

FFTW를 이용한 DFT구현

멀티미디어 프로그래밍

6주차

Recall Fourier Series

- 모든 주기 신호는 정현파 신호의 합으로 주어진다.

$$f(\theta) = a_0 + \sum_{n=1}^{\infty} a_n \cos n\theta + \sum_{n=1}^{\infty} b_n \sin n\theta$$

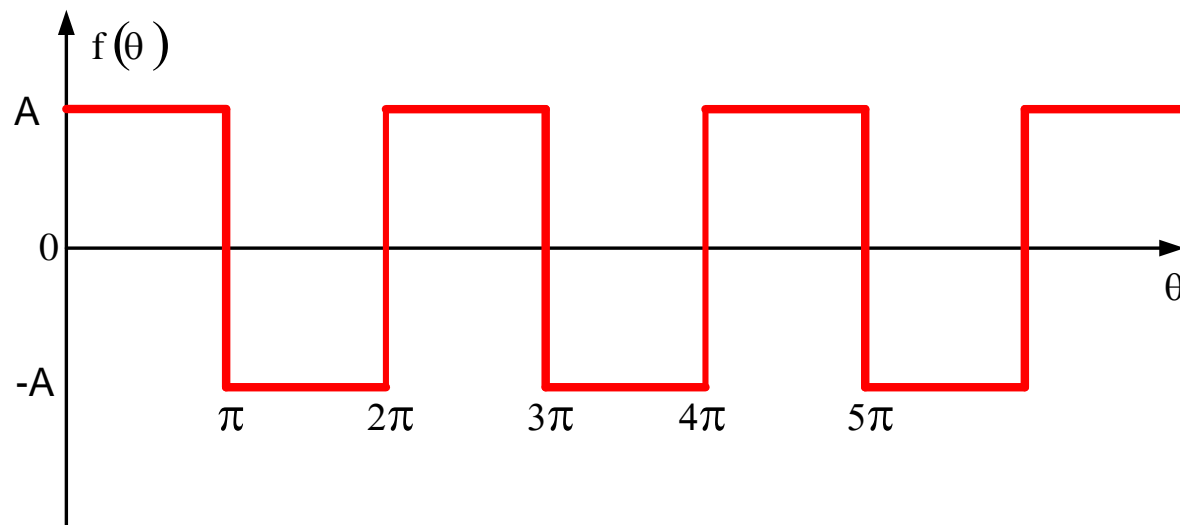
- 각각의 정현파 신호 $\cos n\theta, \sin n\theta$ 를 “기저신호” (basis signal) 라고 부른다.
- 각 정현파에 대한 계수 a_0, a_n, b_n 는 해당 정현파와의 내적(projection)을 통해 구한다.

$$a_0 = \frac{1}{2\pi} \int_0^{2\pi} f(\theta) d\theta$$

$$a_n = \frac{1}{\pi} \int_0^{2\pi} f(\theta) \cos n\theta d\theta \quad n = 1, 2, \dots$$

$$b_n = \frac{1}{\pi} \int_0^{2\pi} f(\theta) \sin n\theta d\theta \quad n = 1, 2, \dots$$

Fourier Series 로 표현된 rectangular 함수



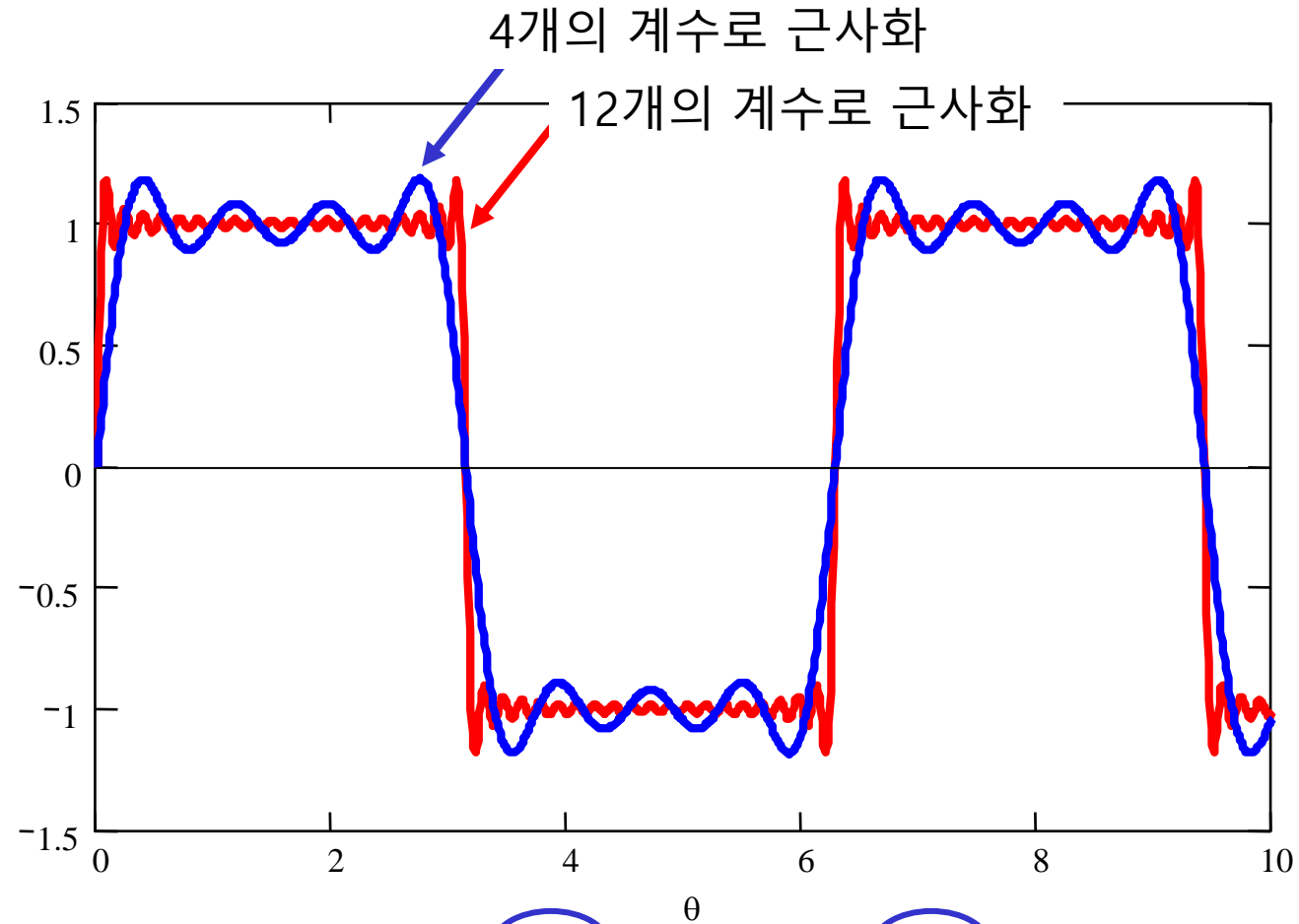
$$\begin{aligned} f(\theta) &= A \quad \text{when} \quad 0 < \theta < \pi \\ &= -A \quad \text{when} \quad \pi < \theta < 2\pi \end{aligned}$$

$$f(\theta + 2\pi) = f(\theta)$$



$$\frac{4A}{\pi} \left(\sin \theta + \frac{1}{3} \sin 3\theta + \frac{1}{5} \sin 5\theta + \frac{1}{7} \sin 7\theta + \dots \right)$$

FS의 근사화



$$f(\theta) = a_0 + \sum_{n=1}^{\infty} a_n \cos n\theta + \sum_{n=1}^{\infty} b_n \sin n\theta$$

The complex form of FS

$$f(\theta) = a_0 + \sum_{n=1}^{\infty} a_n \cos n \theta + \sum_{n=1}^{\infty} b_n \sin n \theta$$

Let us utilize the Euler formulae.

$$\cos \theta = \frac{e^{j\theta} + e^{-j\theta}}{2}$$

$$\sin \theta = \frac{e^{j\theta} - e^{-j\theta}}{2i}$$

$$\begin{aligned} a_n \cos n \theta + b_n \sin n \theta &= a_n \frac{e^{jn\theta} + e^{-jn\theta}}{2} + b_n \frac{e^{jn\theta} - e^{-jn\theta}}{2i} \\ &= a_n \frac{e^{jn\theta} + e^{-jn\theta}}{2} - ib_n \frac{e^{jn\theta} - e^{-jn\theta}}{2} \end{aligned}$$

$$a_n \cos n \theta + b_n \sin n \theta = \left(\frac{a_n - j b_n}{2} \right) e^{j n \theta} + \left(\frac{a_n + j b_n}{2} \right) e^{-j n \theta}$$



Denoting

$$c_0 = a_0 \quad c_n = \left(\frac{a_n - j b_n}{2} \right) \quad c_{-n} = \left(\frac{a_n + j b_n}{2} \right)$$

$$a_n \cos n \theta + b_n \sin n \theta = c_n e^{j n \theta} + c_{-n} e^{-j n \theta}$$



$$f(\theta) = c_0 + \sum_{n=1}^{\infty} (c_n e^{j n \theta} + c_{-n} e^{-j n \theta}) = \boxed{\sum_{n=-\infty}^{\infty} c_n e^{j n \theta}} \quad \leftarrow \text{Complex form of FS}$$

Complex form FS에 대한 FS 계수의 계산

$$a_n = \frac{1}{\pi} \int_0^{2\pi} f(\theta) \cos n \theta d\theta \quad n = 1, 2, \dots$$

$$b_n = \frac{1}{\pi} \int_0^{2\pi} f(\theta) \sin n \theta d\theta \quad n = 1, 2, \dots$$

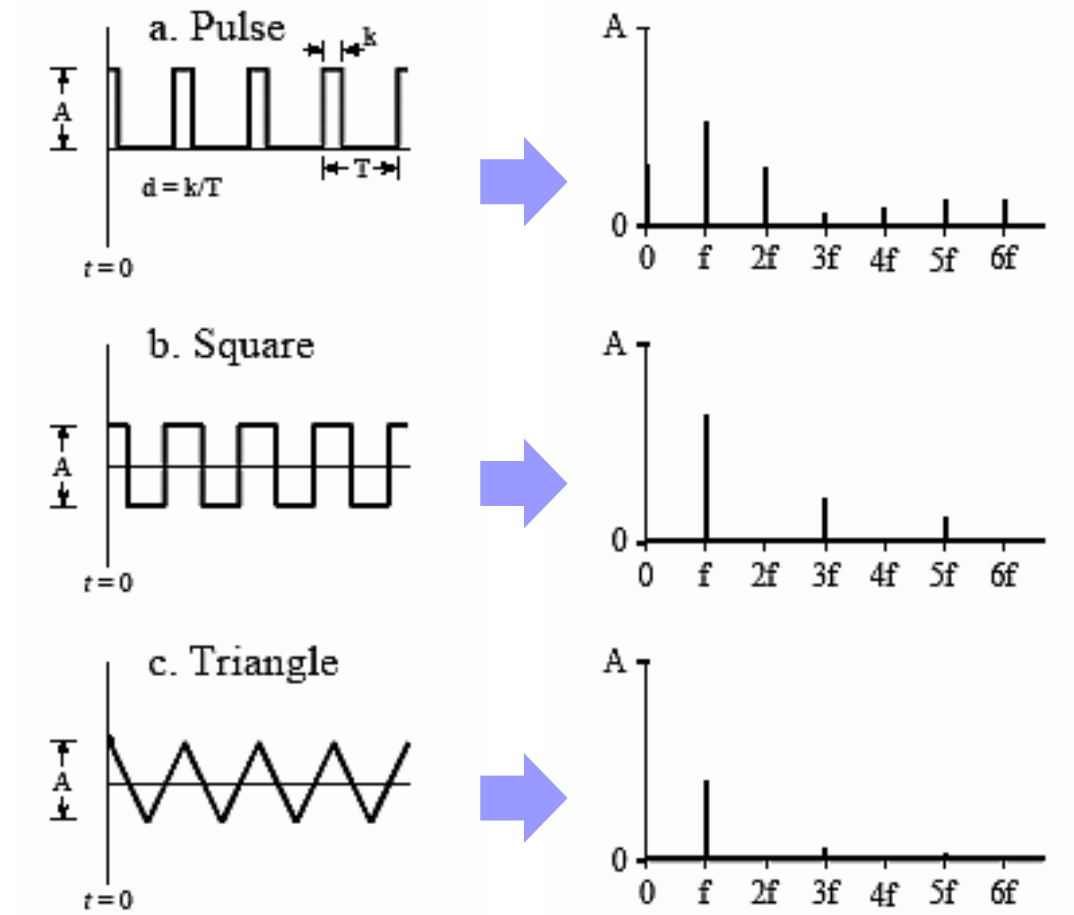
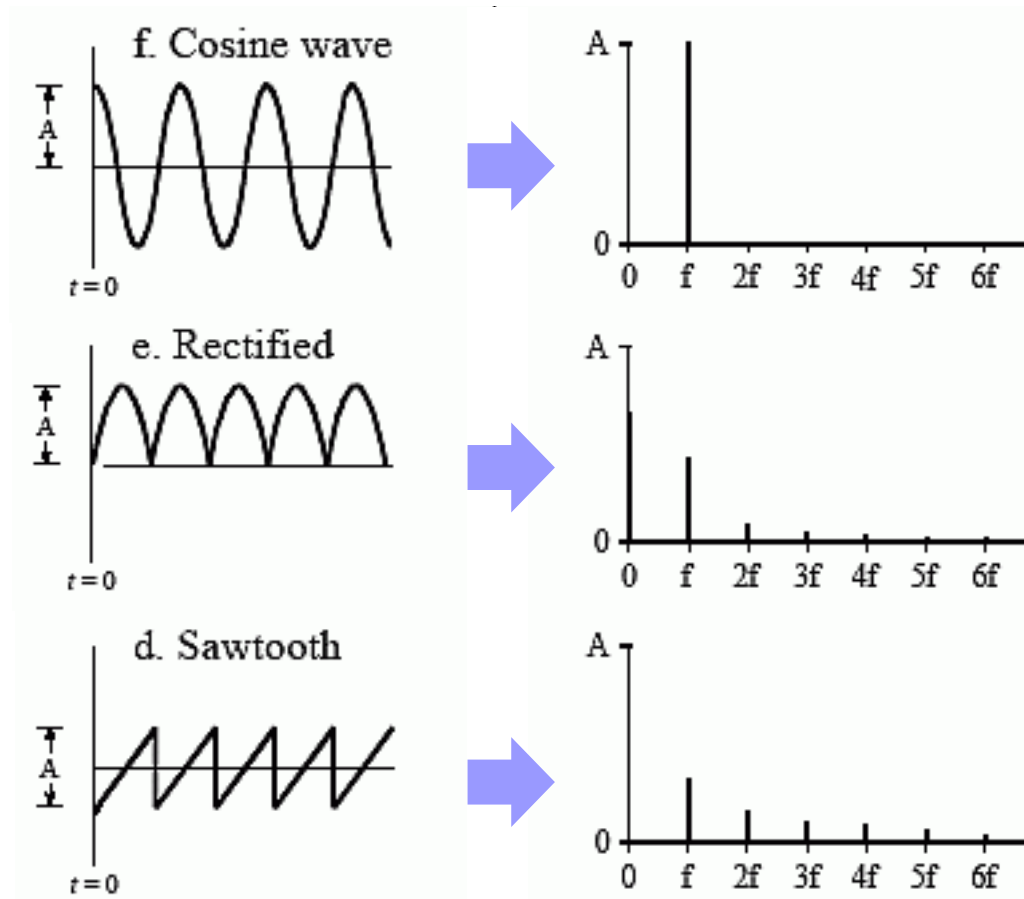
$$c_n = \left(\frac{a_n - j b_n}{2} \right)$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} f(\theta) \cos n \theta d\theta - \frac{j}{2\pi} \int_{-\pi}^{\pi} f(\theta) \sin n \theta d\theta$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} f(\theta) (\cos n \theta - j \sin n \theta) d\theta$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} f(\theta) e^{-jn\theta} d\theta$$

Fourier series pairs



주기가 T인 주기 신호의 Fourier Series

$$f(\theta) = a_0 + \sum_{n=1}^{\infty} a_n \cos n\theta + \sum_{n=1}^{\infty} b_n \sin n\theta \quad \leftarrow \text{주기가 } 2\pi$$

$$f(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos\left(n\frac{2\pi}{T}t\right) + \sum_{n=1}^{\infty} b_n \sin\left(n\frac{2\pi}{T}t\right) \quad \leftarrow \text{주기가 } T$$

주기가 2π

$$a_0 = \frac{1}{2\pi} \int_0^{2\pi} f(\theta) d\theta$$

$$a_n = \frac{1}{\pi} \int_0^{2\pi} f(\theta) \cos n\theta d\theta$$

$$b_n = \frac{1}{\pi} \int_0^{2\pi} f(\theta) \sin n\theta d\theta$$

$$\frac{1}{T} \int_{-T/2}^{T/2} f(t) dt$$

$$\frac{2}{T} \int_{-T/2}^{T/2} f(t) \cos\left(n\frac{2\pi}{T}t\right) dt$$

$$\frac{2}{T} \int_{-T/2}^{T/2} f(t) \sin\left(n\frac{2\pi}{T}t\right) dt$$

주기가 T

주기가 T인 주기 신호의 Fourier Series; for complex FS

$$f(\theta) = \sum_{n=-\infty}^{\infty} c_n e^{jn\theta}$$

$$c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(\theta) e^{-jn\theta} d\theta$$

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{jn\frac{2\pi}{T}t}$$

$$c_n = \frac{1}{T} \int_{-T/2}^{T/2} f(t) e^{-jn\frac{2\pi}{T}t} dt$$

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{jn\omega_0 t}$$

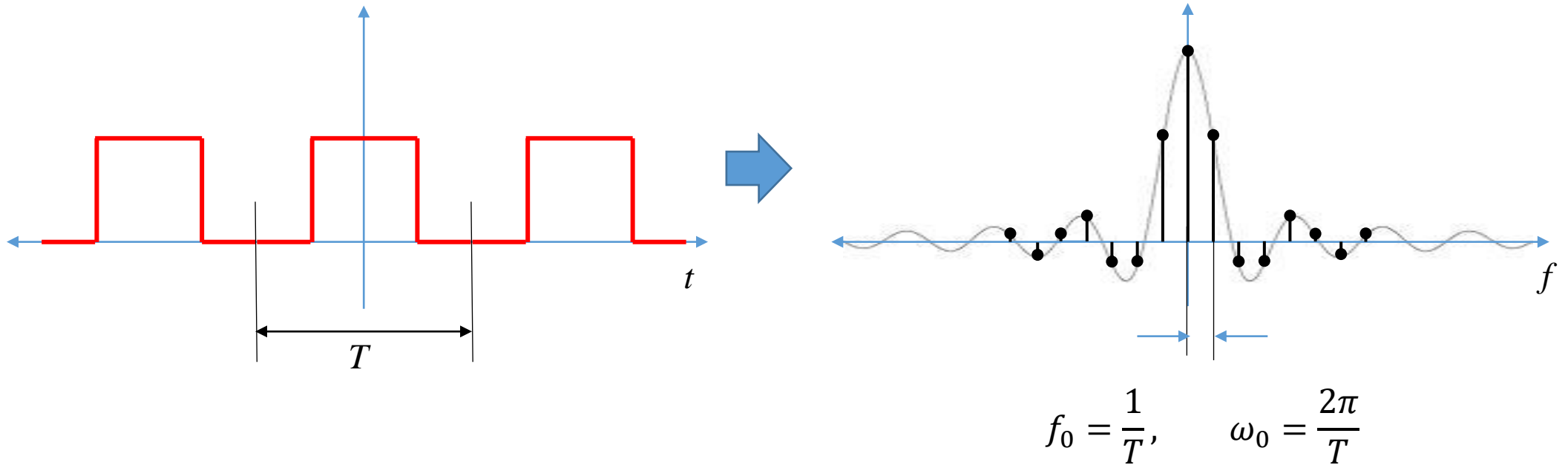
$$c_n = \frac{1}{T} \int_{-T/2}^{T/2} f(t) e^{-jn\omega_0 t} dt$$

$\omega_0 = \frac{2\pi}{T}$ Fundamental angular frequency

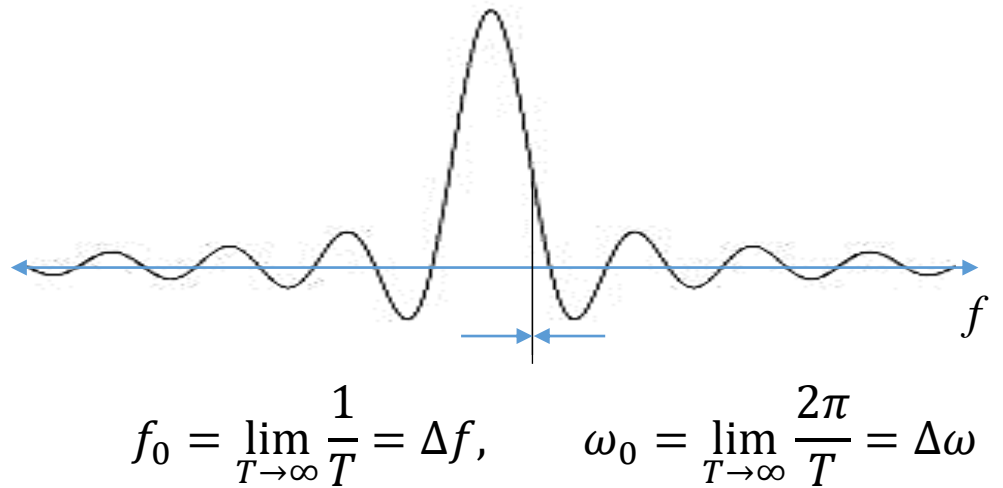
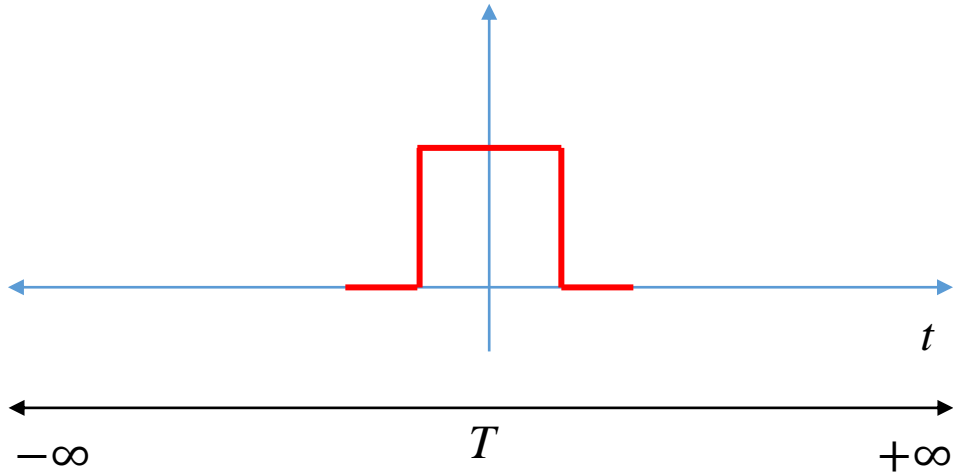
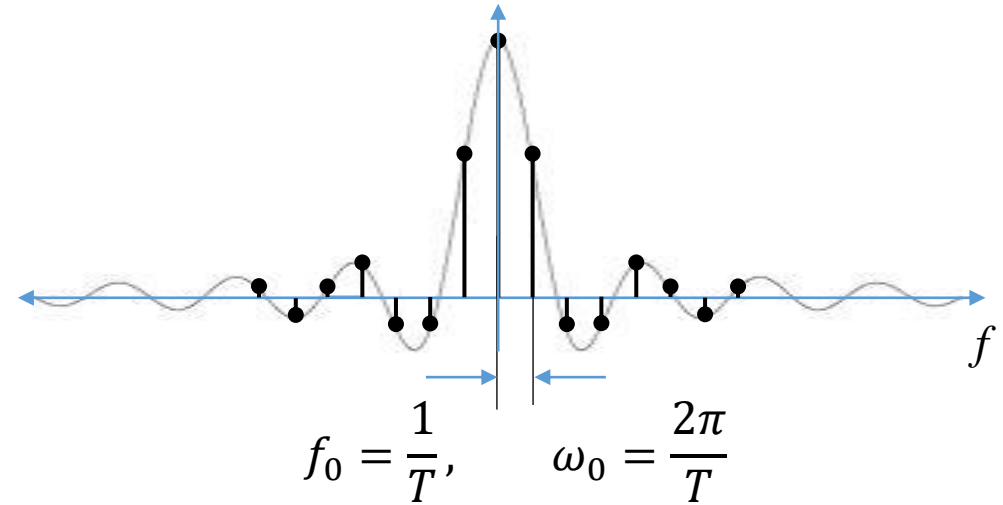
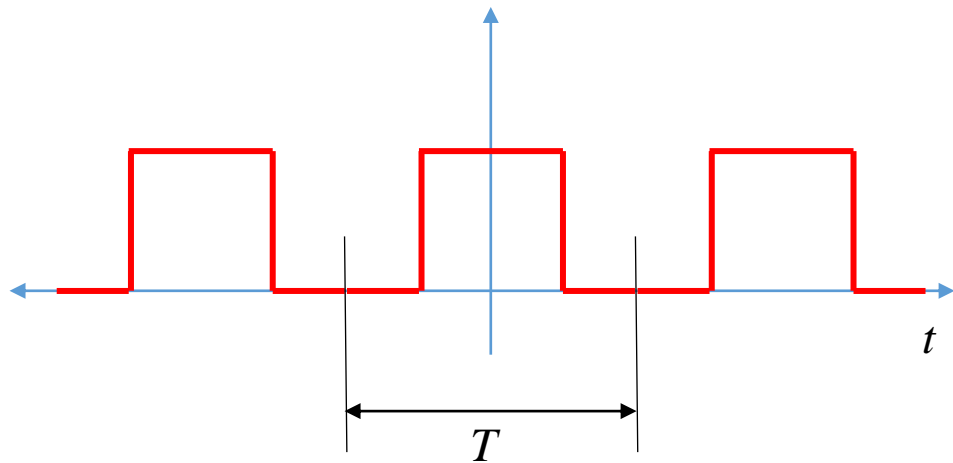
$f_0 = \frac{1}{T}$, Fundamental frequency

$$\omega_0 = 2\pi f_0$$

주기 와 기본주파수와의 관계



비 주기 함수의 주기 함수 표현?



From Fourier Series to Fourier Transform

$$f(t) = \sum_{n=-\infty}^{\infty} \boxed{C_n} e^{jn\omega_0 t}, \quad -\frac{T}{2} < t < \frac{T}{2}, \quad \text{let } T = \frac{2\pi}{\omega_0} \quad C_n = \frac{1}{T} \int_{-T/2}^{T/2} f(t) e^{-jn\omega_0 t} dt$$

$f(t)$ is not periodic function if $T \rightarrow \infty$

$$\omega_0 = \frac{2\pi}{T} \rightarrow d\omega, \quad n\omega_0 \rightarrow \omega$$

$$\begin{aligned} f(t) &= \int_{-\infty}^{\infty} C_n e^{jn\omega_0 t} = \int_{-\infty}^{\infty} \left[\frac{d\omega}{2\pi} \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \right] e^{j\omega t} \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \right] e^{j\omega t} d\omega \end{aligned}$$

$\boxed{F(j\omega)}$

Fourier Series vs. Fourier Transform

FS

정현파 신호 $\cos n\theta$, $\sin n\theta \rightarrow$ FS의 기저신호(basis signal).

각 정현파에 대한 계수 a_0 , a_n , b_n 는 해당 정현파와의 내적(projection)을 통해 구한다.

<- FS 계수

$$a_n = \frac{1}{\pi} \int_0^{2\pi} f(\theta) \cos n\theta d\theta \quad n = 1, 2, \dots \quad b_n = \frac{1}{\pi} \int_0^{2\pi} f(\theta) \sin n\theta d\theta \quad n = 1, 2, \dots$$

FT

정현파 신호 $e^{-i\omega t} = \cos\omega t + j\sin\omega t \rightarrow$ FT의 기저신호(basis signal).

각 정현파에 대한 계수 $F(j\omega)$ 는 해당 정현파 $e^{-i\omega t}$ 와의 내적(projection)을 통해 구한다.

<- FT계수

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

디지털 신호처리 알고리즘

- 수학적으로 유도된 식이 실제로 구현되어야 한다.
 - 프로그램 코드 (C, python, ...)
 - HDL (Hardware Description Language)
- Fourier series 와 Fourier transform:

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{jn\omega_0 t} \quad c_n = \frac{1}{T} \int_{-T/2}^{T/2} f(t) e^{-jn\omega_0 t} dt$$

FS 계수 C_n 은 Discrete
-> 유한 개수. But,
-> 적분식 -> 계산이 불가능.

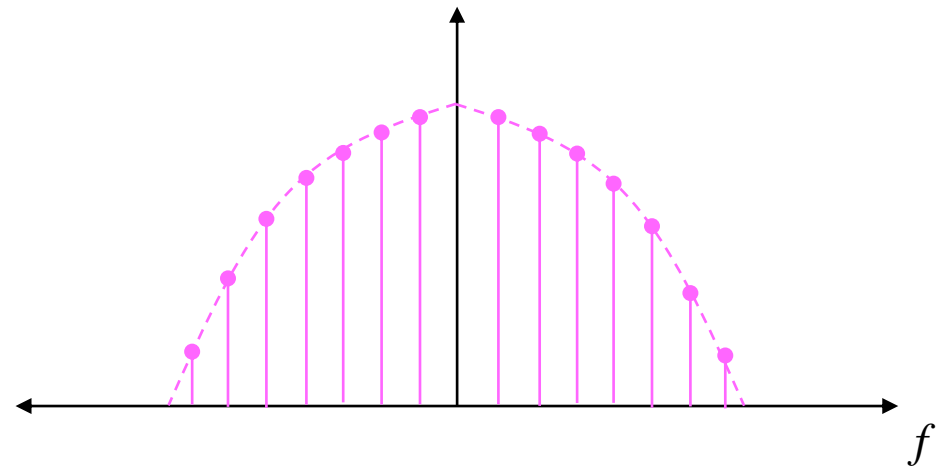
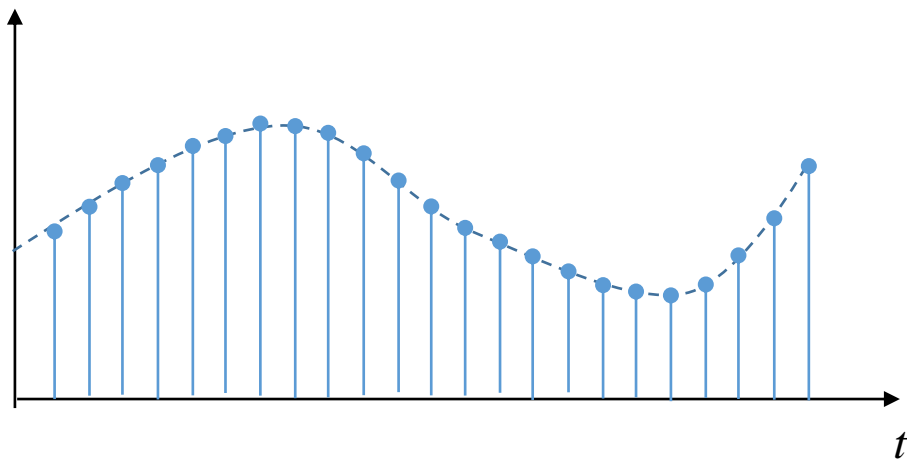
$$F(j\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt$$

$F(\omega)$ 는 continuous
-> 무한 개수 -> 저장 불가능.
-> 적분식 -> 계산이 불가능.

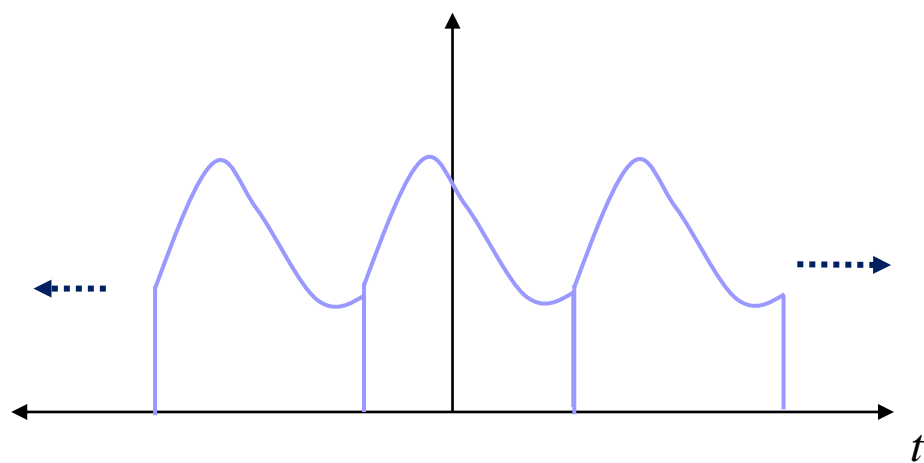
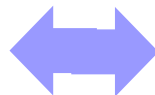
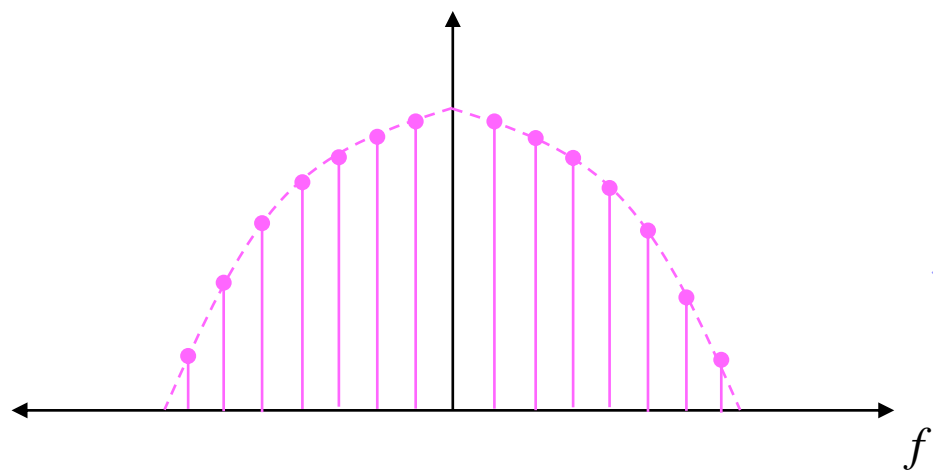
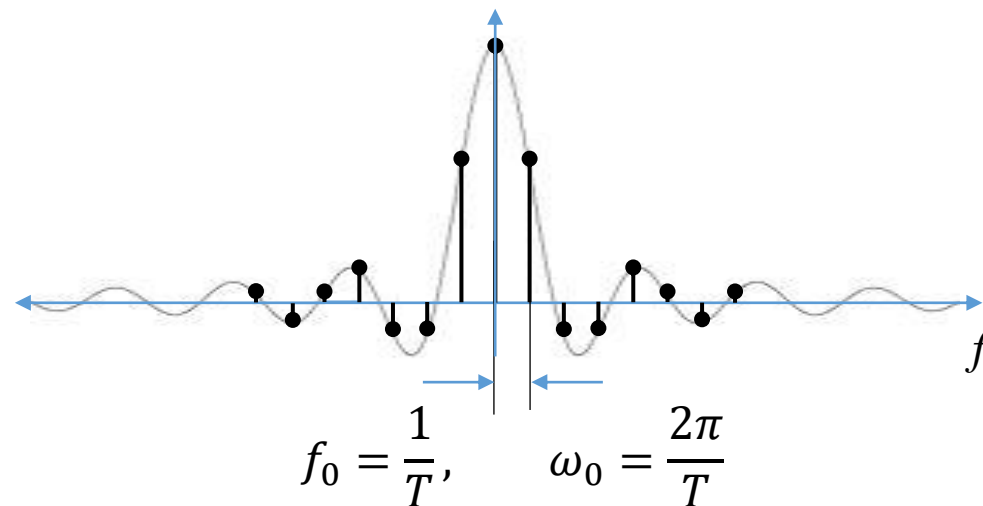
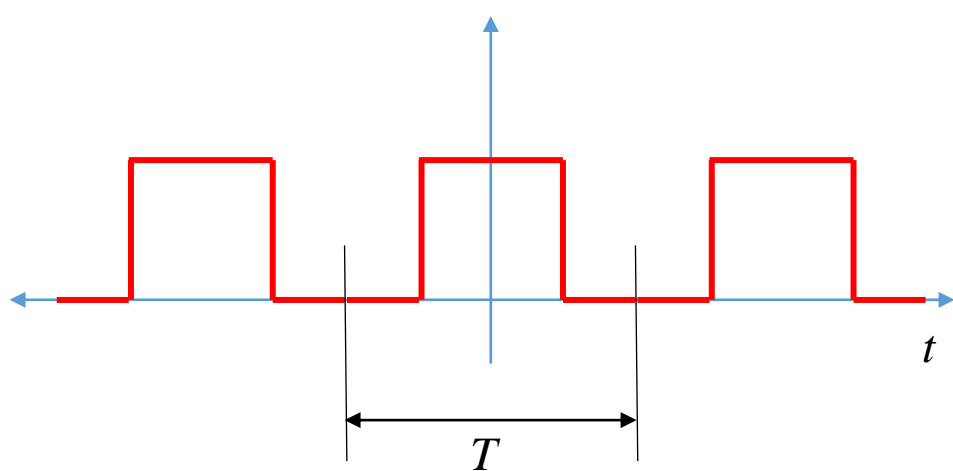
모두 실제 구현이
불가능

Discrete 신호에 대해 Fourier transform을 적용하기 위한 조건.

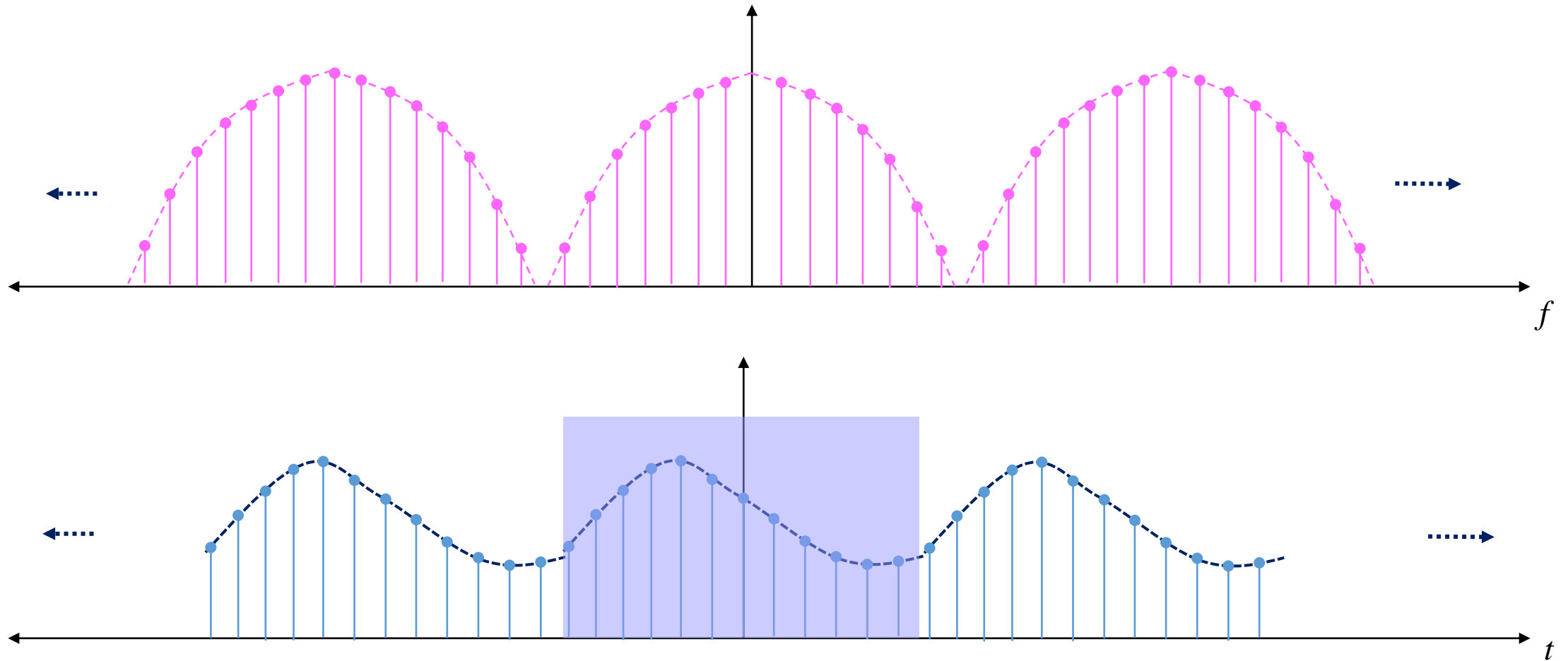
- 비주기 신호에 대한 변환
- 입력 신호 \rightarrow discrete, 유한 개수
- 출력 신호 \rightarrow discrete, 유한 개수



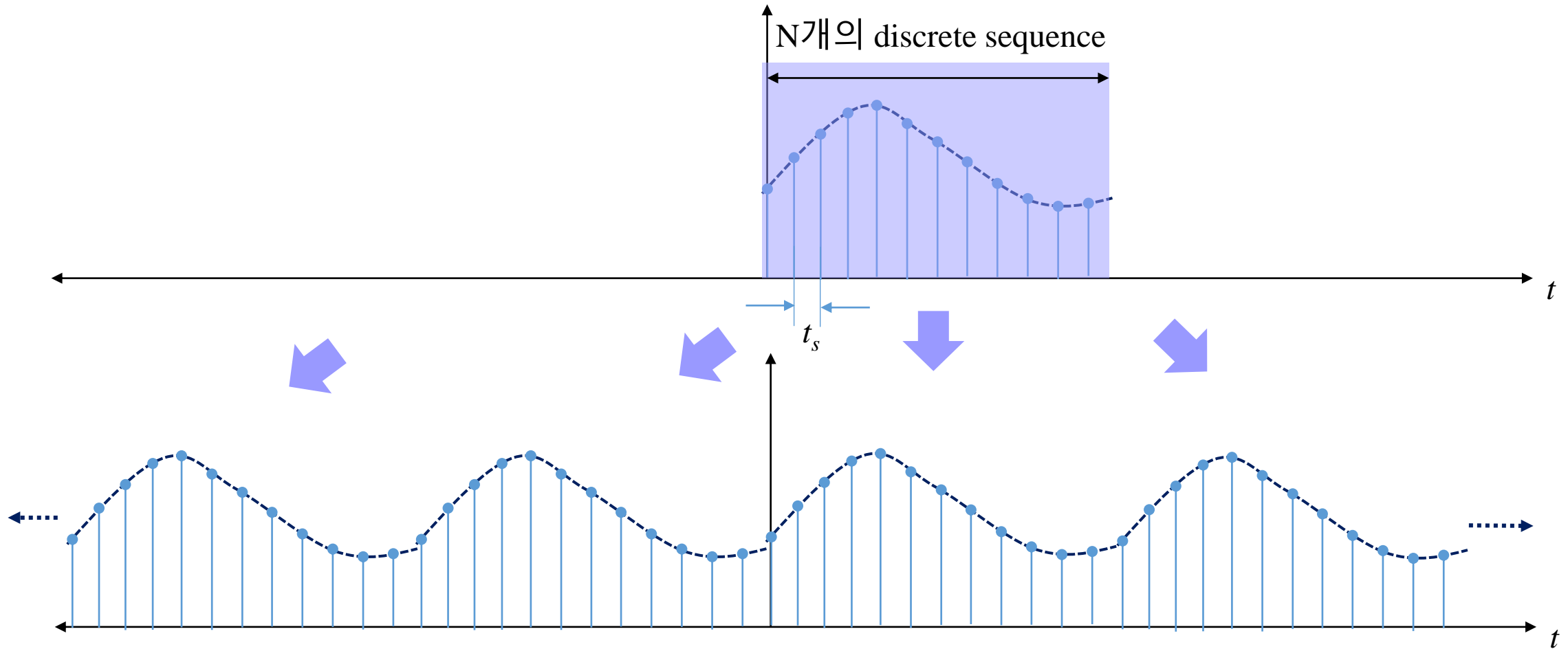
주파수 축에서 discrete 하게 나오기 위한 조건



Time/Frequency 축 모두 discrete하기 위한 조건

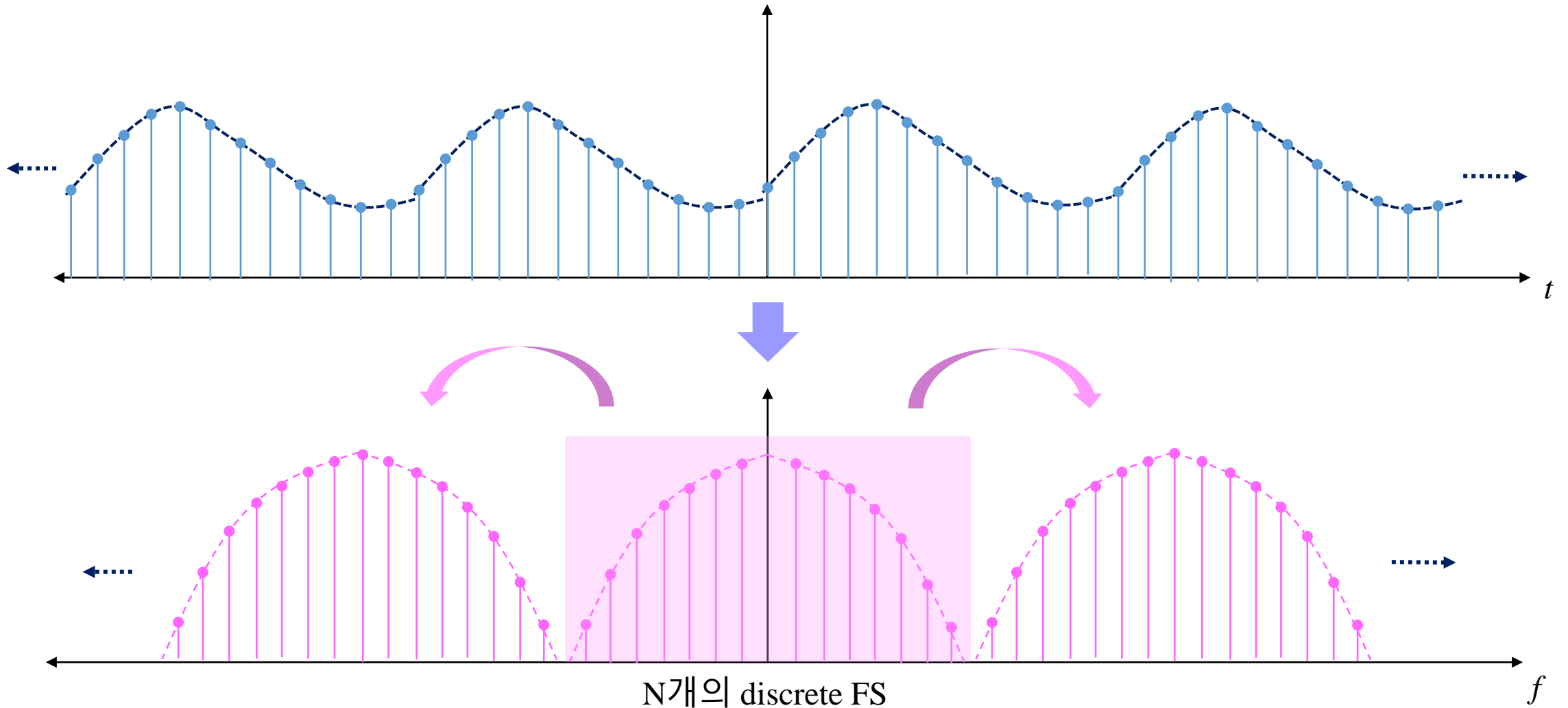


Discrete Fourier transform의 도출1



N개의 discrete sequence 가 반복적으로 나타난다고 가정 \rightarrow 주기가 Nt_s 인 주기함수.

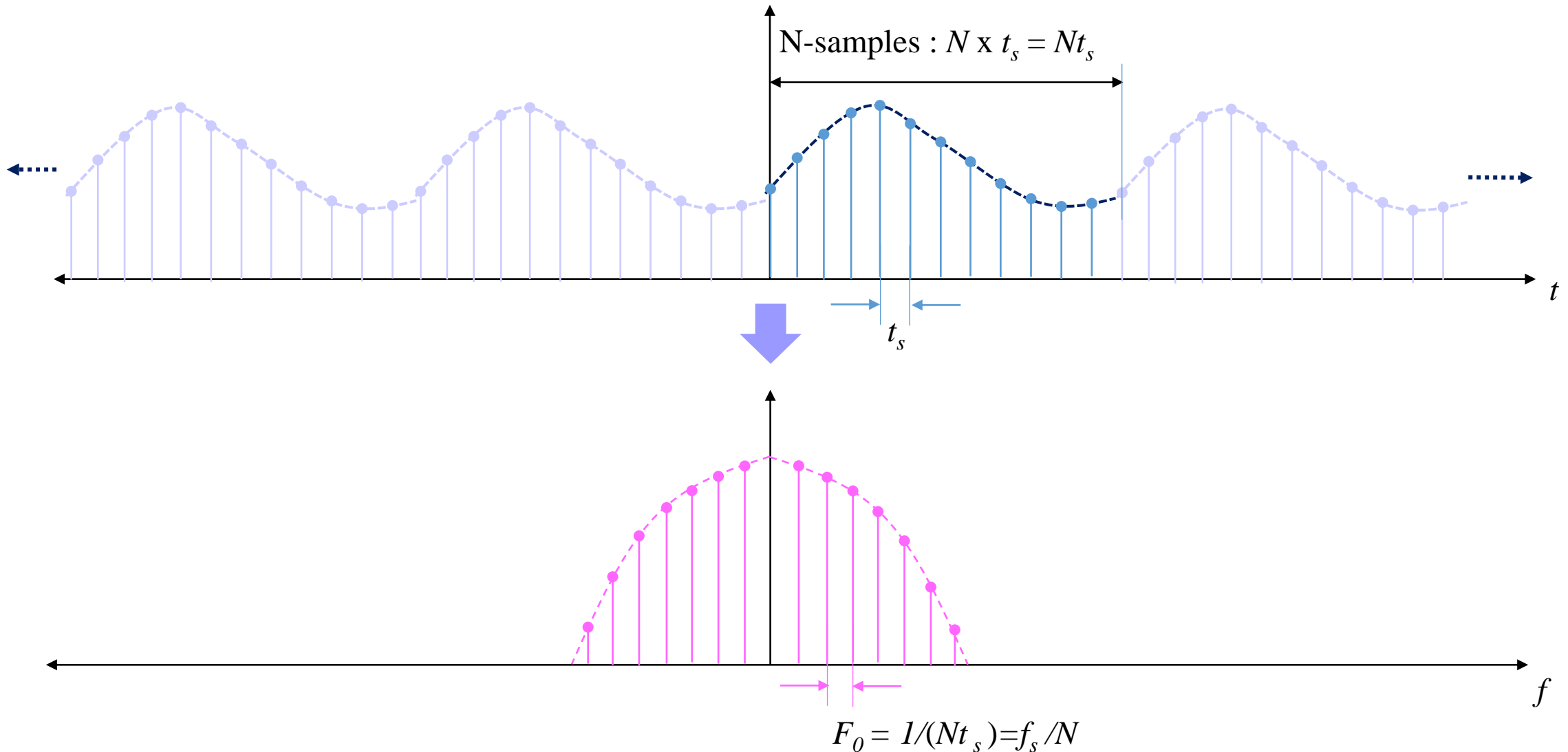
Discrete Fourier transform의 도출2



Discrete Fourier transform (정리)

- FT를 실제로 계산하기 위해 고안된 방법
- Time signal, frequency spectrum 모두 유한개의 discrete sample로 표현됨
 - CFT: continuous time signal \Leftrightarrow continuous frequency spectrum
- Frequency spectrum이 discrete하게 표현되기 위해 time sequence가 반복적으로 나타난다고 가정 (강제로 주기함수로 만듦) -> FS를 적용.
- FS계수는 샘플링 주파수 간격마다 동일하게 나타나므로 샘플링 주파수 구간에 대해서만 FS를 계산한다.

FS 각 계수는 어떤 주파수를 나타내는가?



DFT계수의 유도

$$c_m = \frac{1}{T} \int_{-T/2}^{T/2} x(t) e^{-jm\omega_0 t} dt$$

$$\omega_0 = \frac{1}{Nt_s}, \quad t \rightarrow nt_s$$

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{nm}{N}}$$

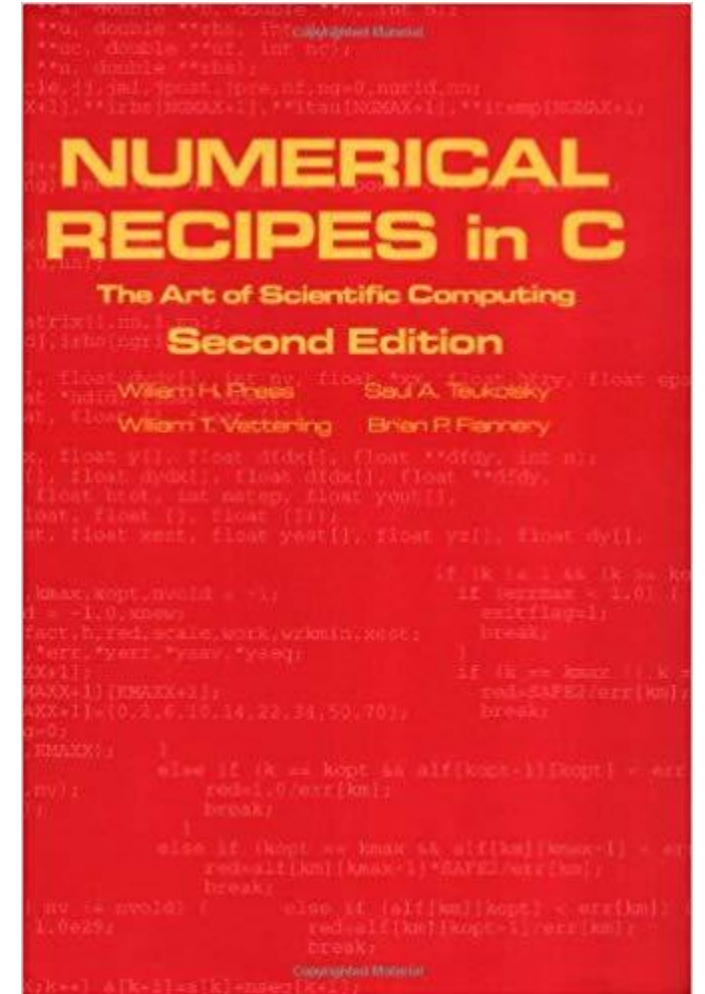
$$0 \leq m \leq N-1$$

DFT of $x(n)$, $0 \leq n \leq N-1$

Library 함수를 이용한 DFT/FFT의 구현

Numerical Recipes in C

- 각종 행렬 연산 및 신호처리 관련 연산 함수 포함.
- Source code를 제공.
 - DFT관련 함수
 - DFT
 - Inverse DFT
 - FFT
 - Inverse FFT
 - 각종 윈도우 함수 생성.
- 강의 자료에 첨부된 “dft.cpp”, “dft.h” 를 download.



fft - Microsoft Visual Studio

빠른 실행(Ctrl+Q)

파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 팀(M) Nsight 도구(T) 테스트(S) 분석(N) 창(W) 도움말(H) 로그인

Release x64 로컬 Windows 디버거

dft.h dft.cpp **fft.cpp**

fft (전역 범위) main(int argc, char ** argv)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #pragma comment(lib, "C:\\main\\work.19\\sound\\fft\\fft64\\libfft3-3.lib")
5 #include "C:\\main\\work.19\\sound\\fft\\fft64\\fft3.h"
6 #include "C:\\main\\work.19\\sound\\fft\\dft\\dft.h"
7
8 int main(int argc, char **argv)
9 {
10     fftw_complex *data, *fft_result, *ifft_result;
11     fftw_plan plan_forward, plan_backward;
12     int i, F_SIZE;
13     COMPLEX *ft_in, *ft_out;
14     clock_t start, end;
15
16     printf("Enter FFT SIZE : ");
17     scanf("%d", &F_SIZE);
18
19     ft_in = (COMPLEX *)malloc(sizeof(COMPLEX)*F_SIZE);
20     ft_out = (COMPLEX *)malloc(sizeof(COMPLEX)*F_SIZE);
21
22     for (i = 0; i < F_SIZE; i++) {
23         ft_in[i].real = i;
24         ft_in[i].imag = i*0.5;
25     }
26
27     start = clock();
28     for (i = 0; i < 100; i++) {
```

Header file을 include 시키고

Source file을 프로젝트에 add.

솔루션 탐색기

솔루션 탐색기 검색(Ctrl+;)

솔루션 'fft' (1개 프로젝트)

- fft
 - 참조
 - 외부 종속성
 - 리소스 파일
 - 소스 파일
 - dft.cpp
 - fft.cpp
 - stdafx.cpp
 - 헤더 파일
 - stdafx.h
 - targetver.h
 - ReadMe.txt

솔루션 탐색기 팀 탐색기 클래스 뷰

속성

main VCppCodeFunction

```
#include <stdio.h>
#include <stdlib.h>
#include "\\main\\work.19\\sound\\fft\\dft\\dft.h"
```

```
int main(int argc, char **argv)
{
    int i, F_SIZE;
    COMPLEX *ft_in, *ft_out;
    clock_t start, end;

    printf("Enter FFT SIZE : ");
    scanf("%d", &F_SIZE);

    ft_in = (COMPLEX *)malloc(sizeof(COMPLEX)*F_SIZE);
    ft_out = (COMPLEX *)malloc(sizeof(COMPLEX)*F_SIZE);

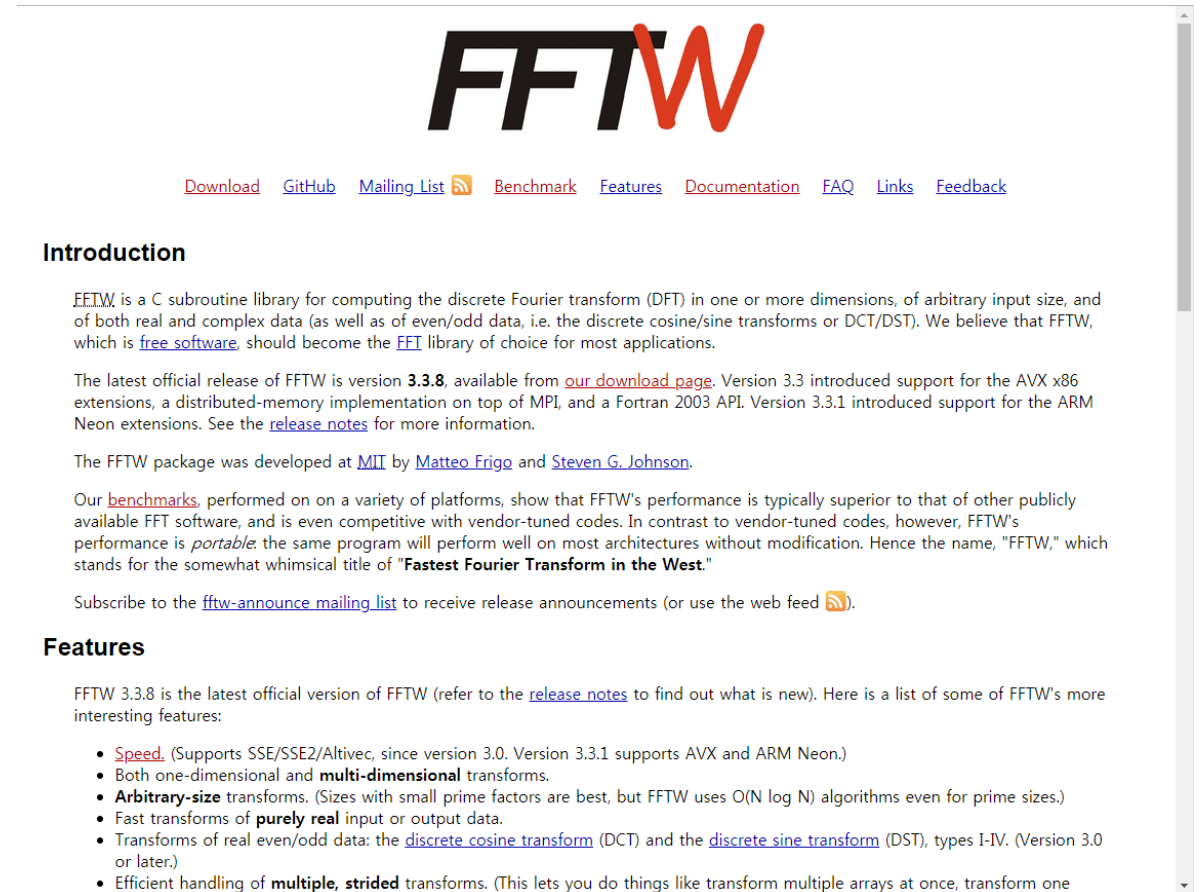
    for (i = 0; i<F_SIZE; i++) {
        ft_in[i].real = i;
        ft_in[i].imag = i*0.5;
    }

    dft(ft_in, ft_out, F_SIZE);
    idft(ft_out, ft_in, F_SIZE);

    fft(ft_in, log2((unsigned)F_SIZE));
    ifft(ft_in, log2((unsigned)F_SIZE));
}
```

FFTW

- MIT에서 개발한 FFT library
- "FFT in the West"
 - FFTW received the [1999 J. H. Wilkinson Prize for Numerical Software](#)
- 강의 자료에 첨부된 아래 파일들을 download.
 - Libfftw3-3.lib
 - Libfftw3-3.dll
 - Fftw3.h



The screenshot shows the official FFTW website. At the top is the FFTW logo, with 'FFT' in black and 'W' in red. Below the logo is a navigation bar with links: Download, GitHub, Mailing List, Benchmark, Features, Documentation, FAQ, Links, and Feedback. The main content area starts with an 'Introduction' section, followed by a paragraph describing FFTW as a C subroutine library for computing the discrete Fourier transform (DFT). It then mentions the latest official release is version 3.3.8, available from the download page. A paragraph follows stating the package was developed at MIT by Matteo Frigo and Steven G. Johnson. Another paragraph discusses benchmarks, noting FFTW's performance is typically superior to other publicly available FFT software. A subscription link for the mailing list is provided. The 'Features' section lists several key capabilities: Speed (supporting SSE/SSE2/Altivec and AVX/ARM Neon), Both one-dimensional and multi-dimensional transforms, Arbitrary-size transforms, Fast transforms of purely real input/output data, Transforms of real even/odd data (DCT and DST), and Efficient handling of multiple, strided transforms.

FFTW

[Download](#) [GitHub](#) [Mailing List](#) [Benchmark](#) [Features](#) [Documentation](#) [FAQ](#) [Links](#) [Feedback](#)

Introduction

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST). We believe that FFTW, which is [free software](#), should become the [FFT](#) library of choice for most applications.

The latest official release of FFTW is version **3.3.8**, available from [our download page](#). Version 3.3 introduced support for the AVX x86 extensions, a distributed-memory implementation on top of MPI, and a Fortran 2003 API. Version 3.3.1 introduced support for the ARM Neon extensions. See the [release notes](#) for more information.

The FFTW package was developed at [MIT](#) by [Matteo Frigo](#) and [Steven G. Johnson](#).

Our [benchmarks](#), performed on a variety of platforms, show that FFTW's performance is typically superior to that of other publicly available FFT software, and is even competitive with vendor-tuned codes. In contrast to vendor-tuned codes, however, FFTW's performance is *portable*: the same program will perform well on most architectures without modification. Hence the name, "FFTW," which stands for the somewhat whimsical title of "**F**astest **F**ourier **T**ransform in the **W**est."

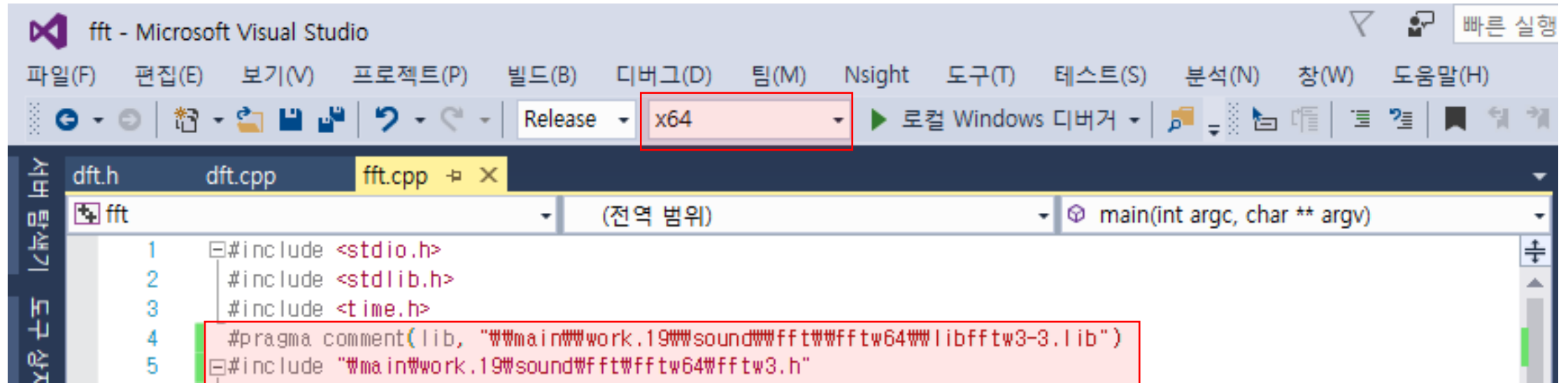
Subscribe to the [fftw-announce mailing list](#) to receive release announcements (or use the web feed [RSS](#)).

Features

FFTW 3.3.8 is the latest official version of FFTW (refer to the [release notes](#) to find out what is new). Here is a list of some of FFTW's more interesting features:

- [Speed](#). (Supports SSE/SSE2/Altivec, since version 3.0. Version 3.3.1 supports AVX and ARM Neon.)
- Both one-dimensional and **multi-dimensional** transforms.
- **Arbitrary-size** transforms. (Sizes with small prime factors are best, but FFTW uses $O(N \log N)$ algorithms even for prime sizes.)
- Fast transforms of **purely real** input or output data.
- Transforms of real even/odd data: the [discrete cosine transform](#) (DCT) and the [discrete sine transform](#) (DST), types I-IV. (Version 3.0 or later.)
- Efficient handling of **multiple, strided** transforms. (This lets you do things like transform multiple arrays at once, transform one

FFTW 사용 시 주의 사항



- 배포된 library가 64bit용이므로 Build 시 "x64"로 설정.
- Library file(libfftw3-3.lib) 경로를 지정해야 함.
- Header file(fftw3.h) 경로를 지정해야 함.
- 실행 홀더에 "libfftw3-3.dll" 파일이 존재해야 함.

```
#include <stdio.h>
#include <stdlib.h>
#pragma comment(lib, "\\main\\work.19\\sound\\fft\\fftw64\\libfftw3-3.lib")
#include "\\main\\work.19\\sound\\fft\\fftw64\\fftw3.h"

int main(int argc, char **argv)
{
    fftw_complex *data, *fft_result, *ifft_result;
    fftw_plan plan_forward, plan_backward;
    int i, F_SIZE;

    printf("Enter FFT SIZE : ");
    scanf("%d", &F_SIZE);

    data = (fftw_complex *)fftw_malloc(sizeof(fftw_complex)*F_SIZE);
    fft_result = (fftw_complex *)fftw_malloc(sizeof(fftw_complex)*F_SIZE);
    ifft_result = (fftw_complex *)fftw_malloc(sizeof(fftw_complex)*F_SIZE);

    plan_forward = fftw_plan_dft_1d(F_SIZE, data, fft_result, FFTW_FORWARD, FFTW_ESTIMATE);
    plan_backward = fftw_plan_dft_1d(F_SIZE, fft_result, ifft_result, FFTW_BACKWARD, FFTW_ESTIMATE);

    for (i = 0; i<F_SIZE; i++) {
        data[i][0] = i;
        data[i][1] = i*0.5;
    }
}
```

```
fftw_execute(plan_forward);  
fftw_execute(plan_backward);
```

```
fftw_destroy_plan(plan_forward);  
fftw_destroy_plan(plan_backward);
```

```
fftw_free(data);  
fftw_free(fft_result);  
fftw_free(ifft_result);
```

```
return 0;  
}
```

DFT/FFT/FFTW speed benchmarking

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#pragma comment(lib, "\\main\\work.19\\sound\\fft\\fftw64\\libfftw3-3.lib")
#include "\\main\\work.19\\sound\\fft\\fftw64\\fftw3.h"
#include "\\main\\work.19\\sound\\fft\\dft\\dft.h"
```

```
int main(int argc, char **argv)
{
    fftw_complex  *data, *fft_result, *ifft_result;
    fftw_plan  plan_forward, plan_backward;
    int  i, F_SIZE;
    COMPLEX  *ft_in, *ft_out;
    clock_t  start, end;

    printf("Enter FFT SIZE : ");
    scanf("%d", &F_SIZE);

    ft_in = (COMPLEX *)malloc(sizeof(COMPLEX)*F_SIZE);
    ft_out = (COMPLEX *)malloc(sizeof(COMPLEX)*F_SIZE);
```



```
for (i = 0; i < F_SIZE; i++) {  
    ft_in[i].real = i;  
    ft_in[i].imag = i*0.5;  
}  
  
start = clock();  
for (i = 0; i < 100; i++) {  
    dft(ft_in, ft_out, F_SIZE);  
    idft(ft_out, ft_in, F_SIZE);  
}  
end = clock();  
printf("[DFT]Eclipse time = %6.3f sec.\n", (end - start)*0.001);
```

```
start = clock();  
for (i = 0; i < 100; i++) {  
    fft(ft_in, log2((unsigned)F_SIZE));  
    ifft(ft_in, log2((unsigned)F_SIZE));  
}  
end = clock();  
printf("[FFT]Eclipse time = %6.3f sec.\n", (end - start)*0.001);
```

```
data = (fftw_complex *)fftw_malloc(sizeof(fftw_complex)*F_SIZE);  
fft_result = (fftw_complex *)fftw_malloc(sizeof(fftw_complex)*F_SIZE);  
ifft_result = (fftw_complex *)fftw_malloc(sizeof(fftw_complex)*F_SIZE);
```

```
plan_forward = fftw_plan_dft_1d(F_SIZE, data, fft_result, FFTW_FORWARD, FFTW_ESTIMATE);  
plan_backward = fftw_plan_dft_1d(F_SIZE, fft_result, ifft_result, FFTW_BACKWARD, FFTW_ESTIMATE);
```

```
for (i = 0; i < F_SIZE; i++) {  
    data[i][0] = i;  
    data[i][1] = i*0.5;  
}
```

```
start = clock();  
for (i = 0; i < 100; i++) {  
    fftw_execute(plan_forward);  
    fftw_execute(plan_backward);  
}  
end = clock();  
printf("FFTW]Eclipse time = %6.3f sec.\n", (end - start)*0.001);
```

```
fftw_destroy_plan(plan_forward);  
fftw_destroy_plan(plan_backward);
```

```
fftw_free(data);  
fftw_free(fft_result);  
fftw_free(ifft_result);
```

```
return 0;  
}
```

```
E:\Wmain\work.19\sound\wfft>.Wx64WreleaseWfft
Enter FFT SIZE : 1024
[DFT]Eclipse time = 0.617 sec.
[FFT]Eclipse time = 0.002 sec.
FFTW]Eclipse time = 0.000 sec.

E:\Wmain\work.19\sound\wfft>.Wx64WreleaseWfft
Enter FFT SIZE : 2048
[DFT]Eclipse time = 2.478 sec.
[FFT]Eclipse time = 0.005 sec.
FFTW]Eclipse time = 0.001 sec.

E:\Wmain\work.19\sound\wfft>.Wx64WreleaseWfft
Enter FFT SIZE : 4096
[DFT]Eclipse time = 10.358 sec.
[FFT]Eclipse time = 0.012 sec.
FFTW]Eclipse time = 0.003 sec.

E:\Wmain\work.19\sound\wfft>.Wx64WreleaseWfft
Enter FFT SIZE : 8192
[DFT]Eclipse time = 44.632 sec.
[FFT]Eclipse time = 0.039 sec.
FFTW]Eclipse time = 0.006 sec.

E:\Wmain\work.19\sound\wfft>.Wx64WreleaseWfft
Enter FFT SIZE : 16384
[DFT]Eclipse time = 190.799 sec.
[FFT]Eclipse time = 0.098 sec.
FFTW]Eclipse time = 0.015 sec.
```

Check points & project

- 사용 컴퓨터에서 DFT/FFT/FFTW speed benchmarking.
- 배포된 코드에서 forward DFT/FFT/FFTW 후 inverse DFT/FFT/FFTW를 수행한 경우 원래의 data값이 나오는지 확인.
- Forward FFT수행 후 저주파 성분을 제거하고 inverse FFT를 수행한 경우 파형 모양 확인.
- Forward FFT수행 후 고주파 성분을 제거하고 inverse FFT를 수행한 경우 파형 모양 확인.