NPEX 음성인식 강의 (6월22~24)

	NPEX 인공신경망 음성인식 강의 일정	
1일차	Sampling, FFT, sampling rate change	
수요일	Speech production and hearing, Mel-frequency bank	
	(실습 1시간 설명, 1시간 코딩) Speech feature extraction, error rate metric 등	
2일차	Speech recognition with HMM, Tri-phone	
목요일	End-to-end acoustic modeling, RNN-based CTC acoustic model	
	(실습 1시간 설명, 1시간 코딩) RNN-CTC model (DeepSpeech2)	
3일차	Neural Transducers, LAS, on-device speech recognition	
금요일	Self-supervised pretraining, recent speech recognition models (ContextNet, Conform	
	(실습 1시간 설명, 1시간 코딩) Beam-search decoding	
강의 성원용교수, 조교 최익수 심규홍(메인조교)		

음성 인식 vs Text understanding

- DNN 구조는 매우 비슷하다. 둘 다 sequence recognition 의 성격이 강하다. RNN, Transformer 가 중요한 역할.
- 디지털 신호처리 + RNN/CNN/Transformer
- 차이:
  - Text – 인간에게 익숙한 매우 압축된 형태의 심볼(feature 인) 글자 또는 단어(word)가 들어간다.
  - Speech – analog signal (continuous signal) 이 들어간다.
    - Analog를 digital data 로 바꾸고 (결과는 방대한 크기)
    - Digital data 를 압축해서 작은 용량의 feature 로 바꾸어야 한다.
  - Speech는 text 보다 길이가 매우 길다. 1초의 대화를 text로 나타내면 단어 하나나 둘, 음성은 sample 수로는 16K 개, frame 개수로는 50~100개.

디지털 신호 처리: Sampling, Fourier transform, Filtering

서울대 전기정보공학부
교수 성원용

신호처리

- Analog 와 Digital signal processing
- Sampling and quantization (A/D conversion, D/A conversion)
  - Nyquist sampling rate
  - μ-Law companding
  - Frequency in sampled domain
- Fourier transform (discrete Fourier transform, fast Fourier transform)
  - Power spectrum
- Digital filtering (convolution)
- Sampling rate conversion

Digital signal (숫자형태의 신호)

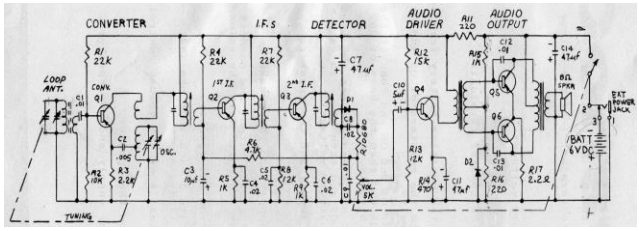
- 원래부터 digital 형태로 만들어진 신호
  - 예, 주가(주식의 가격) 등 통계 데이터
  - 컴퓨터로 생성된 신호 (sine wave sin(2*pi*n/N)
  - CMOS sensor 에서 얻어진 이미지 (원래 정해진 해상도가 있다)
- 아날로그 신호를 sampling 하여 digital 로 저장한 것
  - 음성
  - 많은 센서 신호

주식 가격 신호

- Moving average: 과거 5, 20, 60 일간의 평균

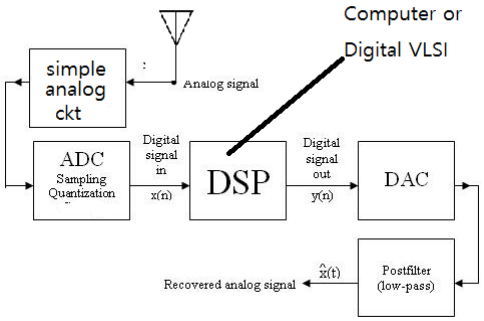
Analog and digital signal processing system

- Analog signal processing system
  - 옛날의 전축, 라디오, TV
  - 내부에 몇 개의 Transistor (신호증폭)과 L (inductor), C (capacitor)가 많이 들어감.
  - LC 공진회로
  - 지금은 Transistor 대신에 OP Amp 많이 사용

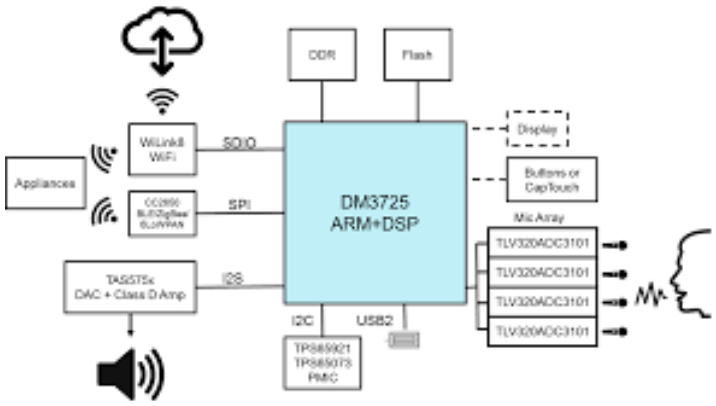


디지털 신호처리 시스템

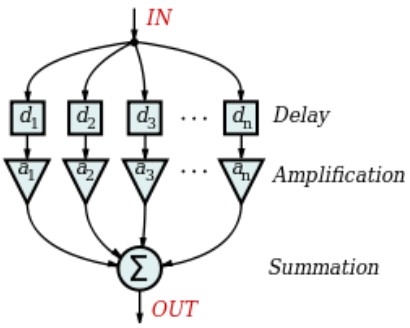
- ADC(Analog digital converter)를 이용하여 analog 신호를 숫자로 바꿈. 그 다음은 컴퓨터(SW)나 digital hardware (adder, memory) 로 처리



Amazon Echo block diagrm



Digital filter



- Bandpass, highpass, lowpass filter 오른쪽 그림과 같이
  - amplification (곱하기, *)
  - 합치기 (더하기, +)
  - 신호를 지연하기 (delay, memory)동작을 이용
- 즉 매우 빠른 컴퓨터만 있으면 라디오를 SW로 만들 수 있다.
  또는 digital VLSI 로 만들 수 있다.
- 현재 HDTV, Smartphone 을 이런 방식으로 만듦.
- 인공신경망도 비슷한 구조 (non-linear activation unit이 필요)

그럼 왜 그전에는 digital 방식을 사용하지 안하였는가?

- Digital 방식은 많은 수의 multiplication, addition 을 필요로 하는데, 이 필요한 회수가 sampling 주파수 (즉, 신호가 오는 주파수)에 비례한다. 그렇기 때문에 고주파 신호처리에는 매우 불리하였다.

승산기(hardware multiplier)의 가격)

- 30년전 TRW 16bit*16bit multiplier, 10MHz 동작
  – 당시 자동차 한대 가격
  (OP amp: 짜장면 가격)

- 현재: 하나의 chip 안에 multiplier (동작 주파수 100MHz 이상) 가격: 10만원 이내
  - Moore의 법칙에 의해서 이렇게 됨.
- 30년전에는 정교한 신호처리를 위해서 digital signal processing 을 사용했다면, 지금은 대부분 cheap signal processing 이 목적이다.

디지털 신호의 예: 주기 N=8 인 sine wave: sine(2pi*n/N)

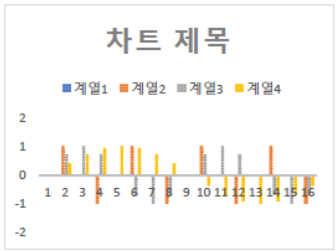
- 값이 N=8 의 주기로 변한다.
- 한 sample 동안에 sine wave 가 2*pi/N radian 만큼 변한다.

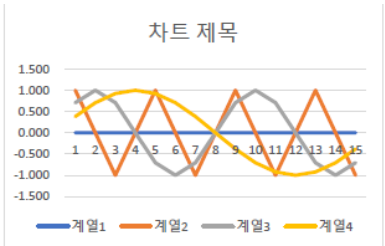
8	0	0.707	1.000	0.707	0.000	-0.707	-1.000	-0.707	0.000	0.707	1.000	0.707	0.000	-0.707	-1.000	-0.707

디지털 신호와 주파수

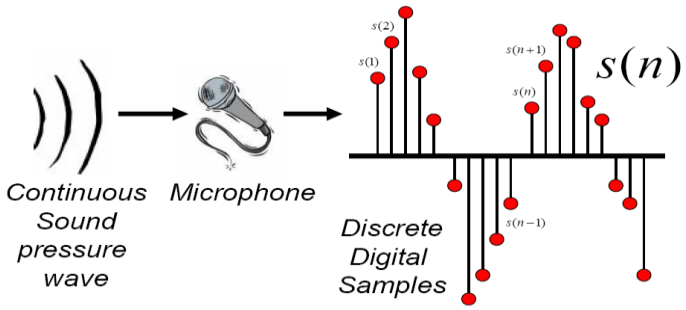
- sin(2*pi*n/N), cos(2*pi*n/N)
  - 주기가 N 인 sine wave. 주파수(rad/sample) 2*pi/N

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0	1.000	0.000	-1.000	0.000	1.000	0.000	-1.000	0.000	1.000	0.000	-1.000	0.000	1.000	0.000	-1.000
8	0	0.707	1.000	0.707	0.000	-0.707	-1.000	-0.707	0.000	0.707	1.000	0.707	0.000	-0.707	-1.000	-0.707
16	0	0.383	0.707	0.924	1.000	0.924	0.707	0.383	0.000	-0.383	-0.707	-0.924	-1.000	-0.924	-0.707	-0.383





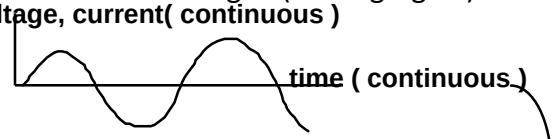
Analog to digital conversion (sampling and quantization)



Analog와 디지털 신호

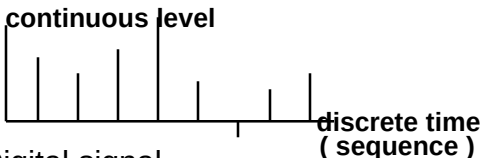
- Continuous time signal(analog signal)

voltage, current(continuous)

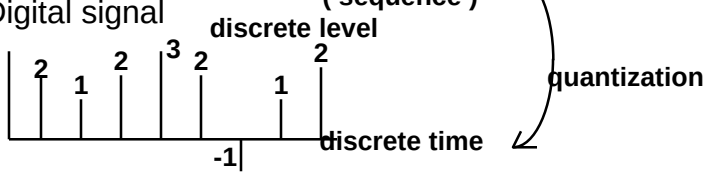


- Discrete time signal

sampling

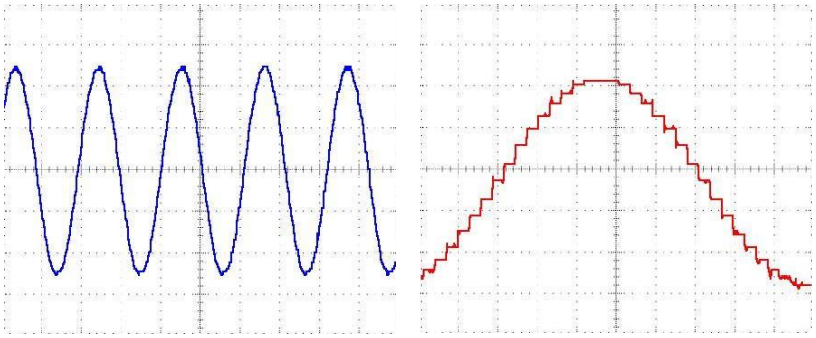


- Digital signal

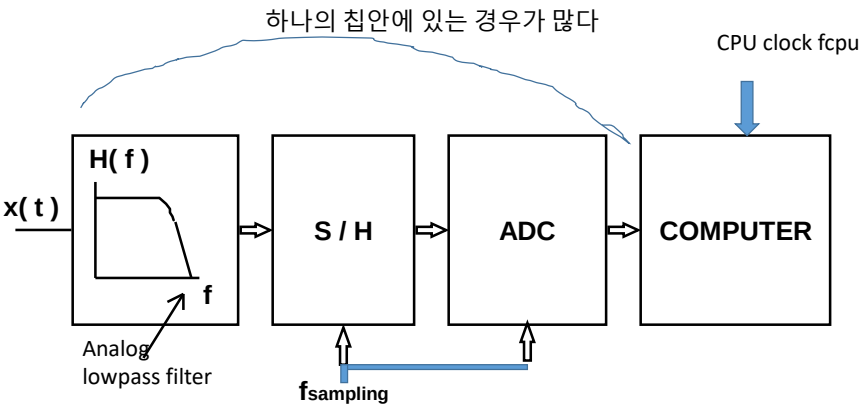


Analog to digital conversion

- Converting a continuously changing waveform (analog) into a series of discrete levels (digital)



ADC 과정



fsampling은 신호를 sampling 하는 주파수이고, fcpu는 CPU 동작속도로 서로 관련이 없다. 단 SW구현을 위해서는 fcpu >> fsampling이 필요하다. 보통 fcpu는 10MHz~3GHz, fsampling은 수 KHz ~ 수십MHz

TLV320AIC3101

- Stereo ADC (Analog Digital Converter)

TEXAS INSTRUMENTS

TLV320AIC3101

SLAS520E – FEBRUARY 2007 – REVISED DECEMBER 2014

TLV320AIC3101 Low-Power Stereo Audio Codec for Portable Audio/Telephony

1 Features

- Stereo Audio DAC
  - 102-dBA Signal-to-Noise Ratio
  - 16/20/24/32-Bit Data
  - Supports Sample Rates From 8 kHz to 96 kHz
  - 3D/Bass/Treble/EQ/De-Emphasis Effects
  - Flexible Power Saving Modes and Performance are Available
- Stereo Audio ADC
  - 92-dBA Signal-to-Noise Ratio
  - Supports Sample Rates From 8 kHz to 96 kHz
  - Digital Signal Processing and Noise Filtering Available During Record
- Extensive Modular Power Control
- Power Supplies:
  - Analog: 2.7 V–3.6 V.
  - Digital Core: 1.525 V–1.95 V
  - Digital I/O: 1.1 V–3.6 V
- Package: 5-mm × 5-mm 32-Pin QFN

2 Applications

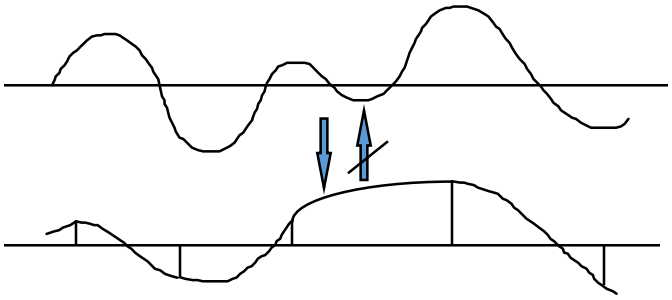
- Digital Cameras
- Smart Cellular Phones

3 Description

The TLV320AIC3101 is a low-power stereo audio codec with stereo headphone amplifier, as well as

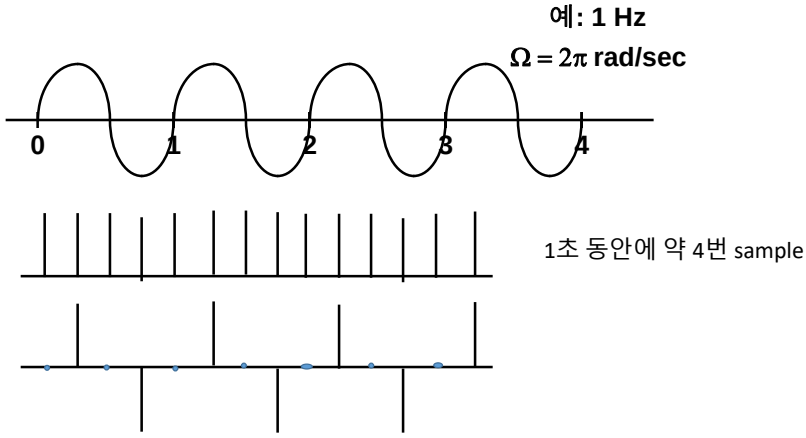
샘플링 주파수의 결정(f_{sample}, 1/T)

T를 키우면 x[n]의 양이 줄어든다.

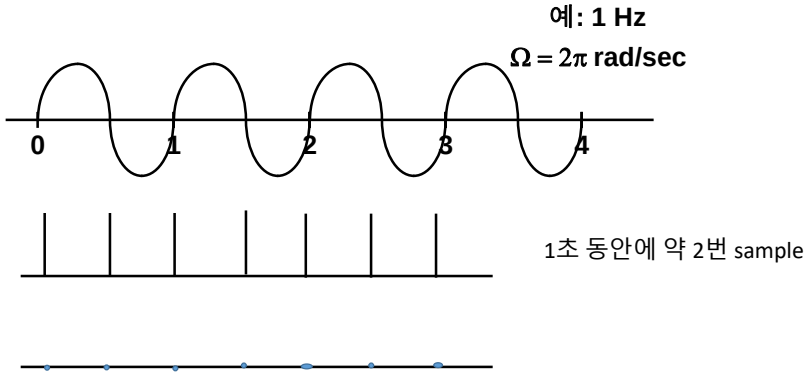


그러나 T를 무한정 크게 할 수는 없다.
aliasing problem 이 있다.

Sine wave 가 들어올 때의 sampling



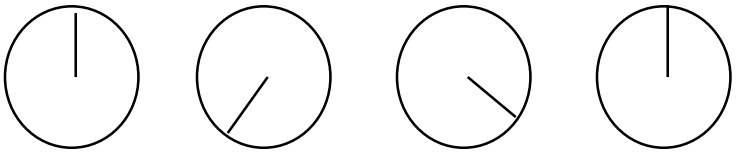
Sine wave 가 들어올 때의 sampling



Sampling

- **한 sample 동안에 신호의 phase가 π보다 더 많이 바뀌면 샘플링이 잘 안된다.**

(1/2 주기, + -, +)



- 2/3 π or 4/3 π ?

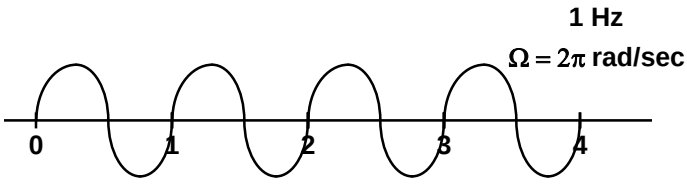
Nyquist sampling theorem

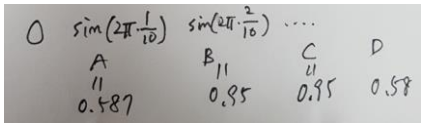
- Aliasing 을 막기 위해서는 input signal 의 maximum frequency 성분이 –fsample/2 ~ fsample/2 사이에 있어야 한다. 즉 1MHz 로 sampling을 하려면, 입력의 최대 주파수가 0.5MHz 이내여야 한다.
- 어떤 입력 신호의 최대 주파수가 fmax 이면 fsampling > 2*fmax 여야 한다.

The concept of frequency in the discrete time domain

continuous . . . Hz, cycle / sec, rad / sec ; sec(절대적 시간)

Discrete: sin(2*pi/N), cos(2*pi/N)





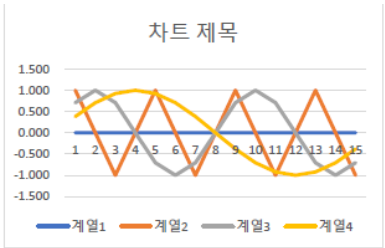
왼쪽과 같이 digital 로 보이는 값의 주파수는?
π/5 rad/sample
또는 그냥 π/5

즉, 샘플된 신호의 원래 주파수 (Hz)는 알 수가 없고, 단지 sample 된 domain 에서의 변화만 안다.

디지털 신호와 주파수

- sin(2*pi/N), cos(2*pi/N)
  - 주기가 N 인 sine wave.

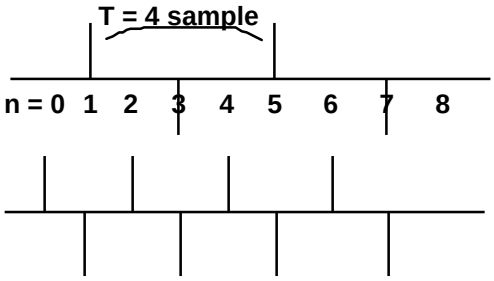
n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0	1.000	0.000	-1.000	0.000	1.000	0.000	-1.000	0.000	1.000	0.000	-1.000	0.000	1.000	0.000	-1.000
8	0	0.707	1.000	0.707	0.000	-0.707	-1.000	-0.707	0.000	0.707	1.000	0.707	0.000	-0.707	-1.000	-0.707
16	0	0.383	0.707	0.924	1.000	0.924	0.707	0.383	0.000	-0.383	-0.707	-0.924	-1.000	-0.924	-0.707	-0.383



The Concept of frequency in the discrete time domain

Discrete . . . cycle / sample, rad / sample

; 절대적 시간 대신 순서만 존재한다.



$\omega = \pi / 2$ **rad / sample**

$\omega = \pi$ **rad /sample**

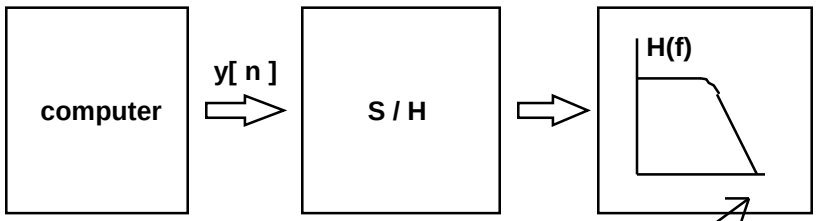
*** Discrete time domain에서 frequency는 -π에서 π까지만 생각하면된다.**

몇 bit ADC 가 필요?

- N bit -> 2^N levels
  - 8bit -> 256 levels
  - 16bit -> 65,536 levels
- 일반적으로 audio 에는 16비트 이상, 전화에는 12 비트 이상, video 에는 8비트 이상을 사용한다. Printer는 1bit (black and white dot)로 나타낸다.
- N 이 크면 – 양자화 잡음 (quantization noise) 작다.
- ADC 가격이 비싸다. 저장에 용량이 더 필요하다. DSP 과정에서 더 높은 bit의 multiplier adder memory 가 필요하다.

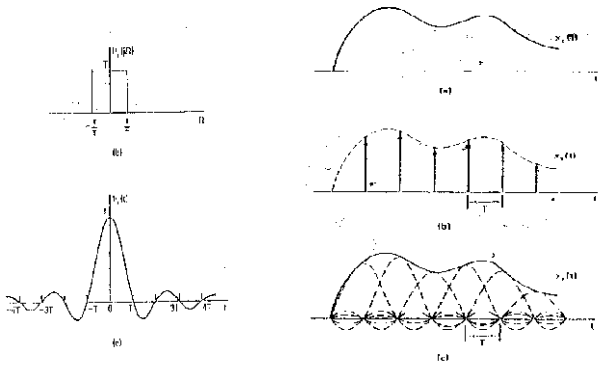
Discrete to continuous conversion

. . . low pass filtering



$$f_c = \frac{f_{sampling}}{2}$$

Discrete to continuous conversion(continued)



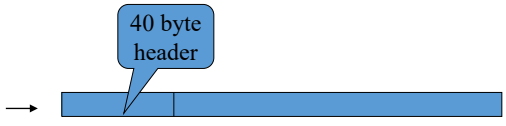
Practical system의 예

- Audio: 인간 귀의 청력 한계는 약 20KHz이다.
- Hifi audio의 경우: CD fsampling = 44.1KHz
- MP3 = 48k, 44.1K and 32kHz 등
- 전화: fsampling = 8KHz → wideband coding 16KHz

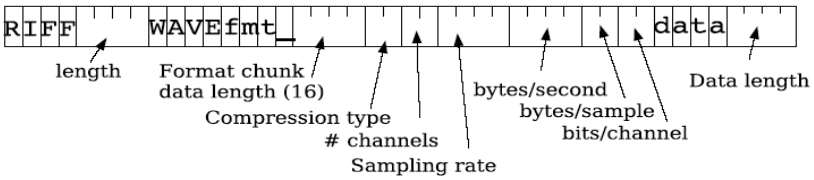
- Image의 경우: printer: 1200dpi (dot per inch)
- iPhone7 LCD: 1334 x 750 픽셀 **해상도**(326ppi)

Digitizing Speech (II)

- **Quantization**
  - Representing real value of each amplitude as integer
  - 8-bit (-128 to 127) or 16-bit (-32768 to 32767)
- **Formats**:
  - 16 bit PCM
  - 8 bit mu-law; log compression
- LSB (Intel) vs. MSB (Sun, Apple)
- **Headers**:
  - Raw (no header)
  - Microsoft wav
  - Sun .au



WAV format



1/5/07

LIBROSA

- *LibROSA* is a python library that has almost every utility you are going to need while working on audio data. Quick light on the features —

1. *Loading and displaying characteristics of an audio file.*
2. *Spectral representations*
3. *Feature extraction and manipulation*
4. *Time-Frequency conversions*
5. *Temporal Segmentation*
6. *Sequential Modeling…etc*

- Installation
- **pypi** : pip install librosa
  conda : conda install -c conda-forge librosa

- librosa.load() —> function returns two things
- 1. An array of amplitudes.
- 2. Sampling rate. If you keep the argument sr = None , it will load your audio file in its original sampling rate. (Note: libROSA can upsample or downsample the signal for you).

Loading Audio

```
file_path = "1995-1826-0003.wav"
samples , sampling_rate = librosa.load(file_path, sr = None, mono = True,
                                       offset = 0.0, duration = None)
len(samples), sampling_rate
```

executed in 6ms, finished 15:54:24 2020-01-08

```
(49440, 16000)
```

Duration

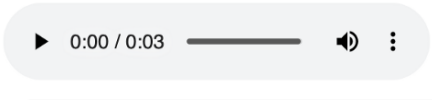
```
duration_of_sound = len(samples)/ sampling_rate
print (duration_of_sound, " seconds")
```

executed in 3ms, finished 15:54:37 2020-01-08

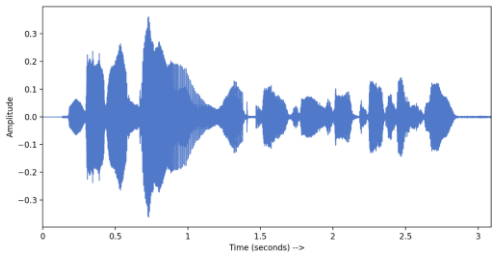
```
3.09  seconds
```

```
from IPython.display import Audio
Audio(file_path)
```

executed in 19ms, finished 00:14:49 2020-01-10




```
from librosa import display
plt.figure()
librosa.display.waveplot(y = samples, sr = sampling_rate)
plt.xlabel("Time (seconds) -->")
plt.ylabel("Amplitude")
plt.show()
```



Digital storage

- CD는 audio 신호를 44.1KHz로 sample 한 후 16bit (2byte)로 ADC한다. 한시간의 음악을 녹음해 두는데 저장장치 얼마만큼이 필요한가?
  - 답: 2byte*44.1K*3600 = 317Mbyte
- 지금 사용하는 LCD 화면의 해상도는 1680*1050이다. 그리고 한 픽셀은 32비트(RGB를 나타냄)를 사용한다. 하나의 화면 버퍼로 필요한 DRAM 용량은 얼마인가? =10.08Mbyte
- 화면의 수평, 수직 해상도가 모두 두배씩이 된다면 화면 버퍼로 필요한 DRAM의 양은? 위의 4배.
- HDTV를 만들기 위해서는 큰 DRAM buffer 필요.

Digital compression

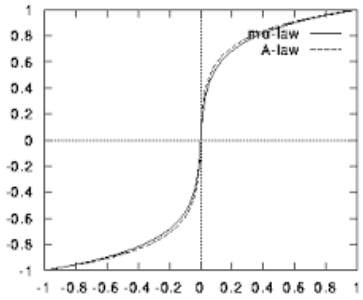
- CD는 audio 신호를 44.1KHz로 sample 한 후 16bit (2byte)로 ADC한다. 한시간의 음악을 녹음해 두는데 저장장치 얼마만큼이 필요한가?
  - 답: 2byte*44.1K*3600 = 317Mbyte
- Flash memory 32Mbyte가 있다. 곡을 몇시간이나 넣어둘 수 있는가? Uncompressed: 약 6분
- MP3는 이 오디오 신호를 compress 해서 32Mbyte flash memory 에 곡을 한시간 이상 저장토록 하였다. MP3는 오디오 압축 표준.
- 어떻게 압축: 신호의 성질을 이용.

Audio (음성)용 ADC, DAC

- 음성전화용 – $f_{sampling}$ 8KHz
- 광대역 전화용 - $f_{sampling}$ 16KHz
- MP3 audio 등 - $f_{sampling}$ 16KHz ~ 48KHz

- Bit 수 – 전화용 12bit, Audio 용 16~24bit

8-bit μ-Law companding



Discrete Fourier Transform

Jean-Baptiste Joseph **Fourier**
1768~1830

Wonyong Sung

Why Fourier transform is important

- 인간은 음성의 waveform 을 보고서 판단하지 못한다. 음성의 시간적인 waveform 은 반사등의 영향으로 phase distortion 이 많다. Phase distortion 은 음성의 waveform 모양을 많이 바꾼다. 그렇지만 frequency domain 의 주파수별 에너지 (spectrum)는 안 바꾼다.
- 귀의 와우(cochlear)가 time-domain waverform 을 frequency domain 으로 바꾼다.

Fourier Transform

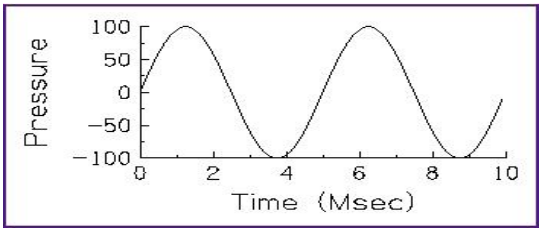
- Property of transforms:
  - They convert a function from one domain to another with no loss of information
- Fourier Transform: converts a function from the time (or spatial) domain to the frequency domain

$$F(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\omega t}\, dt$$

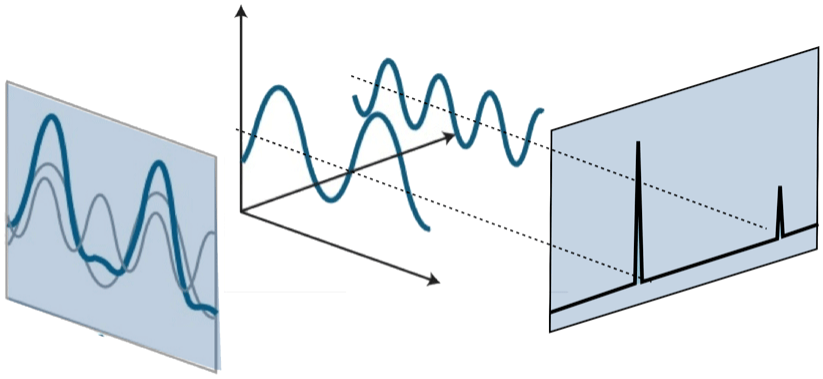
$\cos \omega t - i \sin \omega t$

Time Domain and Frequency Domain

- Time Domain:
  - Tells us how properties (air pressure in a sound function, for example) change over time:

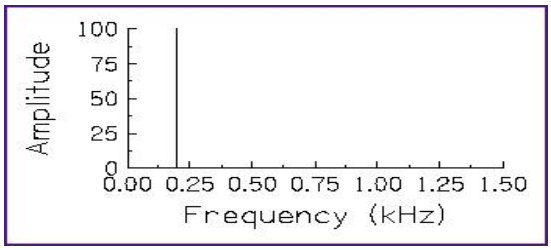


  - Amplitude = 100
  - Frequency = number of cycles in one second = 200 Hz



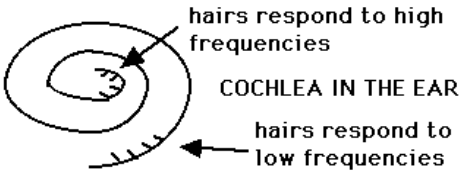
Time Domain and Frequency Domain

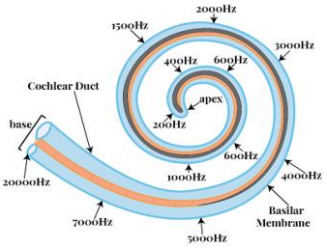
- Frequency domain:
  - Tells us how properties (amplitudes) change over frequencies:



Time Domain and Frequency Domain

- Example:
  - Human ears do not hear wave-like oscillations, but constant tone

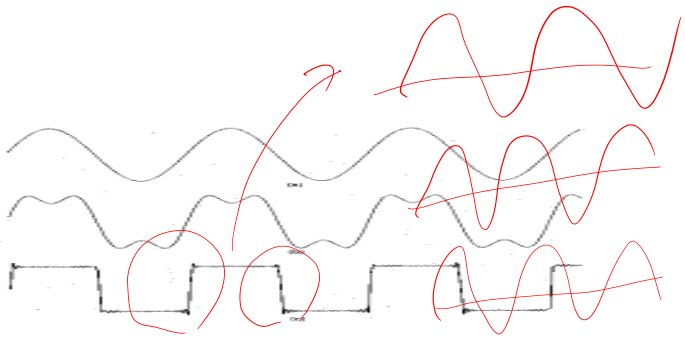




- Often it is easier to work in the frequency domain

Time Domain and Frequency Domain

- In 1807, Jean Baptiste Joseph Fourier showed that any periodic signal could be represented by a series of sinusoidal functions



In picture: the composition of the first two functions gives the bottom one

	Hertz	cpspch	Half-Step	MIDI Note
c 5	523.25	9.00	3	72
b 4	493.88	8.11	2	71
a# 4	466.16	8.10	1	70
a 4	440.0	8.09	0	69
g# 4	415.3	8.08	-1	68
g 4	391.1	8.07	-2	67
f# 4	369.99	8.06	-3	66
f 4	349.23	8.05	-4	65
e 4	329.63	8.04	-5	64
d# 4	311.13	8.03	-6	63
d 4	293.66	8.02	-7	62
c# 4	277.18	8.01	-8	61
c 4	261.63	8.00	-9	60

Fourier Transform

- Because of the property:

EULER's FORMULA

$$e^{i\theta} = \cos\theta + i\sin\theta$$

$$e^{i\omega t} = \cos\omega t + i\sin\omega t$$

$$\text{where } i = \sqrt{-1}$$

- Fourier Transform takes us to the frequency domain:

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-i\omega t}\, dt$$

the Fourier transform; strength of frequency ω contained in f(t)

scale factor for the Fourier Transform F(ω); the original signal in the time domain; the "inverse Fourier transform".

sinusoidally varying "basis" function for the expansion

Discrete Fourier Transform

- In practice, we often deal with discrete functions (digital signals, for example)
- The input is of limited length (n)
- Discrete version of the Fourier Transform is much more useful in computer science:

$$f_j = \sum_{k=0}^{n-1} x_k e^{-\frac{2\pi i}{n} jk} \qquad j = 0, \ldots, n-1$$

- $O(n^2)$ time complexity

두 신호의 correlation

- Sum(n=0, N-1) x(n)*y(n)
- 두개의 같은 주파수의 sine wave를 곱한 후 평균하면 ½ 이 된다.
- 두개의 다른 주파수의 sine wave를 곱한 후 평균하면 0 이 된다.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
2	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	
4	0	1.000	0.000	-1.000	0.000	1.000	0.000	-1.000	0.000	1.000	0.000	-1.000	0.000	1.000	0.000	-1.000	
8	0	0.707	1.000	0.707	0.000	-0.707	-1.000	-0.707	0.000	0.707	1.000	0.707	0.000	-0.707	-1.000	-0.707	
16	0	0.383	0.707	0.924	1.000	0.924	0.707	0.383	0.000	-0.383	-0.707	-0.924	-1.000	-0.924	-0.707	-0.383	
x4*x4		1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000	1.000	8.000
x4*x8		0.707	0.000	-0.707	0.000	-0.707	0.000	0.707	0.000	0.707	0.000	-0.707	0.000	-0.707	0.000	0.707	0.000
x4*16		0.3827	5E-07	-0.924	-1E-06	0.9239	1E-06	-0.383	-2E-12	-0.383	-2E-06	0.9239	4E-06	-0.924	-3E-06	0.3827	0.000

Discrete Fourier Transform

In practice, we often deal with discrete functions
The input is of limited length (N)
Discrete version of the Fourier Transform is much more useful in computer science:

Let $x[n]$ be an N-point signal, and W_N be the N^{th} root of unity. The N-point discrete Fourier Transform of $x[n]$, denoted $X(k) = \text{DFT}\{x[n]\}$, is defined as

$$W_N = \exp\left(-\frac{j2\pi}{N}\right)$$

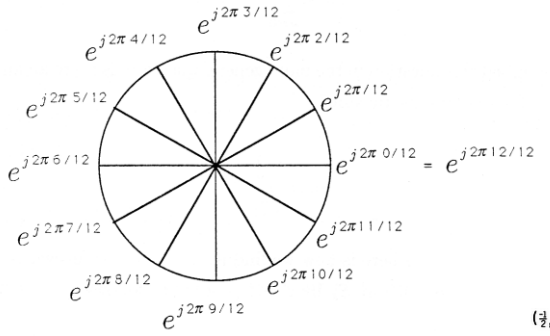
$$X(k) = \sum_{n=0}^{N-1} x[n]\, W_N^{kn}, \quad 0 \le k \le N-1$$

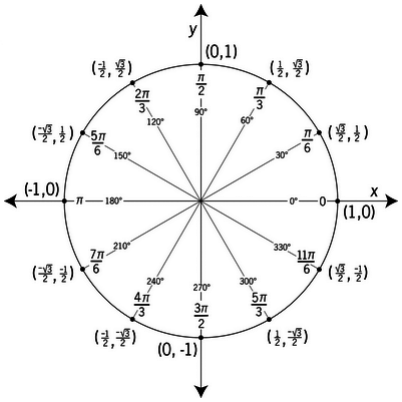
$$= \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi kn}{N}}$$

W_N^k N^{th} Root of Unity

$$W_N = \exp\left(-\frac{j2\pi}{N}\right)$$

1) $W_N^{N/4} = j$
2) $W_N^{N/2} = -1$
3) $W_N^{3N/4} = j$
4) $W_N^{N} = 1$
5) $W_N^{k+N} = W_N^{k}$
6) $W_N^{k+(N/2)} = -W_N^{k}$
7) $W_N^{2k} = W_{N/2}^{k}$
8) $W_N^{*} = W_N^{-1}$





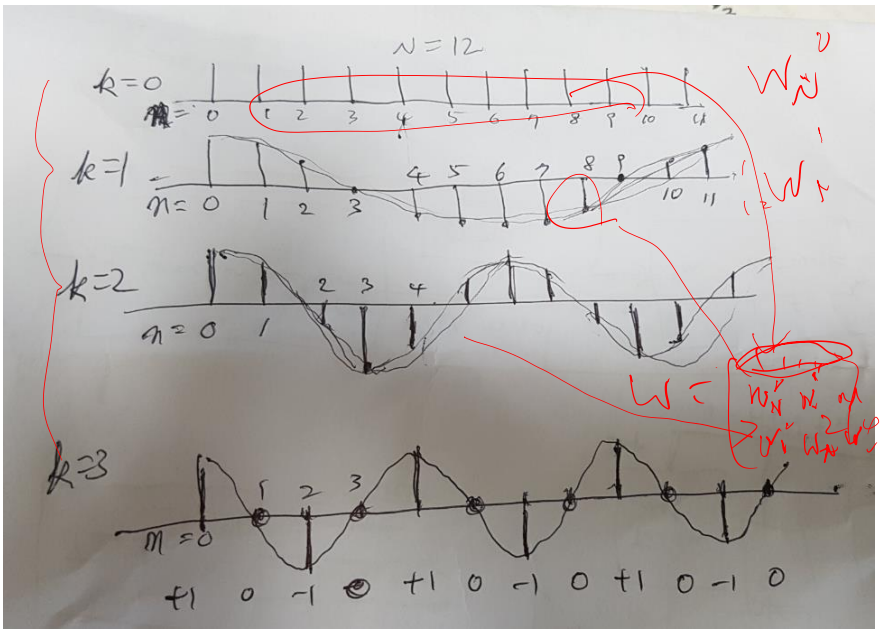
Matrix Formulation

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ \vdots \\ X(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N^1 & W_N^2 & \cdots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \\ x[N-1] \end{bmatrix}$$

$$\underline{X} = W\underline{x}$$

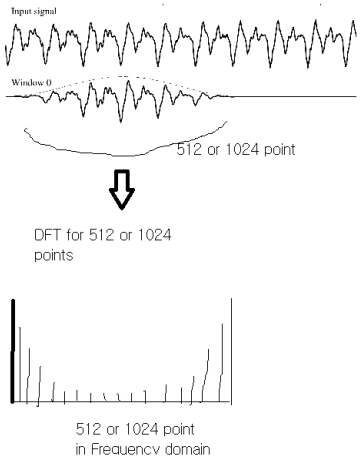
주기가 N인 discrete (cosine, sine) sequence (한 sample에 $2\pi/N$ 씩 진전

주기가 N/2인 discrete (cosine, sine) sequence



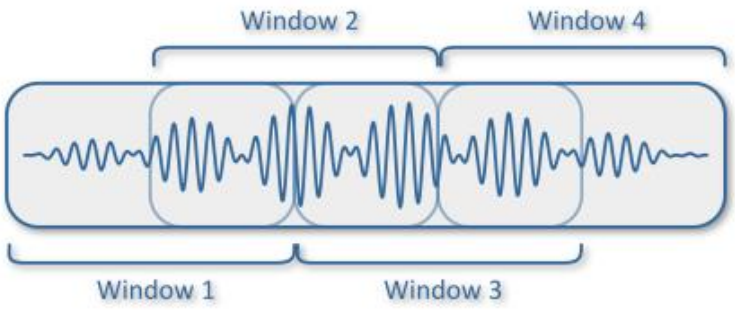
연속적으로 들어오는 신호의 Fourier transform (short-time Fourier transform)

- Windowing and DFT (discrete Fourier transform)



Real-time signal processing

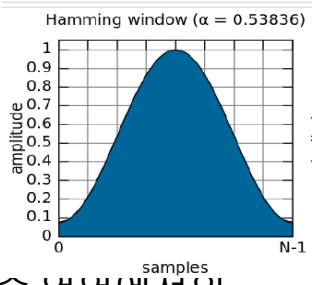
- Sampling -> sliding window -> DFT for each window -> Plot



Windowing

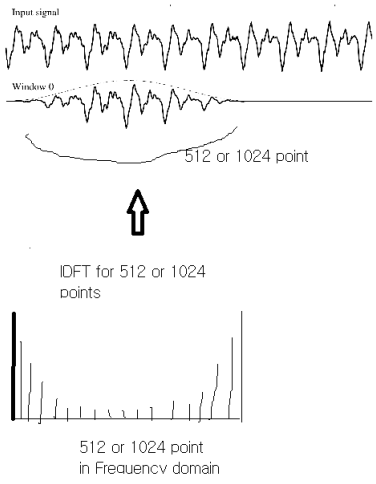
- x[n]의 길이가 N보다 길 때, 이를 N으로 맞추어야 한다.
- Hamming window 가 유명

$$w_0(n) \stackrel{\text{def}}{=} w(n + \tfrac{N-1}{2}) = 0.54 + 0.46 \cos\left(\frac{2\pi n}{N-1}\right)$$



- Rectangular window의 경우 주파수 영역에서의 찌그러짐이 있다.

Inverse DFT: from frequency domain to time(sampled) domain



Inverse Discrete Fourier Transform

Let $X(k)$ be an N-point DFT sequence, and W_N be the N^{th} root of unity. The N-point inverse discrete Fourier Transform of $X(k)$, denoted $x[n] = \text{IDFT}\{X(k)\}$, is defined as

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\, W_N^{-kn}, \quad 0 \le n \le N-1$$

$$= \frac{1}{N}\sum_{k=0}^{N-1} X(k)\, e^{j\frac{2\pi kn}{N}}$$

Matrix Formulation

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ \vdots \\ X(N-1) \end{bmatrix} = \frac{1}{N}\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N^{-1} & W_N^{-2} & \cdots & W_N^{-(N-1)} \\ 1 & W_N^{-2} & W_N^{-4} & \cdots & W_N^{-2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{-(N-1)} & W_N^{-2(N-1)} & \cdots & W_N^{-(N-1)(N-1)} \end{bmatrix}\begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \\ x[N-1] \end{bmatrix}$$

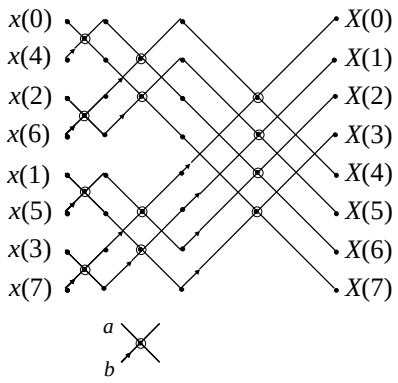
$$\underline{x} = \frac{1}{N} W^* \underline{X}$$

Faster DFT computation?

- Take advantage of the symmetry and periodicity of the complex exponential (let $W_N=e^{-j2\pi/N}$)
  - symmetry: $W_N^{k[N-n]} = W_N^{-kn} = (W_N^{kn})^*$
  - periodicity: $W_N^{kn} = W_N^{k[n+N]} = W_N^{[k+N]n}$
- Note that two length N/2 DFTs take less computation than one length N DFT: $2(N/2)^2<N^2$
- Algorithms that exploit computational savings are collectively called *Fast Fourier Transforms*

8-point FFT

- 8-point Signal Flow Diagram



66

FFT times

- Time (1 multiplication per microsec)

	N	Direct DFT	FFT
2^6	64	.02 sec	.002 sec
2^9	512	1	.02 sec
2^{12}	4096	67	.2
2^{15}	32768	1 hr 11 mins	2
2^{18}	262144	3 days 4 hrs	19

67

DFT Interpretation

DFT sample $X(k)$ specifies the magnitude and phase angle of the k^{th} spectral component of $x[n]$.

$$\text{Magnitude spectrum} = \frac{1}{\text{N}} |X(k)|$$

$$\text{Phase spectrum} = \angle X(k)$$

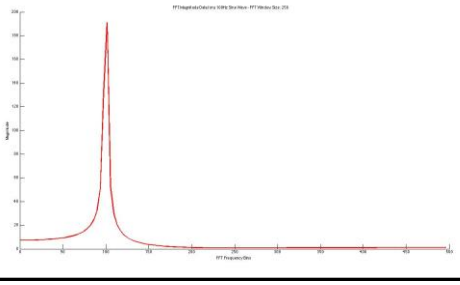
The amount of power that $x[n]$ contains at a normalized frequency, f_k, can be determined from the **power density spectrum** defined as

$$P_N(k) = \frac{|X(k)|^2}{N^2}, \quad 0 \le k \le N-1$$

Complex number 의 magnitude 는 real term 과 imaginary term을 각각 제곱하여 더 한후 square root를 씌운다

DFT의 bin number 와 실제 주파수

- 어떤 signal x(t)를 10KHz로 sample 하였다. 그리고 sampling 된 신호 x[0] ~ x[1023] 을 이용하여 1024 point FFT를 실행하였다.
- 아래 보이는 바와 같이 X[100]에 peak이 발생하였다.
- 몇 Hz에 해당?



DFT의 bin number 와 실제 주파수

- X[k = 0, 1, 2, … N-1]이 얻어지는데, 가상적으로 k = N이 sampling 주파수에 해당
- 즉 DFT domain에서 K=1이 변할 때 continuous time domain의 주파수는 fs/N 또는 rad/sec로 2π*fs/N 만큼씩 변한다.
- 앞의 문제에 대한 답은 100*10KHz/1024 ~=약 1KHz
- Sampling 하기 전에 신호가 lowpass filter fs/2를 거치며, k=N/2~ N-1은 k=0~N/2-1과 symmetric하다 (real input signal의 경우)

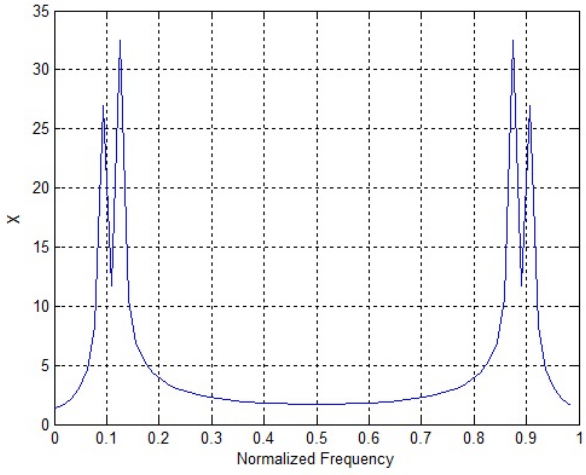
Matrix Formulation

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ \vdots \\ X(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N^1 & W_N^2 & \cdots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \\ x[N-1] \end{bmatrix}$$

$$\underline{X} = W\underline{x}$$

주기가 N인 discrete (cosine, sine) sequence (한 sample에 $2\pi/N$ 씩 진전

주기가 N/2인 discrete (cosine, sine) sequence

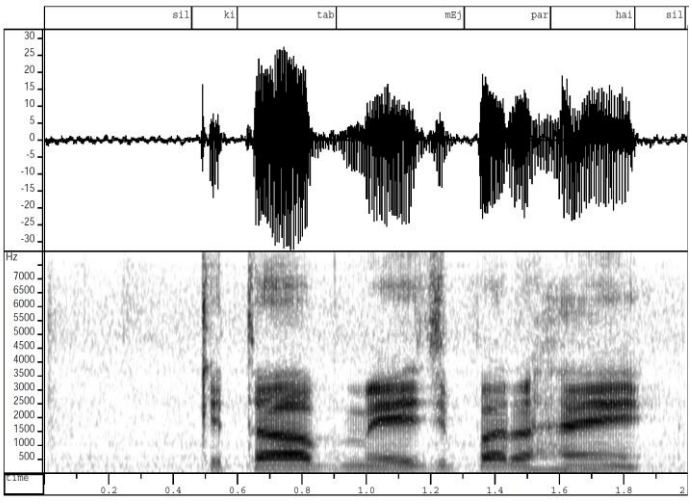


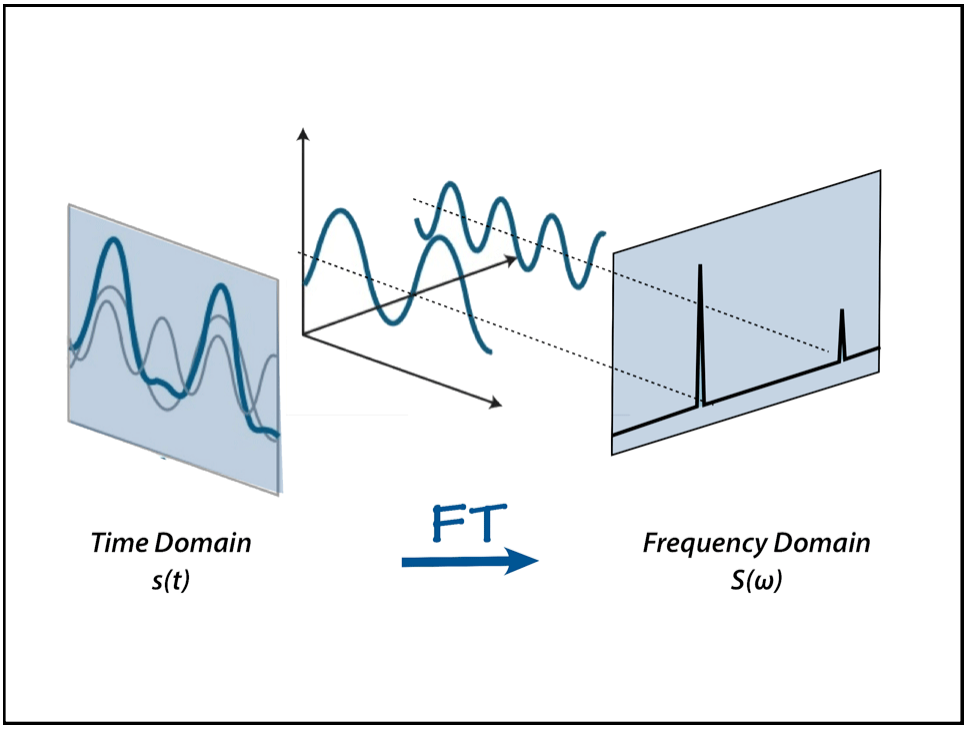
주파수 해상도를 매우 높이려면

- Input data x[n]에 0을 추가해서 한다.
- 길이 1024의 x[n] -> 1024 point FFT
  frequency 해상도 2pi/1024

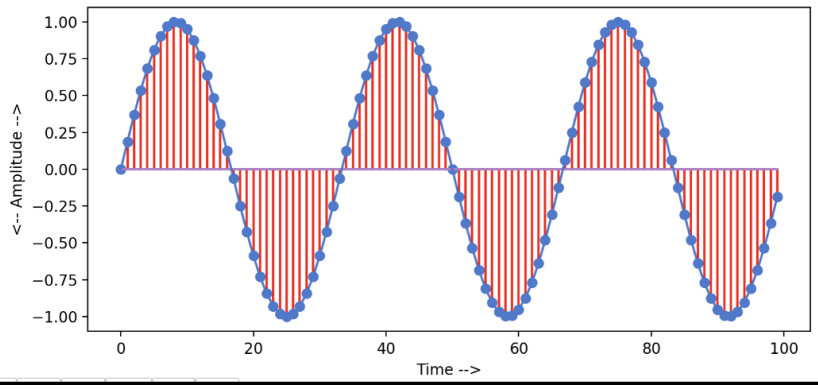
- 길이 1024 x[n] + 1024 0.0 -> 2048 point FFT
  frequency 해상도 2pi/2048

Spectrogram
sampl된 입력신호-> sliding window -> FFT

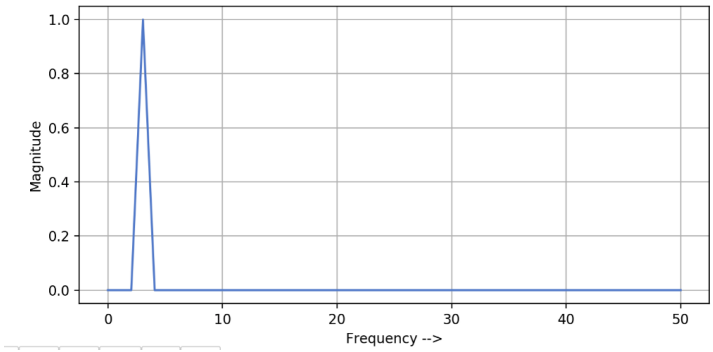


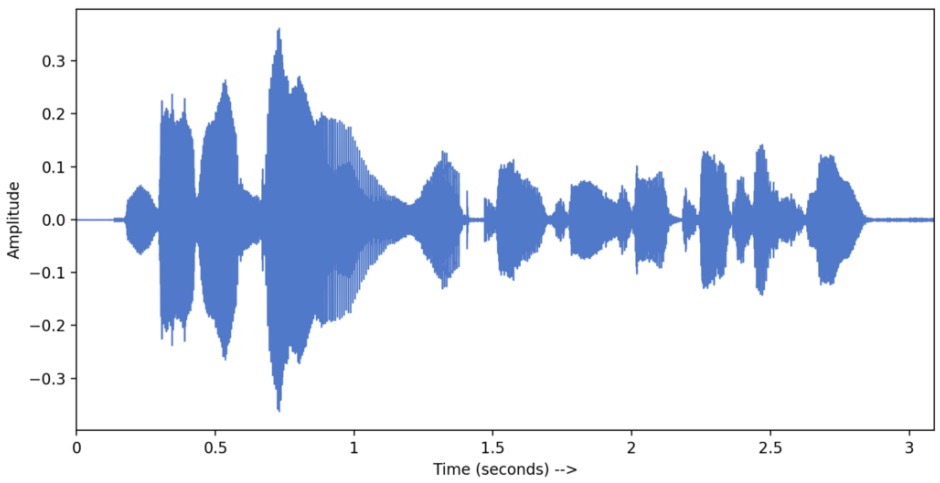



```
samples = 100
f = 3
x = np.arange(samples)
y1 = np.sin(2*np.pi*f * (x/samples))
plt.figure()
plt.stem(x,y1, 'r', )
plt.plot(x,y1)
plt.xlabel("Time -->")
plt.ylabel("<-- Amplitude -->")
plt.show()
```

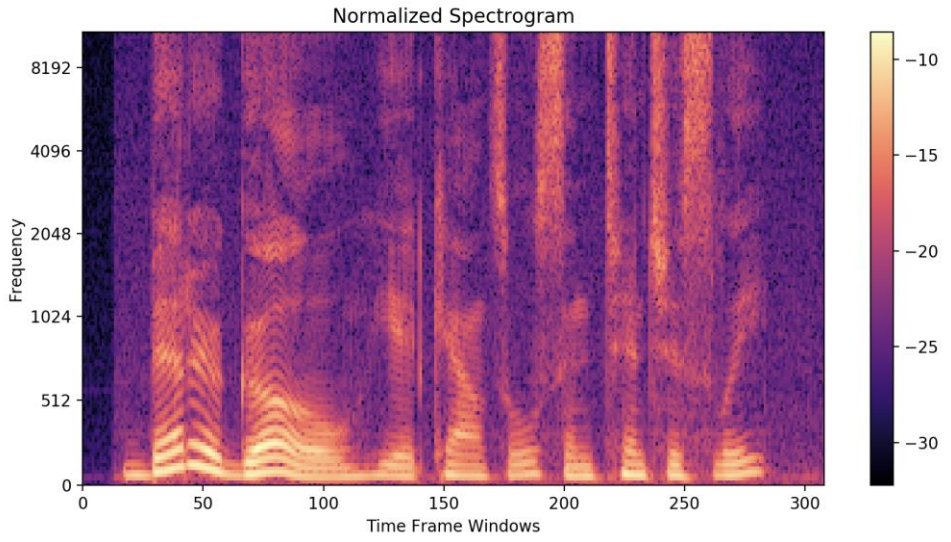


```python
import scipy
def fft_plot(audio, sampling_rate):
    n = len(audio)
    T = 1/sampling_rate
    yf = scipy.fft(audio)
    xf = np.linspace(0.0, 1.0/(2.0*T), n/2)
    fig, ax = plt.subplots()
    ax.plot(xf, 2.0/n * np.abs(yf[:n//2]))
    plt.grid()
    plt.xlabel("Frequency -->")
    plt.ylabel("Magnitude")
    return plt.show()
```



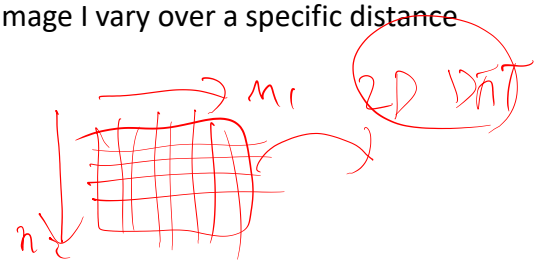


```
def spectrogram (samples, sample_rate, stride_ms = 10.0,  window_ms = 20.0, max_freq = None, eps = 1e-14):
    stride_size = int(0.001 * sample_rate * stride_ms)
    window_size = int(0.001 * sample_rate * window_ms)

    # Extract strided windows
    truncate_size = (len(samples) - window_size) % stride_size
    samples = samples[:len(samples) - truncate_size]
    nshape = (window_size, (len(samples) - window_size) // stride_size + 1)
    nstrides = (samples.strides[0], samples.strides[0] * stride_size)
    windows = np.lib.stride_tricks.as_strided(samples,
                                shape = nshape, strides = nstrides)

    assert np.all(windows[:, 1] == samples[stride_size:(stride_size + window_size)])

    # Window weighting, squared Fast Fourier Transform (fft), scaling
    weighting = np.hanning(window_size)[:, None]

    fft = np.fft.rfft(windows * weighting, axis=0)
    fft = np.absolute(fft)
    fft = fft**2

    scale = np.sum(weighting**2) * sample_rate
    fft[1:-1, :] *= (2.0 / scale)
    fft[(0, -1), :] /= scale

    # Prepare fft frequency list
    freqs = float(sample_rate) / window_size * np.arange(fft.shape[0])

    # Compute spectrogram feature
    ind = np.where(freqs <= max_freq)[0][-1] + 1
    specgram = np.log(fft[:ind, :] + eps)
    return specgram
```



Applications

- In image processing:
  - Instead of time domain: *spatial domain* (normal image space)
  - *frequency domain:* space in which each image value at image position F represents the amount that the intensity values in image I vary over a specific distance related to F



Extending DFT to 2D (cont'd)

- Special case: f(x,y) is N x N.
- Forward DFT

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{ux+vy}{N})},$$

$$u,v = 0,1,2,\ldots, N-1$$

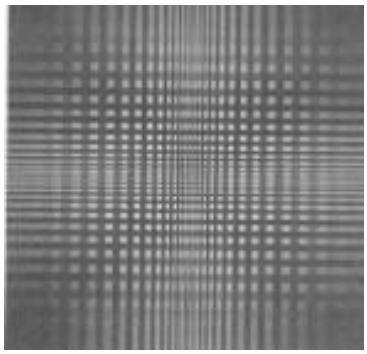
- Inverse DFT

$$f(x,y) = \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(\frac{ux+vy}{N})},$$

$$x,y = 0,1,2,\ldots, N-1$$

Applications: Frequency Domain In Images

- *Spatial frequency* of an image refers to the rate at which the pixel intensities change
- In picture on right:
  - High frequences:
    - Near center
  - Low frequences:
    - Corners



Digital Filter Design & Implementation

Wonyong Sung

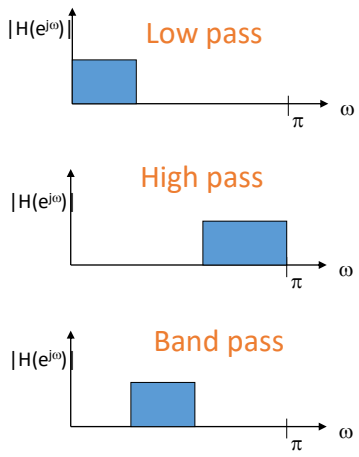
2 x 1.5 hrs

Digital Filters

- Types of Digital Filters:
  - low-pass, band-pass, high-pass, notch-filter, allpass, etc.
- FIR and IIR Digital Filters
- Filters for sampling rate conversion

Wonyong Sung
Multimedia Systems Lab SNU

Types of Digital Filters

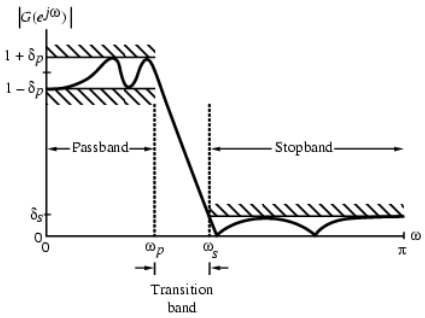


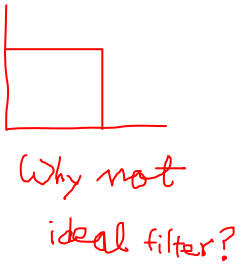
- Usages:
  - Low pass: anti-aliasing, smoothing, noise reduction
  - High pass: DC removal, baseline wander reduction
  - Band pass: noise reduction
- Design:
  - Choose FIR or IIR filter coefficients to approximate desired frequency response.
  - Usually the designed filter coefficients are not unique! Leaving large design space to be explored. Passband, stopband ripples.

Wonyong Sung
Multimedia Systems Lab SNU

Digital filter specifications

- For example the magnitude response $|G(e^{j\omega})|$ of a digital lowpass filter may be given as indicated below





* Transition bandwidth is important for filter order determination.

dB (decibel)

- 신호의 크기나 시스템의 증폭율 등을 나타내는 단위. 전력(power)의 비로 따진다: dB
- Bel 은 Bel of A = log10 A
  -> decibe은 bel의 10배 값
- 신호의 크기가 k 배 되면 power는 k^2 배가 된다.
- 어떤 amp를 지났더니 신호가 10배 되었다.
  - 이 경우 신호의 power는 100배가 되었다.
  - Bel 값은 log10 (100) = 2
  - dB 값은 20
  - 비슷하게 신호가 100배가 되면 40dB
  - 비슷하게 신호가 1/100이 되면 -40dB
  - 신호 크기가 1.12배가 되면, 20*(log10 1.12)=1dB

Digital filter specifications

Filter specification parameters

- ω_p- passband edge frequency
- ω_s- stopband edge frequency
- δ_p - peak ripple value in the passband
- δ_s - peak ripple value in the stopband

Wonyong Sung
Multimedia Systems Lab SNU

Filter 복잡도

- Wp 와 Ws의 차이를 transition bandwidth라 하며 이 사이가 좁을 수록 filter 의 차수가 높아야 한다 (복잡하다). 보통 0.05π 정도는 준다.
- Passband ripple은 0dB (이상적, 불가능한 값)이상으로 0.5 dB, 1 dB 등으로 준다. 이 값이 클 수록 필터의 ripple 은 크지만 차수가 낮아진다.
- Stopband ripple 또는 rejection 은 stopband 에서의 감쇄된 값으로 -40dB (amplitude로 1/100)에서 -80dB (1/10000)의 값을 가진다. 이 감쇄가 클 수록 차수가 높은 어려운 필터가 된다.

Wonyong Sung
Multimedia Systems Lab SNU

간단한 filter

- y[n] = x[n] – x[n-1]

- 이 필터는 어떤 특징을 가지는가?
- DC를 집어 넣으면, 즉 x[n] = x[n-1] 이 경우는 y[n]으로 0 이 나온다.
- X[n] = cos(πn)을 넣는다. (매우 높은 주파수 신호)
- y[n] = cos(πn)- cos(π(n-1))
  = cos(πn)- cos(πn-π) = 2 cos(πn)

High pass filter 가 된다.

일반적인 구조의 FIR filter

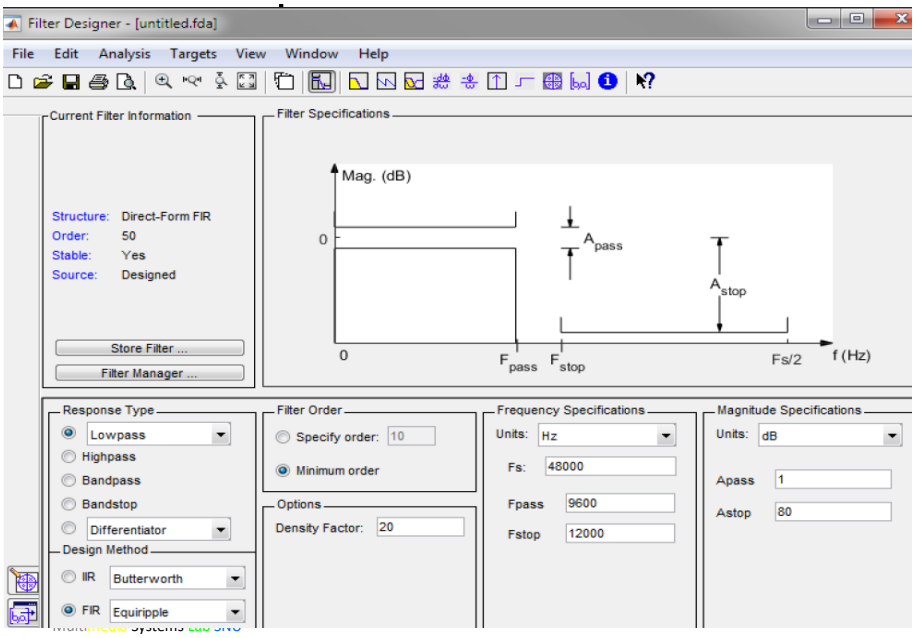
- FIR filter (Finite Impulse Response) filter: input signal 을 delay한 후 각각을 계수로 multiply 한 후 모두 더한다.
  - ❖ **The basic FIR Filter equation is**

$$y[n] = \sum h[k].x[n-k] \quad \text{for k = 0 to M-1}$$

where h[k] is an array of constants (filter coefficients)

M=3 인 경우
y[0] = h0*x[0]+h1*x[-1] +h2*x[-2]
y[1] = h0*x[1]+h1*x[0] +h2*x[-1]
y[2] = h0*x[2]+h1*x[1] +h2*x[0]
y[3] = h0*x[3]+h1*x[2] +h2*x[1]

```
//N 개의 출력 계산 y[0] ~ y[N-1]
For (n=0; n<N; n++) {
y[n]=0;
        For (k = 0;k<M;k++)
//inner loop
            y[n] = y[n] + h[k]*x[n-k];}
```



Filter design specification

- Filter structure: FIR filter direct form (무난)
- Filter의 종류: lowpass, high pass, bandpass 등
- Design method: Equiripple (CAD 방법)
- Sampling frequency: 현재 사용하는 system의 sampling frequency를 주어도 되고, Fs =1 로 두고 설계해도 됨. Fs =1에서 Fpass = 0.1인 것과 Fs = 10KHz에서 Fpass = 1KHz인 것은 동일한 digital filter
- Passband ripple (여기에서는 1dB), stopband attenuation (여기에서는 80dB)를 준다.
- 그러면 금방 filter order M 을 정하고 계수를 준다.

FIR Digital Filter Order Estimation

Kaiser's Formula:

$$N \cong \frac{-20\log_{10}(\sqrt{\delta_p \delta_s})}{14.6(\omega_s - \omega_p)/2\pi}$$

- ie *N* is inversely proportional to transition band width and not on transition band location
- 설계된 filter 의 order 가 너무 크다 (즉 계산 많이 든다). 그래서 줄여야 한다. 그러면 transition bandwidth를 늘리고, attenuation 을 작게 해야 한다 (80dB=> 60dB =>40dB)

95

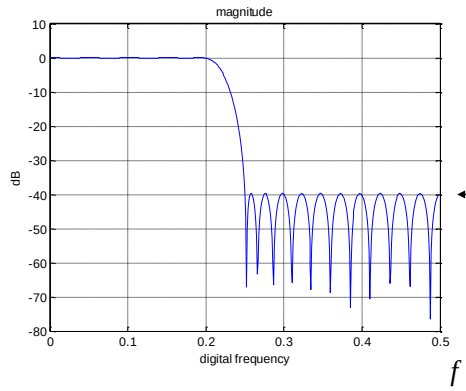
Example: Low Pass Filter

Passband $f = 0.2$

Stopband $f = 0.25$ with attenuation 40dB

Choose order N=40 > 37

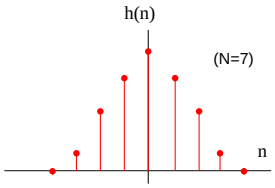
$|H(f)|$



OK!!!

Linear phase filter - symmetric FIR

- h(n) = h(-n)
- Evaluate the frequency response (assuming that N is odd) and h(n) is real-valued



$$H(z) = \sum_{n=n_1}^{n_2} h(n)\ z^{-n} = \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} h(n)\ z^{-n}$$

if $h(n) = h(-n)$ we get

$$H(e^{j2\pi\Omega}) = h(0) + \sum_{n=1}^{\frac{N-1}{2}} h(n)\ (e^{-j2\pi n\Omega} - e^{+j2\pi n\Omega})$$

$$H(e^{j2\pi\Omega}) = h(0) + 2\sum_{n=1}^{\frac{N-1}{2}} h(n)\ \cos[2\pi n\Omega]$$

The **frequency response is real**: phase shift is 0 or 180 degrees

Wonyong Sung
Multimedia Systems Lab SNU

T-filter

- Free on-line FIR filter design SW

http://t-filter.engineerjs.com/

- Sampling frequency는 주어진 최대 주파수의 두배로 가정하여 설계된다.
- 아래의 경우는 bandpass filter 인데, 최대주파수가 0.5Hz 즉 sampling freq 는 1.0 Hz
- Passband는 0.0125 ~ 0.01875

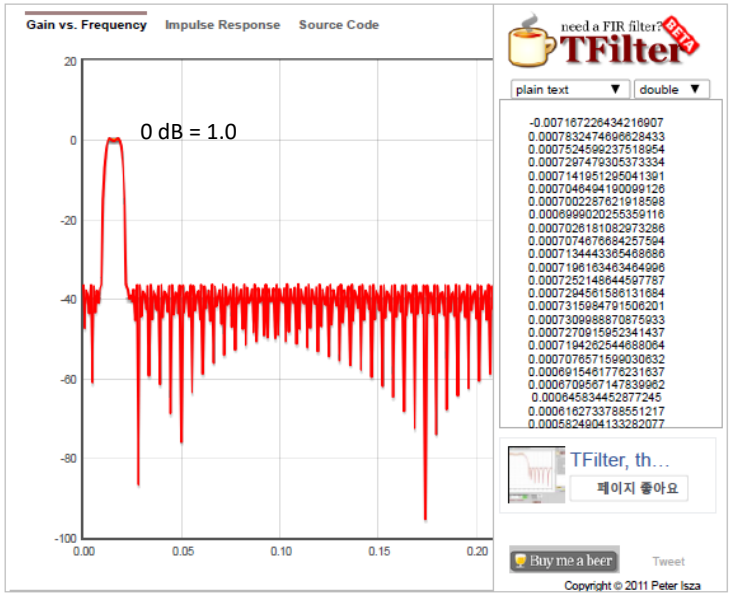
add passband add stopband !predefined ▼

from	to	gain	ripple/att.	act. rpl	
0 Hz	0.009375 H:	0	-40 dB	-36.41 dB	
0.0125 Hz	0.01875 Hz	1	0.5 dB	0.56 dB	
0.021875 H:	0.5 Hz	0	-40 dB	-36.41 dB	

sampling freq.	1 Hz
desired #taps	minimum
actual #taps	549

DESIGN FILTER

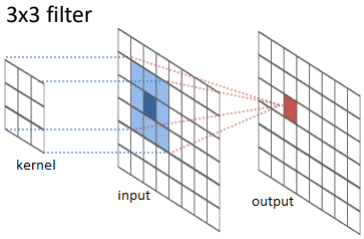
T



Wo
Mu...

2D FIR filter (2D convolution)

- 2 dimensional filtering: image 신호처리에 사용함.
- Input x[n1, n2] -> h[n1, n2] -> y[n1, n2]




```
//N² 개의 출력 계산 y[0,0] ~ y[N-1, N-1]
For (n1=0; n1<N; n1++) {
   For (n2=0; n2<N; n2++) {
y[n1, n2]=0.0;
            For (k1 = 0; k1<M; k1++) {
                        For (k2 = 0;k2<M;k2++)
//inner loop
                  y[n1, n2] = y[n1, n2] + h[k1,
k2]*x[n1-k1, n2-k2];}}}
```

Wonyong Sung
Multimedia Systems Lab SNU

Filtering rate change

- 16K sampling -> 8K sampling
- or 8K sampling -> 16 Ksampling

- 16K sampled signal -> Lowpass filtering with fstop = pi/2 or less -> downsample (take every other samples) -> 8K sampled signal

- 8K sampled signal -> upsampling with 0 for every other one -> lowpass filtering with fstop = pi/2 or less -> 16K sampled signal

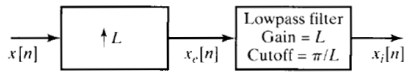


Figure 4.24 General system for sampling rate increase by L.

Thus, the Fourier transform of the output of the expander is a frequency-scaled version of the Fourier transform of the input; i.e., ω is replaced by ωL so that ω is now normalized by

$$\omega = \Omega T'. \tag{4.87}$$

This effect is illustrated in Figure 4.25. Figure 4.25(a) shows a bandlimited continuous-time Fourier transform, and Figure 4.25(b) shows the discrete-time Fourier transform of

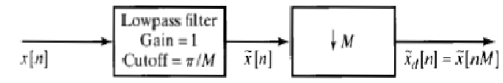


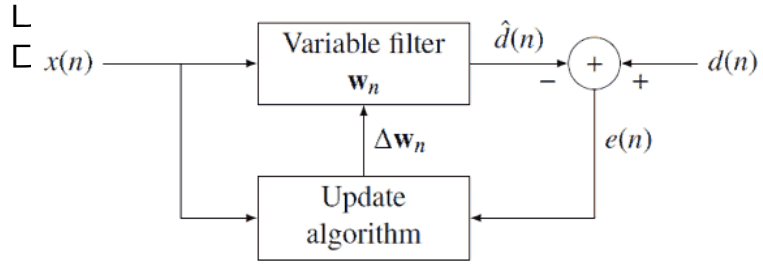
Figure 4.23 General system for sampling rate reduction by M.

From the preceding discussion, we see that a general system for downsampling by a factor of M is the one shown in Figure 4.23. Such a system is called a *decimator*, and downsampling by lowpass filtering followed by compression has been termed *decimation* (Crochiere and Rabiner, 1983).

Adaptive filter

- Filter 의 계수를 신호에 따라서 바꾼다. 즉, 환경 조건에 따라서 필터의 계수를 자동으로 바꾼다.
  - Adaptive noise cancelling
    - AirPods Pro noise cancelling

- ㄴ
  ㄷ



Summary

- Digital signal processing 은 digital 연산(승, 가산)과 memory (지연)를 이용하여 신호처리를 수행한다.
- SW를 이용하여 구현할 경우 매우 쉽게 시스템의 parameter 등을 바꿀 수 있으므로 유연한 시스템 설계가 가능하다.
- Sampling frequency 가 높은 고속 신호처리에는 계산량이 매우 많이 필요하였으나, 현재 고속의 컴퓨터를 이용하기 때문에 SW를 이용한 시스템 구현이 가능하다.
- Digital filter 는 FIR filter 와 IIR filter 등이 있다. 계수를 환경 조건에 따라 바꿀 수 있는 adaptive filter 도 있다.