[수치해석] Digital Signal Processing2

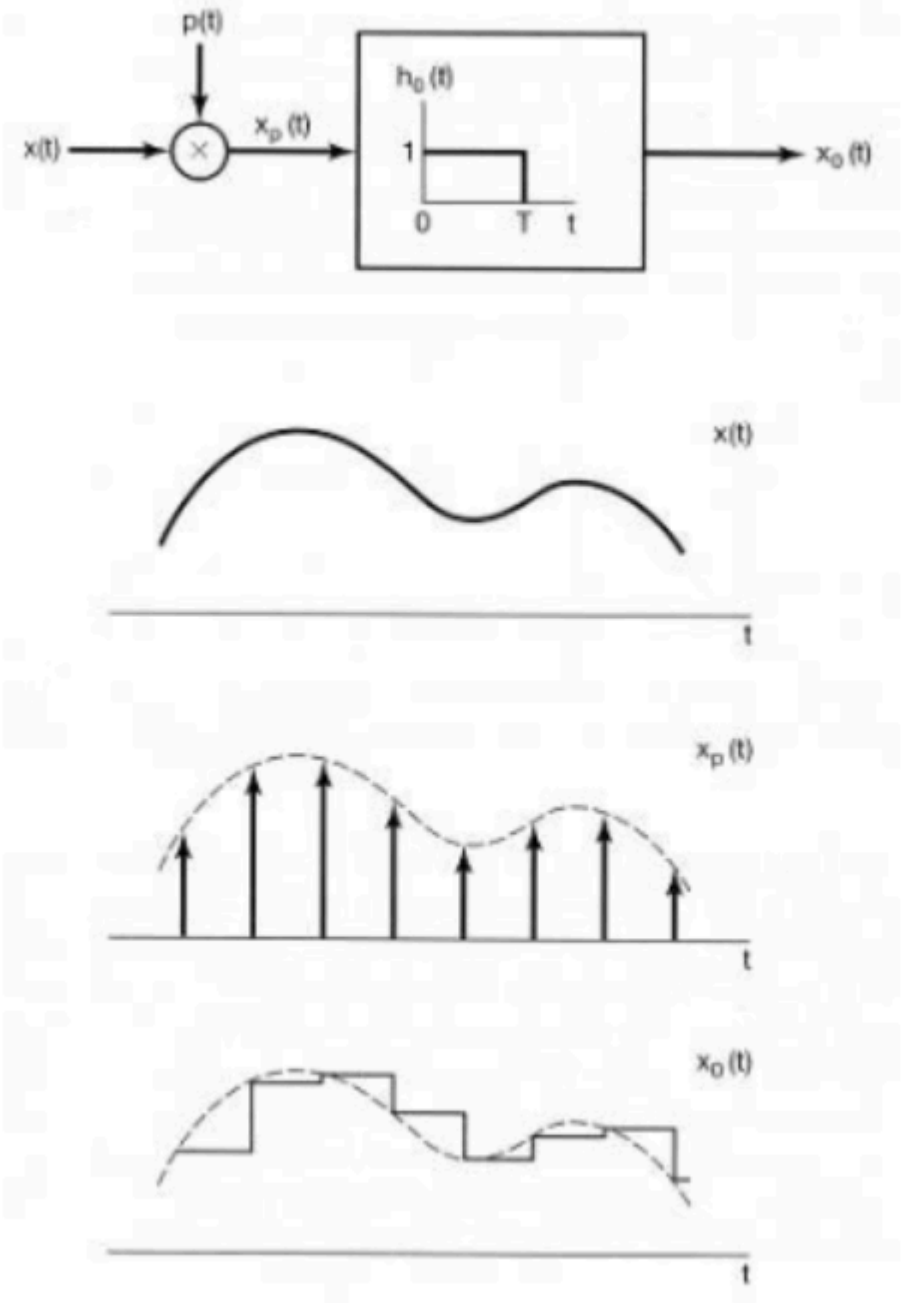
Sampling aperture

> **Sampling이 이루어지는 시간**

이상적으로는 Sampling이 특정 순간에 짧게 이루어지는 것이 좋다.

하지만, 대부분의 경우 짧게 이루어지지 못하고 **특정 시간 동안 해당 Signal**을 받게 된다.

- Signal을 파악하고 유지하는데 특정 시간이 걸리기 때문에 **Finite aperture**이라고도 한다.



Sampling을 할 때, **아날로그-디지털 변환(ADC) 회로의 한계로 $h_0(t)$와 같이 Signal이 잡혀 있다.**

$$H_0(\omega) = e^{-j\omega T/2}\left[\frac{2\sin(\omega T/2)}{\omega}\right]$$

- $w = 2\pi f$

- 시간이 길어질수록 T가 길어져 사인파가 길어진다.

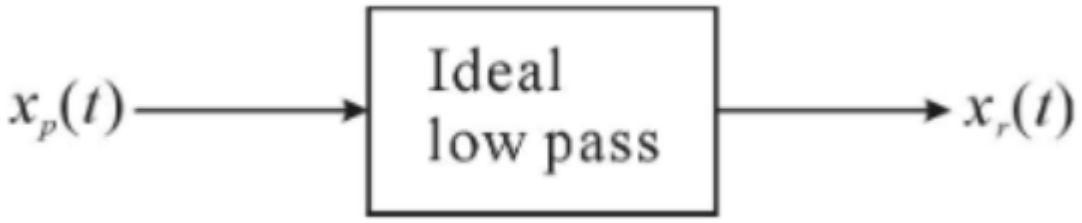
결국 **마지막 Sampling이 진행되는 시간 자체가 길면 $x_p(t)$가 손상**된다.

- 길면 길수록 평균에 가까워지는 효과가 있어 High frequency 정보가 손상된다.

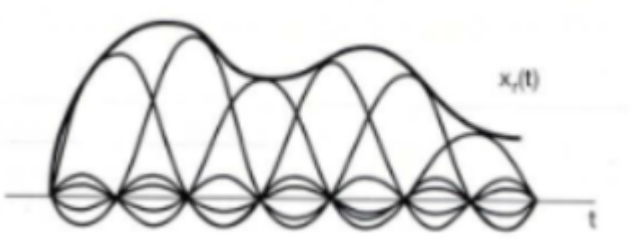
Reconstruction

> **Interpolation을 이용하여 Sampling한 신호를 원래의 신호로 되돌리는 것**

Sampling signal에 Low pass를 통과시켜 원래의 Signal을 얻을 수 있다.

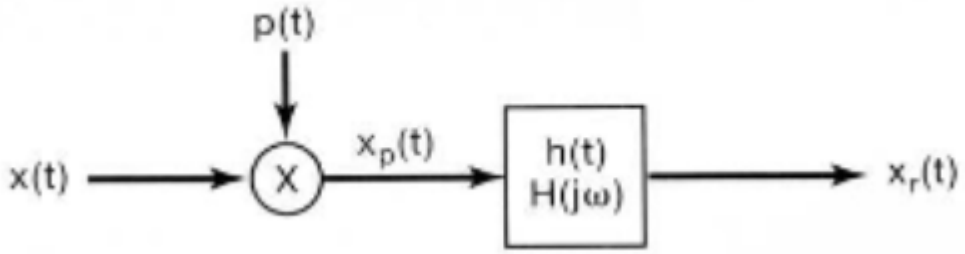


각 Sample 하나당 Sinc를 구성하고 합치면 원래 신호가 된다.



Linear interpolation

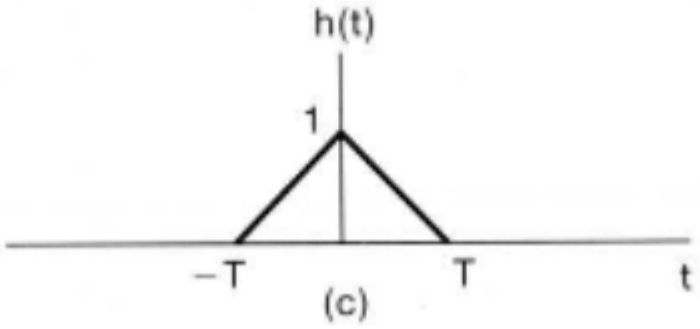
아래 상황에서 Sampling된 $x_p(t)$를 Interpolation하여 Recontruction하는 $h(t)$를 생각해보자.



- $h(t)$: 신호 영역에서의 Interpolation filter
- $H(jw)$: 주파수 영역에서의 Interpolation filter

Interpolation의 $h(t)$는 다음과 같다.

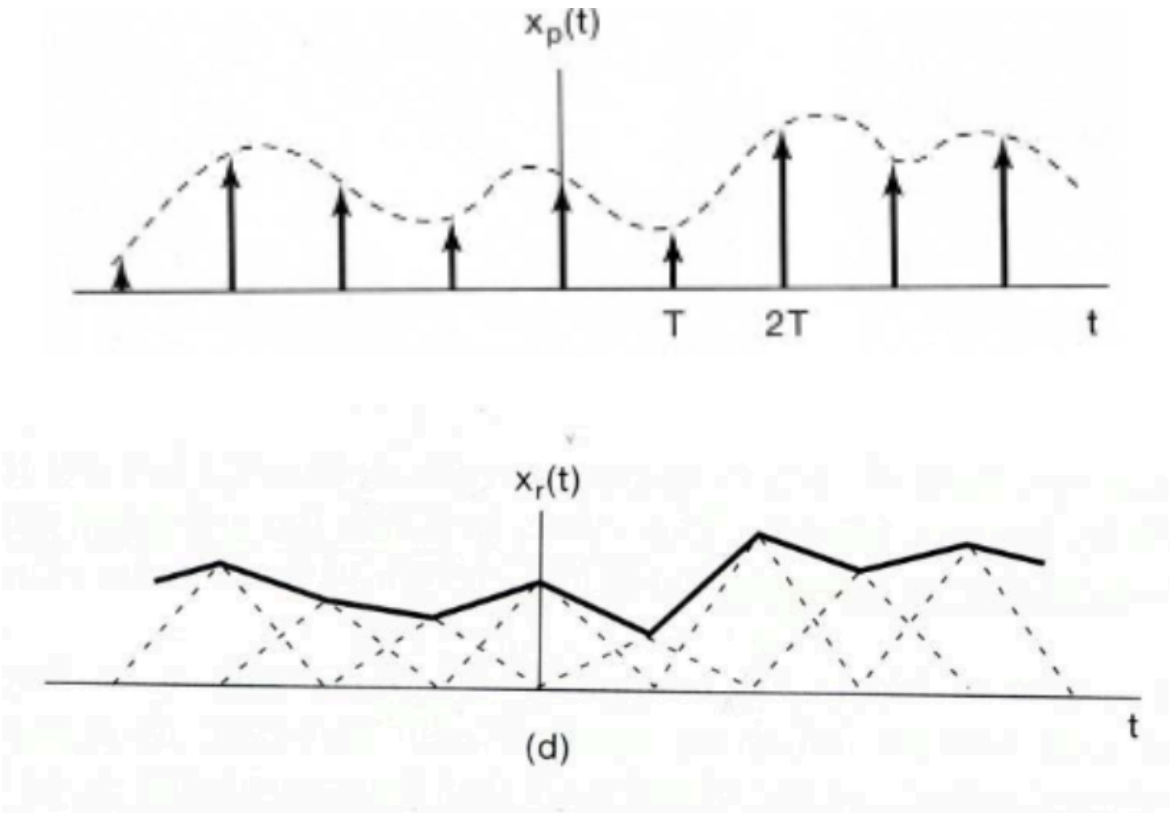
$$h(t) = \begin{cases} 1 - \frac{|t|}{T}, & |t| \leq T \\ 0, & \text{otherwise} \end{cases}$$



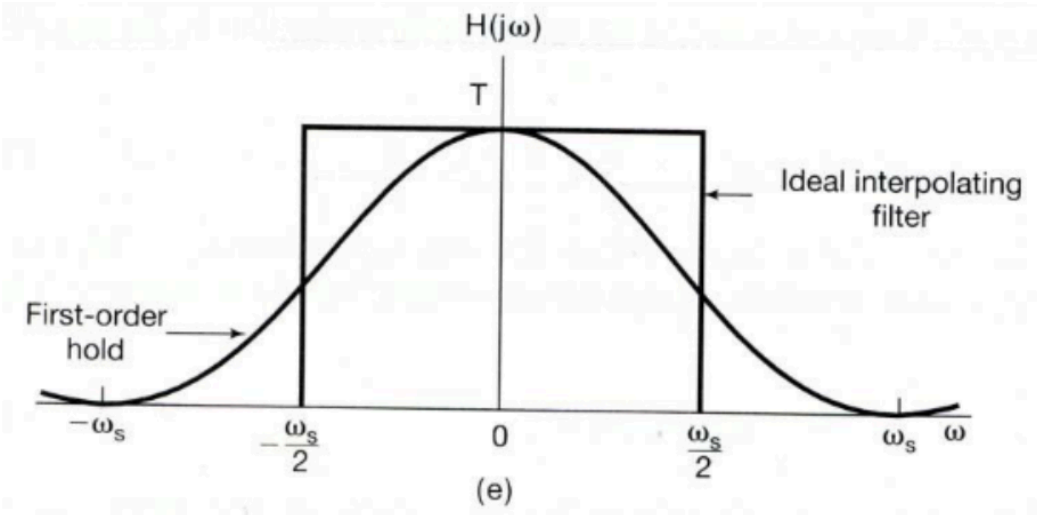
- 이 때의 h(t)를 Signal processing의 Linear system의 H[t(x)]로 생각할 수 있다.

Sampled signal의 각 Sample에 위 Filter를 곱하면 다음과 같다.

- 이후, 각 Sample에서의 값을 모두 더한다.
- 여기서 $x_p * h(t)$로 Convolution을 하는 것과 동일하다.



Reconstruction filter를 주파수 영역에서 보면 $sinc^2$ 형태를 띈다.



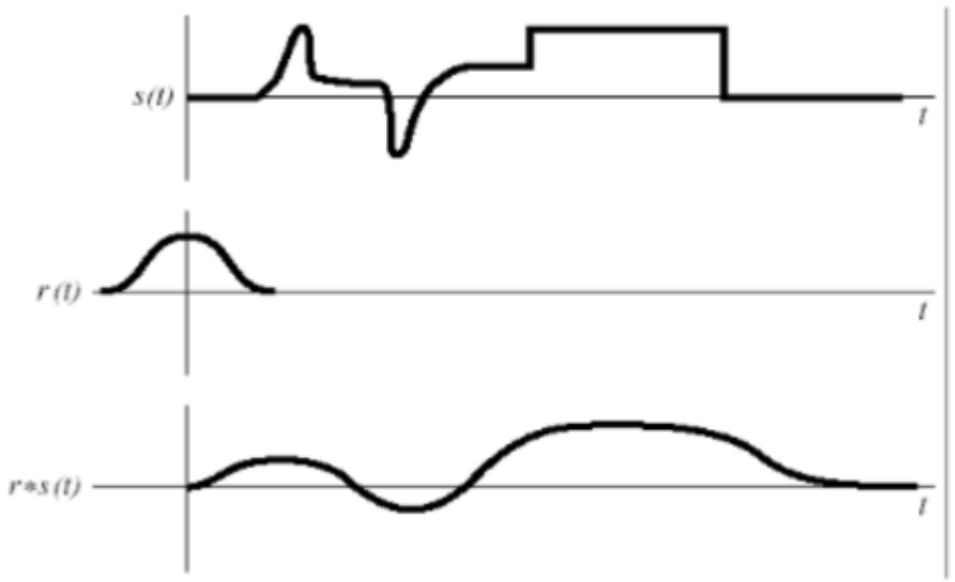
- 주파수 영역에서는 정사각형이 Ideal interpolation filter인데 $H(jw)$가 이를 근사함을 확인할 수 있다.

Convolution

> **입력을 무한히 많은 Impulse들의 합으로 분해하고, 각 Impluse에 대한 시스템의 반응을 시간 이동하며 모두 더한 것**

$$g * h \equiv \int_{-\infty}^{\infty} g(\tau)h(t-\tau)d\tau$$

Input: $s(t)$, Impulse response: $h(t)$, Output: $r * s(t)$



- $r(t)$을 좌우로 뒤집어 $s(t)$ 위에서 작은 영역씩 이동하며 곱한 각 값들을 모두 더하면 $r * s(t)$가 계싼된다.
- 좌우로 뒤집는 이유는 $t-\tau$에서 τ가 커질수록 $t-\tau$가 감소하기 때문이다.

Discrete한 경우는 아래와 같이 쓸 수 있다.

$$(r * s)_j \equiv \sum_{k=-M/2+1}^{M/2} s_{j-k}r_k$$

Convolution theorem

> **시간 영역에서 Convolution은 주파수 영역에서 곱과 동일하다**

$$g * h \Longleftrightarrow G(f)H(f)$$

Convolution 연산이 계산량이 많고 복잡해서 Fourier transform을 이용하여 주파수 영역으로 변환한 후 곱하는 것이 효율적이다.

Problem in DFT of finite duration

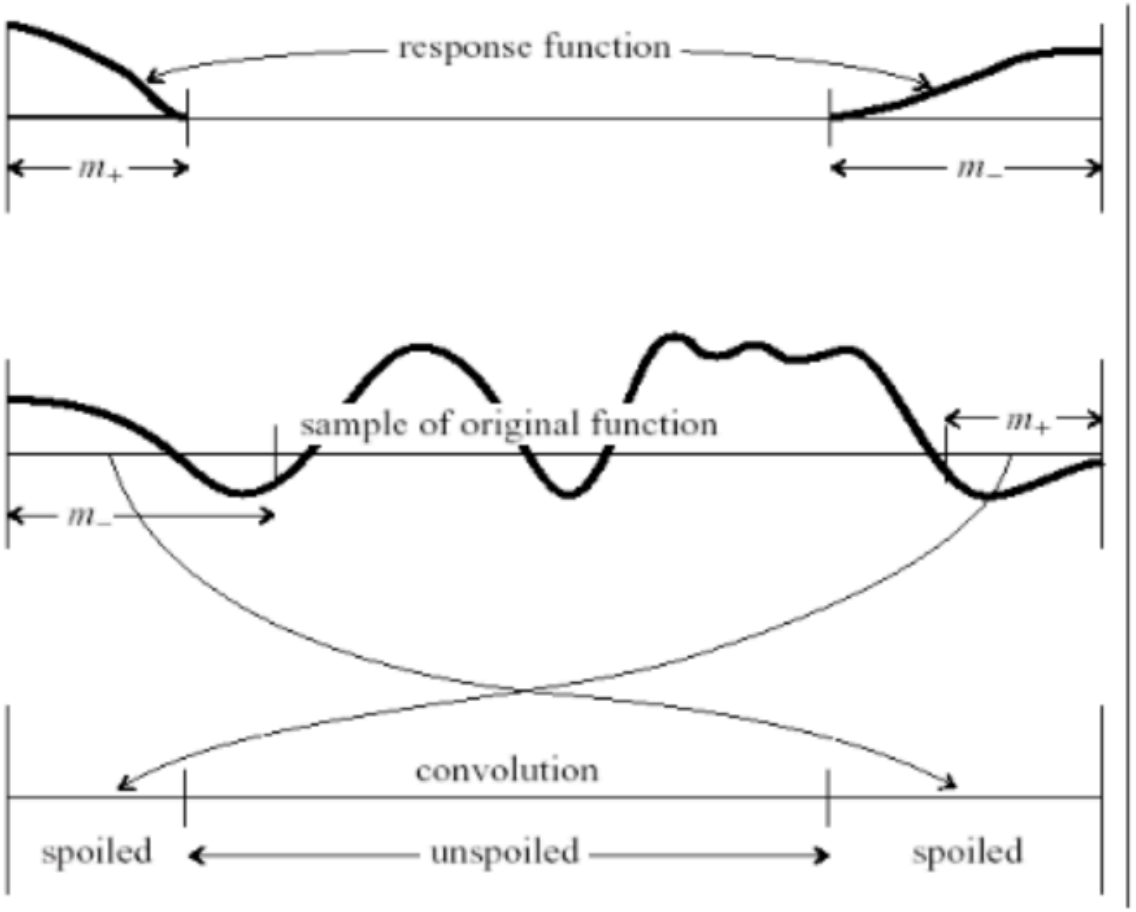
> **신호의 처음이나 끝 부분이 Filter size 보다 작아 정보가 훼손되는 현상**

Discrete Fuerier transformation에서는 Signal이 주기성을 갖고 원형처럼 이어져 있다고 생각한다.

이때, 신호의 처음이나 끝 부분에서는 **반대쪽의 정보를 참고하게 되고, 이것이 정보를 훼손**한다.

- **Spoiled**
- **End effect**라고 한다.

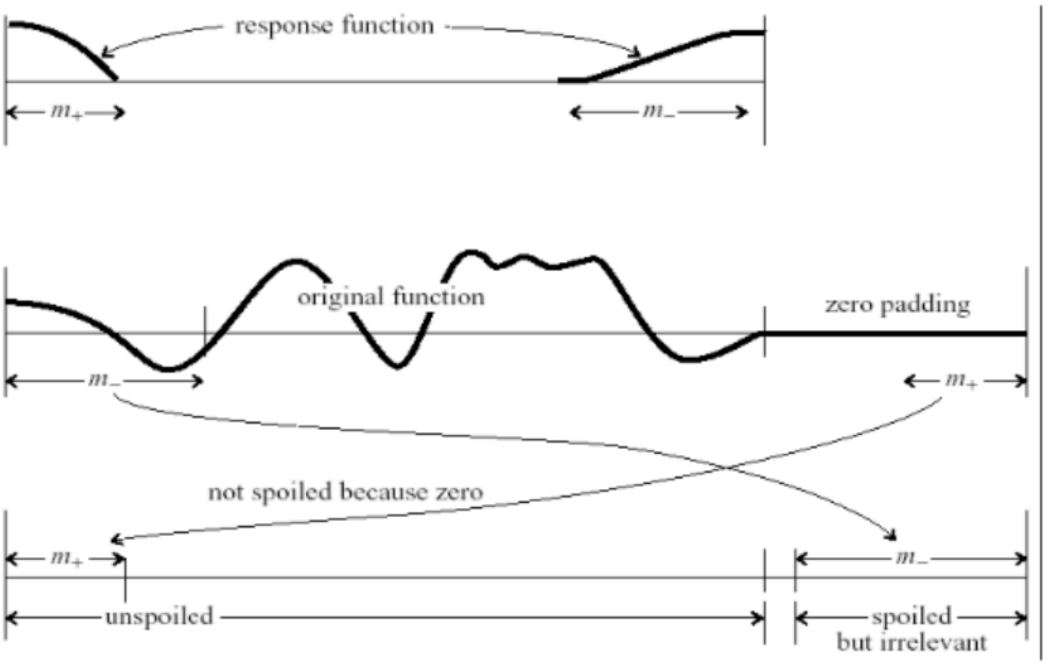
Response function (Filter)가 Spoiled된 결과를 생성한다.



Solution: Zero padding

뒤 부분에 Zero-padding을 추가하면 **앞부분의 정보가 손실되지 않는다.**

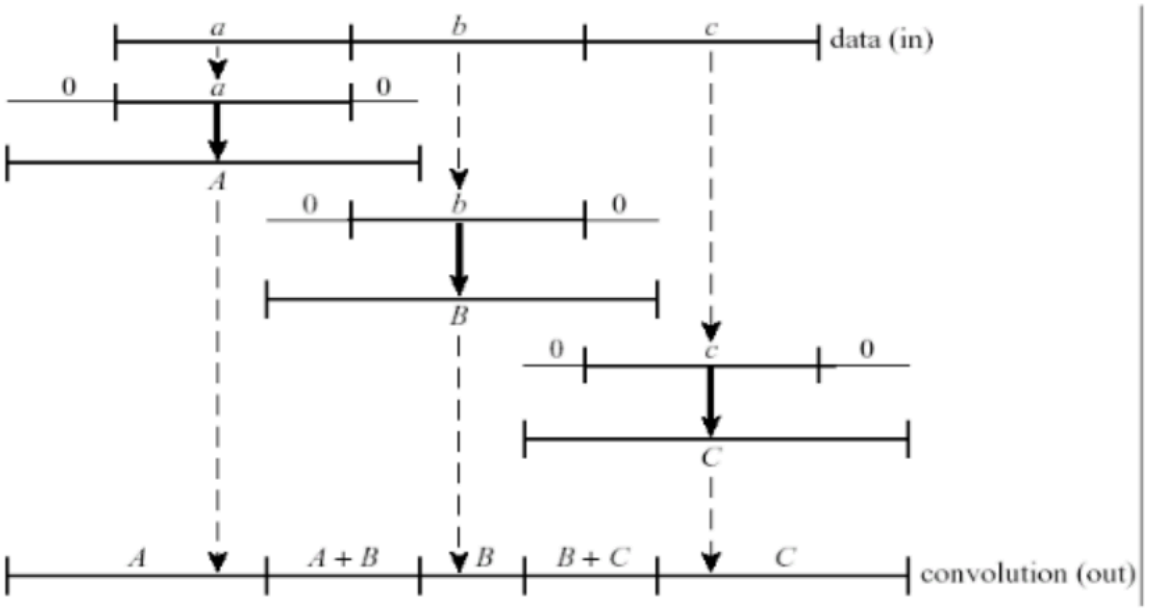
여전히 뒷 부분의 정보는 손실되지만 **Zero-padding에 해당하는 영역은 실제 신호에 대응되지 않아 무시가 가능하다.**



Overlap - add method

굉장히 큰 Data를 한번에 Convolution 연산을 하는 것은 메모리와 시간 부담이 크다.

긴 신호를 여러 Block으로 나누어 각각 FT를 이용해 Linear convolution을 적용하고 그 결과를 합치는 방법이다.



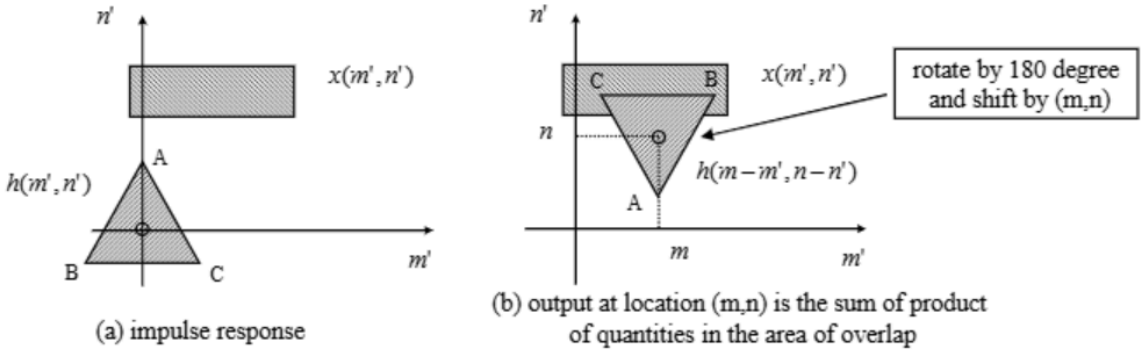
2D Convolution

Input: x, Kerne: h

$$y(m,n) = h(m,n) * x(m,n) = \sum_{m'} \sum_{n'} x(m\prime, n\prime) h(m - m\prime, n - n\prime)$$

Kernel 내에서 변수를 m', n'으로 보면 $(m - m')$, $(n - n')$ = $-(m' - m)$, $-(n - n')$이다.

- 따라서 Kernel을 m, n만큼 평행 이동하고 180도 회전한 것 (좌우, 상하 모두 회전)과 동일하다.



Fourier transform

연속인 경우 다음과 같다.

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{2\pi i f t} dt$$

$$h(t) = \int_{-\infty}^{\infty} H(f) e^{-2\pi i f t} df$$

Discrete fourier transform

연속 Fourier transform을 적분의 정의를 이용해 근사한 것이다.

$$\begin{aligned} H(f_n) &= \int_{-\infty}^{\infty} h(t) e^{2\pi i f_n t} dt \\ &\approx \sum_{k=0}^{N-1} h_k e^{2\pi i f_n t_k} \Delta \\ &= \Delta \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N} \end{aligned}$$

DFT

$$H_n \equiv \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N}$$

- 이걸 직접 계산하면 각 H_n을 계산할 때 $O(N)$, 전체 H_1 H_N을 계산할 때는 $O(N^2)$의 시간이 든다.

Inverse DFT (IDFT)

$$h_k = \frac{1}{N}\sum_{n=0}^{N-1} H_n e^{-2\pi i k n/N}$$

Fast Fourier Transform (FFT)

> **DFT를 짝수 인덱스, 홀수 인덱스로 분할하여 속도를 높이는 방법**

아래는 FFT AIgorithms이다.

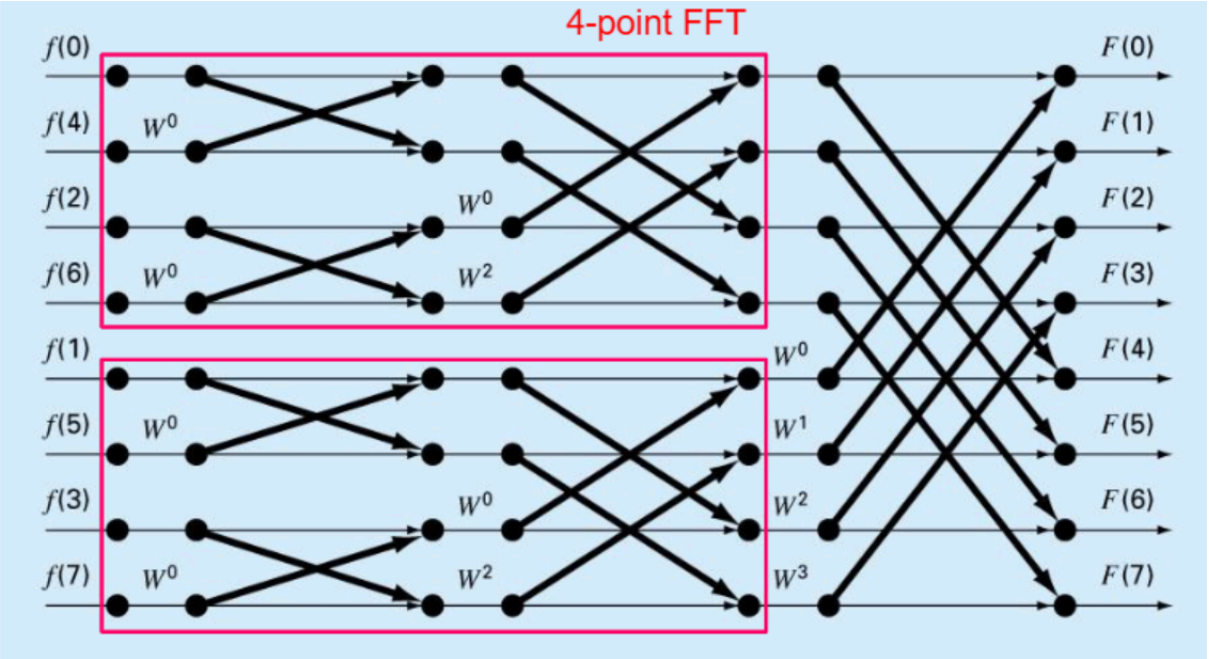
Cooly-tukey method

$$H_n = \sum_{k=0}^{N-1} W^{nk} h_k \quad \text{where} \quad W \equiv e^{2\pi i/N}$$

방법은 다음과 같다.

$$\begin{aligned} F_k &= \sum_{j=0}^{N-1} f_j e^{2\pi i j k/N} \\ &= \sum_{j=0}^{N/2-1} f_{2j} e^{2\pi i k(2j)/N} + \sum_{j=0}^{N/2-1} f_{2j+1} e^{2\pi i k(2j+1)/N} \\ &= \sum_{j=0}^{N/2-1} f_{2j} e^{2\pi i j k/(N/2)} + W^k \sum_{j=0}^{N/2-1} f_{2j+1} e^{2\pi i j k/(N/2)} \\ &= F_k^e + W^k F_k^o \end{aligned}$$

총 시간 복잡도가 $O(NlogN)$이 된다.



Sande-tukey algorithms

가장 먼저 N이 2의 배수가 되도록 Sampling한다.

$$N = 2^m$$

DFT의 공식은 다음과 같다.

$$F_j = \sum_{k=0}^{N-1} f_k e^{-i2\pi jk/N} = \sum_{k=0}^{N-1} f_k w^{jk}$$
$$\text{where} \quad w = e^{-i2\pi/N}$$

위 식을 아래와 같이 변환한다.

$$
\begin{aligned}
F_j &= \sum_{k=0}^{N/2-1} f_k e^{-i2\pi jk/N} + \sum_{k=N/2}^{N-1} f_k e^{-i2\pi jk/N} \\
&= \sum_{k=0}^{N/2-1} f_k e^{-i2\pi jk/N} + \sum_{m=0}^{N/2-1} f_{m+N/2} e^{-i2\pi j(m+N/2)/N} \\
&= \sum_{k=0}^{N/2-1} (f_k + e^{-i\pi j} f_{k+N/2}) e^{-i2\pi jk/N} \\
&\quad \text{where} \quad e^{-i\pi j} = (-1)^j
\end{aligned}
$$

이제 위 식에서 **짝수와 홀수**를 구별한다.

짝수

$$
\begin{aligned}
F_{2j} &= \sum_{k=0}^{N/2-1} (f_k + f_{k+N/2}) e^{-i2\pi \cdot 2jk/N} \\
&= \sum_{k=0}^{N/2-1} (f_k + f_{k+N/2}) e^{-i2\pi jk/(N/2)} \\
&= \sum_{k=0}^{N/2-1} (f_k + f_{k+N/2}) w^{2jk}
\end{aligned}
$$

홀수

$$
\begin{aligned}
F_{2j+1} &= \sum_{k=0}^{N/2-1} (f_k - f_{k+N/2}) e^{-i2\pi (2j+1)k/N} \\
&= \sum_{k=0}^{N/2-1} (f_k - f_{k+N/2}) e^{-i2\pi k/N} e^{-i2\pi jk/(N/2)} \\
&= \sum_{k=0}^{N/2-1} (f_k - f_{k+N/2}) w^k w^{2jk}
\end{aligned}
$$

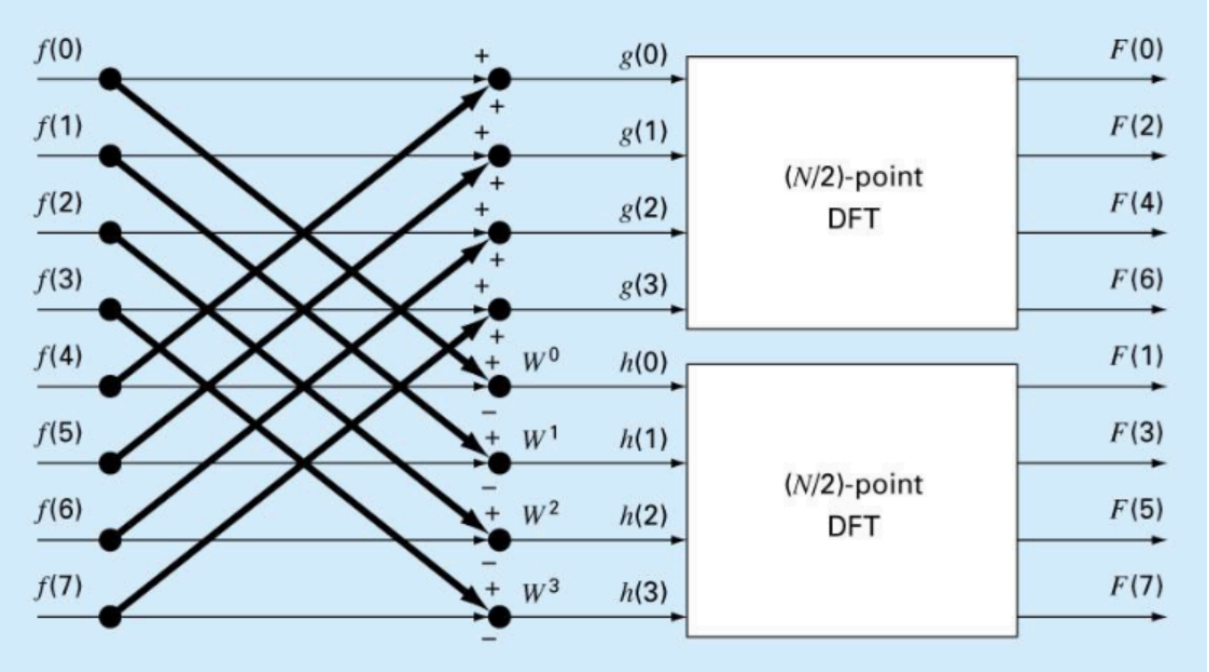
짝수 및 홀수 주파수 성분을 각각 묶으면, N-point DFT는 두 개의 (N/2)-point DFT로 분해되며, 이때 입력은 **half-size sequence**로 재구성된다.

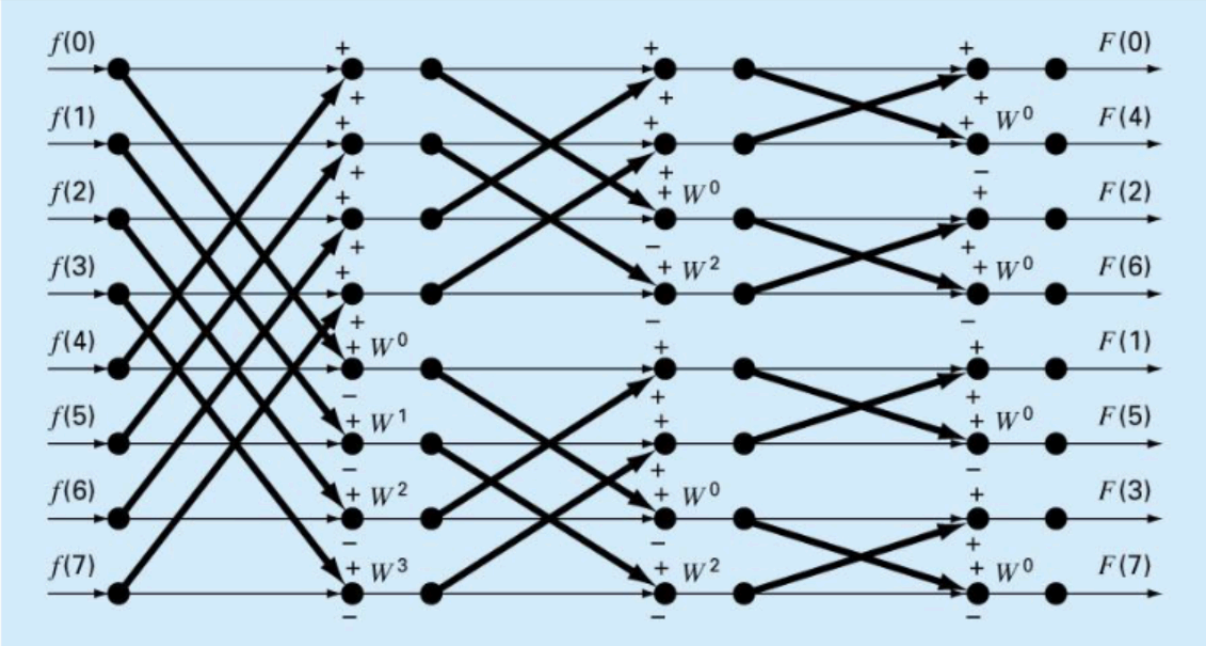
Half size sequence

$$g_k = f_k + f_{k+N/2}$$
$$h_k = (f_k - f_{k+N/2})w^k$$

Half-size Transforms

$$F_{2j} = G_j$$
$$F_{2j+1} = H_j$$

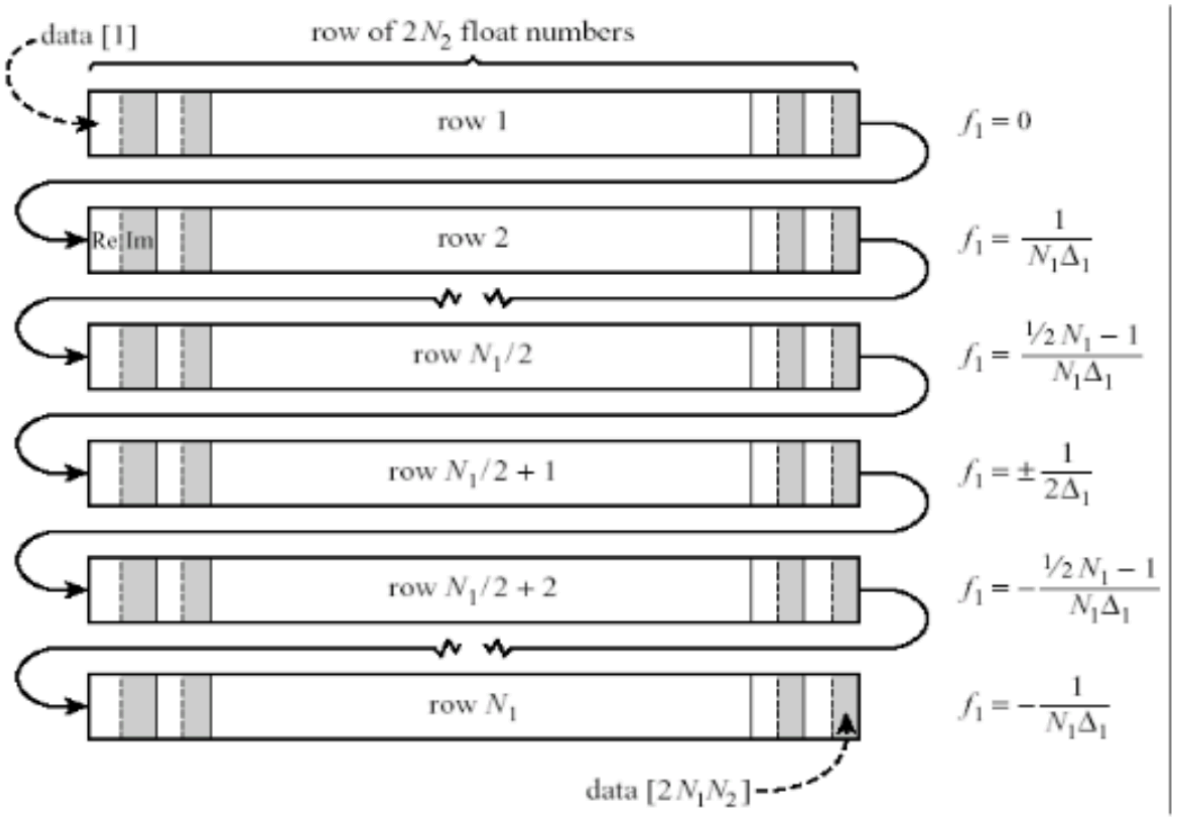




2D FFT

$$H(n_1, n_2) \equiv \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} \exp(2\pi i k_2 n_2 / N_2)\ \exp(2\pi i k_1 n_1 / N_1)\ h(k_1, k_2)$$

$$\begin{aligned} H(n_1, n_2) &= \text{FFT-on-index-1}\,(\text{FFT-on-index-2}\,[h(k_1, k_2)]) \\ &= \text{FFT-on-index-2}\,(\text{FFT-on-index-1}\,[h(k_1, k_2)]) \end{aligned}$$



* Generalization to L-dimension

$$H(n_1, \ldots, n_L) \equiv \sum_{k_L=0}^{N_L-1} \cdots \sum_{k_1=0}^{N_1-1} \exp(2\pi i k_L n_L / N_L) \times \cdots \times \exp(2\pi i k_1 n_1 / N_1)\; h(k_1, \ldots, k_L)$$