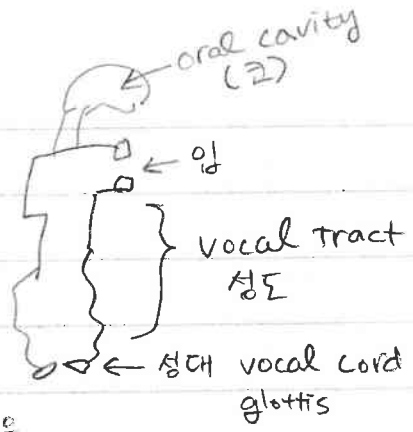
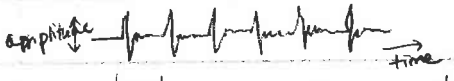
2021. 1. 6. Vocoder design - LPC

- speech production model (voiced)

성대의 떨림으로 인해 생긴 기본음이 성도(vocal tract)를 거쳐 주파수 특성이 입혀지고 입과 코를 통해 배출됨



- 기본음: glottal pulse, excitation. 펄적을



주기 pitch. $F_0 = \frac{1}{\text{pitch}}$ (기본주파수)

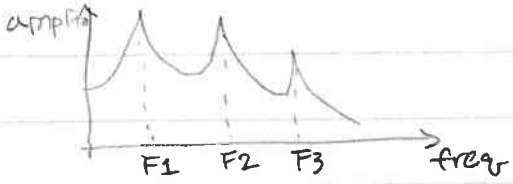
- 성도 (vocal tract): tube 이론에 따라 공명이 생기는 주파수에서 energy가 커짐. (이론적으로 무한대이지만 사람의 성도에서 흡수됨)

이때의 공명주파수들을 formant라고 부르고 F1, F2, F3.. 와 같이 쓴다 linguist들은 보통 F1과 F2로 모음들을 구분하며 가끔 F3도 필요하다

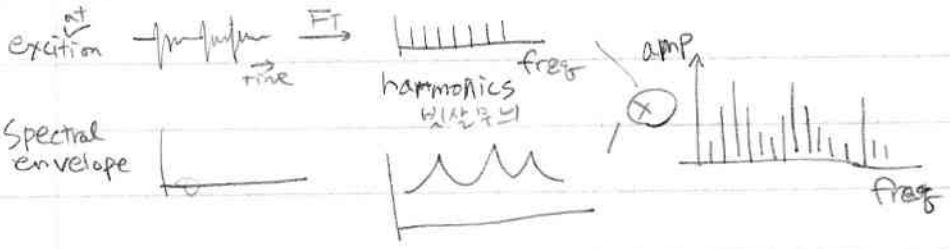
* formant frequency들은 정확히 (deterministically) 추정하기 어려워서 음성인식에 직접적으로 쓰이진 않는다.

* 입의 모양도 formant frequency에 영향을 준다

* 코는 공명뿐 아니라 (비강 커지는 현상이) 생기지 않고 energy가 흡수되는 현상이 생기며 이를 zero라고 한다. ([n] [m] [ŋ])



- 음성의 생성 (주파수 영역)



(?) time domain에서는 모델링 가능한가?

2021. 1. 6. Vocoder - LPC

주파수 영역: $X(\omega) = H(\omega) \cdot E(\omega)$ (E(ω): excitation) — 단순한 곱하기

($H(\omega)$: impulse response of envelope)

시간영역: $x[t] = h[t] * e[t]$ convolution

$= \sum_{\tau=-\infty}^{\infty} e[t-\tau] \cdot h[\tau]$

⇒ h[t] 와 e[t] 를 어떻게 분리할 수 있나?

주파수 $X(\omega) = H(\omega) \cdot E(\omega) \Leftrightarrow E(\omega) = \frac{X(\omega)}{H(\omega)} = A(\omega) \cdot X(\omega)$ ($* A(\omega) = \frac{1}{H(\omega)}$)

시간 $e[t] = a[t] * x[t]$

a[t]의 order를 finite하게 제한 ⇒ M (8KHz에서는 10)

$e[t] = \sum_{\tau=0}^{M} x[t-\tau] \cdot a[\tau] = a_0 x[t] + a_1 x[t-1] + a_2 x[t-2] + a_M x[t-M]$

$\Leftrightarrow a_0 x[t] = -\sum_{\tau=1}^{M} a_\tau x[t-\tau] + e[t]$

$\Rightarrow$ (let $a_0 = 1$) $x[t] = -\sum_{\tau=1}^{M} a_\tau x[t-\tau] + e[t]$

t번째 sample을 M개의 이전 sample들의 선형결합 (linear combination) 으로 추정 (prediction) 하고

이때의 추정 error는 e[n] 이다.

- 유도방법 linear predictive coding

$\{a_1, a_2, \dots a_M\} = \arg\min_{\{a_k\}} \sum_t e^2[t]$ 주어진 sample들의 squared error를 최소화

where $e[t] = x[t] + \sum_{\tau=1}^{M} a_\tau x[t-\tau]$

- 의미: $E(\omega) = A(\omega) \cdot X(\omega) \Rightarrow X(\omega) = \frac{E(\omega)}{A(\omega)}$

$A(\omega) = 0$ 가 될때 $X(\omega) \to \infty$ (resonance, 즉 formants)

all-pole model 이라고 함 (∞ 가 되므로)

- 구현: MATLAB `[a, g] = lpc(x, p)`

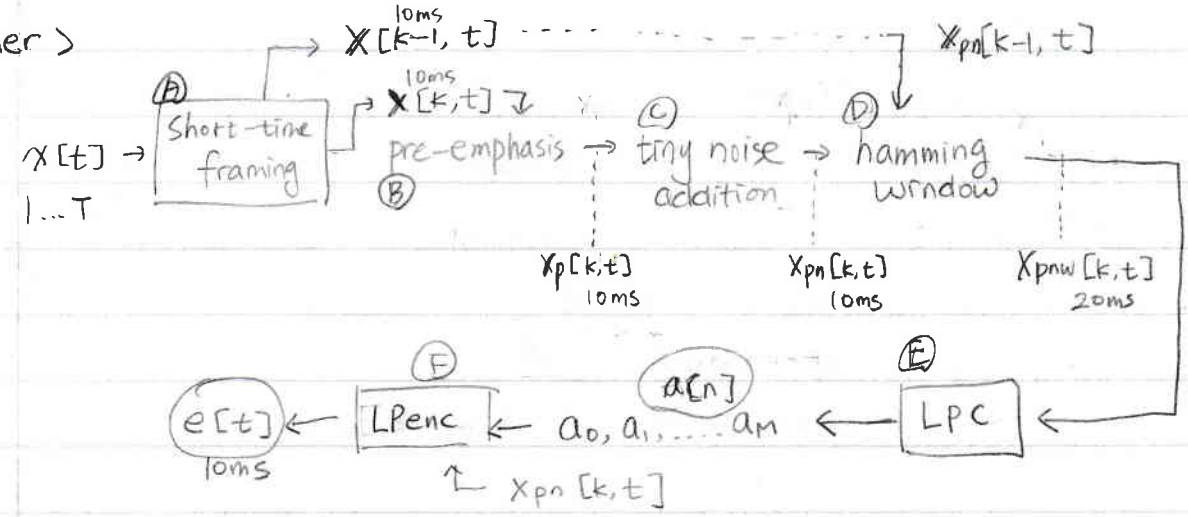
(a_τ, excitation, signal, M: order)

python lpctorch, librosa.core.lpc(y, order)

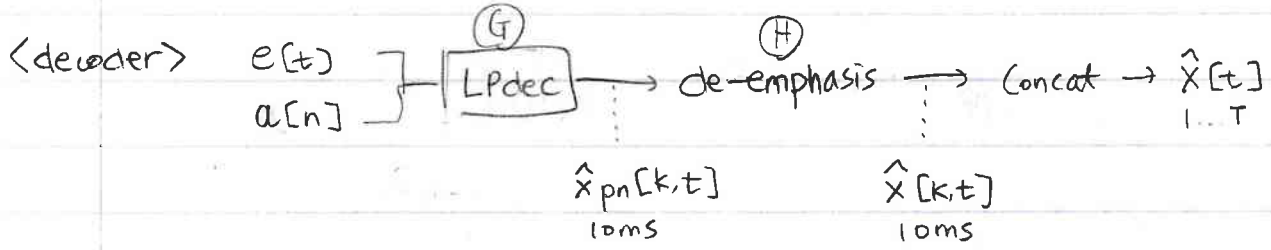
- (주의): 코드에 따라 a_0를 포함/미포함. 부호: ⊗ (a의 부호가 뒤집혀짐) ($x[t] + \sum a_\tau x[t-\tau]$ / $x[t] - \sum a_\tau x[t-\tau]$)

2021. 1.6. Vocoder-LPC

<encoder>



<decoder>



Ⓐ Short-time framing

- 길이가 정해져 있지 않은 입력 신호를 10ms 나누는 연산
- 10ms = 0.01 × 8000 (sampling frequency 8kHz) = 80 samples
  = 0.01 × 16000 (Fs = 16kHz) = 160 samples

※ 이전과정 : .wav 파일을 읽어서 x[t] : np.ndarray of T×1 에 저장

python

```
def wavfilevocoder(wavfilename, M, Ts=0.01):
    x, Fs = wavread(wavfilename)   # 적당한 wav file read 함수 사용
    Ns = int(Fs * Ts)              # shift sample length
    T = len(x)    K = (T+Ns-1) // Ns   # ceiling, frame 개수
    A = np.ndarray (M+1, K)
    E = np.ndarray (Ns, K)
    for k in range(K):
        xk = x[(k*Ns):((k+1)*Ns)]   # 마지막 frame 적절히 처리 필요
        A[:,k], E[:,k] = LPenc(xk, M)
    return A, E
```

2021. 1. 6. vocoder - LPC

Ⓟ pre-emphasis

defined by $H_p(z) = 1 - \alpha z^{-1}$ (a_0, a_1) $\alpha = 0.97, 0.98, 1.0$

$\Leftrightarrow x_p[t] = x[t] - \alpha x[t-1]$

사람의 목소리는 vocal tract을 거치면서 고주파 energy가 흡수됨

이것을 spectral tilt라고 하며 주파수가 높을수록 반사가 많이 되면서 signal energy가 흡수되기 때문

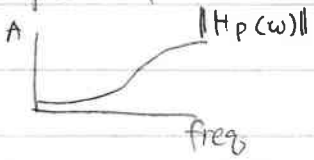
구체적으로 -6 dB / octave 로 알려짐

✱ 1 octave = 주파수가 두배가 되는 것

$-6\,dB = 20 \log_{10} A \Leftrightarrow A = 10^{-\frac{6}{20}} \simeq 0.5$

즉, 주파수가 두배가 되면 energy가 절반이 됨

✱ pre-emphasis filter의 impulse response의 magnitude는

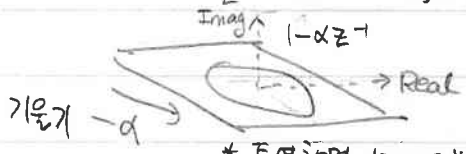


즉, 약간의 high-pass filter의 성질을 가진다.

✱ (Q) $1 - \alpha z^{-1}$ 은 linear equation 인데 왜 impulse response는 linear가 아닌가?

⇒ Fourier transform은 complex z domain 에서

$z = e^{j\omega} = \cos\omega + j\sin\omega$ $(0 \le \omega \le 2\pi)$에 대해서만 계산된 것임. 즉,



* 투영하면 linear line이 아닌 sine 곡선이 됨

$H_p(\omega) = 1 - \alpha e^{j\omega \cdot (-1)} = 1 - e^{-j\omega}$ (쉽게 계산하기 위해 let $\alpha = 1$)

$= 1 - (\cos\omega - j\sin\omega)$

$= (1 - \cos\omega) + j\sin\omega$

$\text{Real}(H_p(\omega)) = 1 - \cos\omega$

$\text{Imag}(H_p(\omega)) = j\sin\omega$

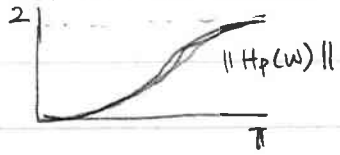
$\|H_p(\omega)\|^2 = H_p(\omega) \cdot \underbrace{H_p^*(\omega)}_{\text{Complex conjugate}} = (1 - \cos\omega + j\sin\omega)(1 - \cos\omega - j\sin\omega)$

$= (1 - \cos\omega)^2 - j^2\sin^2\omega = 1 - 2\cos\omega + \cos^2\omega + \sin^2\omega$

$= 2 - 2\cos\omega = 2(1 - \cos\omega) \Rightarrow$

2021. 1. 6 LPC vocoder

Ⓑ pre-emphasis Ⅱ



즉 $\|H_p(w)\| = \|1 - \alpha z^{-1}\|$ 은

sine 곡선에 가까운 가장 간단한

highpass filter 가 되고

0 → π 의 평균 기울기는 2 이다

⇒ 6dB/octave boost

* 일반적으로 FIR filter를 이용하여 high-/low-/bandpass filter를

$y[n] = a_0 x[n] + a_1 x[n-1] + \dots + a_M x[n-M]$

설계하면 order M에 따라 precision 정확도가 결정된다. 즉,

stopband에서 날카롭게 변하지 않고

sine 함수들의 중첩으로 표현되며

side lobe 들의 수는 M과 같다

pre-emphasis filter 는 가장 간단한 HPF이다

python

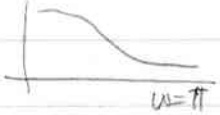
```
def preemphasis(x, xmem=0, alpha=0.98):
    xp = np.ndarray(x.shape)
    xp[0] = x[0] - alpha * xmem
    xp[1:] = x[1:] - alpha * x[0:end-1]
    return xp
```

xmem : 첫번째 sample 계산을 위한 memory

이전 frame의 마지막 sample - $x[k-1, N_s]$

* $preemphasis^{-1}$ = deemphasis (IIR)

$\frac{1}{1-\alpha z^{-1}} \rightarrow x_d[t] = x_p[t] + \alpha\, x_d[t]$



원래대로 복원시켜줌

2021.1.6 LPC Vocoder

© tiny noise addition

mean 0, Std 1.0 인 Gaussian random

$x_{pn}[t] = x_p[t] + \varepsilon \cdot randn(0,1)$

신호처리 과정에서 전체 frame sample들이 0이면 frame energy가 0가 되고 특히 log filterbank energy 계산할때 $-\infty$가 되어서 수학에 문제가 됨

따라서 매우 작은 크기의 Gaussian noise를 더해준다.

입력 x가 (코드에 따라) $[-1, 1]$ 이면 $\varepsilon = 10^{-6} \sim 10^{-10}$

$[-2^{15}, 2^{15}-1] = [-32768, 32767]$ 이면 $\varepsilon = 10^{-3}$

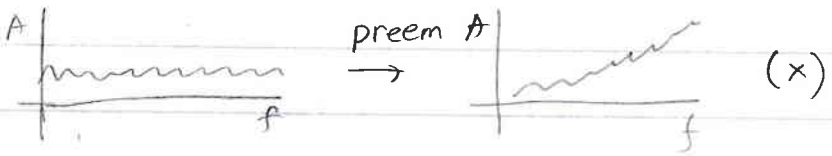
정도가 적당하다.

raw PCM wav file은 short int이고 그대로 memory로 올리는 경우

반드시 사용하는 wavread의 출력을 확인해 볼것

* 왜 preemphasis 이후에 하는가?

→ Gaussian noise는 flat white 인데, preemphasis에 의해 high frequency noise가 되면 매우 annoying



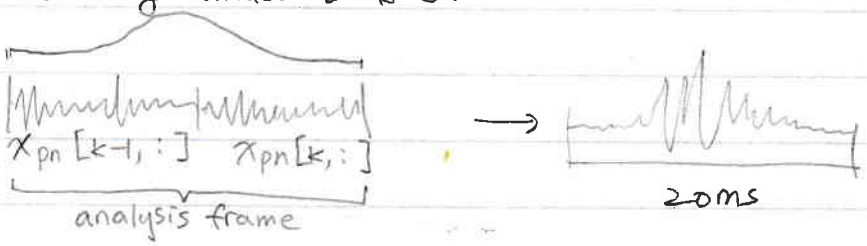
(×)

LPC 등 나중 단계에 영향을 주기 때문에 preemphasis 이후에 해야 한다

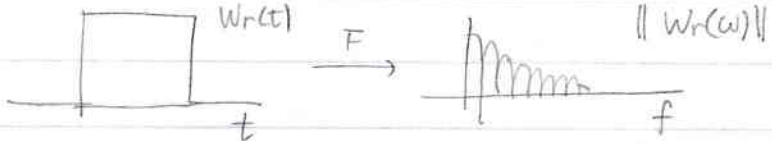
2021.1.6 LPC vocoder

ⓓ hamming window

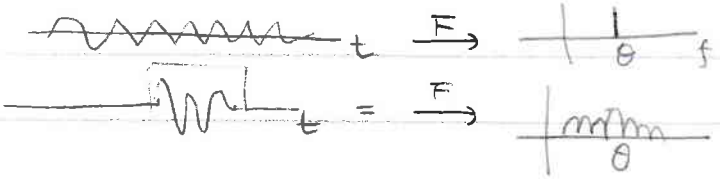
- 1) 이전의 10ms frame과 붙여서 20ms를 만듦
- 2) hamming window를 곱함.



* 왜 window를 곱하는가? → 원래 모든 LTI (linear time invariant) system은 domain을 $(-\infty, \infty)$로 가정하는데 20ms만 취하면 [0, 20ms]의 rectangular window를 곱하는 것과 같다.

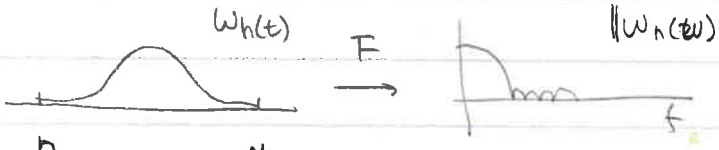


따라서, 다음과 같은 pure sine의 경우 $W_r[t]$가 convolution되어

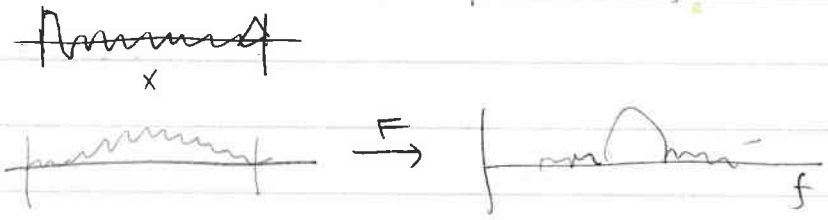


깔끔한 선이 아닌

hamming window는



main lobe는 커지지만 side lobe는 작아진다.



smoothed but 정확해진다.

* 왜 hanning window 안쓰는가? analysis에는 (LPC, MFCC) hamming 사용.

→ hanning window는 마지막 boundary 값들이 0이고 overlap-add 에서

2021. 1. 6. LPC Vocoder

Ⓔ LPC

- LPC envelope을 구할 때 충분한 정보를 주기 위해서 이전 frame을 concat하여 20ms 사용

$a[0], \ldots, a[M] = lpc(w_{ham}(320) \cdot (x_{pn}[k-1,:], x_{pn}[k,:]))$

MATLAB `[a, g] = lpc(x, p)`

python 여러가지 방법이 있는 것 같지만 try `librosa.core.lpc(y, order)`

* a의 sign을 주의한다. manual 읽어보기

Ⓕ LPCenc : $e[t] = a[t] * x[t]$

- Ⓔ에서 구한 lpc filter coefficients $a[n]$을 이용하여 excitation 구한다

- 입력 : $x_{pn}[k,:]$ 10ms 출력 : $e[k,:]$ 10ms
  $x_{pn}[k-1, -M:end]$ #이전 frame의 M개의 샘플

```
def LPCenc(x_pn, a_k, x_mem):
    #a_0 = 1.0 확인필요
    T = len(x_pn)
    e = np.ndarray(T)
    for t=0 to T-1:
        # 처음 x의 M개의 sample들은 memory 이용
        # 그다음 sample들은 memory 사용하지 않고 convolution
```

2021. 1.6. LPC vocoder

Ⓖ LPCdec: $X(\omega) = \frac{E(\omega)}{A(\omega)}$ IIR filtering

def LPCdec(e, a, $\underbrace{\hat{x}_{pn,mem}}_{\text{memory}}$): returns $\hat{x}_{pn}$

$$e[t] = a[t] * x[t] = \sum_{\tau=0}^{M} x(t-\tau)\cdot a[\tau]$$

$$= \underbrace{a_0}_{=1} x[t] + a_1 x[t-1] + \dots + a_M x[t-M]$$

$$\Leftrightarrow\ x[t] = e[t] - a_1 x[t-1] - a_2 x[t-2] - \dots - a_M x[t-M]$$

$$\Rightarrow\ \hat{x}_{pn}[k,t] = e[k,t] - a[k,1]\cdot\hat{x}_{pn}[t-1] - a[k,2]\,\hat{x}_{pn}[t-2] - \dots - a[k,M]\cdot\hat{x}_{pn}[t-M]$$

(* $\hat{x}_{pn}$ 의 M개의 memory가 필요함)

→ $\hat{x}_{pn}[k-1, -M:\text{end}]$ 활용

→ 이전 frame에서 생성한 $\hat{x}_{pn}$ 값 저장

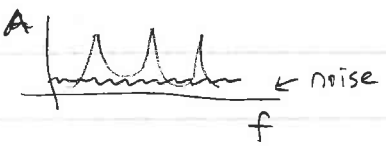
㊟(주의) a는 e에 대하여 IIR filter 이므로 발산할 수 있다

Ⓗ de-emphasis

$$x_p[t] = x[t] - \alpha x[t-1] \Leftrightarrow x[t] = x_p[t] + \alpha x[t-1]$$

$$\Rightarrow\ \hat{x}[k,t] = \hat{x}_{pn}[k,t] + \underbrace{\hat{x}[k,t-1]}_{\text{size 1의 memory 필요}}$$

* noise의 영향



de-emph →

낮아있는 잔주잡음은

저주파가 강조된 "pink" noise

⇒ 없앨 필요 없음

2021.1.6. LPC vocoder

(summary of the assignment 1)

1) encoder 작성

```
def wavfile_vocoder(wavfile, M, Ts=0.01):
    # read wave file
    # for every 10ms:   (Ⓐ short-time framing)
        extract x[k, 0:Ns]   (10 ms)
        Ⓑ pre-emphasis → xp[k, :]
        Ⓒ add noise → xpn[k, :]
        Ⓓ concat(xpn[k-1], xpn[k, :]) and multiply
          hamming window → xpnw[k, :]  (20ms)
        Ⓔ LPC → a[k, 0...M]
        Ⓕ LPenc → e[k, 0:Ns-1]
    return a[k, :], e[k, :]
```

2) decoder 작성

```
def LPdecoder(a[1...k, 0...M], e[1...K, 1...Ns])
    # Ⓖ LPdec → x̂pn[k, :]
    # Ⓗ deemphasis → x̂[k, :]
    # concatenate
    # return x̂[1....T]
```

3) 복원성능:

$\varepsilon = 10^{-6}$ 정도를 넣었을때 40dB 정도

$e_x[t] = x[t] - \hat{x}[t]$

$$SNR = 20\log_{10}\frac{\|signal\|}{\|noise\|} = 20\log_{10}\frac{\sqrt{\sum_{t=1}^{T} x^2[t]}}{\sqrt{\sum e^2[t]}}$$

$$= 10\log_{10}\frac{\sum x^2[t]}{\sum e^2[t]} \geq 40$$

약 1% 이내의 error