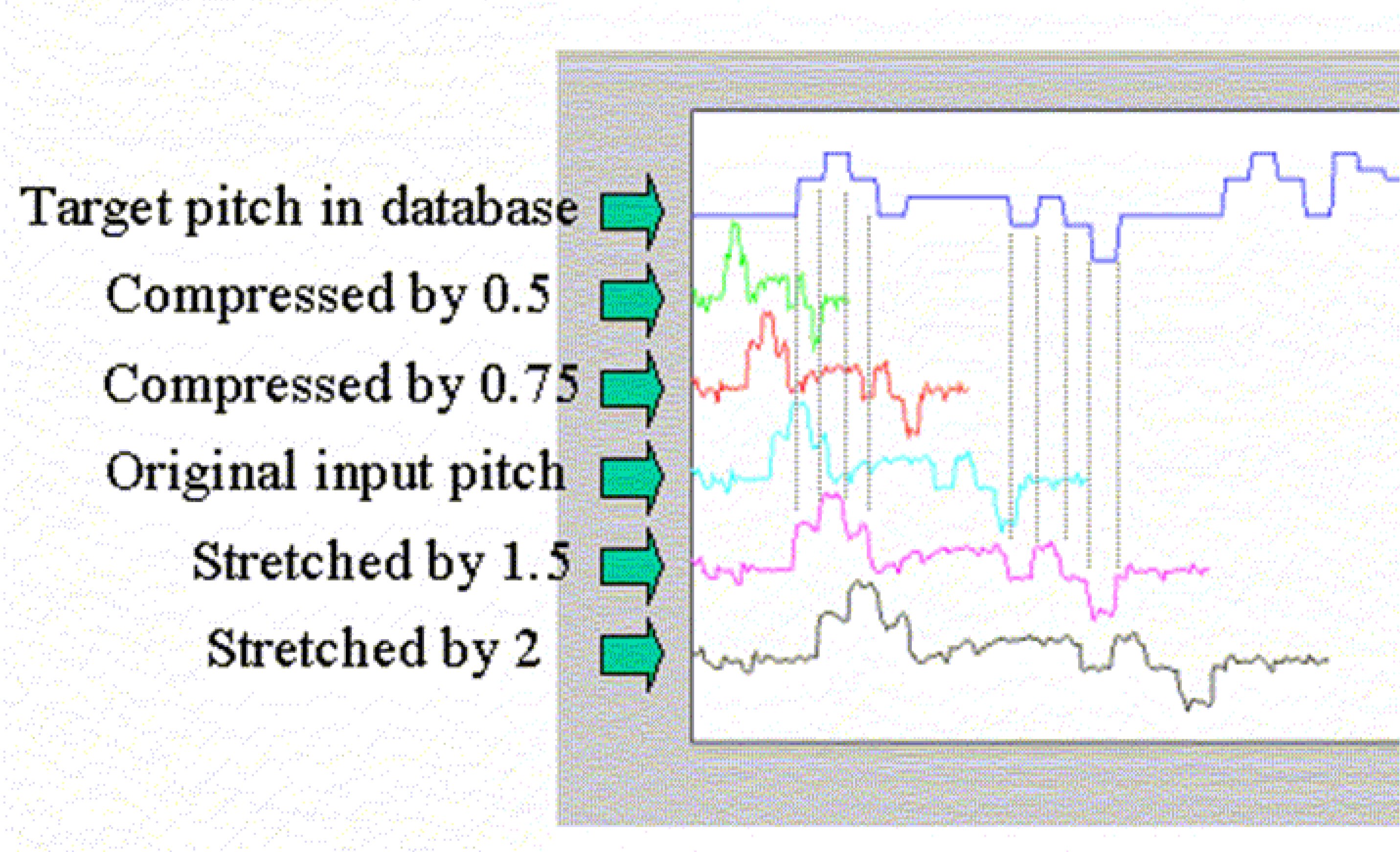
Linear scaling (LS for short), also known as uniform scaling or global scaling, is the most straightforward frame-based method for melody recognition. Typical linear scaling involves the following three steps:

1. Use interpolation to expand or compress the input pitch vector linearly. The **scaling factor**, defined as the length ratio between the scaled and the origianl vectors, could range from 0.5 to 2.0. If we take the step of 0.1 for the scaling factor, it leads to 16

expanded or compressed versions of the original pitch vector. Of course, the **scaling-factor bounds** ([0.5, 2.0] for instance) and the **resolution** (number of scaled pitch vectors, 16 for instance) are all adjustable parameters.

2. Compare these 16 time-scaled versions with each song in the database. The minmum distance is then defined as the distance of the input pitch vector to a song in the database. You can simply use L1 or L2 norm to compute the distance, with appropriate processing for key transposition, to be detailed later.
3. For a given input pitch vector, compute the distances to all songs in the database. The song with the minimum distance is the most likely song for the input pitch vector.

The following plot is a typical example of linear scaling, with a scaling-factor bounds of [0.5, 2] and a resolution of 5. When the scaling factor is 1.5, the minimum distance is achieved.



Simple as it is, in practice, we still need to consider the following detailed issues:

1. Method for interpolation: Simple linear interpolation should suffice. Other advanced interpolations may be tried only if they will not make the computation prohibitive.

2. Distance measures: We can use L_1 norm (the sum of absolute difference of elements, also known as taxicab distance or Manhattan distance) or L_2 norm (square root of the sum of squared difference of elements, also known as Euclidean distance) to compute the distance.

   Hint

   L_p norm of vectors x and y = $[\Sigma|x_i-y_i|^p]^{1/p}$

3. Distance normalization: Usually we need to normalize the distance by the length of the vector to eliminate the biased introduced via expansion or compression.
4. Key transposition: To achieve invariance in key transposition, we need to shift the input pitch vector to achieve a minimum distance when compared to the songs in the database. For different distance measures, we have different schemes for key transposition:
   - For L_1 norm, we can shift the input pitch vector to have the same median as that of each song in the database.
   - For L_2 norm, we can shift the input pitch vector to have the same mean as that of each song in the database.
5. Rest handling: In order to preserve the timing information, we usually replace the rest with previous non-rest pitch for both input pitch vector and songs in the database. One typical example is "Row, Row, Row Your Boat" (original site, local copy).

Characteristics of LS for melody recognition can be summarized as follows:

- If the user's singing or humming is at a constant pace, LS usually gives satisfactory performance.
- LS is also very efficient both in its computation and the one-shot way to handle key transposition.

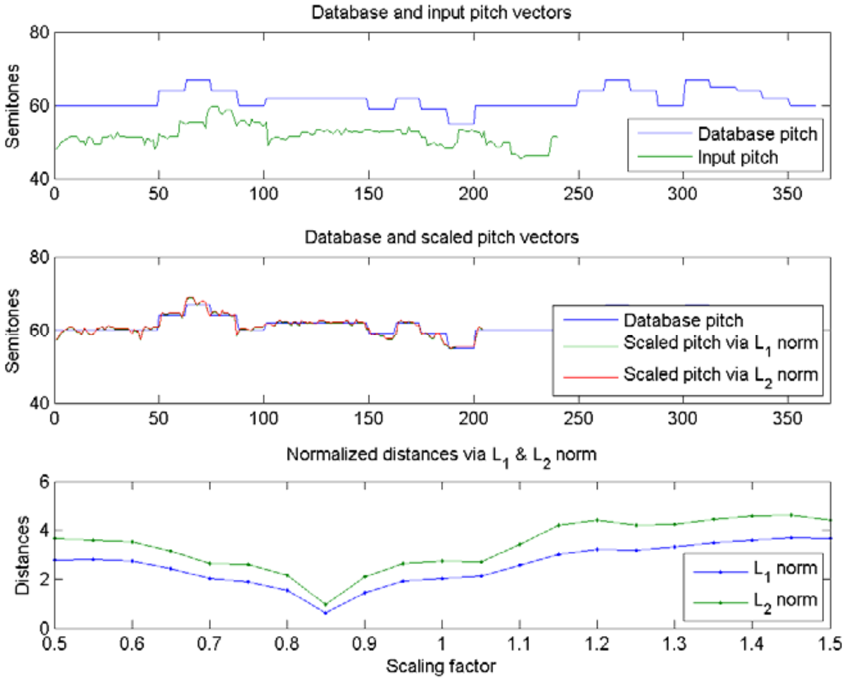
The following example is a typical result of using LS for distance comparison:

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 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

Note that in order to save computation, linScalingMex uses the square of L_2 norm instead of L_2 norm literally. Therefore we need to do square root in order to get the correct distance based on L_2 norm.

16-4 DTW of Type-1 and 2

Old Chinese version

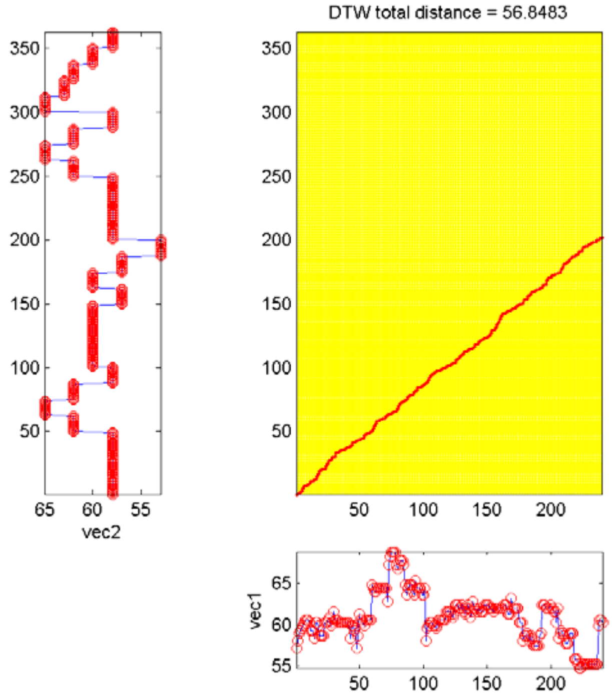
DTW (Dynamic Time Warping) is another commonly used method for frame-based approach to QBSH in melody recognition. The major advantages of DTW is its high recognition rate for QBSH. This is partly due to its robustness in dealing with undesirable pitch vectors, especially for type-1 DTW which allow skipping of sporadic unlikely pitch values. On the other hand, the major disadvantage of DTW is its requirement for massive computation, which is worsen by the fact that there is no one-shot scheme for dealing with key transposition. Since the applications of DTW are quite extensive in numerous different fields, research on speeding up DTW is a rather hot topic.

For technical details of DTW, please refer to the chapter on dynamic programming. In this section, we shall give some examples of using DTW for QBSH. First of all, we can plot the mapping path when comparing a query input with a database entry using type-1 DTW, as follows:

Example 1**Input file** mr/mrDtw1Plot01.m

```
% inputPitch: input pitch vector
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: database pitch vector
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 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];
n=length(inputPitch);
meanPitch=mean(dbPitch(1:n));
inputPitch=inputPitch-mean(inputPitch)+meanPitch;	% Shift input pitch to have the same mean

anchorBeginning=1;	% Anchor beginning
anchorEnd=0;		% Anchor end
m=11;			% Number of pitch shifts for key transposition
pitchStep=linspace(-2, 2, m);
dtwDist=zeros(1, m);	% DTW distances for different pitch shifts
for i=1:length(pitchStep)
	newInputPitch=inputPitch+pitchStep(i);
	dtwDist(i) = dtw1(newInputPitch, dbPitch, anchorBeginning, anchorEnd);
end
[minValue, index]=min(dtwDist);
optInputPitch=inputPitch+pitchStep(index);
[minDist, dtwPath, dtwTable]=dtw1(optInputPitch, dbPitch, anchorBeginning, anchorEnd);
dtwPlot(inputPitch+pitchStep(index), dbPitch, dtwPath);
```

Output figure



In the above example of type-1 DTW, we need to be aware of two issues:

- For QBSH application, we have two types of comparisons:
  a. Comparison from beginning: The acoustic input corresponds to the beginning of a song in the database. This is exactly the case in the above example, so we set anchorBeginning=1 and anchorEnd=0.
  b. Comparison from anywhere: The acoustic input corresponds to the middle of a song. Under such a condition, we need to set anchorBeginning=0 and anchorEnd=0.

  In fact, if we can define the "beginning position" of a query input, then all comparisons can be viewed as "comparison from beginning" with different beginning positions. Several commonly used "beginning positions" are listed next, according to their complexities:

  3. The beginning of a song: This is not a realistic assumption since the most well-known part of a song does not necessarily starts from the beginning.

4. The refrain (chorus or burden, 副歌) of a song: However, refrains of a song cannot be defined objectively.
5. The beginning of each senence: This is most likely to be the case in karaoke systems since such information is already embedded in the songs in order to display the lyrics in a sentence-by-sentence manner.
6. The beginning of each note: This is the final catchall which requires a lot of computation.

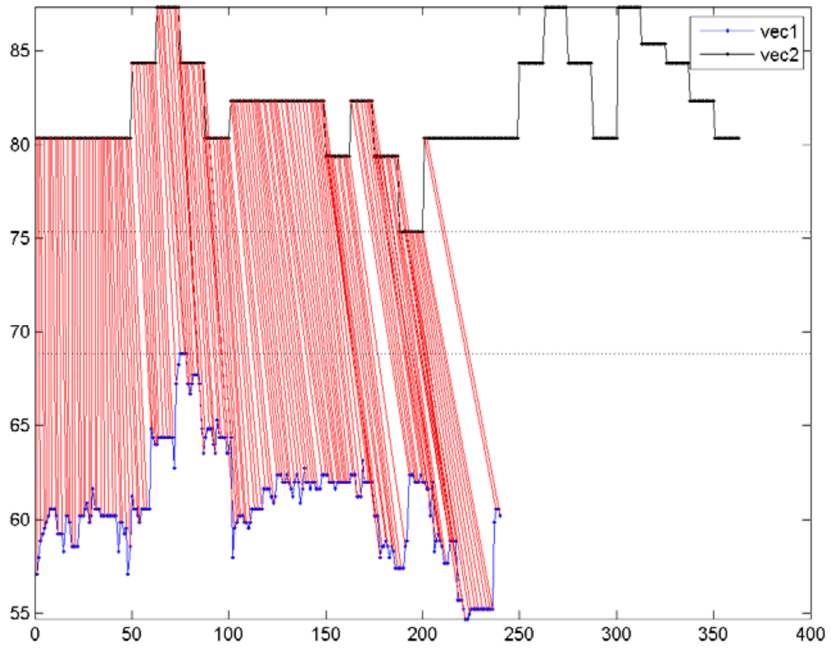
- To deal with key transposition, we use a linear search between the interval of [-2, 2] with a resolution of 11. It is possible to use some sort of heuristic search, such as the one mentioned in section 2 of this chapter.

Besides plotting the mapping path, we can also employ two other methods for displaying the result of DTW, as mentioned in the chapter of dynamic programming. One of them is the mapping between two curves in a 2D plane, as shown next:

Example 2**Input file** mr/mrDtw1Plot02.m

```
% inputPitch: input pitch vector
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: database pitch vector
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 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 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];
n=length(inputPitch);
meanPitch=mean(dbPitch(1:n));
inputPitch=inputPitch-mean(inputPitch)+meanPitch;      % Shift input pitch to have the same mean

anchorBeginning=1;   % Anchor beginning
anchorEnd=0;         % Anchor end
m=11;                % Number of pitch shifts for key transposition
pitchStep=linspace(-2, 2, m);
dtwDist=zeros(1, m);   % DTW distances for different pitch shifts
for i=1:length(pitchStep)
        newInputPitch=inputPitch+pitchStep(i);
        dtwDist(i) = dtw1(newInputPitch, dbPitch, anchorBeginning, anchorEnd);
end
[minValue, index]=min(dtwDist);
optInputPitch=inputPitch+pitchStep(index);
[minDist, dtwPath, dtwTable]=dtw1(optInputPitch, dbPitch, anchorBeginning, anchorEnd);
dtwPlot2(inputPitch+pitchStep(index), dbPitch, dtwPath);
```

Output figure

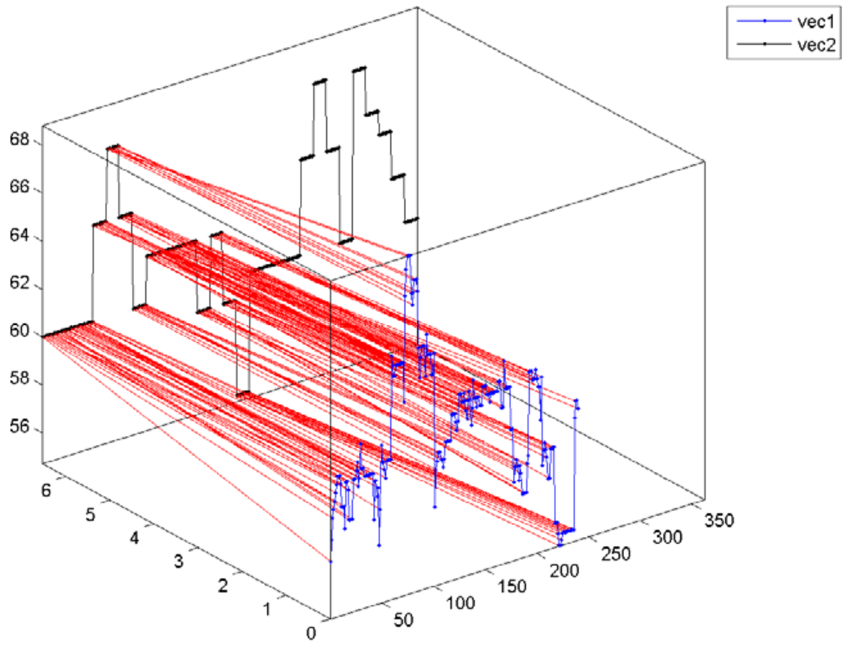


Similarly, we can plot the mapping between two curves in a 3D space, as shown next:

Example 3**Input file** mr/mrDtw1Plot03.m

```
% inputPitch: input pitch vector
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: database pitch vector
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 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];
n=length(inputPitch);
meanPitch=mean(dbPitch(1:n));
inputPitch=inputPitch-mean(inputPitch)+meanPitch;      % Shift input pitch to have the same mean

anchorBeginning=1;   % Anchor beginning
anchorEnd=0;                        % Anchor end
m=11;                               % Number of pitch shifts for key transposition
pitchStep=linspace(-2, 2, m);
dtwDist=zeros(1, m);  % DTW distances for different pitch shifts
for i=1:length(pitchStep)
        newInputPitch=inputPitch+pitchStep(i);
        dtwDist(i) = dtw1(newInputPitch, dbPitch, anchorBeginning, anchorEnd);
end
[minValue, index]=min(dtwDist);
optInputPitch=inputPitch+pitchStep(index);
[minDist, dtwPath, dtwTable]=dtw1(optInputPitch, dbPitch, anchorBeginning, anchorEnd);
dtwPlot3(inputPitch+pitchStep(index), dbPitch, dtwPath);
```

Output figure



Hint

If you try the above example in MATLAB, you can actually rotate the plot to get a better view of the mapping between these two curves.

We can simply change "dtw1" to "dtw2" in these examples to obtain the results of type-2 DTW. In practice, the performance of type-1 and type-2 DTW is likely to be data dependent. There is no guarantee as which one is the clear winner for different data sets.

As long as we have grasped the characteristics of DP, we can devise tailored comparison methods suitable for different scenarios of melody recognition.

16-5 DTW of Type-3

Old Chinese version

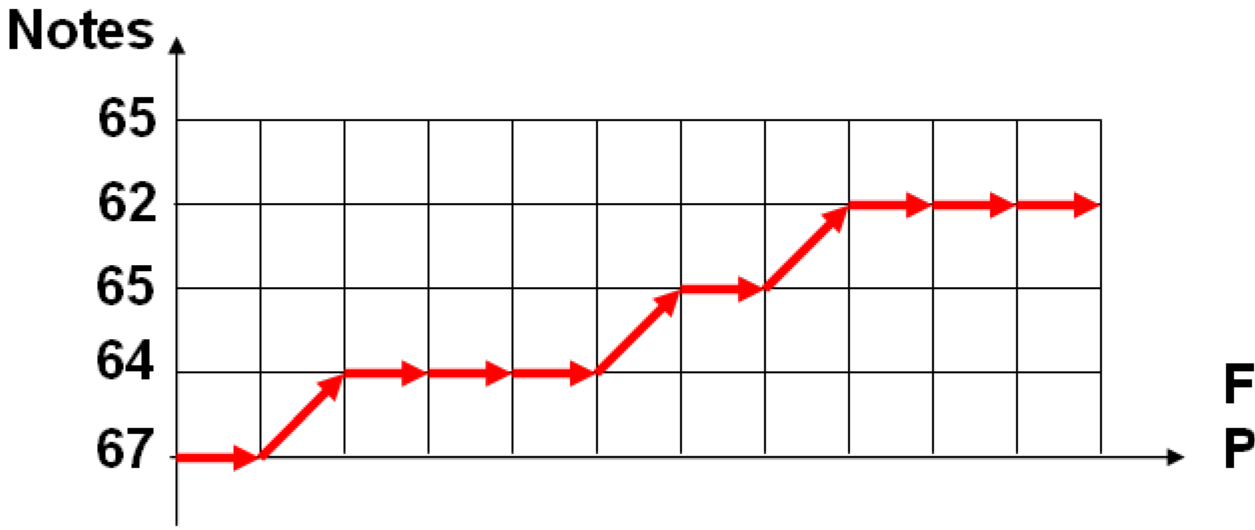
Once we grasp the principle of DP, we can modify DTW for our needs. In this section, we shall introduce another version of DTW with the following charactersitics:

- Query input: This is a frame-based pitch vector. (For our previous task on pitch labeling, the corresponding time unit for each pitch point is 256/8000 = 1/31.25 = 0.032 s = 32 ms.)
- Reference song: This is a note-based pitch vector in which the note duration is discarded for simplicity.

Let **t** be the input query vector and **r** be the reference vector. The optimum-value function D(i, j), defined as the mininum distance between **t**(1:i) and **r**(1:j), can be expressed in the following recursion:

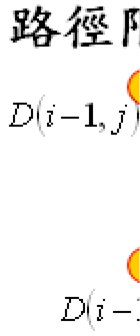
D(i, j) = min(D(i-1,j), D(i-1, j-1))+|**t**(i)-**r**(j)|

Please refer to the following figure:



遞迴算式（**based on log prob.**）：

$$D(i, j) = | t(i) - r(j) | + \min \begin{cases} D(i-1, j) \\ D(i-1, j-1) \end{cases}$$



For simplicity, we shall refer to DTW of this type as type-3 DTW, with the following characteristics:

- Type-3 DTW computes the distance between a frame-based pitch vector and a note-based pitch vector. Therefore the computational complexity is lower than those of type-1 and type-2 DTW>

- The note duration is not used in the note-based pitch vector for comparison. Hence the recognition rate should not be as good as type-1 and type-2 DTW.
- There is no method for one-shot key transposition for type-3 DTW. So we have to rely on trial-and-error method for key transposition.

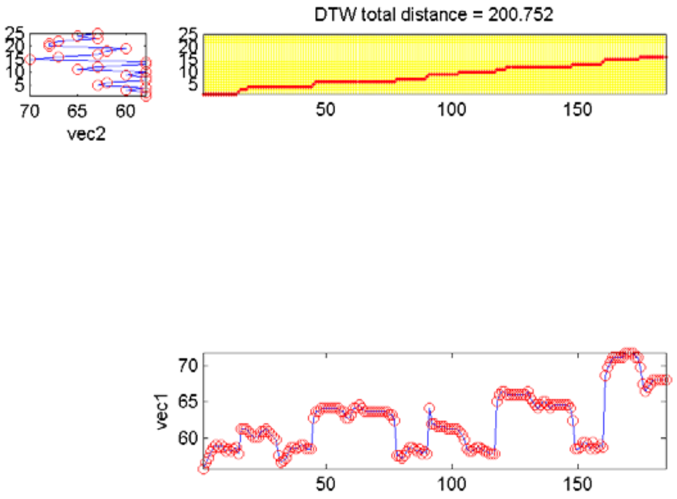
The following is a typical example of using type-3 DTW for melody alignment:

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 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 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



In the above example, before using type-3 DTW, we have performed the following preprocessing:

1. Key transposition: We assume the tempo of the query input is the same as the reference song. Therefore we convert the note into frame-based pitch vector for computing the mean value based on the length of the input query. We then shift the input query to have the same mean of the reference song. We can replace this simplified operation by a more precise method for key transposition.
2. Rest handling: We simply delete all rests in both the input query and the reference song. Again, this is a simplified operation which can be replaced by a more delicate procedure for rest handling.

After the alignment of type-3 DTW in the above example, we can plot the original input PV, shifted PV, and the induced PV, as follows:

Example 2**Input file** mr/dtw3inducedPitch01.m

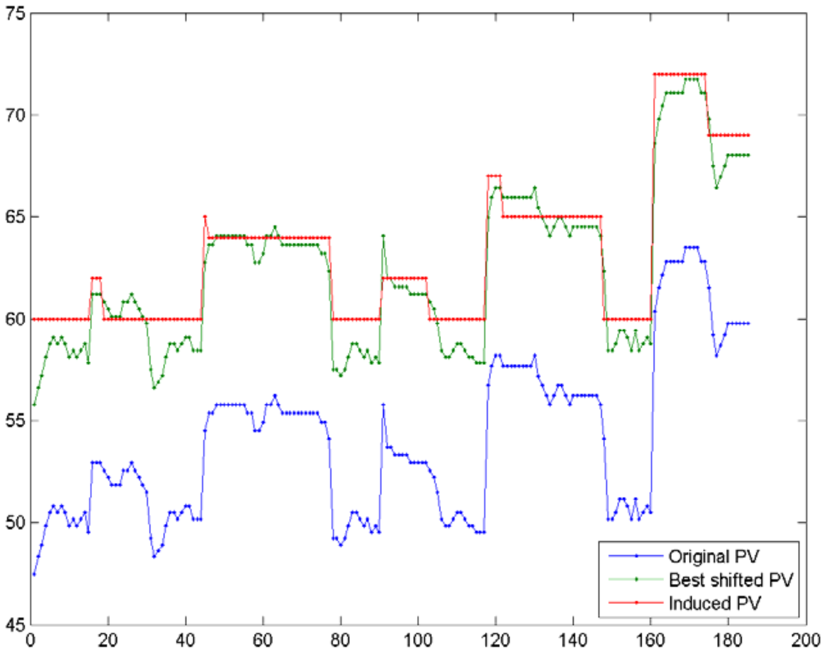
```
pv=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 47.485736 48.330408 48.917323 49.836778 50.478049 50.807818 50.478049
50.807818 50.478049 49.836778 50.154445 49.836778 50.154445 50.478049 49.524836 0 0 52.930351 52.930351
52.930351 52.558029 52.193545 51.836577 51.836577 51.836577 52.558029 52.558029 52.930351 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 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, 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 (刪除休止符)
[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



In the above example, the green line is the origina input PV, the green line is the shifted PV, and the red line is the induced PV. Since the discrepancy between the shifted and induced PVs is still too big, we can conclude that the key transposition is satisfactory. It is likely that the tempo of the query input is not close to that of the reference song. The reference song is "Happy Birthday" and we can hear the related files:

- MIDI file of "Happy Birthday"
- User's singing clip
- Query pitch vector (with rests)
- Query pitch vector (without rests)
- Shifted query pitch vector (without rests)
- Induced notes

If we want to do a better job in the alignment, we need to improve key transposition. A straightforward method is to do a linear (exhaustive) search of 81 comparisons with the range [-2, 2], as shown in the following example:

Example 3**Input file** mr/dtw3inducedPitch02.m

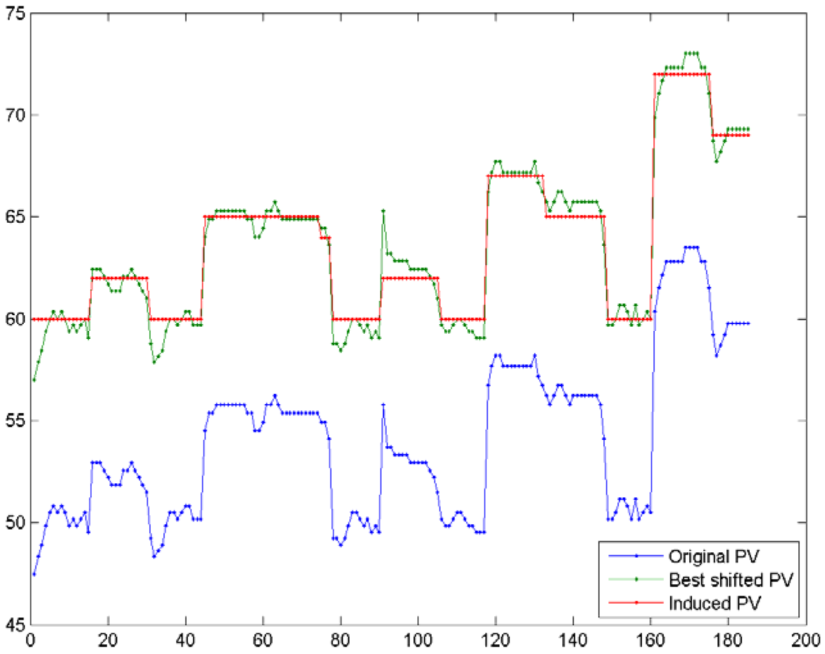
```
pv=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 47.485736 48.330408 48.917323 49.836778 50.478049 50.807818 50.478049
50.807818 50.478049 49.836778 50.154445 49.836778 50.154445 50.478049 49.524836 0 0 52.930351 52.930351
52.930351 52.558029 52.193545 51.836577 51.836577 51.836577 52.558029 52.558029 52.930351 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 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 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



Due to a better key transposition, the alignment of type-3 DTW is improved significantly with a much less DTW distance. The related files are shown next:

- Induced notes

Hint

In general, if we want to perform melody recognition, the exhaustive search for key transposition is impractical due to its excessive computational load. Some heuristic search, such as the binary-like search mentioned in section 2 of this chapter, should be employed instead for such purpose.

In the above example, we can still find some obvious mistake for the alignment. For instance, the fifth induced is too short since it only covers 3 frames. To solve this problem and to improve type-3 DTW in general, we have the following stragegies:

- Set the range of frame numbers being mapped to each note: For instance, the duration of the frames being mapped to a note should between the range [0.5, 2] of the duration of the note. This can enhance the performance since the duration of each note is used.

- Use rests: Except for the leading and trailing rests, any rest in the query input indicates the end of the previous note and the beginning of the next note. We can use this cue for better alignment and query by singing/humming.

We can simply modify our type-3 DTW to meet the above two requirement in order to increase the precision of alignment and the recognition rates of query by singing/humming.

16-6 LCS and Edit Distance

Old Chinese version

We can use note-based approach for melody recognition. First of all, we need to segment the input query pitch vector into music notes. Assume the input pitch vector is represented by pv(i), i=1~n, then the most straightforward method for note segmentation can be described as the pseudo code shown next:

```
for i=1:n
    if |pv(i)-pv(i-1)|>θ & the current note is long enough
        Finish the current note and start a new note
    else
        Add pv(i) to the current note
    end
end
```

The following example demonstrates the use of pv2note.m for note segmentation:

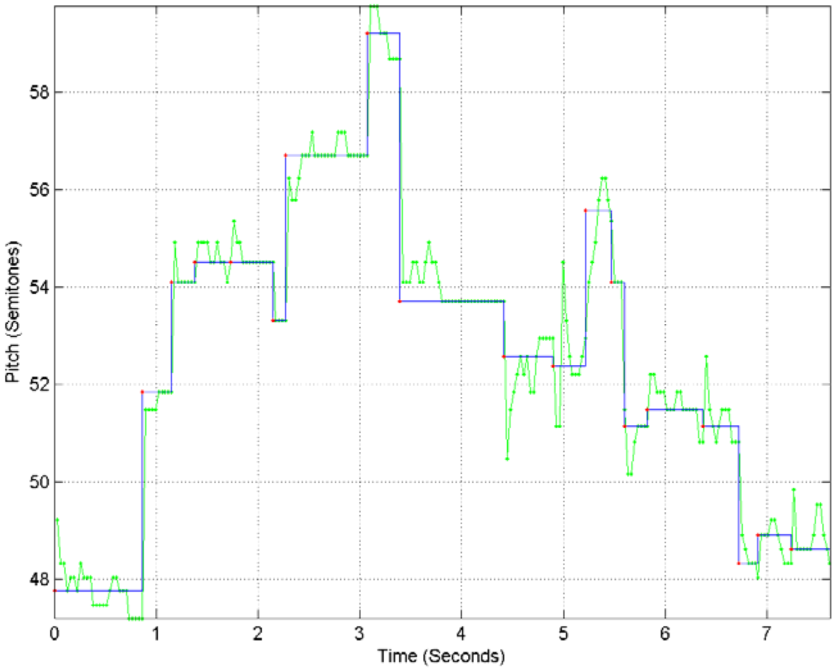
Example 1**Input file** mr/noteSegment01.m

```
pitchVector=[49.21841 48.33041 48.33041 47.76274 48.04425 48.04425 47.76274 48.33041 48.04425 48.04425 48.04425
47.48574 47.48574 47.48574 47.48574 47.48574 47.76274 48.04425 48.04425 47.76274 47.76274 47.76274 47.21309
47.21309 47.21309 47.21309 47.21309 51.48682 51.48682 51.48682 51.48682 51.83658 51.83658 51.83658 51.83658
51.83658 54.92247 54.09792 54.09792 54.09792 54.09792 54.09792 54.09792 54.92247 54.92247 54.92247 54.92247
54.50529 54.50529 54.92247 54.50529 54.50529 54.09792 54.50529 55.34996 54.92247 54.92247 54.50529 54.50529
54.50529 54.50529 54.50529 54.50529 54.50529 54.50529 54.50529 54.50529 53.31086 53.31086 53.31086 53.31086
56.23796 55.78827 55.78827 56.23796 56.69965 56.69965 56.69965 57.17399 56.69965 56.69965 56.69965 56.69965
56.69965 56.69965 56.69965 57.17399 57.17399 57.17399 56.69965 56.69965 56.69965 56.69965 56.69965 56.69965
56.69965 59.76274 59.76274 59.76274 59.21309 59.21309 59.21309 58.68037 58.68037 58.68037 58.68037 54.09792
54.09792 54.09792 54.50529 54.50529 54.09792 54.09792 54.50529 54.92247 54.50529 54.50529 54.09792 53.69992
53.69992 53.69992 53.69992 53.69992 53.69992 53.69992 53.69992 53.69992 53.69992 53.69992 53.69992 53.69992
53.69992 53.69992 53.69992 53.69992 53.69992 53.69992 53.69992 50.47805 51.48682 51.83658 52.19354 52.55803
52.19354 52.55803 51.83658 51.83658 52.55803 52.93035 52.93035 52.93035 52.93035 52.93035 51.14399 51.14399
54.50529 53.31086 52.55803 52.19354 52.19354 52.19354 52.55803 52.93035 54.09792 54.50529 54.92247 55.78827
56.23796 56.23796 55.78827 55.34996 54.09792 54.09792 54.09792 51.48682 50.15444 50.15444 50.80782 51.14399
51.14399 51.14399 51.14399 52.19354 52.19354 51.83658 51.83658 51.83658 51.48682 51.48682 51.48682 51.83658
51.83658 51.48682 51.48682 51.48682 51.48682 51.48682 50.80782 50.80782 52.55803 51.48682 51.14399 50.80782
51.14399 51.48682 51.48682 51.48682 50.80782 50.80782 50.80782 48.91732 48.62138 48.33041 48.33041 48.33041
48.04425 48.91732 48.91732 48.91732 49.21841 49.21841 48.91732 48.62138 48.33041 48.33041 48.33041 49.83678
48.62138 48.62138 48.62138 48.62138 48.62138 48.91732 49.52484 49.52484 48.91732 48.62138 48.33041];
timeStep=256/8000;
pitchTh=0.8;
minNoteDuration=0.1;
plotOpt=1;
note=pv2note(pitchVector, timeStep, pitchTh, minNoteDuration, plotOpt);
fprintf('Hit return to hear the pitch vector...'); pause; fprintf('\n');
pvPlay(pitchVector, timeStep);
fprintf('Hit return to hear the segmented note...'); pause; fprintf('\n');
notePlay(note, 1);
```

Output message

```
Warning: Could not find an exact (case-sensitive) match for 'noteplot'.
d:\users\jang\matlab\toolbox\audioProcessing\notePlot.m is a case-insensitive match
and will be used instead.
You can improve the performance of your code by using exact
name matches and we therefore recommend that you update your
usage accordingly.  Alternatively, you can disable this warning using
warning('off','MATLAB:dispatcher:InexactMatch').
> In pv2note at 42
  In noteSegment01 at 6
  In goWriteOutputFile at 32
Hit return to hear the pitch vector...
Hit return to hear the segmented note...
```

Output figure



In fact, the method implemented by pv2note.m is the simplest way for note segmentation. Possible enhancements include:

1. Finish the current note whenever there is a rest.
2. If the singer has a good absolute pitch, we can round off each note to the nearest integer. On the other hand, if the singer has a good relative pitch, we can shift up and down to find a best shift amount that minimizes the absolute error.
3. We can also employ the concept of DP to find the best way for note segmentation, such that the difference between the original pitch vector and the segmented notes is minimized.

Once we finish note segmentation, we can use the note-based representation for melody recognition, as follows.

In the first method, we only consider the pitch difference between two neighboring notes, regardless of the note's absolute pitch and duration. For instance, if the note pitch is [69, 60, 58, 62, 62, 67, 69], we can convert it into a ternary string [D, D, U, S, U, U] where D (down), U (up) and S (same) are used to described the relationship between two neighboring notes. Once we have convert both the input query and the reference song into two ternary string vectors, we can invoke LCS (longest common subsequence) or ED (edit distance) to compute their distance.

In the second method, we use the note pitch directly, regardless of the note duration. For instance, if the note pitch is [69, 60, 58, 62, 62, 67, 69], we can simply invoke type-1 or type-2 DTW to compute the distance. Since we are using the note pitch directly, we need to perform key transposition before invoking DTW. On the other hand, if we are using the difference of the note pitch for comparison, then we do not need to invoke key transposition.

In the third method, we need to consider both note pitch and note duration. We can still use DTW-like method for comparison, except that we need to consider note pitch and note duration separately. For note pitch, we can take the difference to deal with key variation. For note duration, we can take the ratio of neighboring note duration to take care of tempo variation. We can then invoke DTW that compute the cost function as a weighted average of pitch and duration cost.

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。）

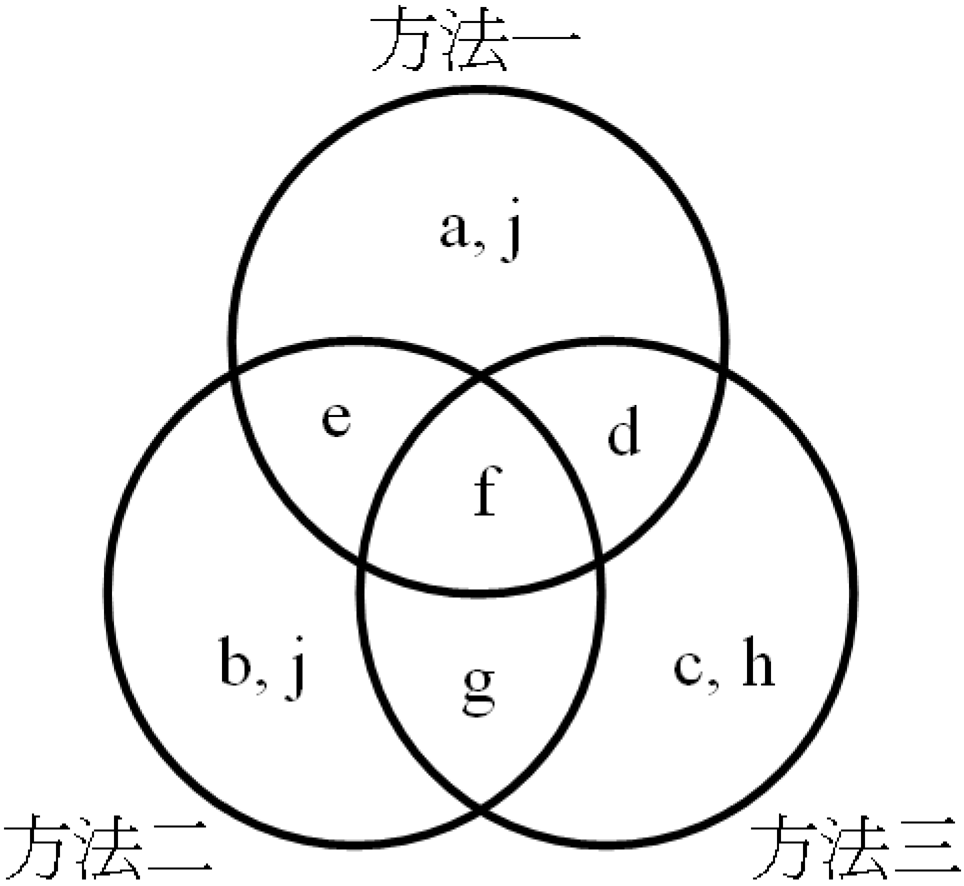
當資料庫的歌曲越來越多時，比對所花的時間也會越來越慢，但對於一個實際應用的系統而言，等待時間應該以不超過五秒為原則，因此我們必須在有限的時間內，找出各種辨識方法的組合，

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。

第 16 章作業

Old Chinese version

1. (*) **Median for minimizing L-1 norm**： For a given vector **x** = [x_1, ... x_n], prove that the median of elements in **x** can minimize the following objective function based on L-1 norm:

$$\text{Median of } \mathbf{x} = \arg\min_u \Sigma|x_i-u|$$

2. (*) **Mean for minimizing L-2 norm**: For a given vector **x**= [x_1, ... x_n], prove that the mean of elements in **x** can minimize the following objective function based on L-2 norm:

$$\text{Mean of } \mathbf{x} = \arg\min_u \Sigma(x_i-u)^2$$

3. (**) **Function for for note segmentation**: Write an m-file function myPv2note.m for note segmentation, with the following usage:

   note = myPv2note(pitchVec, timeStep, pitchTh, minNoteDuration)

   where

   - pitchVec: The input pitch vector
   - timeStep: The time for each pitch point
   - pitchTh: Pitch threshold for creating a new note
   - minNoteDuration: The minimum duration of a note
   - note: The output note vector of the format [pitch, duration, pitch, duration, ...], where the units for pitch and duration are semitone and second, respectively.

   Besides using pitch difference, you also need to use rests (zeros in pitchVec) for note segmentation. The duration of a rest note can be shorter than minNoteDuration. How to test your function:

   f. Please use the pv file of "Happy Birthday" for note segmentation. You can hear the original wave clip and the corresponding midi file.
   g. Please use the pv file of "Twinkle Twinkle Little Star" for note segmentation. You can hear the original wave clip and the corresponding midi file.

   Hint:

   - You can start with the pv2note.m in the Melody Recognition Toolbox, which do not take rests into consideration.
   - You can use pvPlay.m to play the pitch vector, and use notePlay.m to play the output notes.

4. (**) **Function for linear scaling**：Pleae write an m-file function mylinearScaling.m for linear scaling, with the following usage:

   [minDist, bestVec1, allDist] = myLinearScaling4mr(pitchVec1, pitchVec2, lowerRatio, upperRatio, resolution, distanceType)

   where

   - pitchVec1: pitch vector of the input query. This is the vector to be expanded or contracted.
   - pitchVec2: pitch vector of the reference song
   - lowerRatio: lower ratio for contraction
   - upperRatio: upper ratio for expansion
   - resolution: total number of contraction or expansion
   - distanceType: distnace type, 1 for L_1 norm, 2 for L_2 norm.
   - minDist: mininum distance of the linear scaling
   - bestVec1: the pitch vector of the input query after shifting, expansion/contraction that can achieve the mininum distance
   - allDist: all distances of linear scaling

   Note that:

   - If the expanded pitchVec1 is greater than the length of pitchVec2, then stop doing linear scaling.
   - For L-1 norm, we need to do median justification. For L-2 norm, we need to do mean justification.
   - You need to perform distance normalization.
   - You can assume the input vectors do not contain rests.

   How to test your program:

   - Please run this example this example, but replace linScalingMex with myLinearScaling to see if you can get the same result.

- Use your function in the programming contest of melody recognition to see if you can achieve the same recognition rate.

Hint: You can use "interp1" for interpolation, "median" for computing median, "mean" for computing mean, "norm" for computing L_p norm. If you do not know how to use "interp1", the following is an example:

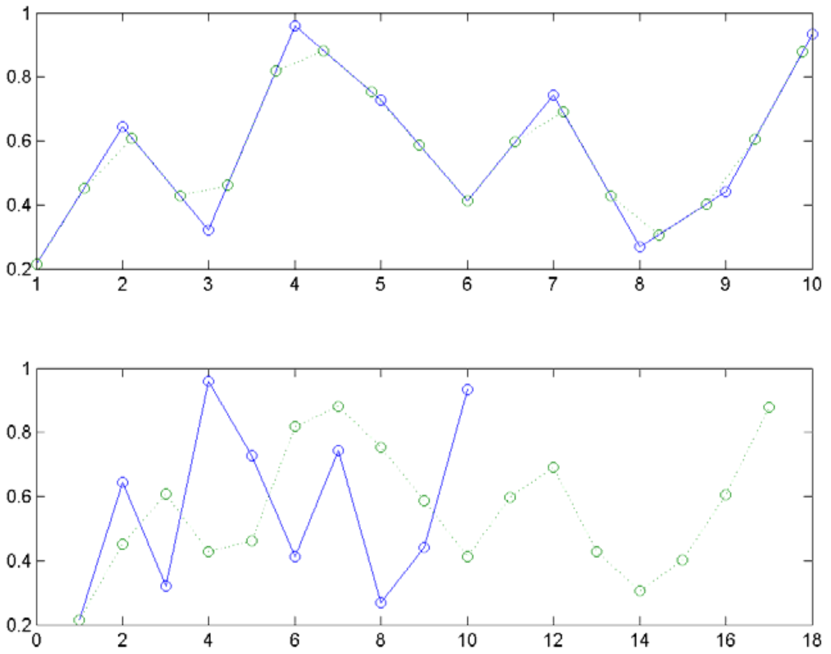
Example 1**Input file** interpDemo01.m

```
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. (***) **Function for type-3 DTW：** Please write a m-file function myDtw3.m for type-3 DTW, with the usage:

[minDist, dtwPath, dtwTable] = myDtw3(vec1, vec2)

You can start from the following skeleton:

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));
```

To test your function:

. Try this example, but replace dtw3mex by myDtw3 to see if you can obtain the same result.

a. Use your function in the programming contest of melody recognition, with 1 key transposition. What is the recognition rate?

b. Use your function in the programming contest of melody recognition, with 5 key transposition. What is the recognition rate?

6. (***) **Methods for improving type-3 DTW：** Write a m-file function myDtw3b.m for type-3 DTW, with the enhancements mentioned at the end of the section covering type-3 DTW:

. Set the range of frame numbers being mapped to each note: For instance, the duration of the frames being mapped to a note should between the range [0.5, 2] of the duration of the note. This can enhance the performance since the duration of each note is used.

a. Use rests: Except for the leading and trailing rests, any rest in the query input indicates the end of the previous note and the beginning of the next note. We can use this cue for better alignment and query by singing/humming.

b. Use both of the above enhancements.

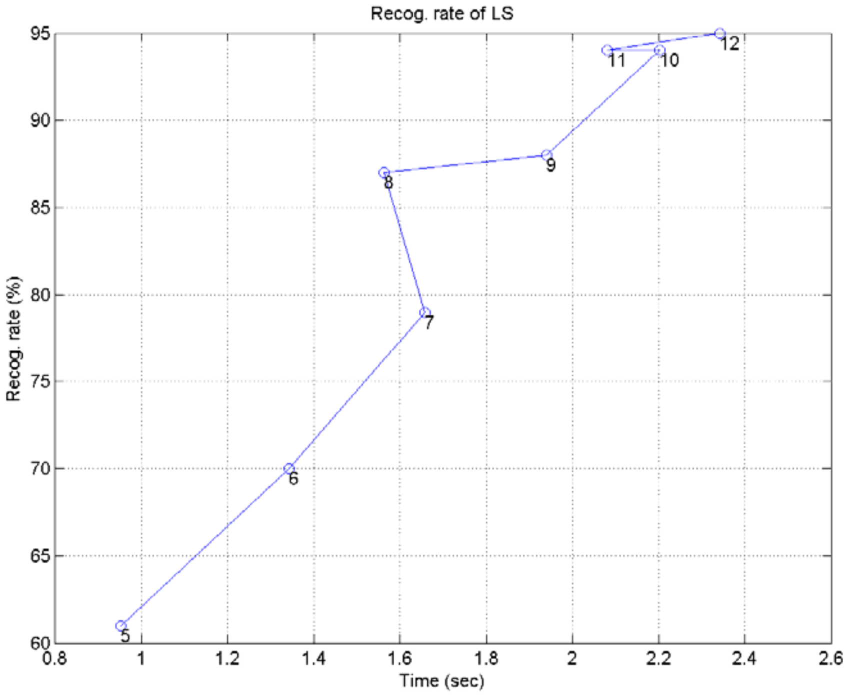
Please test your function as follows:

c. Use your function in the programming contest of melody recognition, with L-1 norm and 1 key transposition. What is the three recognition rates?

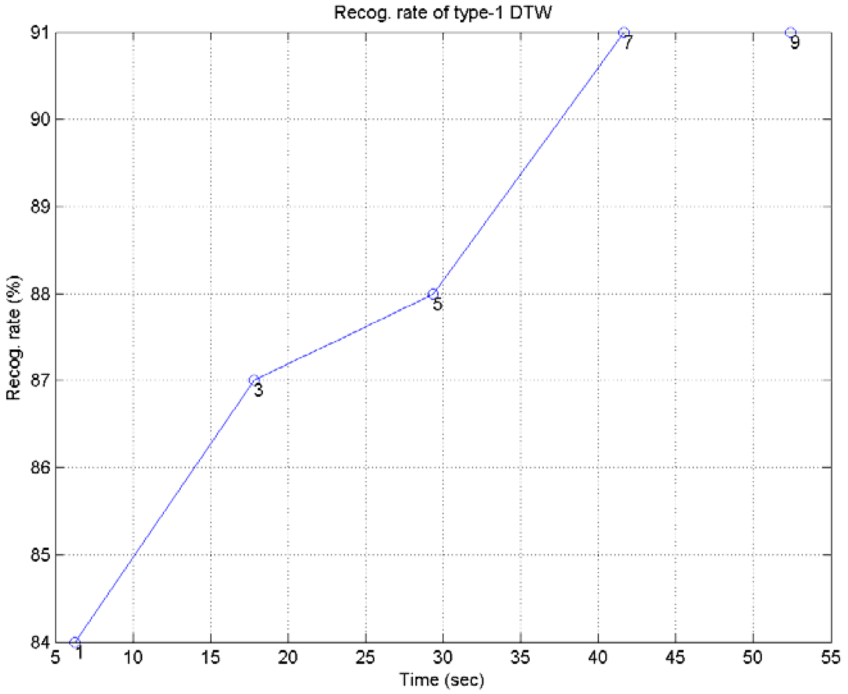
d. Use your function in the programming contest of melody recognition, with L-1 norm and 5 key transposition. What is the three recognition rates?

7. (***) **Melody recognition rates w.r.t. LS resolutions and no. of key transpositions：** Before attempting this exercise, you should first fully understand the example program in the programming contest of melody recognition. You tasks are explained next.

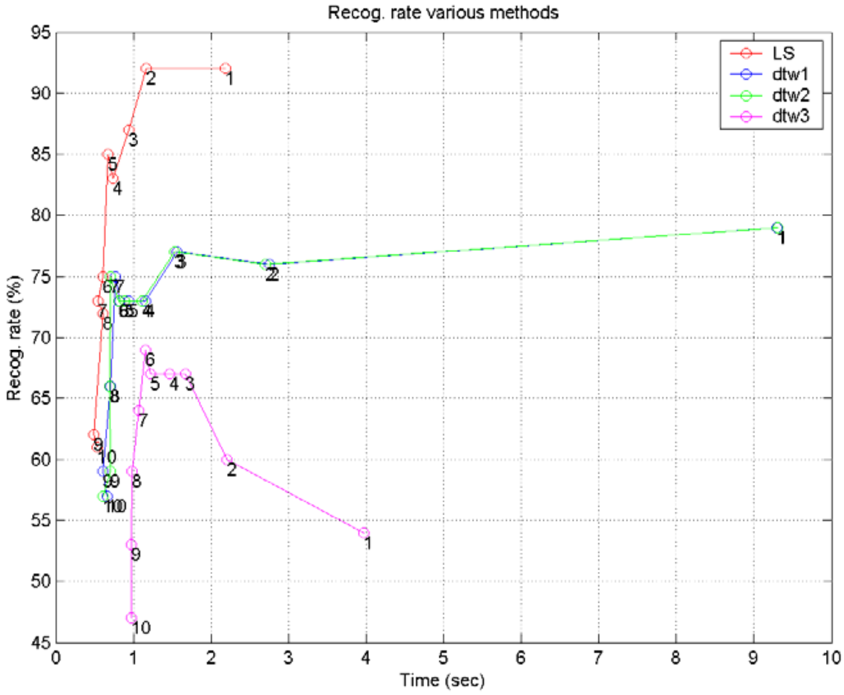
. Use linear scaling for melody recognition, and plot the recognition rate as a function of the computation time, but parameterized by the resolution for linear scaling from 5 to 12. Your plot should be similar to the following one:



a. Use type-1 DTW for melody recognition, and plot the recognition rate as a function of the computation time, but parameterized by the numbers of key transposition of [1, 3, 5, 7, 9]. Your plot should be similar to the following one:



8. Hint:
   - If the computing time is too long, you can use the first 100 wave files for this exercise.
   - It would be easier for you to do the computation and plotting if you modify the example program in the programming contest for melody recogntion into a single function.
9. (***) **Melody recognition rates w.r.t. pitch vector reduction ratio**： Before attempting this exercise, you should first fully understand the example program in the programming contest of melody recognition. The pitch rate of our query pitch vector is fs/frameSize = 8000/256 = 31.25. If the computing time is too long, we can simply down-sample the pitch vector to reduce the size of the DTW table. In other words, we can explore the effect of PVRR (pitch vector reduction ratio) on the recognition rates of various methods. For a given value of pvrr, the pitch rate becomes 31.25/pvrr. You need to plot the recognition rates of 4 methods (LS, DTW1, DTW3, DTW3) with respect to computation time, but parameterized by PVRR which varies from 1 to 10. Your plot should be similiar with the following one:



Hint:

- If the computing time is too long, you can use the first 100 wave files for this exercise.
- It would be easier for you to do the computation and plotting if you modify the example program in the programming contest for melody recogntion into a single function.

10. (***) **Creation of a prototypical QBSH system**: In this exercise, you are going to write a MATLAB script to create a prototypical QBSH system. Basically, the user can sing/hum to the microphone for 8 seconds and the system can list the top-10 ranking of the retrieved songs based on their similarity to the query input. Your system should follow the following steps:

    . Load the song database. This can be achieved by using

    ```
    songDb=songDbRead('childSong', 0);
    ```

    You can check the contents of songDb by typing

    ```
    dispStructInHtml(songDb);
    ```

a. Convert the "track" field of each song to PV representation and attach the PV to a field "pv" of songDb. You can use the command `note2pv`.
b. Do the recording with fs=8000, nbits=8.
c. Perform pitch tracking by using the command `wave2pitchByAcf`. Plot the result of pitch tracking (by supplying appropriate arguments to the command) after each recording to make sure the pitch tracking result is satisfactory.
d. Handle rests by using the command `restHandle`.
e. Compare the query PV to each PV in the database using linear scaling.
f. Display the top-10 retrieved songs based on the LS distances.

You system should be able to let the user sing/hum as many songs as they want until ctrl-c is hit. When you demo your system to TA, make sure you can have at least 3 humming/singing inputs to have the correct answer in top-10 results. (Hint: You need to use the Melody Recognition Toolbox.)

11. (***) **Programming contest: melody recognition**: Please follow the link for more information.

Chapter 17: HTK

17-1 HTK Introduction (HTK 簡介)

Old Chinese version

HTK (Hidden Markov Model Toolkit) is a public-domain software for training HMM (Hidden Markov Models), mostly for the application of automatic speech recognition. Most of the related information can be found at the HTK website:

http://htk.eng.cam.ac.uk/

HTK was originally developed by the Machine Intelligence Lab of Engineering Department at Cambridge University. In 1999, Microsoft bought HTK from its owner Entropic Inc., and made it public-domain open-source software for enhancing the speech technology through collective efforts. As a result, now we can download the source of HTK from its website directly.

The implementation of automatic speech recognition involves advanced techniques of HMM training and evaluation, which are not easily mastered by common programmers. Since the availability of HTK source code, the entry barrier became lower, which advances the research of speech technology rapidly. Currently most of the research labs in unversities and industry are using HTK for their research and development. As a result, HTK is the de facto standard tool for developing automatic speech recognition.

This chapter will use several basic examples to demonstrate the use of HTK. We have tried to keep the examples as simple as possible while not omitting any important features of HTK. After getting familiar with these examples, the users can further consult HTK manual for other advanced features for your own research and development.

17-2 HTK Example: Digit Recognition (HTK 基本範例一：數字辨識)

Old Chinese version

To implement CHMM, usually we rely on HTK (Hidden Markov Model Toolket) for corpus preparation and training. In this section, we shall use a simple example to demonstrate the use of HTK. In this example, we want to construct CHMM for the recognition for the digits from 0 to 9 in Mandarin. You will be able to know how to do corpus training, and how to compute the recognition rates.

In this homepage, we intend to display files of several extensions (mlf, scp, template, cfg, pam, init, hmm, net, grammar, etc) via iframe tags in the web browser directly. To this end, you need to cancel the application associations for these extensions. (Otherwise there will be popup windows to ask if you want to download the files, etc.) You can use the following batch file to cancel the extension-application associations:

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=
```

Please run the above batch file under the DOS prompt. Then you can reload this homepage such that all files can be displayed on within iframe tages in this page correctly.

Hint

You can use the command "assoc" to specify or delete file association with an application. For instance, if you want to delete file association with the extension "scp", you can type "assoc .scp=" under the DOS prompt.

Before move on, you need to download the following files for this section:

- HTK commands: After decompression, there are two directories "bin.win32" and "bin", which contain HTK commands and other data-processing commands, respectively. You need to add these two directories to the system's search path via 「控制台/系統/進階/環境變數/系統變數」. If the decompressed directory is put as c:\htk, then you can also append the search path with the DOS window as follows:

```
set path=%path%;c:\htk\bin.win32;c:\htk\bin
```

- Corpus and training scripts: There are two directories:
  - training: Training scripts.
  - waveFile: Corpus of digits 0 ~ 9 in Mandarin.

  Please place the directory "chineseDigitRecog" in a path without blanks. (Do not put them on the desktop since its path contains blanks.)

Open a DOS window and change directory to chineseDigitRecog/training. Type "goSyl13.bat" in the DOS window to start training and performance evaluation. (You can also execute goSyl13.m within MATLAB to get the same results.) If the command runs smoothly until two confusion matrices are displayed on the screen, then everything is set up correctly.

Before start our corpus training, we need to prepare two files manually. The first file is "digitSyl.pam" which specifies how to decompose the phonetic alphabets of each digit into the corresponding acoustic models. For simplicity, our current approach use each syllable as an acoustic model, as follows:

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
```

The above decomposition is a rough but simple way to define syllable-based acoustic models for this application. More delicate decompositions based on monophones or biphones will be explained later.

Hint

The extension "pam" represents phonetic alphabets to model. This is also referred to as the dictionary file in HTK.

The second file is digitSyl.mlf, which define the contents (in the form of acoustic models) of each utterance, as follows:

Example（htk/chineseDigitRecog/training/digitSyl.mlf）：

In the above file, we use sil for silence. We add leading and trailing sil around ling to indicate the utterance of "0" is surrounded by silence.

Hint

- The extension of "mlf" stands for master label file, which is used to record the contents of each utterance.

- HTK uses feature file names to map to the contents in an mlf file. Hence the feature file names should be unique. (The wave file names do not have to be unique.)

In the following, we shall explain the contents of the MATLAB script goSyl13.m and the DOS batch file goSyl13.bat. Both these files involve three major tasks:

I. Extraction of acoustic features of MFCC.
II. Corpus training based on EM to find the optimum parameters.
III. Performance evaluation based on recognition rate.

We shall introduce these three steps in detail.

I. Acoustic feature extraction of MFCC

1. Create output directories

We need to create 3 directories for holding output files:

- output: For various intermediate output files
- output\feature: For feature files of all utterances.
- output\hmm: For HMM parameters during training

The MATLAB commands are

```
mkdir('output');
mkdir('output/feature');
mkdir('output/hmm');
```

The batch command is:

```
for %%i in (output output\feature output\hmm) do mkdir %%i > nul 2>&1
```

If the directories exist, the batch command will not display any warning messages.

Hint

The batch command listed above is copied directly from goSyl13.bat. If you want to execute the command within DOS prompt, be sure to change %%i to %i. Same for the following discussion.

2. Generate digitSyl.mnl and digitSylPhone.mlf

The MATLAB comand for generating syl2phone.scp is:

```
3.    fid=fopen('output\syl2phone.scp', 'w'); fprintf(fid, 'EX'); fclose(fid);
```

The corresponding batch command is:

```
@echo EX > output\syl2phone.scp
```

The contents of syl2phone.scp are shown next:

原始檔（htk/chineseDigitRecog/training/output/syl2phone.scp）：（灰色區域按兩下即可拷貝）

```
EX
```

The string "EX" represents "expand", which serves to expand syllables into acoustic models to be used with the HTK command "HLED". This command is used to generate digitSyl.mnl and digitSylPhone.mlf, as shown next:

```
HLEd -n output\digitSyl.mnl -d digitSyl.pam -l * -i output\digitSylPhone.mlf output\syl2phone.scp
digitSyl.mlf
```

In the above expression, input files are in blue while output files are in red. The output file digitSyl.mnl lists all the used acoustic models:

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

The extension "mnl" represents model name list.

The file digitSylPhone.mlf contains the results of converting the syllable information in digitSyl.mlf into acoustic models for corpus training, as follows:

Example（htk/chineseDigitRecog/training/output/digitSylPhone.mlf）：

In this example, since we are using syllable-based acoustic models, the contents of digitSylPhone.mlf are the same as those in digitSyl.mlf.

4. Generate wav2fea.scp

Before extracting acoustic features, we need to specify the file mapping between each utterance (with extension .wav) and its corresponding feature file (with extension .fea). This mapping is specified in the file wav2fea.scp, which can be generated by the following MATLAB commands:

```
wavDir='..\waveFile';
waveFiles=recursiveFileList(wavDir, 'wav');
outFile='output\wav2fea.scp';
fid=fopen(outFile, 'w');
for i=1:length(waveFiles)
	wavePath=strrep(waveFiles(i).path, '/', '\');
	[a,b,c,d]=fileparts(wavePath);
	fprintf(fid, '%s\t%s\r\n', wavePath, ['output\feature\', b, '.fea']);
end
fclose(fid);
```

The corresponding batch command is much simpler:

```
(for /f "delims=" %%i in ('dir/s/b wave\*.wav') do @echo %%i output\feature\%%~ni.fea)>
output\wav2fea.scp
```

The contents of wav2fea.scp are shown next:

Example（htk/chineseDigitRecog/training/output/wav2fea.scp）：

From the contents of wave2fea.scp, we know that all the feature files will be put under "output\feature" with a file extension of "fea".

15. Use HCopy.exe for acoustic feature extraction

Now we can use HTK command "HCopy" to generate MFCC feature files for all utterances:

16.
```
HCopy -C mfcc13.cfg -S output\wav2fea.scp
```

In the above expression, mfcc13.cfg is a configuration file which specifies parameters for generating MFCC, with the following contents:

原始檔（htk/chineseDigitRecog/training/mfcc13.cfg）：（灰色區域按兩下即可拷貝）

```
#NATURALREADORDER = TRUE
SOURCEKIND = WAVEFORM
SOURCEFORMAT = WAV
TARGETKIND = MFCC_E
TARGETRATE = 100000.0
WINDOWSIZE = 200000.0
PREEMCOEF = 0.975
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
USEHAMMING = T
DELTAWINDOW = 2
ACCWINDOW= 2
```

The meanings of these parameters can be found in the HTK manual.

II. Corpus training based on EM to find the optimum parameters

1. Generate file lists in trainFea.scp and testFea.scp

We need to generate file lists for training and test sets, with the following MATLAB commands:

2.
```
outFile='output\trainFea.scp';
```

```
fid=fopen(outFile, 'w');
for i=1:460
        wavePath=strrep(waveFiles(i).path, '/', '\');
        [a,b,c,d]=fileparts(wavePath);
        fprintf(fid, '%s\r\n', ['output\feature\', b, '.fea']);
end
fclose(fid);
outFile='output\testFea.scp';
fid=fopen(outFile, 'w');
for i=461:length(waveFiles)
        wavePath=strrep(waveFiles(i).path, '/', '\');
        [a,b,c,d]=fileparts(wavePath);
        fprintf(fid, '%s\r\n', ['output\feature\', b, '.fea']);
end
fclose(fid);
```

From the above program, it is obvious that the first 460 files are for training, while all the others are for test. The corresponding batch commands are:

```
for %%i in (train test) do (
 for /f %%j in (%%i.list) do @echo output\feature\%%j.fea
) > output\%%iFea.scp
```

Note that the above code segment read contents from files train.list and test.list (which are prepared in advance), and generates files trainFea.scp and testFea.scp for corpus training and recognition rate computation, respectively. The contents of trainFea.scp are:

Example（htk/chineseDigitRecog/training/output/trainFea.scp）：

18. Generate HMM template file

For corpus training, we need to generate an HMM template file to specify the model structure, such as how many states in an acoustic model, how many streams in a state, and how many Gaussian components in a stream, and so on. The HTK command is:

```
outMacro.exe P D 3 1 MFCC_E 13 > output\template.hmm
```

where

- P: HMM system type, which is fixed to "P" for the time being.
- D: Types of the covariance matrix, which could be "InvDiagC", "DiagC", or "FullC". The "D" is the above command represents "DiagC", which is the most commonly used setting.
- 3: Number of states for a model
- 1: Indicate each state has 1 stream with 1 Gaussian component. (For example, "5 3" indicates there are 2 streams in a state, with 5 and 3 Gaussian components, respectively.)
- MFCC_E: The acoustic parameters are MFCC and energy.
- 13: Dimension of the feature vector.

If this is done with MATLAB, we need to invoke genTemplateHmmFile.m, as follows:

```
feaType='MFCC_E';
feaDim=13;
outFile='output\template.hmm';
stateNum=3;
mixtureNum=[1];
streamWidth=[13];
genTemplateHmmFile(feaType, feaDim, stateNum, outFile, mixtureNum, streamWidth);
```

The generated template.hmm specifies an HMM of 3 states, with 1 stream per state, and 1 component per stream, with the following contents:

Example（htk/chineseDigitRecog/training/output/template.hmm）：

Since this file is used to specify the structure of HMM, all the parameters are given preset reasonable values. Moreover, the states are given indices from 2 to 4 since the first and last states are dummy in HTK convention.

20. Populate HMM template using all corpus

In the next step, we need to compute the initial HMM parameters from the corpus and put them into template.hmm to generate output\hcompv.hmm. By doing so, we can have a set of parameters (for a single HMM of 3 states) which is a better guess than the preset value in template.hmm. Later on, we should copy the parameters to all of the HMMs for EM training. The following command can populate output\template.hmm to generate output\hcompv.hmm:

```
HCompV -m -o hcompv.hmm -M output -I output\digitSylPhone.mlf -S output\trainFea.scp
output\template.hmm
```

The contents of the generated output\hcompv.hmm are:

Example（htk/chineseDigitRecog/training/output/hcompv.hmm）：

From the contents of output/hcompv.htmm, it can be observed that:

- The transition probabilities are not changed.
- The mean and variance of each Gaussian have been changed to the same values for all components. These values are obtained via MLE (maximum likelihood estimate) based on all corpus.

22. Copy the contents of hcompv.hmm to generate macro.init

In this step, we need to copy the contents of hcompv.hmm to each acoustic model, with the following MATLAB commands:

```
% Read digitSyl.mnl
modelListFile='output\digitSyl.mnl';
models = textread(modelListFile,'%s','delimiter','\n','whitespace','');
% Read hcompv.hmm
hmmFile='output\hcompv.hmm';
fid=fopen(hmmFile, 'r');
contents=fread(fid, inf, 'char');
```

```
contents=char(contents');
fclose(fid);
% Write macro.init
outFile='output\macro.init';
fid=fopen(outFile, 'w');
source='~h "hcompv.hmm"';
for i=1:length(models)
        target=sprintf('~h "%s"', models{i});
        x=strrep(contents, source, target);
        fprintf(fid, '%s', x);
end
fclose(fid);
```

The corresponding DOS batch commands are:

```
 (for /f %%i in (output\digitSyl.mnl) do @sed 's/hcompv.hmm/%%i/g' output\hcompv.hmm) > output\macro.init
```

The generated HMM parameter file is macro.init, with the following contents:

Example（htk/chineseDigitRecog/training/output/macro.init）：

This file contains the HMM parameters of 11 (sil、ling、i、er、san、si、wu、liou、qi、ba、jiou) acoustic models.

42. Use mxup.scp to modify macro.init to generate macro.0

We copy output\macro.init to output\hmm\macro.0 first, and then use HHEd.exe to modify macro.0, with the following MATLAB commands:

```
fid=fopen('output\mxup.scp', 'w'); fprintf(fid, 'MU 3 {*.state[2-4].mix}'); fclose(fid);
copyfile('output/macro.init', 'output/hmm/macro.0');
cmd='HHEd -H output\hmm\macro.0 output\mxup.scp output\digitSyl.mnl';
dos(cmd);
```

The corresponding batch commands are:

```
copy /y output\macro.init output\hmm\macro.0
(@echo MU 3 {*.state[2-4].mix}) > output\mxup.scp
```

```
HHEd -H output\hmm\macro.0 output\mxup.scp output\digitSyl.mnl
```

The contents of mxup.scp are shown next:

原始檔（htk/chineseDigitRecog/training/output/mxup.scp）：（灰色區域按兩下即可拷貝）

```
MU 3 {*.state[2-4].mix}
```

Its function is to increase the number of mixture components (of states 2 ~ 4) from 1 to 3. The contents of the generated macro.0 are:

Example（htk/chineseDigitRecog/training/output/hmm/macro.0）：

From the above contents, we can observe that the variances of the three mixtures of a given state are the same. But their mean vectors are different in order to better cover the dataset.

47. Perform re-estimation to generate macro.1~macro.5

Now we can start training to find the best parameters for each acoustic model, with the MATLAB commands:

```
emCount=5;
for i=1:emCount
          sourceMacro=['output\hmm\macro.', int2str(i-1)];
          targetMacro=['output\hmm\macro.', int2str(i)];
          fprintf('%d/%d:  產生  %s...\n', i, emCount, targetMacro);
          copyfile(sourceMacro, targetMacro);
          cmd=sprintf('HERest -H %s -I output\\digitSylPhone.mlf -S output\\trainFea.scp output\\digitSyl.mnl', targetMacro);
          dos(cmd);
end
```

The corresponding batch commands are:

```
set current=0
:loop
 set /a prev=current
 set /a current+=1
 copy /y output\hmm\macro.%prev% output\hmm\macro.%current%
 set cmd=HERest -H output\hmm\macro.%current% -I output\digitSylPhone.mlf -S output\trainFea.scp
output\digitSyl.mnl
 echo %cmd%
 %cmd%
if not %current%==5 goto :loop
```

In the above commands, we use macro.0 as the initial guess for corpus training to generate macro.1, and then use macro.1 to perform re-estimation to generate macro.2. This re-estimation is repeated five times to generate macro.1 ~ macro.5 in "output\hmm\macro.*".

Hint

Since we do not have phone-level transcription, HTK will use flat start (or equal division) to assign an utterance uniformly to its transcribed sequence of models and states.

III. Performance evaluation based on recognition rate

1. Use digit.grammar to generate digit.net

After corpus training, we need to evaluate the recognition rate based on a test data set. First of all, we need to construct the lexicon net, as follows:

2.
```
 Hparse digit.grammar output\digit.net
```

The contents of grammer.txt are:

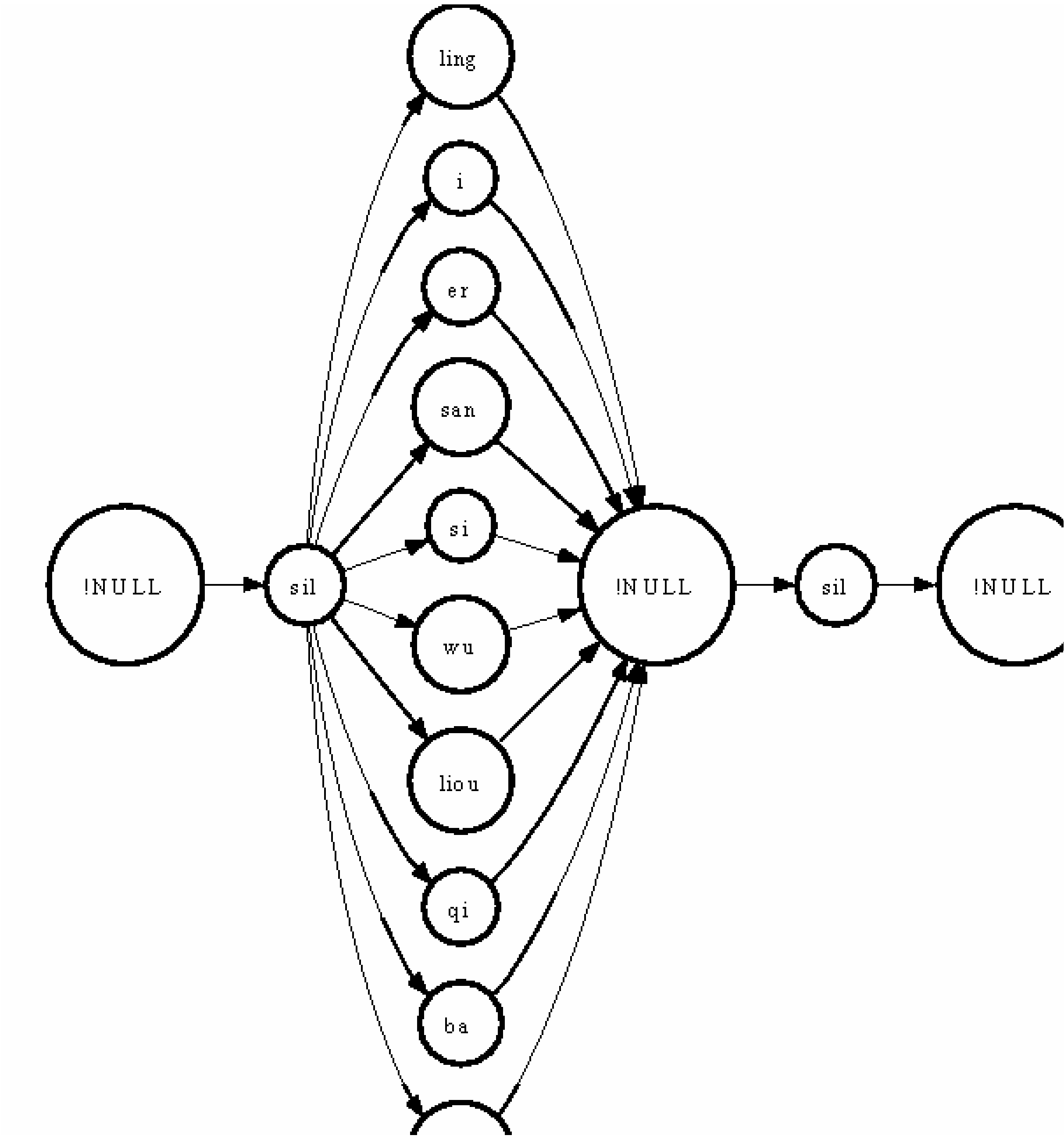
原始檔（htk/chineseDigitRecog/training/digit.grammar）：（灰色區域按兩下即可拷貝）

```
$syl=( ling | i | er | san | si | wu | liou | qi | ba | jiou );
(sil $syl sil)
```

The contents of the generated digit.net are:

Example（htk/chineseDigitRecog/training/output/digit.net）：

The schematic diagram of the net is shown next:



Hint

Note that "!NULL" is a dummy node whose inputs can connected to its fanout directly.

3. <u>Evaluate the recognition rates for both the training and test sets</u>

This is achieved by the following HTK commands:

4.
```
HVite -H output\macro -l * -i output\result_test.mlf -w output\digit.net -S output\testFea.scp
digitSyl.pam output\digitSyl.mnl
```

The contents of the output file result_test.mlf are:

Example（htk/chineseDigitRecog/training/output/result_test.mlf）：

By using a similar command, we can also generate the recognition rate of the training set.

5. Generate the confusion matrices for both inside and outside tests

Finally, we can use the following commands to generate the confusion matrices:

```
findstr /v "sil" output\result_test.mlf > output\result_test_no_sil.mlf
findstr /v "sil" digitSyl.mlf > output\answer.mlf
HResults -p -I output\answer.mlf digitSyl.pam output\result_test_no_sil.mlf > output\outsideTest.txt
type output\outsideTest.txt
```

The confusion matrix for the outside test is:

Example（htk/chineseDigitRecog/training/output/outsideTest.txt）：

Similarly, the confusion matrix for the inside test is:

Example（htk/chineseDigitRecog/training/output/insideTest.txt）：

As usually, the outside test is not as good as the inside test. One possible reason is that the training corpus is not big enough to cover a variety of accents from different individuals. Other possibilities could be the structures of the acoustic models. In the subsequent sections, we shall explore other model structures to improve the performance.

17-3 Digit Recognition: Varying MFCC Dimensions (數字辨識：改變 MFCC 維度)

Old Chinese version

In the previous section, we have demonstrated how to use HTK for Mandarin digit recognition. In this and the following sections, we shall change various settings (such as acoustic features, acoustic model configuration, etc) to improve the recognition rates.

For modularity, we have packed the basic training and test programs into an m-file function htkTrainTest.m. This function takes a structure variable that specifies all the parameters for training, and generates the final test results.

If we keep the configuration of the acoustic models, we can still change the acoustic features. In the previous section, we used a feature type of 13-dimensional MFCC_E. We can now change it to 26-dimensional MFCC_E_D or MFCC_E_D_Z. Furthermore, we can change it to 39-dimensional MFCC_E_D_A or MFCC_E_D_A_Z. For simplicity, we have use the string representations for various feature types, as explained next.

- E: Append energy.
- D: Apply delta operator.
- A: Apply acceleration operator.
- Z: Apply cepstrum mean subtraction (CMS).

The following exmaple uses 26-dimensional MFCC_E_D_Z for recognition:

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%
```

The corresponding batch file is goSyl26.bat.

Furthermore, the following example uses 39-dimensional MFCC_E_D_A_Z:

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%
```

The corresponding batch file is goSyl39.bat.

In the batch files, since we have not pack them into functions, the contents of batch files seem more complicated. But in fact, from goSyl13.bat to goSyl26.bat, only two lines have been changed. You can use the following command to verify their difference:

```
fc goSyl13.bat goSyl26.bat
```

Similarly, you can use the same method to verify the difference between goSyl26.bat and goSyl39.bat.

17-4 Digit Recognition: Changing Acoustic Models (數字辨識：改變 Model 單位)

Old Chinese version

In the previous sections, we use a syllable as an acoustic model. In this section, we shall decompose a syllable into phones and use each phone as an acoustic model. These phones are called **monophones** since they are independent of their following phones. If we have more training data and want to distinguish phone models in a more detailed manner, we can use the so-called **biphones** which is **right-context dependent** (RCD for short).

The new pam file for using monophone acoustic models is digitMonophone.pam, as shown next:

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
```

In fact, we only need to replace digitSyl.pam with digitMonophone.pam, then we can proceed with all the same training and test procedures covered in the previous sections to get the results, as shown in the following example:

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%
```

The generated list of monophones are shown next:

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
```

The corresponding mlf file are shown next:

Example（htk/chineseDigitRecog/training/output/digitMonophone.mlf）：

The following examples uses 26-dimensional MFCC_E_D_Z:

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%
```

The following examples uses 39-dimensional MFCC_E_D_A_Z:

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%
```

17-5 Digit Recognition: Changing MFCC Dimensions and Gaussian Component Numbers (數字辨識：改變 MFCC 維度和 Gaussian 個數)

Old Chinese version

In this section, we shall change both the numbers of Gaussians as well as the dimensions of acoustic feature vectors.

In the next example, we use 13-dimensional MFCC and plot the recognition rates of inside and outside tests as functions of the number of Gaussians:

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);
```

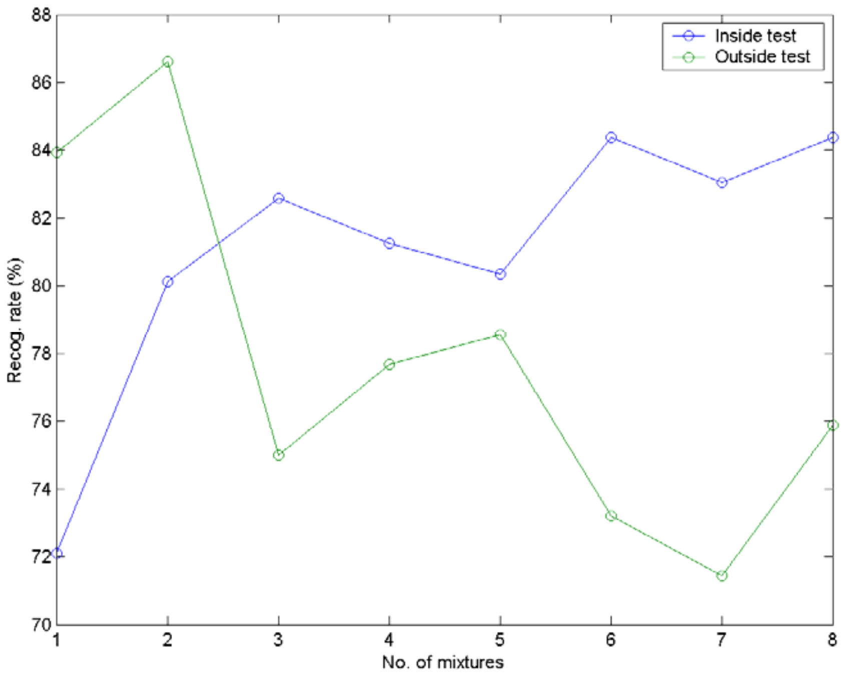
```
end


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



In the above example, we have packed the corpus training and test in an m-file function htkTrainTest.m. Each time this function is invoked, it will do feature extraction. If the feature type is fixed, we can omit this part by changing this function for saving computation time.

We can also use MFCC of dimensions 13, 26, and 39 to plot the recognition rates of inside and outside tests as functions of the number of Gaussian components, as shown in the following example:

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
```

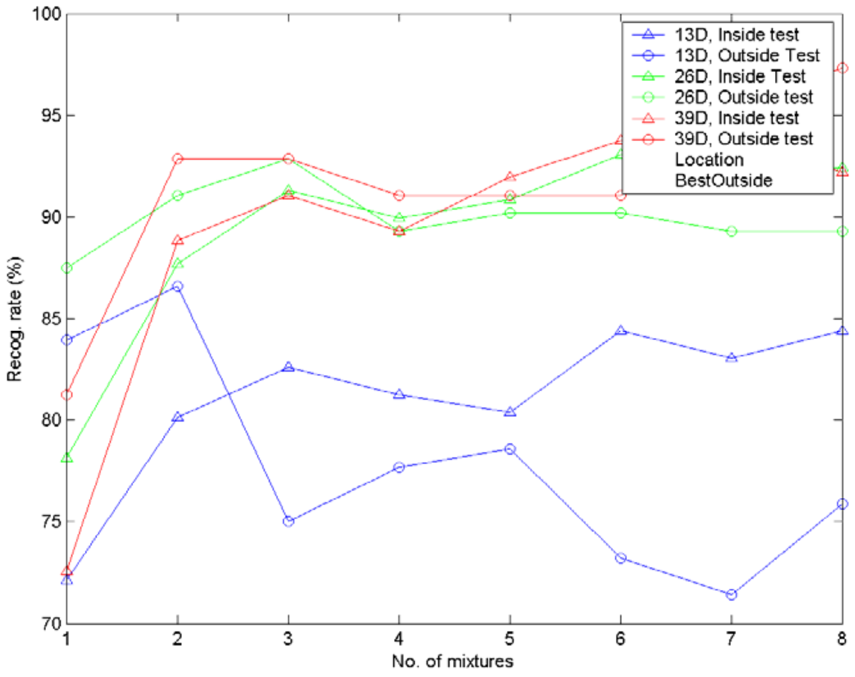
```
Pruning-Off
Pruning-Off
Pruning-Off
====== 8/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 1/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 2/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
====== 3/8
Pruning-Off
Pruning-Off
Pruning-Off
Pruning-Off
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



The above example take a longer time since it involves 24 cases of training and test. (3 dimensions × 8 Gaussians = 24 cases).

第 17 章作業

Old Chinese version

1. (**) **Mandarin Digit Recognition**：Please follow the steps in goSyl13.m to obtain the inside and outside-test recognition rates, where the traing and test copora will be given by TA in the class. The major part of this exercise is to prepare related scripts and parameter files, so your task is condensed to the writing of an m-file script goGenFile4htk.m which collects information of wave files and generates the following files for HTK training:
   a. digitSyl.mlf
   b. wav2fea.scp
   c. trainFea.scp & testFea.scp

   Then you can start training and computing recognition rates. For this part, you need to write an m-file script goHtkTrainTest.m to display the results. In particular, you need to compute the recognition rates for both inside and outside tests, when the dimension of MFCC is set to 13, 26, and 39, respectively. Please show the confusion matrices to TA for your demo. My results are

   - Based on goSyl13.m: inside test 86.38%, outside test 79.51%。
   - Based on goSyl26.m: inside test 92.07%, outside test 87.25%。
   - Based on goSyl39.m: inside test 95.17%, outside test 89.53%。

   Please be aware of the following facts:

   - Feature files cannot take Chinese name since HTK does not support.
   - Every feature file name should be unique, so you need to convert the Chinese directories into numbers, plus the original file names to form a unique name for each feature file. For instance, 912508 鄒銘軒\3a_7436_16017.wav ===> 00002-3a_7436_16017.fea.
   - HTK is case sensitive, so you need to make sure a file name should appear correctly in a file list, and so on.

   Hint: You can use recursiveFileList.m to retrieve all wave files under a given directory.

2. (***) **Programming contest: Mandarin digit recognition**： Repeat the previous exercise by trying all kinds of methods to obtain the maximum performance defined as the average recognition rate of both inside and outside tests. Please record all the related settings (dimension of acoustic feature, unit for acoustic model, number of states, number of streams, number of mixtures, etc) in method.txt, together with the description of your approach. Please upload the following files:
    - The method description file method.txt
    - The final macro file, with file name "final.mac"
    - All the other necessary files for computing recognition rates

   TA will use these files to reproduce your recognition rates.

3. (**) **English letters recognition**： Please repeat the exercise 1, but use the corpus of English letters instead. TA will give the training and test corpora in the class.
4. (**) **Programming contest: English letters recognition**： Please repeat the exercise 2, but use the corpus of English letters instead. TA will give the training and test corpora in the class.

Chapter 18: 語音辨識前處理

18-1

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：

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       ㄓㄤ
長        CaG       ㄔㄤ
門        mrN       ㄇㄣ
阜        fu        ㄈㄨ
...
```

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：

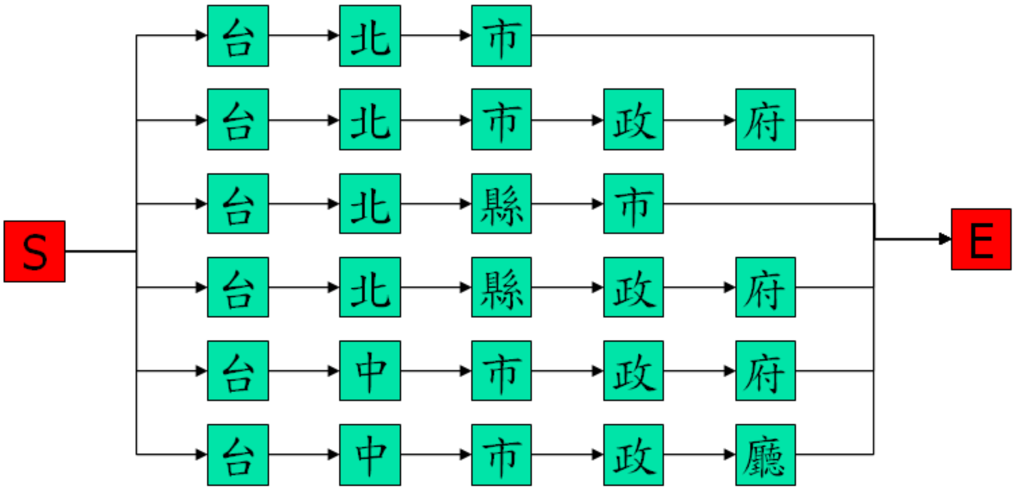
a. Linear Net
b. Tree Net
c. Double-ended Tree Net

以下列可辨識語句為範例：

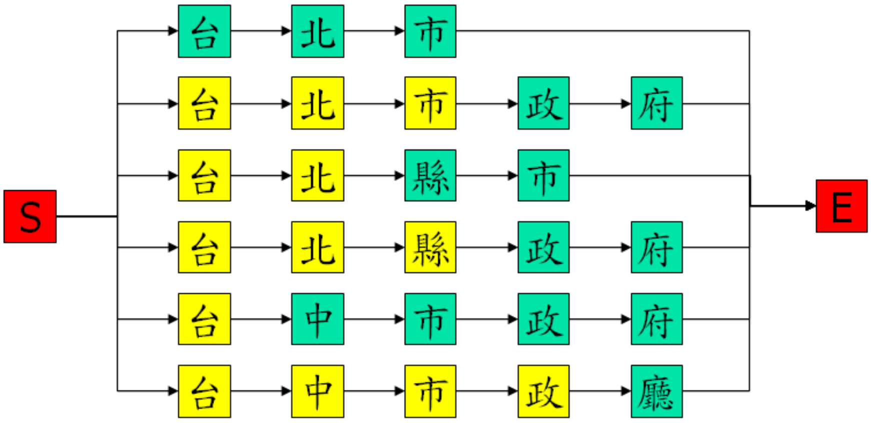
原始檔（政府機關名稱.rt）：（灰色區域按兩下即可拷貝）

```
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
```

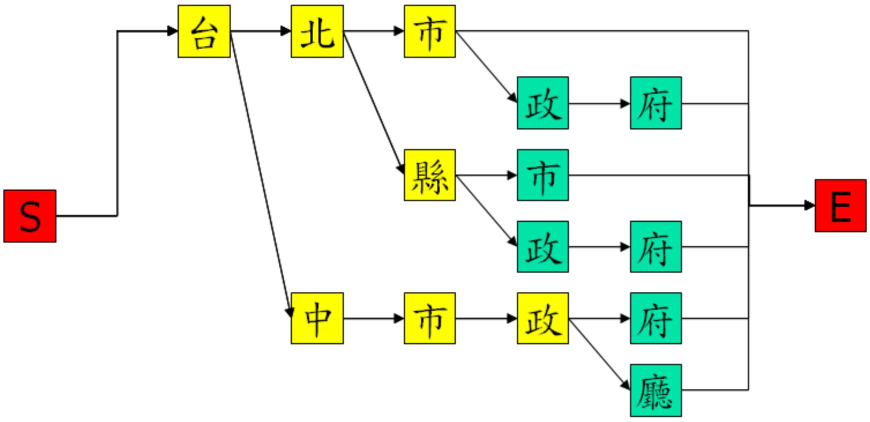
我們可以產生最簡單的 linear net，圖示如下：



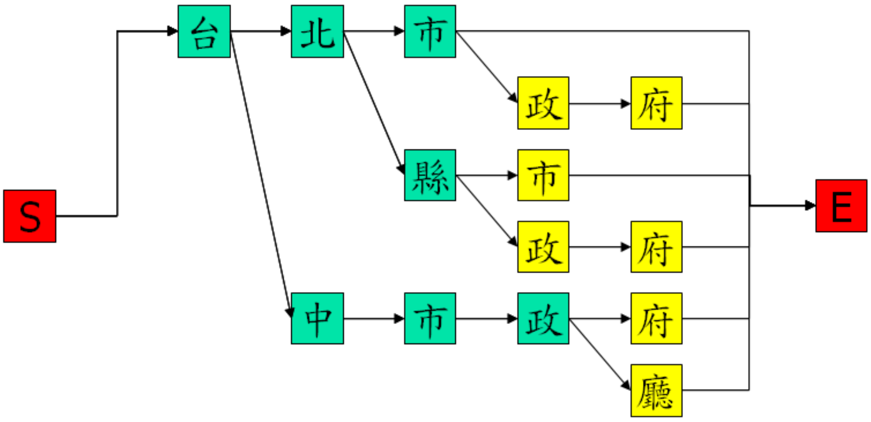
如果將每一條路徑向左對齊，並進行排序，可以找出重複的節點，如下圖之黃色節點：



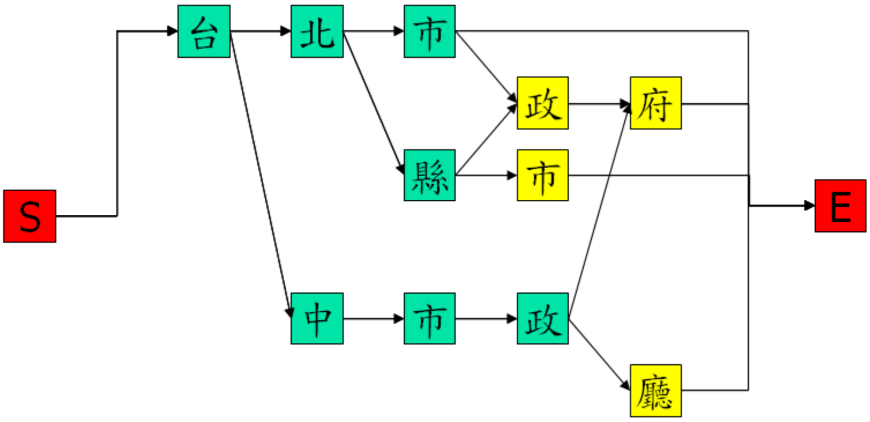
此時我們可以將這些在同一欄且發音相同的黃色節點合併成一個節點，如下：



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。

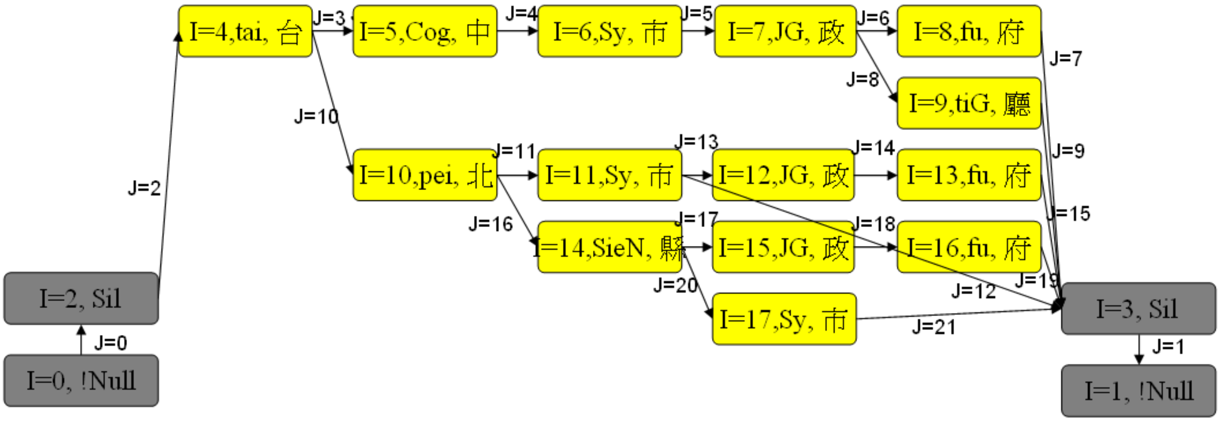
在上述說明中，我們是將 linear net 中的所有路徑向左對齊來進行排序，如果我們改成向右對齊來進行排序，也可以得到另一組 tree net 及 double-end tree net。

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：



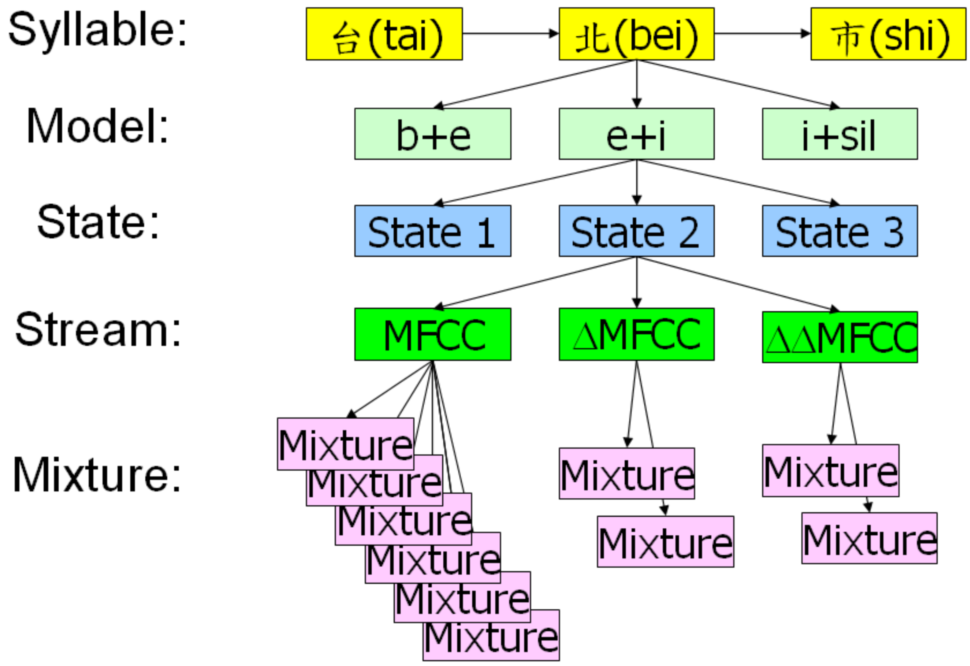
相關投影片請見此連結。

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：

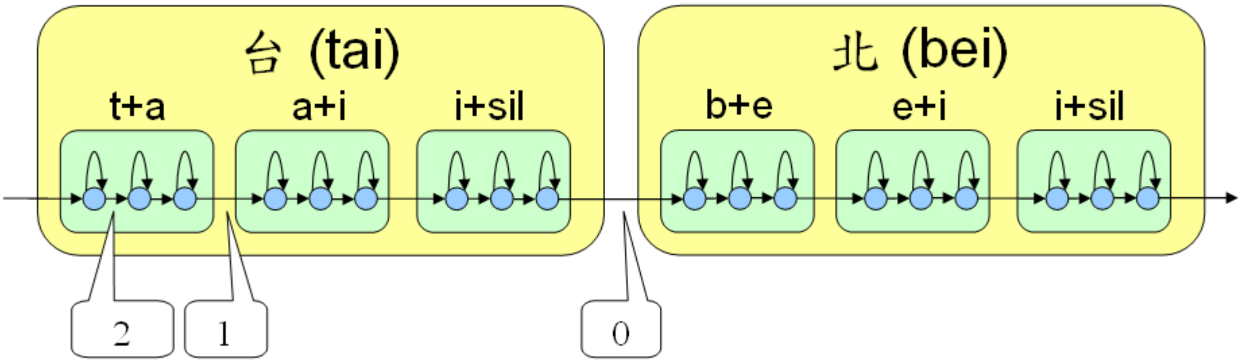
- 聲學模型（Acoustic Model，或簡稱 Model）：使用於 HMM 的一個抽象單位，通常一個聲學模型包含數個狀態。我們可以使用音節或是音素作為一個聲學模型。
- 音節（Syllables）：完整發音的單位，以中文來說，一個字元對應一個音節；以英文來說，一個詞彙可以對應到數個音節，例如 tomorrow 有三個音節。
- 音素（Phoneme）：或簡稱 Phone，是發音的最小單位，例如「大」的發音可以拆解成ㄉ和ㄚ兩個音素，但是音素的拆解並非一成不變，例如碰到滑母音，我們通常就會將一個注音符號拆成兩個音素，例如ㄞ、ㄟ、ㄠ、ㄡ等，這幾個母音在發音過程中，都會呈現連續的變化。
- Monophone：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。