# Speech Recognition에 필요한 Audio 기초

조희철

2020년 10월 11일

### ♠ bit rate & sampling rate

- 파일 크기(byte): (재생시간) x (bit rate) / 8
- 여기서 bit rate(비트전송률)는 초당 얼마나 많은 data를 가지고 있는지를 의미한다. 예를 들어, CD audio는 2-channel, 16 bit(Quantization, bit depth), sampling rate(44100)
- audio file의 비트전송률은 (Windows 10) 파일 속성에서 확인할 수 있다.

$$16 \times 2 \times 44100 = 1,411,200 \text{ bit}$$

- sampling rate: 1초에 들어 있는 data 수.
- bit depth(Quantization): sampling rate이 시간 축인 $x$축을 얼마나 세분하냐를 결정하는 것이라면, bit depth는 $y$축을 얼마나 정밀하게 세분하냐를 결정한다.

```
import sox # sox.exe를 설치후, pip install sox --> https://sourceforge.net/projects/sox/

sox.file_info.sample_rate(audio_filename)
sox.file_info.bitdepth(audio_filename)
sox.file_info.bitrate(audio_filename) # '352k'를 float로 변환하는 과정에 bug가 있다.
```

### ♠ Discrete Fourier Transforms

- Fourier Transform: 음파와 같은 시간에 대한 신호(함수)를 주파수 성분으로 분해할 수 있게 해준다. 주어진 신호를 (서로 다른 주기를 가진) 주기함수들의 결합으로 분해한다. 여기서 사용되는 주기함수는 복소 주기함수($\{e^{-2\pi ik}\}$)들로 구성된다.
- Discrete Fourier Transform(DFT): 컴퓨터에서 처리할 수 있는 data는 이산적이다. 이 이산적인 data에 Fourier Transform을 적용한 것이 Discrete Fourier Transform이다.
- FFT(Fast Fourier Transform): DTF을 좀 더 효율적으로 계산할 수 있게 해주는 알고리즘이다.
- STFT(Short Time Fourier Transform): 우리가 다루어야 하는 음성이나 음악은 하나의 발음이나 음이 계속되는 것이 아니다. 시간에 따라 변한다. 그렇게 때문에 음파를 시간단위로 나누어서 각각에 Fourier Transform을 적용하는 것을 STFT라 한다. 자르는 단위를 frame-length(또는 window-length)라 한다. 그리고 자를때 겹치지 않게 자르지 않고, data의 손실을 막기 위해 겹치는 방식(overlap)으로 자른다. 잘라진 하나의 구간을 frame이라 부른다. 이 frame을 이동시키는 간격을 hop-length라 한다. 당연히, hop-length는 frame-length보다 작아야 frame이 서로 겹치게 된다.

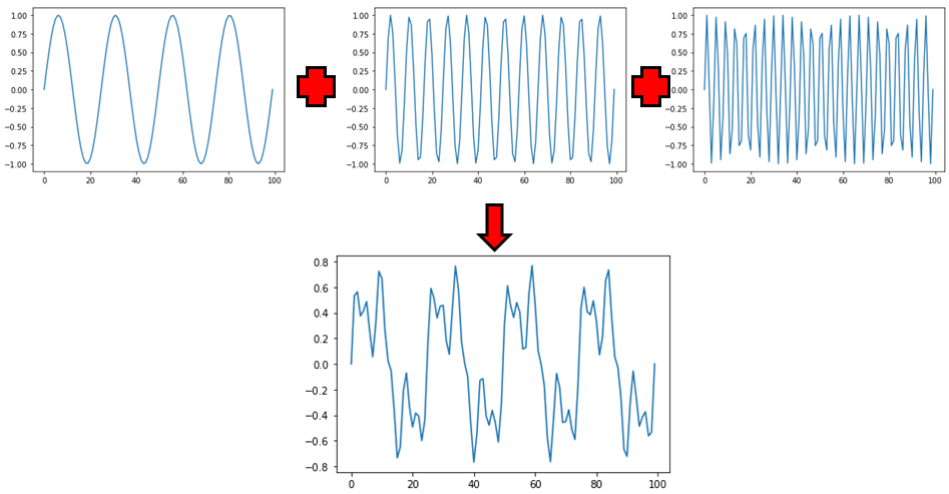


그림 1: Sound(음파)는 단순 Sine파의 결합으로 볼 수 있다. 이 결합된 음파가 어떤 단순파의 결합으로 만들어진 것인지 분석할 수 있게 해주는 것이 Fourier Transform이다. 이런 과정을 time domain(음파)를 frequency domain으로 변환한다고 한다. 그림에서는 각기 다른 주파수의 sine파가 5 : 3 : 2로 결합된 음파를 보여주고 있다. Fourier Transform을 통해 어떤 주파수의 성분이 얼마나 결합되어 있는지 알아낼 수 있다. 이산적인 data를 다루어야 하기 때문에, Discrete Fourier Transform을 사용한다.

- DTF의 결과로 복수소로 이루어진 벡터가 생성되는데, 복소수를 다루기 어렵기 때문에 절대값이나 절대값의 제곱을 취한다. 절대값 취한 것을 magnitude spectrogram, 절대값의 제곱한 것을 power spectrogram이라 부른다.

```
N = 400; T = 1.0 / 800.0
x = np.linspace(0.0, N*T, N)
y = np.sin(50.0 * 2.0*np.pi*x) + 0.5*np.sin(80.0 * 2.0*np.pi*x)
yf = np.fft.fft(y) #shape: (400,) complex numbers
```

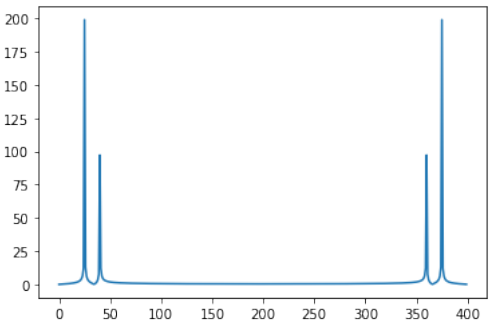


그림 2: DTF: FFT의 absolute 값의 그래프. Nyquist critical frequency[1]에 의해 $-\frac{1}{2T} \sim \frac{1}{2T}$의 주파수 영역이 구해지고, 양의 주파수/음의 주파수에 대한 값이 conjugate가 된다. 대칭적인 결과가 redundant하므로, STFT에서는 절반 크기의 output을 만들어 낸다.

```
scale, sr = librosa.load(audio_filename)
n_fft = 100
fft = np.fft.fft(scale,n=n_fft) # (100,)
rfft = np.fft.rfft(scale,n=n_fft) # (51,)
print(np.allclose(fft[:n_fft//2+1],rfft)) # True
```

[1]sampling time이 $T$일 때, $\frac{1}{2T}$ 이상의 주파수는 측정할 수 없다(aliasing-디지털 신호 처리 과정에서 발생하는 노이즈).

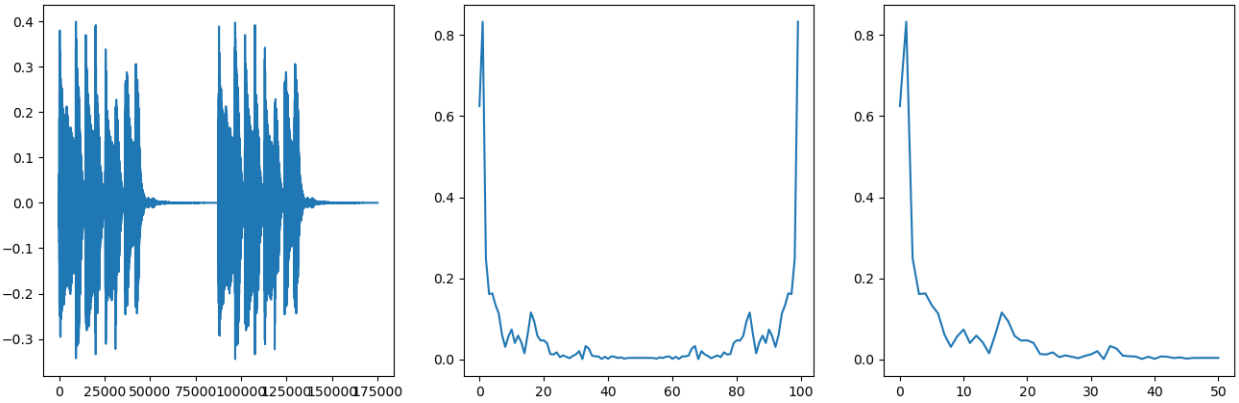


그림 3: fft vs rfft: rfft는 대칭적인 부분을 고려해서 계산량을 줄인다. librosa.stft는 애초에 절반만 return한다.

```
N = 5
signal = np.random.rand(N)
fft = np.fft.fft(signal)

result = []
for k in range(N):
      temp = 0
      for m in range(N):
            temp += signal[m]*np.exp(-2*np.pi*1j*m*k/N)
      result.append(temp)
```

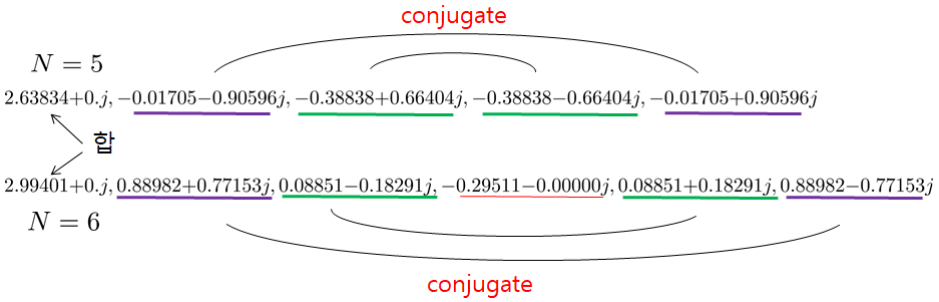
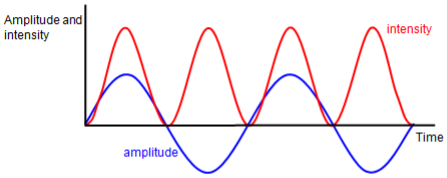



그림 4: FFT의 결과: 첫번째 값은 data의 합이 되고, 그 이후의 값들은 서로 대칭적인 conjugate관계이다. librosa의 stft에서는 hann window를 적용하기 때문에, numpy에서의 fft 결과와는 다르다. 참고로, python에서는 수학에서의 허수단위 $i = \sqrt{-1}$를 $j$로 표시한다.

## ♠ 소리의 요소들

- Intensity($I$): 단위 면적당 에너지를 뜻한다. amplitude($A$)의 제곱에 비례한다. $I \propto A^2$
- Threshold of hearing(TOH): $10^{-12} W/m^2$
- Decibel: TOH를 기준으로 log값을 취한 것

$$dB(I) = 10 \times \log_{10}\left(\frac{I}{I_{TOH}}\right)$$




| Source | Intensity | Intensity level | × TOH |
|---|---|---|---|
| Threshold of hearing (TOH) | $10^{-12}$ | 0 dB | 1 |
| Whisper | $10^{-10}$ | 20 dB | $10^{2}$ |
| Pianissimo | $10^{-8}$ | 40 dB | $10^{4}$ |
| Normal conversation | $10^{-6}$ | 60 dB | $10^{6}$ |
| Fortissimo | $10^{-2}$ | 100 dB | $10^{10}$ |
| Threshold of pain | 10 | 130 dB | $10^{13}$ |
| Jet take-off | $10^{2}$ | 140 dB | $10^{14}$ |
| Instant perforation of eardrum | $10^{4}$ | 160 dB | $10^{16}$ |

Table 1.1 from [Müller, FMP, Springer 2015]

그림 5: intensity는 amplitude의 제곱에 비례한다.

- Timbre: Color of sound, Difference between two sounds with same intensity, frequency, duration.
- **배음(Overtone)**: An overtone is any frequency greater than the fundamental frequency of a sound. Using the model of Fourier analysis, the fundamental and the overtones together are called **partials**. **Harmonics**, or more precisely, harmonic partials, are partials whose frequencies are numerical integer multiples of the fundamental(including the fundamental, which is 1 times itself). 이 배음들이 어떻게 결합되어지는가에 따라 음색이나 악기 고유의 소리가 결정된다.

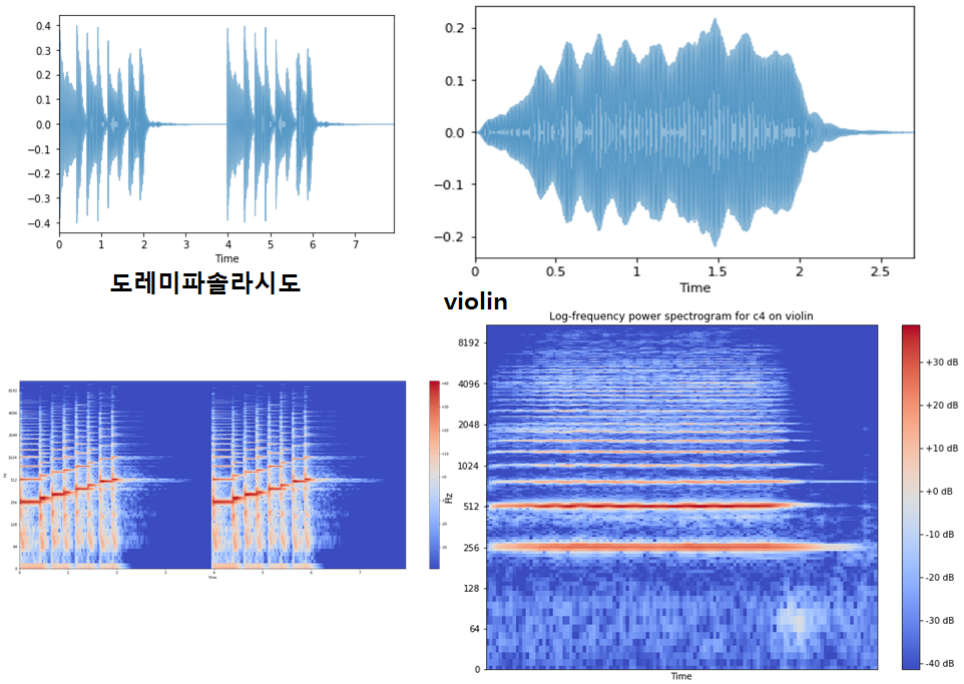



그림 6: 배음(overtone) 구조: fundamental frequency로 256이 보이고, 이것의 정수배에 해당하는 주파수들도 보인다.

- Formant: 음성학에서 인간 성대의 공명에서 발생하는 스텍트럼의 local maximum을 말한다.
- Additive Synthesis[2]: Additive synthesis is a sound synthesis technique that creates timbre by adding sine waves together.
- waveform 비교: 그림 (8)

[2] https://teropa.info/harmonics-explorer/

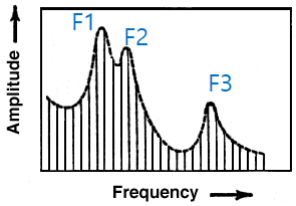



그림 7: Formant: local maximum을 왼쪽 부터 first formant frequency($F1$), second formant frequency($F1$), ... 로 부른다. formant frequency간의 간격이 중요하다. 이 간격(어떤 의미에서 주기)를 잘 파악하는 것이 MFCC이다. 여기서 $y$축의 Amplitude는 중요하지 않다.

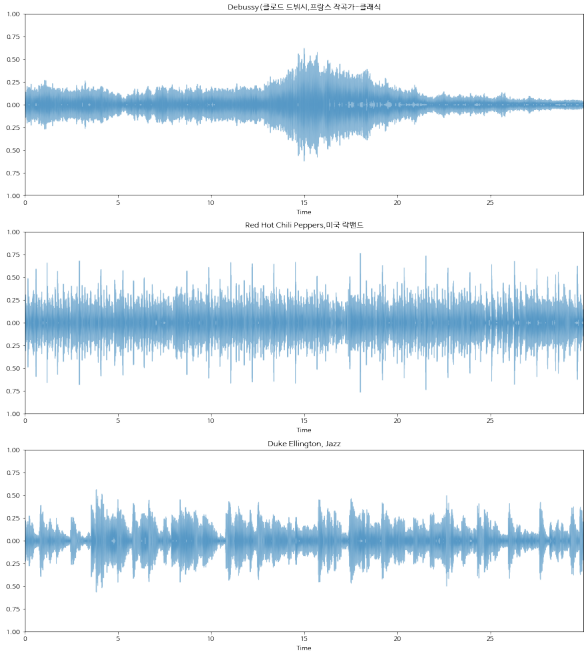



그림 8: waveform 비교: 위에서 부터 클래식음악, 락, 째즈.

### ♠ Time-domain audio Features

Time-domain Feature들은 전통적인 Machine Learning 분야에서 많이 활용되었으나, 딥러닝이 활성화되면서 중요성이 낮아졌다.

- Amplitude envelope: 각 frame의 최대값을 모아 놓은 것.
- RMS(Root Mean Square Energy): 각 frame에서 제곱값의 평균에 root를 취한 값. 각각의 frame마다 계산된다. 직접 계산할 수도 있고, librosa.feature.rms를 이용해도 된다.
- Zero Crossing Rate: Number of times a signal crosses the horizontal axis.
  `librosa.feature.zero_crossing_rate`

### ♠ Mel Filter Bank

- $m = 2595 \log_{10}(1 + \frac{f}{700})$

$$M(f) = 2595 \log_{10}(1 + \frac{f}{700})$$

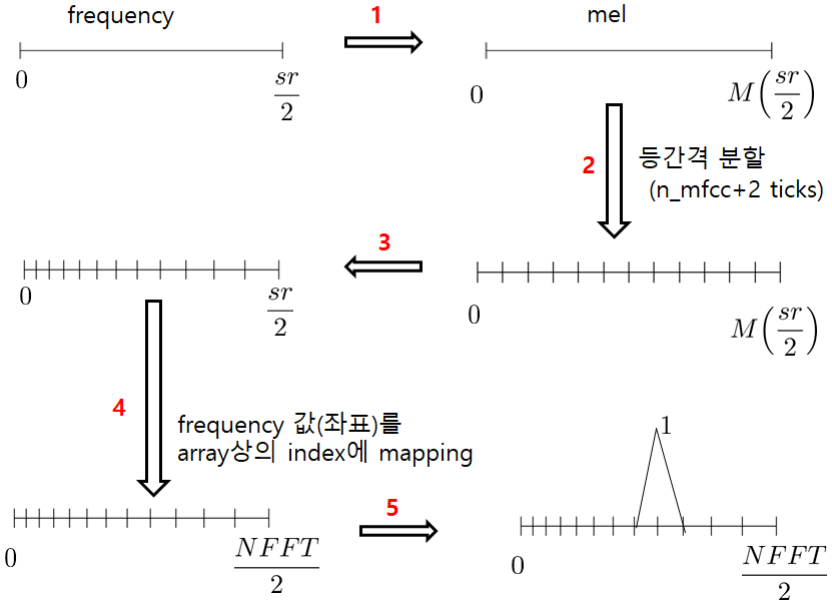



그림 9: mel filter bank(mel basis matrix) 생성 과정: 4번째 과정에서 선택된 주파수는 끝점 포함해서 (n_mfcc+2) 개이고, array index는 0부터 NFFT/2까지 있다. 따라서 $0, 1, 2, \cdots,$ NFFT/2 중에서 (n_mfcc+2) 개 만큼의 index만 선택된다. index중에서 선택된 곳에서는 0 또는 1의 값을 가지게 하고, 그 외의 index들에 대한 값은 ($\Lambda$)모양의 직선상에 있게 값을 잡아 줄 수 있다. 그리고, 처음에 최대 주파수를 지정해 주어야 되는데, default로 sampling-rate의 절반을 사용한다. 주파수는 상대적인 것이기 때문에 critical한 것은 아니다.

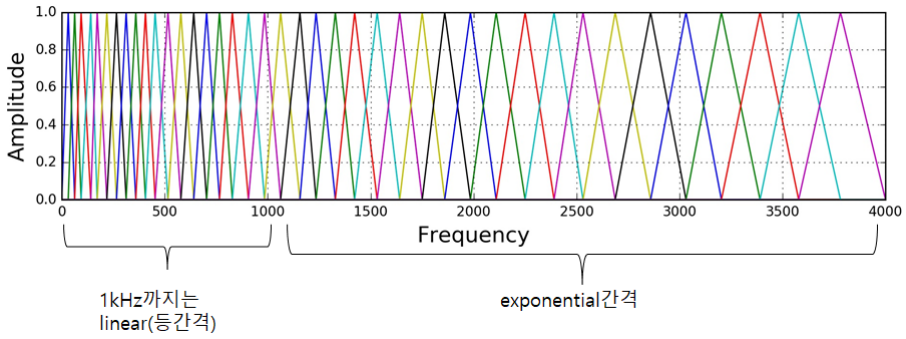



그림 10: mel filter bank

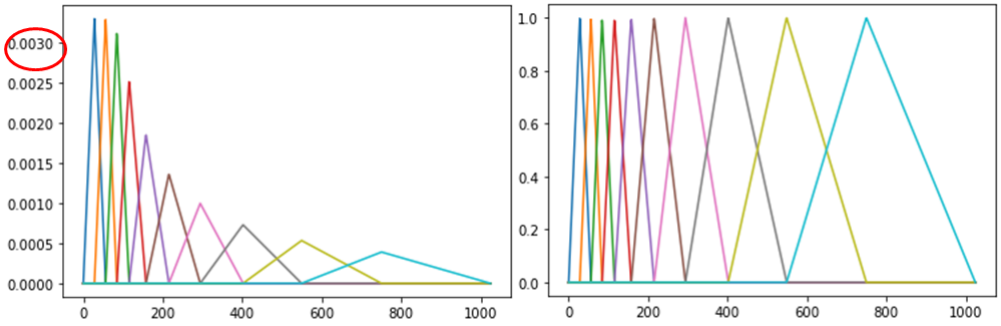


그림 11: librosa.filters.mel: 왼쪽 norm=1(default)로 하면, band의 폭(삼각형의 밑변 길이)으로 값을 나누어준다. 이 경우는 filter weight의 합이 동일하다. 즉 각 column합이 일정하다.
오른쪽: norm=None. normalization이 되지 않았기 때문에, mel spectrogram을 구하면 값이 크다.

## ♠ MFCC

- MFCC(Mel Frequency Cepstral Coefficients)는 Spectrum of Spectrum이라 할 수 있다. Frequency data에 다시 Discrete Cosine Transform(또는 Inverse DFT)을 적용하기 때문이다.
- Therefore, we can apply Discrete Cosine Transform (DCT) to decorrelate the filter bank coefficients and yield a compressed representation of the filter banks[3].
- MFCC는 log mel-spectrogram을 Discrete Cosine Transform하면 된다. Typically, for Automatic Speech Recognition (ASR), the resulting cepstral coefficients 2-13 are retained and the rest are discarded; `num_ceps = 12`. The reasons for discarding the other coefficients is that they represent fast changes in the filter bank coefficients and these fine details don't contribute to Automatic Speech Recognition (ASR).
- log mel-spectrogram의 feature dimension이 $n$이면, DCT를 적용해도 $n$차원이 된다. 이 중에서 앞 부분을 필요한 만큼 선택하면 된다. 첫번째 feature는 값(data의 합)이 너무 작아(음수), 다른 feature에 악영향을 주기 때문에 제거하는 경우도 있다.

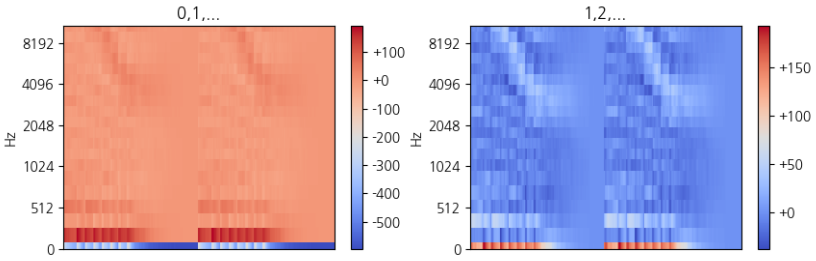



그림 12: MFCC의 첫번째 featrue는 입력 data의 합이다(Fourier/Cosine Transform의 첫번째 값은 data의 합). 첫번째 feature 제거 전/후 비교. 왼쪽은 제일 아래쪽 라인의 값이 너무 작아서 다른 값들이 의미를 가지지 못하게 된다.

- 딥러닝이 활성화되기 이전의 머신러닝에서는 Feature의 선택이 중요했기 때문에, mel filter bank에 기인하여 나타나는 Feature 간의 상관관계를 줄이기 위해서 MFCC가 활용되었다. GMM 같은 모델에서는 이런 상관관계가 제거된 Feature가 유용하게 활용된다.
- 딥러닝이 출현한 이후에는 Feature 간의 비선형 관계를 제거해 버리는 MFCC의 중요성이 낮아졌다.
- MFCC lifter: 결과의 상위 인덱스에 해당하는 값들이 작기 때문에, lifter를 곱해서 증폭시킨다.
  `librosa.feature.mfcc(..., lifter)`

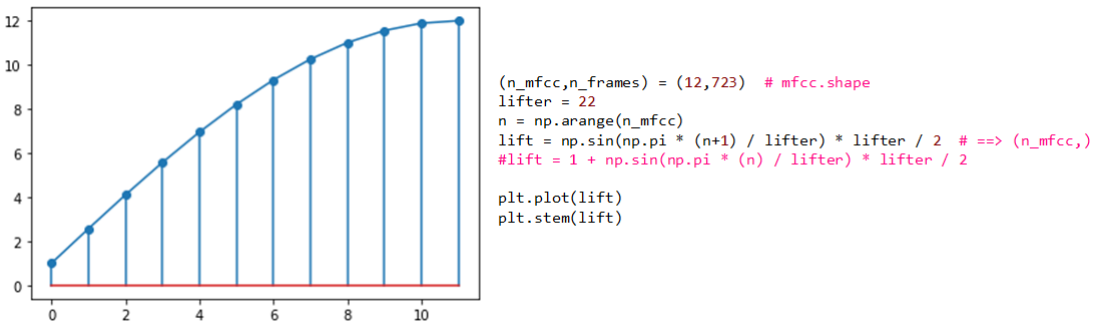



그림 13: MFCC lifter: MFCC output에 sine 값을 곱해준다.

- MFCC의 장점:
  - spectrum의 디테일은 무시하고, 크게 보는 feature.

[3] `https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html`

  - 배음 구조를 잘 파악한다.
  - 음정(pitch)의 차이는 무시한다. 음정을 무시해도 되는 task에 적합하다. 악보를 그리는 것과 같은 task에는 적합하지 않다.
  - formant[4] 구조를 잘 파악한다.

- MFCC의 단점:
  - noise에 약하다.
  - 음성합성에 적합하지 않다. 압축성이 강한 MFCC로 부터 audio를 생성하거나 복원하는 것이 어렵다.

### ♠ Preemphasis

- Pre-emphasis is a very simple signal processing method which increases the amplitude of high frequency bands and decrease the amplitudes of lower bands. In simple form it can be implemented as

$$y_t := x_t - \alpha x_{t-1}$$

- 고주파의 노이즈 제거에서 더욱 중요했는데, 요즘 speech recongnition에서는 불필요하다는 견해도 있다[5].
- STFT를 적용하기 전 waveform에 적용하면 된다.

```
from scipy import signal
N = 5; k = 0.97
wav = np.random.rand(N)

result1 = signal.lfilter([1, -k], [1], wav)
result2 = wav - np.array([0,]+list(k*wav[:-1]))

print(np.allclose(result1,result2))
```

### ♠ Windowing Function

- Hann window: 각 프레임의 처음과 끝에서의 불연속을 최소화하기 위해, Hann window function을 곱해서 STFT를 구한다.

$$w(n) = 0.5\Big(1 - \cos\big(\frac{2\pi n}{M-1}\big)\Big), \quad 0 \le n \le M-1$$

- time domain에서 곱해지는 Hann window function은 frequency domain에서의 convolution이 된다. 따라서 frequency를 filtering하는 효과가 있다.
- Hamm window:

$$w(n) = 0.54 - 0.46\cos\big(\frac{2\pi n}{M-1}\big), \quad 0 \le n \le M-1$$

[4] https://en.wikipedia.org/wiki/Formant

[5] https://www.quora.com/Why-is-pre-emphasis-i-e-passing-the-speech-signal-through-a-first-order-high-pass-filter-required-i

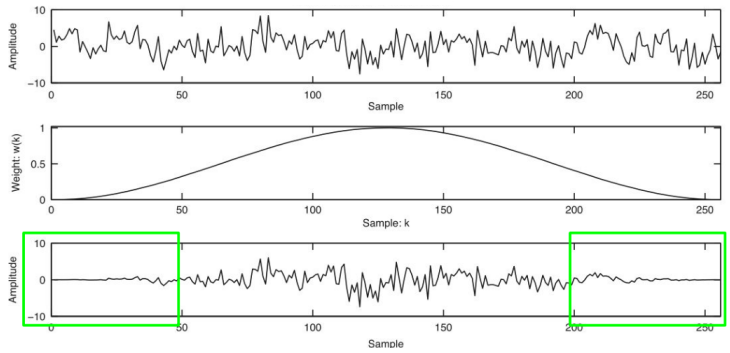
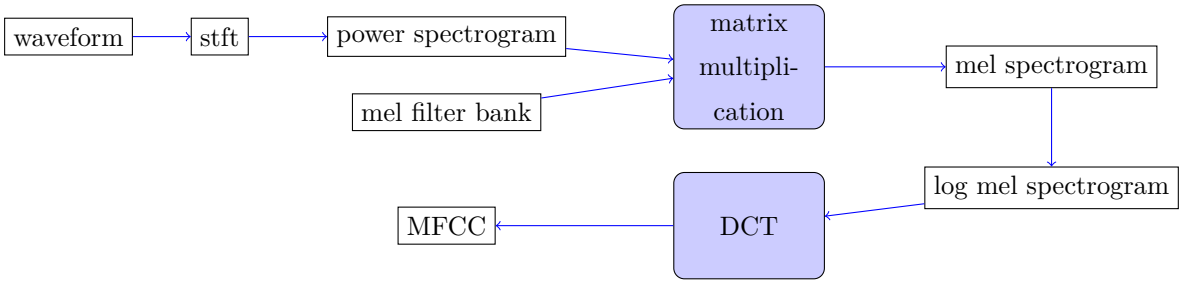



그림 14: Hann Window

### ♠ librosa API




- librosa.load: wav파일 load. $-1 \sim 1$ 사이 값.
- librosa.display.waveplot(y,sr,alpha=0.5): `sample_rate`을 넣어줌으로써, x축이 시간축으로 잘 보여진다. alpah는 선명도.
- librosa.display.specshow: 세로축(axis=0, 아래에서 위), 가로축(axis=1, 왼쪽에서 오른쪽).
  - x축: `sample_rate, hop_length`을 넣어줌으로써, x축이 시간축으로 잘 보여진다. `(입력길이 * hop_length/sample_rate)` = duration(time). `x_axis='time'`이 설정되어야 눈금이 표시된다.
  - y축: `y_axis='log'`로 설정하면, log scale로 보여진다. y축 값은 0 ~ 8000 정도의 값이 입력되는 data에 상관없이 표시된다. `y_coords`를 넣어주면 y축 눈금이 맞게 표시된다. 어쨌든 상대적인 위치를 본다고 하면, `y_coords`를 넣지 않아도 된다.
- `librosa.power_to_db`: Convert a power spectrogram (amplitude squared) to decibel (dB) units. 즉 제곱한 것을 넣어야 한다.
- `librosa.amplitude_to_db`: Convert an amplitude spectrogram to dB-scaled spectrogram.This is equivalent to `power_to_db(S**2)`, but is provided for convenience. 제곱하지 않은 것을 넣으면, 내부에서 제곱해준다.
- `librosa.core.magphase`: Separate a complex-valued spectrogram D into its magnitude (S) and phase (P) components, so that D = S * P.
- amplitude vs magnitude: amplitude는 부호를 가진 vector이고, magnitude는 scalar 값이다.

[6] `https://www.dropbox.com/s/3nmwun0s1dd25tw/scale.wav?dl=0`

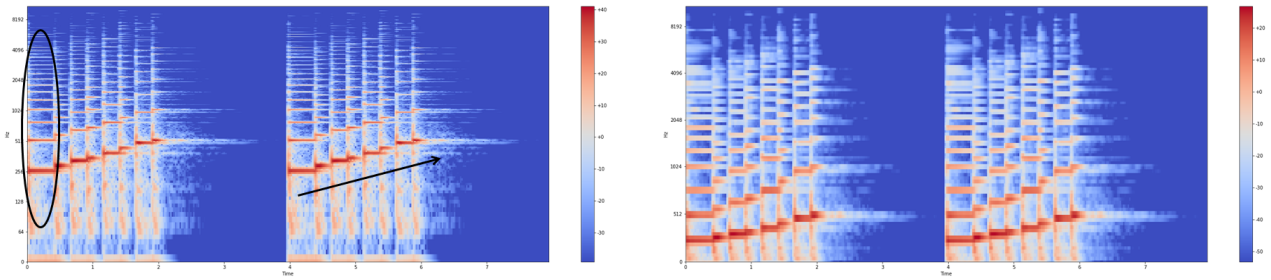



그림 15: log spectrogram vs log mel-spectrogram: '도레미파솔라시도'가 두번 나오는 sound[6]에 대한 spectrogram. 배음 구조(fundamental frequency의 정수배)와 주파수가 올라가는 구조가 잘 보인다. 오른쪽의 mel spectrogram은 차원을 줄이면서도 spectrogram의 특징을 잘 반영하고 있다. MFCC는 더 압축되는 방식이라, 이런 특징이 시각적으로 잘 보이지 않는다.

- librosa.feature.melspectrogram: waveform을 입력하면, magnitude spectrogram을 power(default=2)해서 mel filter bank를 곱한다.

```
# melspectrogram를 한번에 구하는 것과, mel filter bank를 거쳐서 구한 것의 결과가 동일하다.
scale, sr = librosa.load(wav_file) # sr=22050, (174943,)
mel_spectrogram = librosa.feature.melspectrogram(scale, sr=sr, n_fft=2048, hop_length=512,
    n_mels=10) # (10,342)

S_scale = librosa.stft(scale, n_fft=2048, hop_length=512) # (1025, 342)
filter_banks = librosa.filters.mel(n_fft=2048, sr=22050, n_mels=10) # mel matrix
mag_spectrogram = librosa.magphase(S_scale,power=2)[0] # (magnitude,phsase)[0]
mel_spectrogram2 = np.matmul(filter_banks,mag_spectrogram) # (10, 342)

print(np.allclose(mel_spectrogram,mel_spectrogram2)) # True
```

- `librosa.feature.mfcc(y,n_mfcc=40)`: waveform을 mel-spectrogram으로 변환 후, mfcc를 만든다. 내부적으로는 `librosa.filters.mel`, `librosa.feature.melspectrogram`의 default parameter들이 사용된다.

```
# mfcc를 한번에 구하는 것과, melspectrogram을 거쳐서 구한 것의 결과가 동일하다.
n_mfcc = 20

mfcc = librosa.feature.mfcc(scale,n_mfcc=n_mfcc) # hop_length=512,n_fft=2048,
    n_mels(default=128) 크기로 생성후, 앞쪽
    return
mfcc_ = librosa.feature.mfcc(scale,n_mfcc=80,hop_length=512,n_fft=2048) # n_mels(default=128) 크
    기로 생성후, 앞쪽 80개 return

mel_spectrogram = librosa.feature.melspectrogram(scale, sr=sr) # default: n_fft=2048,
    hop_length=512, n_mels=128
log_mel_spectrogram = librosa.power_to_db(mel_spectrogram)
print('shape of log_mel_spectrogram: ', log_mel_spectrogram.shape)
mfcc2 = librosa.feature.mfcc(S=log_mel_spectrogram,n_mfcc=n_mfcc) # output shape, min(n_mels,
    n_mfcc)

print(mfcc.shape,mfcc2.shape)
print("2가지 mfcc 결과 비교: ", np.allclose(mfcc,mfcc2)) # True
```

```
mfcc3 = dct(log_mel_spectrogram, type=2, axis=0, norm='ortho')[1:n_mfcc, :]
print(mfcc3.shape)
print("직접 구한 결과와 비교: ", np.allclose(mfcc[1:],mfcc3)) # True
```

- librosa.feature.delta(mfcc, order=1, axis=-1): delta of mfcc, delta delta of mfcc를 구할 때 사용. 시간에 대한 미분이 필요하므로, axis=-1
- `librosa.feature.chroma_stft`: Chromagram은 소리를 octave에 상관없이 12음정에 mapping 시킨다.

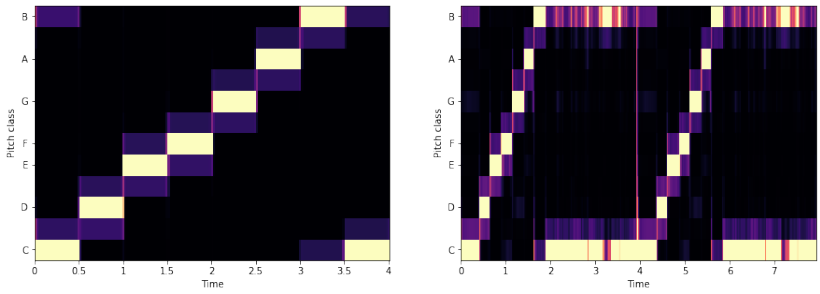



그림 16: Chromagram: '도레미파솔라시도'의 chromagram. 왼쪽은 단순 sine파 합성이고, 오른쪽을 악기로 (2번 반복) 연주된 것이다.