DSP Lab. Week 11
Filtering (FIR, IIR, Poles&Zeros)

Kyuheon Kim
Media Lab. Rm567
kyuheonkim@khu.ac.kr
Last update : September 2, 2019

3-PT AVERAGE SYSTEM

Filtering (FIR)

- ADD 3 CONSECUTIVE NUMBERS
  - Do this for each "n"

the following input–output equation

Make a TABLE

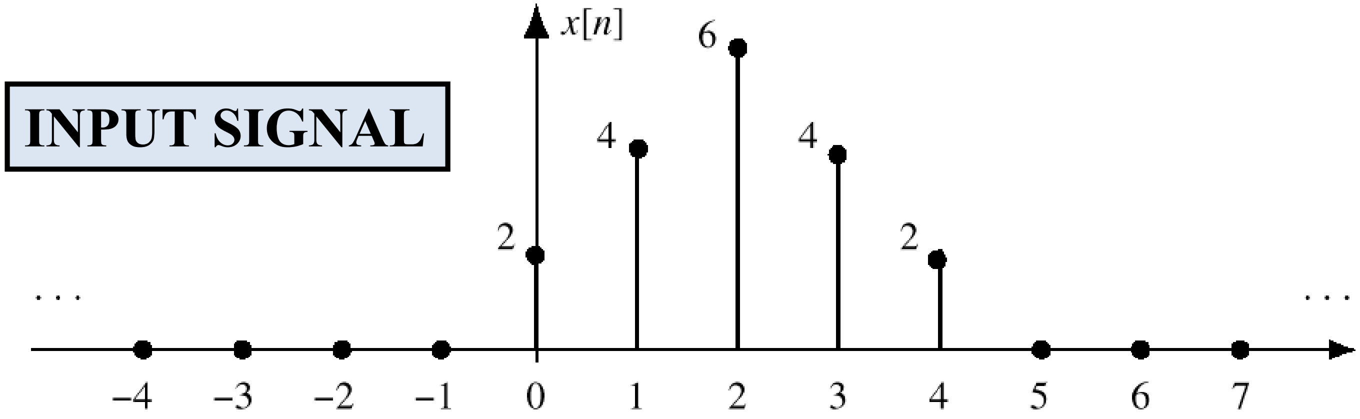
$$y[n] = \tfrac{1}{3}(x[n] + x[n+1] + x[n+2])$$

n	$n < -2$	-2	-1	0	1	2	3	4	5	$n > 5$
$x[n]$	0	0	0	2	4	**6**	**4**	**2**	0	0
$y[n]$	0	$\frac{2}{3}$	2	4	$\frac{14}{3}$	**4**	2	$\frac{2}{3}$	0	0

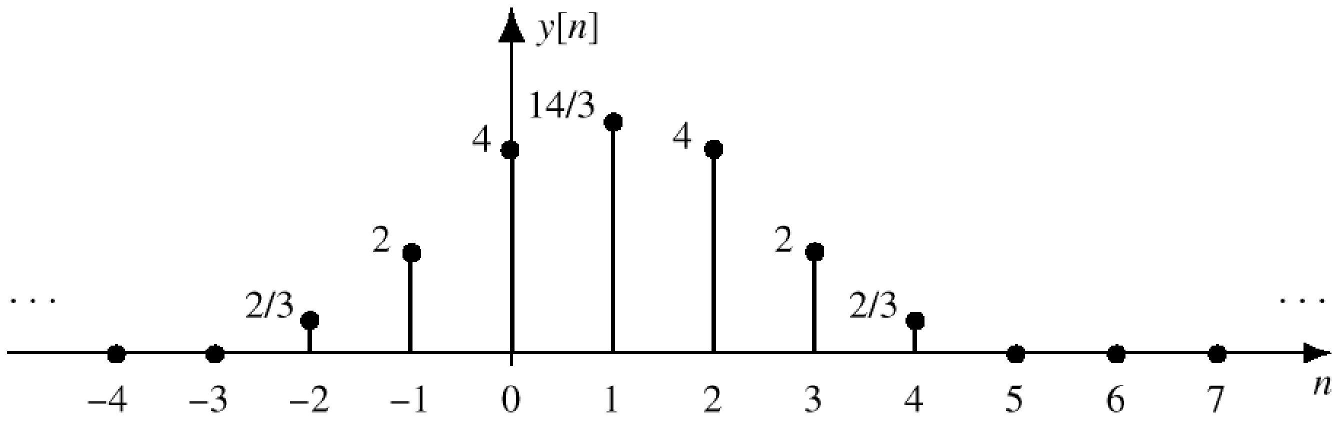
n=0 $y[0] = \frac{1}{3}(x[0] + x[1] + x[2])$

n=1 $y[1] = \frac{1}{3}(x[1] + x[2] + x[3])$

INPUT SIGNAL



$$y[n] = \frac{1}{3}(x[n] + x[n+1] + x[n+2])$$



OUTPUT SIGNAL

- FILTER COEFFICIENTS $\{b_k\}$
  - DEFINE THE FILTER

$$y[n]=\sum_{k=0}^{M} b_k x[n-k]$$

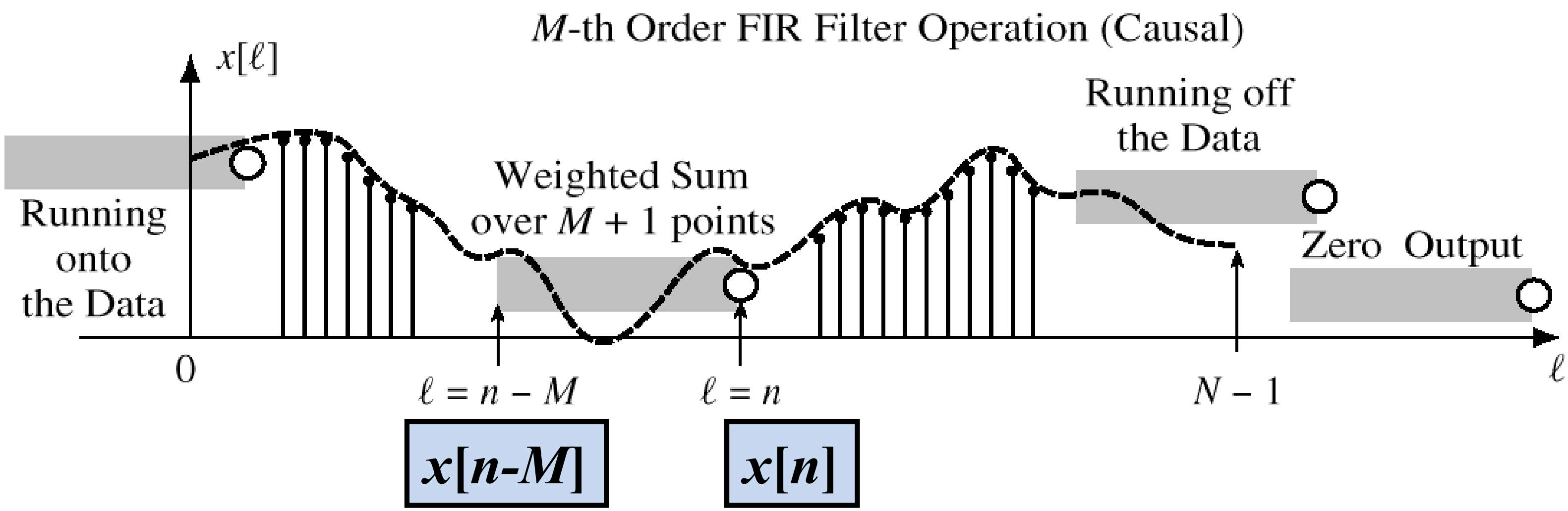
  - For example,

$$\boxed{b_k=\{3,-1,2,1\}}$$

$$\begin{aligned} y[n] &= \sum_{k=0}^{3} b_k x[n-k] \\ &= 3x[n]-x[n-1]+2x[n-2]+x[n-3] \end{aligned}$$

- SLIDE a WINDOW across x[n]

$$y[n] = \sum_{k=0}^{M} b_k x[n-k]$$



FILTERING EXAMPLE

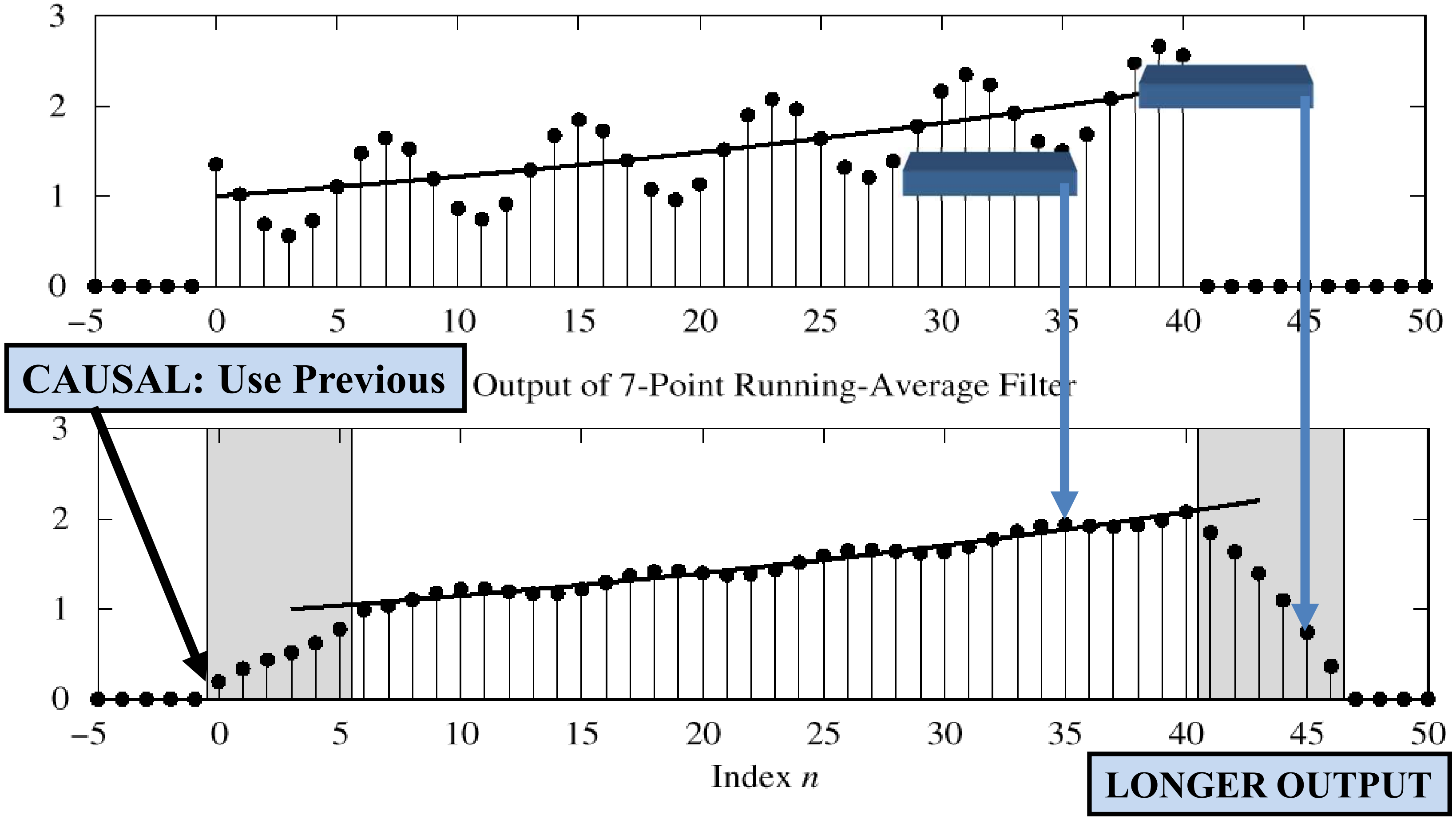
- 7-point AVERAGER
  - Removes cosine
    - By making its amplitude (A) smaller

$$y_7[n] = \sum_{k=0}^{6} \left(\tfrac{1}{7}\right) x[n-k]$$

- **3-point AVERAGER**
  - **Changes A slightly**

$$y_3[n] = \sum_{k=0}^{2} \left(\tfrac{1}{3}\right) x[n-k]$$

$$\text{Input}: x[n] = (1.02)^n + \cos(2\pi n/8 + \pi/4) \quad \text{for } 0 \le n \le 40$$



FIR IMPULSE RESPONSE

- Convolution = Filter Definition
  - Filter Coeffs = Impulse Response

n	$n<0$	0	1	2	3	$\dots$	M	$M+1$	$n>M+1$
$x[n]=\delta[n]$	0	1	0	0	0	0	0	0	0
$y[n]=h[n]$	0	b_0	b_1	b_2	b_3	$\dots$	b_M	0	0

$$y[n]=\sum_{k=0}^{M} b_k x[n-k]$$

$$y[n]=\sum_{k=0}^{M} h[k]x[n-k]$$

CONVOLUTION

- Output = Convolution of x[n] & h[n]
  - NOTATION:
  - Here is the FIR case:

$$y[n] = h[n] * x[n]$$

FINITE LIMITS

$$y[n] = \sum_{k=0}^{M} h[k]x[n-k]$$

Same as $\mathbf{b_k}$

FINITE LIMITS

Filtering (IIR)

- INFINITE IMPULSE RESPONSE FILTERS
  - Define IIR DIGITAL Filters
  - Have FEEDBACK: use PREVIOUS OUTPUTS

$$y[n] = \sum_{l=1}^{N} a_l y[n-l] + \sum_{k=0}^{M} b_k x[n-k]$$

- **Show how to compute the output y[n]**
  - ✓ **FIRST-ORDER CASE (N=1)**
  - ✓ **Z-transform: Impulse Response h[n] ←→ H(z)**

- ADD PREVIOUS OUTPUTS

$$y[n] = a_1 y[n-1] + b_0 x[n] + b_1 x[n-1]$$

PREVIOUS
FEEDBACK

FIR PART of the FILTER
FEED-FORWARD

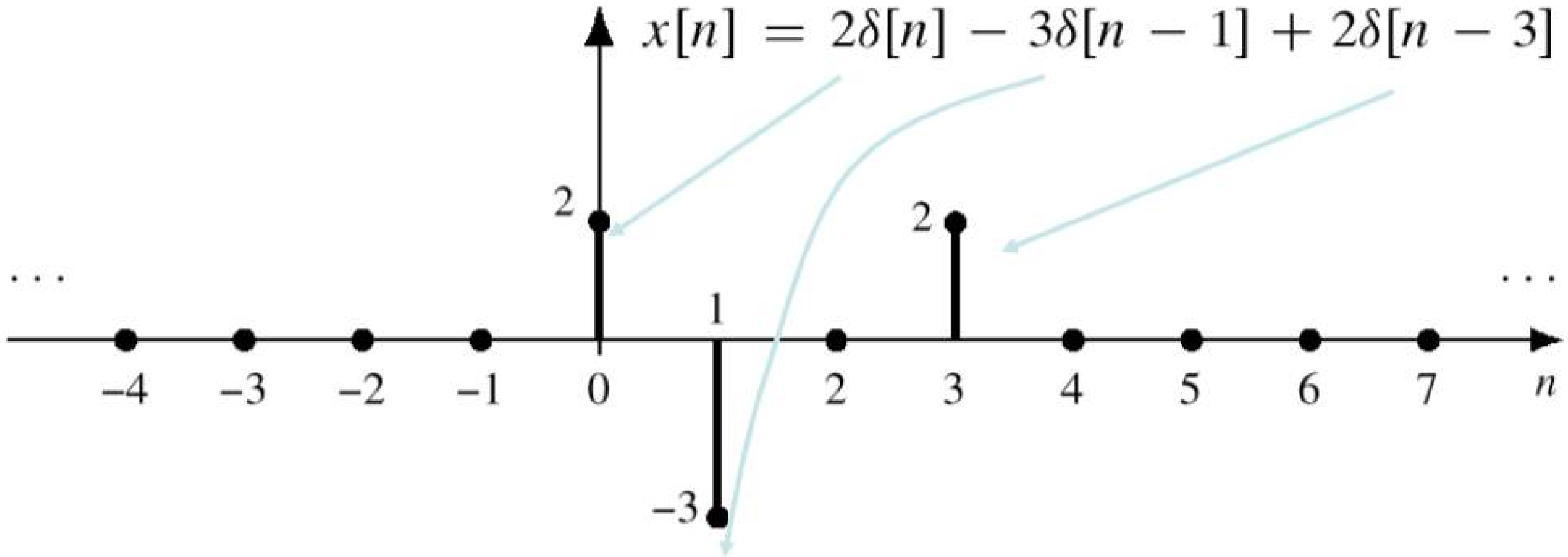
$$y[n] = 0.8y[n-1] + 3x[n] - 2x[n-1]$$

❖ **CAUSALITY**

- **NOT USING FUTURE OUTPUTS or INPUTS**

$$\boxed{y[n] = 0.8y[n-1] + 5x[n]}$$

$$x[n] = 2\delta[n] - 3\delta[n-1] + 2\delta[n-3]$$



- **FEEDBACK DIFFERENCE EQUATION:**

$$\boxed{y[n] = 0.8y[n-1] + 5x[n]}$$

- **NEED y[-1] to get started**

$$\boxed{y[0] = 0.8y[-1] + 5x[0]}$$

- **y[n] = 0, for n<0**
- **BECAUSE x[n] = 0, for n<0**

INITIAL REST CONDITIONS

1. The input must be assumed to be zero prior to some starting time n_0, i.e., $x[n] = 0$ for $n < n_0$. We say that such inputs are *suddenly applied.*
2. The output is likewise assumed to be zero prior to the starting time of the signal, i.e., $y[n] = 0$ for $n < n_0$. We say that the system is *initially at rest* if its output is zero prior to the application of a suddenly applied input.

$$y[n] = a_1 y[n-1] + b_0 x[n]$$

$$h[n] = a_1 h[n-1] + b_0 \delta[n]$$

n	$n<0$	0	1	2	3	4
$\delta[n]$	0	1	0	0	0	0
$h[n-1]$	0	0	b_0	$b_0(a_1)$	$b_0(a_1)^2$	$b_0(a_1)^3$
$h[n]$	0	b_0	$b_0(a_1)$	$b_0(a_1)^2$	$b_0(a_1)^3$	$b_0(a_1)^4$

From this table it is obvious that the general formula is

$$h[n] = \begin{cases} b_0(a_1)^n & \text{for } n \geq 0 \\ 0 & \text{for } n < 0 \end{cases}$$

$$h[n] = b_0(a_1)^n u[n]$$

$$u[n] = 1, \quad \text{for } n \geq 0$$

❖ POLYNOMIAL Representation

$$H(z) = \sum_{n=-\infty}^{\infty} h[n] z^{-n}$$

APPLIES to Any SIGNAL

❖ SIMPLIFY the SUMMATION

$$H(z) = \sum_{n=-\infty}^{\infty} b_0 (a_1)^n u[n] z^{-n} = b_0 \sum_{n=0}^{\infty} a_1^n z^{-n}$$

$$H(z) = b_0 \sum_{n=0}^{\infty} a_1^n z^{-n} = b_0 \sum_{n=0}^{\infty} (a_1 z^{-1})^n$$

$$= \frac{b_0}{1 - a_1 z^{-1}} \quad \text{if } |z| > |a_1|$$

❖ ANOTHER FIRST-ORDER IIR FILTER:

$$y[n] = a_1 y[n-1] + b_0 x[n] + b_1 x[n-1]$$

$$h[n] = b_0 (a_1)^n u[n] + b_1 (a_1)^{n-1} u[n-1]$$

z^{-1} is a shift

$$H(z) = \frac{b_0}{1 - a_1 z^{-1}} + \frac{b_1 z^{-1}}{1 - a_1 z^{-1}} = \frac{b_0 + b_1 z^{-1}}{1 - a_1 z^{-1}}$$

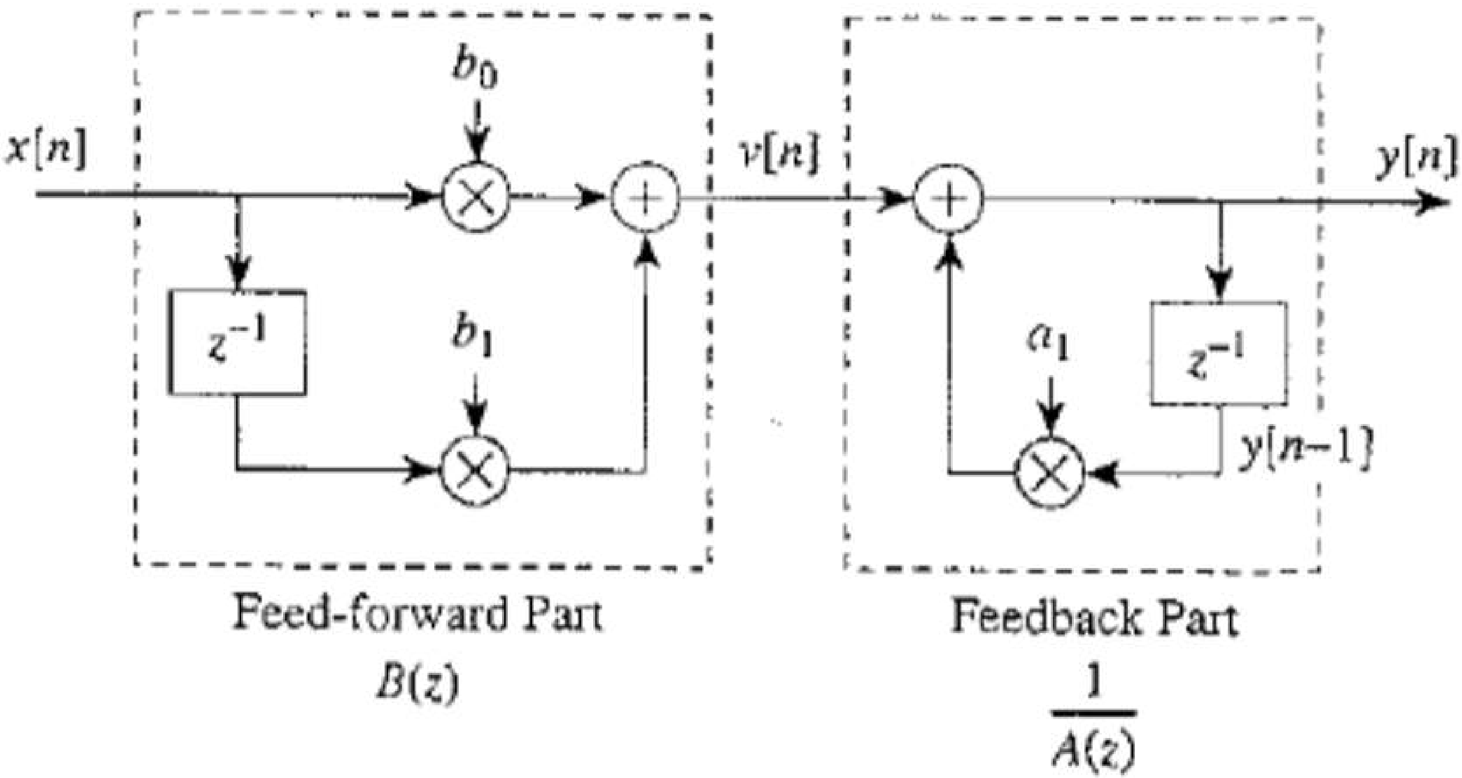
❖ NOTE the FILTER COEFFICIENTS

$$Y(z) - a_1 z^{-1} Y(z) = b_0 X(z) + b_1 z^{-1} X(z)$$

$$(1 - a_1 z^{-1}) Y(z) = (b_0 + b_1 z^{-1}) X(z)$$

$$H(z) = \frac{Y(z)}{X(z)} = \boxed{\frac{b_0 + b_1 z^{-1}}{1 - a_1 z^{-1}}} = \frac{B(z)}{A(z)}$$

Direct form I



Direct form II

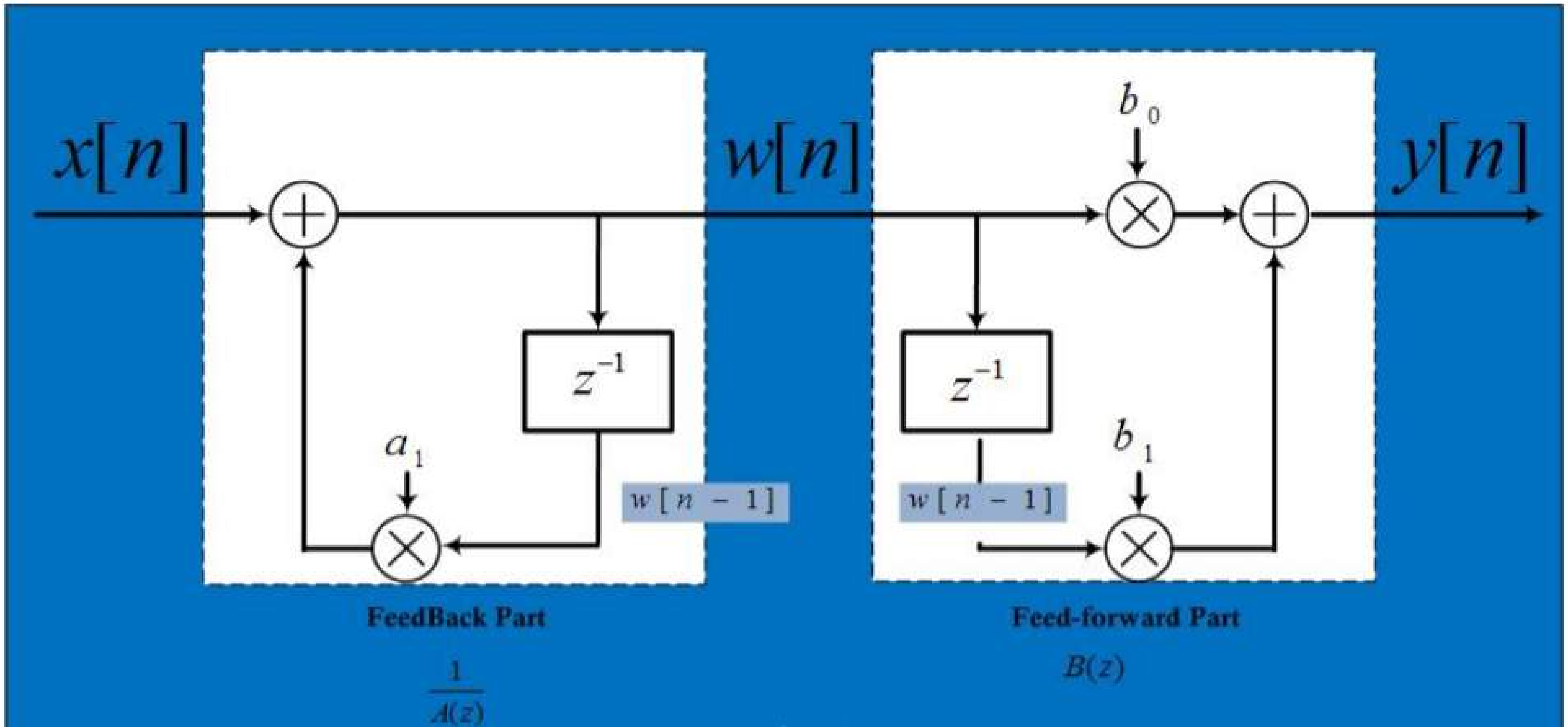
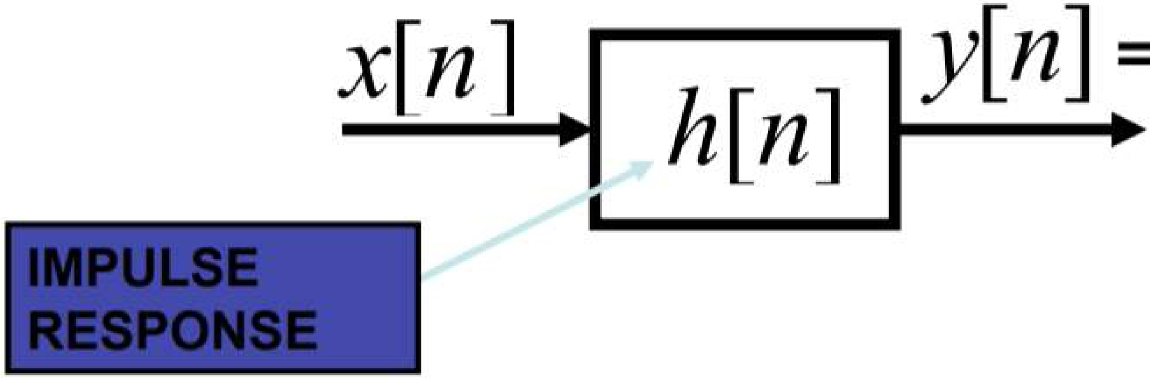


Figure 8.6

❖ MULTIPLICATION of z-TRANSFORMS

$$Y(z) = H(z)X(z)$$

❖ CONVOLUTION in TIME-DOMAIN



$$y[n] = h[n] * x[n]$$

Poles & Zeros

❖ ROOTS of Numerator & Denominator

$$H(z) = \frac{b_0 + b_1 z^{-1}}{1 - a_1 z^{-1}} \rightarrow H(z) = \frac{b_0 z + b_1}{z - a_1}$$

$$b_0 z + b_1 = 0 \quad \Rightarrow z = -\frac{b_1}{b_0}$$

ZERO: H(z)=0

$$z - a_1 = 0 \quad \Rightarrow z = a_1$$

POLE: H(z) → inf

❖ Denominator is QUADRATIC

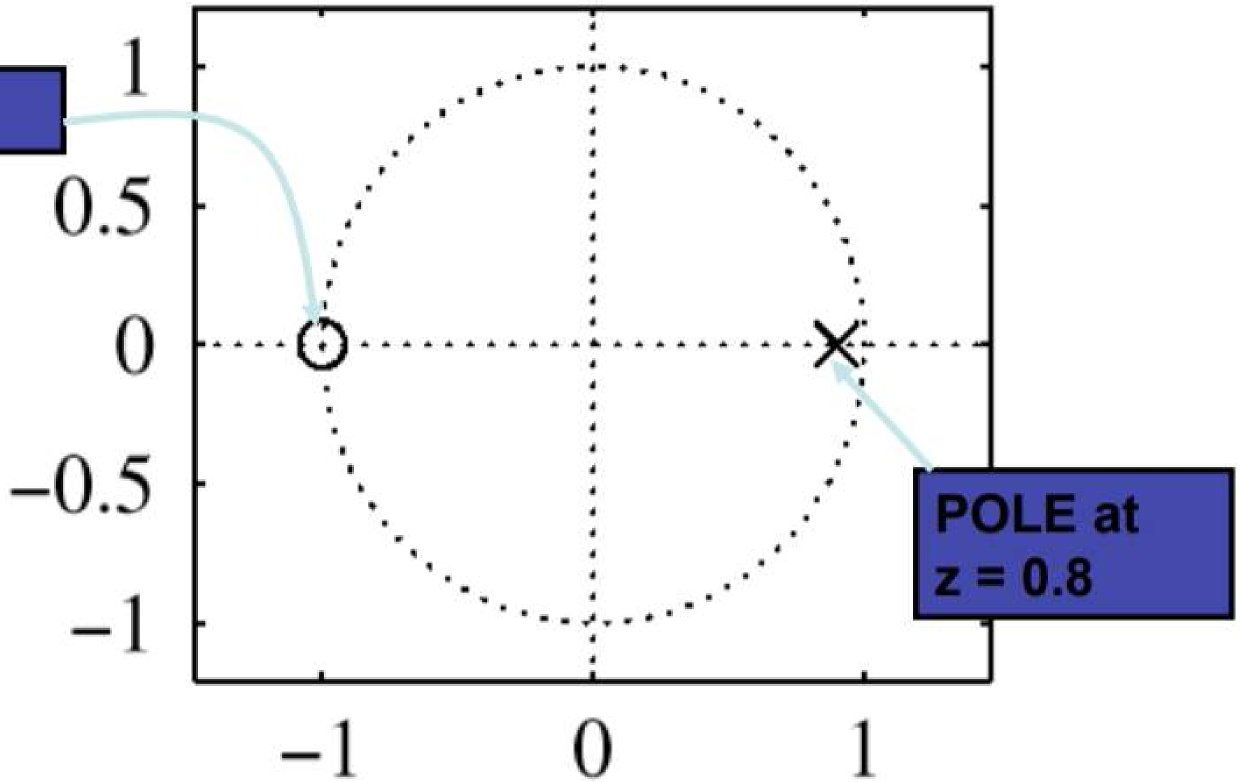
- 2 Poles: REAL
- or COMPLEX CONJUGATES

$$\boxed{\frac{a_1 \pm \sqrt{a_1^2 + 4a_2}}{2}}$$

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}} = \frac{b_0 z^2 + b_1 z + b_2}{z^2 - a_1 z - a_2}$$

PROPERTY OF REAL POLYNOMIALS

A polynomial of degree N has N roots. If all the coefficients of the polynomial are real, the roots either must be real, or must occur in complex conjugate pairs.



$$\frac{2+2z^{-1}}{1-0.8z^{-1}}$$

Stability

- **Nec. & suff. condition:**

$$\sum_{n=-\infty}^{\infty} |h[n]| < \infty$$

$$h[n] = b(a)^n u[n] \Leftrightarrow H(z) = \frac{b}{1 - az^{-1}}$$

$$\sum_{n=0}^{\infty} |b||a|^n < \infty \text{ if } |a| < 1 \Rightarrow$$

Pole must be Inside unit circle

❖ When Does the TRANSIENT DIE OUT ?

STEADY-STATE RESPONSE AND STABILITY

A stable system is one that does not "blow up." This intuitive statement can be formalized by saying that the output of a stable system can always be bounded ($|y[n]| < M_y$) whenever the input is bounded ($|x[n]| < M_x$).[3]

$$y[n] = a_1 y[n-1] + b_0 x[n]$$

$$H(z) = \frac{b_0}{1 - a_1 z^{-1}}$$

$$h[n] = b_0 a_1^n u[n]$$

$$\text{need } |a_1| < 1$$

A causal LTI IIR system with initial rest conditions is stable if all of the poles of its system function lie strictly inside the unit circle of the z-plane.

Inverse Z Transform

SHORT TABLE OF z-TRANSFORMS			
	$x[n]$	$\Longleftrightarrow$	$X(z)$
1.	$ax_1[n] + bx_2[n]$	$\Longleftrightarrow$	$aX_1(z) + bX_2(z)$
2.	$x[n - n_0]$	$\Longleftrightarrow$	$z^{-n_0}X(z)$
3.	$y[n] = x[n] * h[n]$	$\Longleftrightarrow$	$Y(z) = H(z)X(z)$
4.	$\delta[n]$	$\Longleftrightarrow$	1
5.	$\delta[n - n_0]$	$\Longleftrightarrow$	z^{-n_0}
6.	$a^n u[n]$	$\Longleftrightarrow$	$\dfrac{1}{1 - az^{-1}}$

FIR ex1

예제) y[n] = x[n] – x[n-1] 의 일차 차분 시스템을 고려해본다. H($e^{j\hat{w}}$) 의 magnitude 와 phase를 구하여라. fs = 1000 Hz

일차 차분 시스템의 h[n] = $\delta[n] - \delta$[n-1] 이므로 $H\left(e^{j\hat{w}}\right) = 1 - e^{-j\widehat{w}}$

예제) y[n] = x[n] – x[n-1] 의 일차 차분 시스템을 고려해본다. H($e^{j\hat{w}}$) 의 magnitude 와 phase를 구하여라. fs = 1000 Hz

일차 차분 시스템의 h[n] = $\delta[n]$ – δ[n-1] 이므로 $H\left(e^{j\hat{w}}\right) = 1 - e^{-j\hat{w}}$

```
#include<iostream>
#include<fstream>
#include"complex.h"
using namespace std;

#define PI 3.141592

void main()
{
    ofstream out_k, out_mag, out_phs;
    int fs = 1000;
    int L = 3 * fs;

    complex *H = new complex[L];

    out_k.open("k.txt");
    out_mag.open("mag.txt");
    out_phs.open("phs.txt");
    if ((!out_k.is_open()) && (!out_mag.is_open()) && (!out_phs.is_open()))
    {
        cerr << "Error for new file" << endl;
        exit(101);
    }

    for (int k = 0; k < L; k++) {
        H[k] = complex(1.0, 0) - complex(cos(-2.*PI*k / (double)fs), sin(-2.*PI*k / (double)fs));
    }

    for (int k = 0; k < L; k++) {
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
        out_phs << H[k].phase() << endl;
    }


    system("pause");
    return;
}
```

예제) y[n] = x[n] – x[n-1] 의 일차 차분 시스템을 고려해본다. H($e^{j\hat{w}}$) 의 magnitude 와 phase를 구하여라. fs = 1000 Hz

일차 차분 시스템의 h[n] = $\delta[n]$ – δ[n-1] 이므로 $H\left(e^{j\hat{w}}\right) = 1 - e^{-j\hat{w}}$

```
#include<iostream>
#include<fstream>
#include"complex.h"
using namespace std;

#define PI 3.141592

void main()
{
    ofstream out_k, out_mag, out_phs;
    int fs = 1000;
    int L = 3 * fs;

    complex *H = new complex[L];

    out_k.open("k.txt");
    out_mag.open("mag.txt");
    out_phs.open("phs.txt");
    if ((!out_k.is_open()) && (!out_mag.is_open()) && (!out_phs.is_open()))
    {
        cerr << "Error for new file" << endl;
        exit(101);
    }

    for (int k = 0; k < L; k++) {
        H[k] = complex(1.0, 0) - complex(cos(-2.*PI*k / (double)fs), sin(-2.*PI*k / (double)fs));
    }

    for (int k = 0; k < L; k++) {
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
        out_phs << H[k].phase() << endl;
    }


    system("pause");
    return;
}
```

출력을 3주기로 설정 (결과 비교를 위해)

예제) y[n] = x[n] – x[n-1] 의 일차 차분 시스템을 고려해본다. H($e^{j\hat{w}}$) 의 magnitude 와 phase를 구하여라. fs = 1000 Hz

일차 차분 시스템의 h[n] = $\delta[n]$ – δ[n-1] 이므로 $H\left(e^{j\hat{w}}\right) = 1 - e^{-j\hat{w}}$

```
#include<iostream>
#include<fstream>
#include"complex.h"
using namespace std;

#define PI 3.141592

void main()
{
    ofstream out_k, out_mag, out_phs;
    int fs = 1000;
    int L = 3 * fs;

    complex *H = new complex[L];

    out_k.open("k.txt");
    out_mag.open("mag.txt");
    out_phs.open("phs.txt");
    if ((!out_k.is_open()) && (!out_mag.is_open()) && (!out_phs.is_open()))
    {
        cerr << "Error for new file" << endl;
        exit(101);
    }

    for (int k = 0; k < L; k++) {
        H[k] = complex(1.0, 0) - complex(cos(-2.*PI*k / (double)fs), sin(-2.*PI*k / (double)fs));
    }

    for (int k = 0; k < L; k++) {
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
        out_phs << H[k].phase() << endl;
    }


    system("pause");
    return;
}
```

File 설정

예제) y[n] = x[n] – x[n-1] 의 일차 차분 시스템을 고려해본다. H($e^{j\hat{w}}$) 의 magnitude 와 phase를 구하여라. fs = 1000 Hz

일차 차분 시스템의 h[n] = $\delta[n]$ – δ[n-1] 이므로 $\boxed{H(e^{j\hat{w}}) = 1 - e^{-j\hat{w}}}$

```
#include<iostream>
#include<fstream>
#include"complex.h"
using namespace std;

#define PI 3.141592

void main()
{
    ofstream out_k, out_mag, out_phs;
    int fs = 1000;
    int L = 3 * fs;

    complex *H = new complex[L];

    out_k.open("k.txt");
    out_mag.open("mag.txt");
    out_phs.open("phs.txt");
    if ((!out_k.is_open()) && (!out_mag.is_open()) && (!out_phs.is_open()))
    {
        cerr << "Error for new file" << endl;
        exit(101);
    }

    for (int k = 0; k < L; k++) {
        H[k] = complex(1.0, 0) - complex(cos(-2.*PI*k / (double)fs), sin(-2.*PI*k / (double)fs));
    }

    for (int k = 0; k < L; k++) {
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
        out_phs << H[k].phase() << endl;
    }


    system("pause");
    return;
}
```

예제) y[n] = x[n] – x[n-1] 의 일차 차분 시스템을 고려해본다. H($e^{j\hat{w}}$) 의 magnitude 와 phase를 구하여라. fs = 1000 Hz

일차 차분 시스템의 h[n] = $\delta[n]$ – δ[n-1] 이므로 $H\left(e^{j\hat{w}}\right) = 1 - e^{-j\hat{w}}$

만약 h[n]부터 코딩으로 진행한다면?

```
#include<iostream>
#include<fstream>
...
{
    ofstream out_k, out_mag, out_phs;
    int fs = 1000;
    int L = 3 * fs;

    complex *H = new complex[L];

    out_k.open("k.txt");
    out_mag.open("mag.txt");
    out_phs.open("phs.txt");
    if ((!out_k.is_open()) && (!out_mag.is_open()) && (!out_phs.is_open()))
    {
        cerr << "Error for new file" << endl;
        exit(101);
    }

    for (int k = 0; k < L; k++) {
        H[k] = complex(1.0, 0) - complex(cos(-2.*PI*k / (double)fs), sin(-2.*PI*k / (double)fs));
    }

    for (int k = 0; k < L; k++) {
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
        out_phs << H[k].phase() << endl;
    }


    system("pause");
    return;
}
```

예제) y[n] = x[n] – x[n-1] 의 일차 차분 시스템을 고려해본다. $H(e^{j\hat{w}})$ 의 magnitude 와 phase를 구하여라. fs = 1000 Hz

일차 차분 시스템의 $\boxed{h[n] = \delta[n] - \delta[n\text{-}1]}$ 이므로 $H(e^{j\hat{w}}) = 1 - e^{-j\hat{w}}$

```
#include<iostream>
#include<fstream>
```

만약 h[n]부터 코딩으로 진행한다면?

```
    ofstream out_k, out_mag, out_phs;
```

```
int n0 = 0, n1 = 1;
complex* H1 = new complex[fs];
complex* H2 = new complex[fs];
for (int k = 0; k < fs; k++) {
    H1[k] = complex(1., 0)*complex(cos(-2.*PI*k*n0/(double)fs), sin(-2.*PI*k*n0/(double)fs));
    H2[k] = complex(1., 0.)*complex(cos(-2.*PI*k * n1 / (double)fs), sin(-2.*PI*k*n1 / (double)fs));
    H[k] = H1[k] - H2[k];
}
```

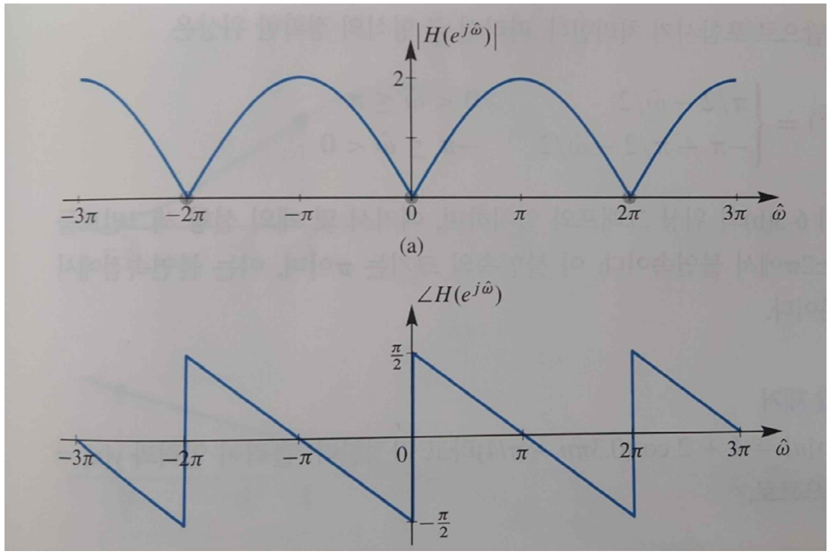
DFT

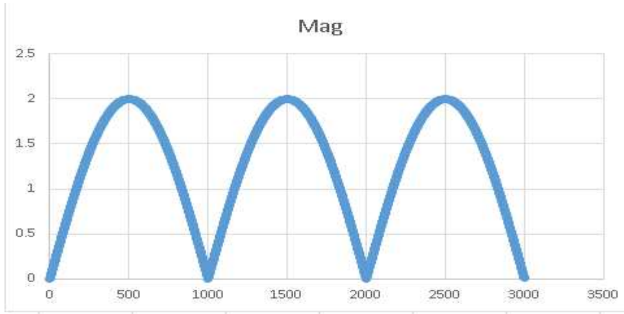
```
        H[k] = complex(1.0, 0) - complex(cos(-2.*PI*k / (double)fs), sin(-2.*PI*k / (double)fs));
    }

    for (int k = 0; k < L; k++) {
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
        out_phs << H[k].phase() << endl;
    }


    system("pause");
    return;
}
```

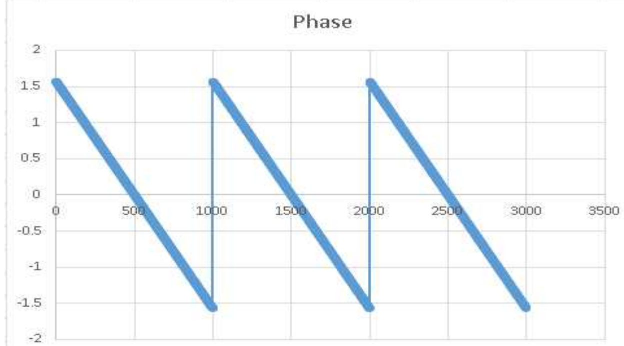
예제) y[n] = x[n] – x[n-1] 의 일차 차분 시스템을 고려해본다. $H(e^{j\hat{w}})$ 의 magnitude 와 phase를 구하여라. fs = 1000 Hz

일차 차분 시스템의 h[n] = $\delta[n] - \delta$[n-1] 이므로 $H\left(e^{j\hat{w}}\right) = 1 - e^{-j\widehat{w}}$

예제 결과 비교 (DSP First vs C++ code)







FIR ex2

예제) y[n] =$\sum_{k=0}^{L-1} x[n-k]$ 의 L-포인트 이동 합 시스템(디리클레 형)의 Magnitude를 구하시오.

L = 11, fs = 1000

$$D_L(e^{j\hat{w}}) = \frac{\sin\left(\frac{\hat{w}L}{2}\right)}{\sin(\frac{\hat{w}}{2})} e^{-j\hat{w}\frac{(L-1)}{2}}$$

예제) y[n] =$\sum_{k=0}^{L-1} x[n-k]$ 의 L-포인트 이동 합 시스템(디리클레 형)의 Magnitude를 구하시오.

L = 11, fs = 1000

$$D_L(e^{j\hat{w}}) = \frac{\sin\left(\frac{\hat{w}L}{2}\right)}{\sin(\frac{\hat{w}}{2})} e^{-j\hat{w}\frac{(L-1)}{2}}$$

```
#include<iostream>
#include<fstream>
#include"complex.h"
using namespace std;

#define PI 3.141592

void Dirichlet(int L, int fs, ofstream& out_k, ofstream& out_mag);


void main()
{
    ofstream out_k, out_mag;
    int fs = 1000;
    int L = 11;


    out_k.open("k.txt");
    out_mag.open("mag.txt");
    if ((!out_k.is_open()) && (!out_mag.is_open()))
    {
        cerr << "Error for new file" << endl;
        exit(101);
    }

    Dirichlet(L, fs, out_k, out_mag);


    system("pause");
    return;
}
```

예제) y[n] =$\sum_{k=0}^{L-1} x[n-k]$ 의 L-포인트 이동 합 시스템(디리클레 형)의 Magnitude를 구하시오.

L = 11, fs = 1000

$$D_L(e^{j\hat{w}}) = \frac{\sin\left(\frac{\hat{w}L}{2}\right)}{\sin(\frac{\hat{w}}{2})} e^{-j\hat{w}\frac{(L-1)}{2}}$$

```
#include<iostream>
#include<fstream>
#include"complex.h"
using namespace std;

#define PI 3.141592

void Dirichlet(int L, int fs, ofstream& out_k, ofstream& out_mag);


void main()
{
    ofstream out_k, out_mag;
    int fs = 1000;
    int L = 11;


    out_k.open("k.txt");
    out_mag.open("mag.txt");
    if ((!out_k.is_open()) && (!out_mag.is_open()))
    {
        cerr << "Error for new file" << endl;
        exit(101);
    }

    Dirichlet(L, fs, out_k, out_mag);


    system("pause");
    return;
}
```

예제) y[n] =$\sum_{k=0}^{L-1} x[n-k]$ 의 L-포인트 이동 합 시스템(디리클레 형)의 Magnitude를 구하시오.

L = 11, fs = 1000

$$D_L(e^{j\hat{w}}) = \frac{\sin\left(\frac{\hat{w}L}{2}\right)}{\sin(\frac{\hat{w}}{2})} e^{-j\hat{w}\frac{(L-1)}{2}}$$

```
#include<iostream>
#include<fstream>
#include"complex.h"
using namespace std;

#define PI 3.141592

void Dirichlet(int L, int fs, ofstream& out_k, ofstream& out_mag);


void main()
{
    ofstream out_k, out_mag;
    int fs = 1000;
    int L = 11;


    out_k.open("k.txt");
    out_mag.open("mag.txt");
    if ((!out_k.is_open()) && (!out_mag.is_open()))
    {
        cerr << "Error for new file" << endl;
        exit(101);
    }

    Dirichlet(L, fs, out_k, out_mag);


    system("pause");
    return;
}
```

예제) y[n] =$\sum_{k=0}^{L-1} x[n-k]$ 의 L-포인트 이동 합 시스템(디리클레 형)의 Magnitude를 구하시오.

L = 11, fs = 1000

```
#include<iostream>
#include<fstream>
#include"complex.h"
```

```
void Dirichlet(int L, int fs, ofstream& out_k, ofstream& out_mag)
{
    complex *H = new complex[fs];
    complex upper, bottom;
    double lim = 11.;

    for (int k = 0; k < fs; k++) {
        bottom = complex(sin(2.*PI*k / (double)(2. * fs)), 0.);
        if (bottom.mag()==0.0) {
            H[k] = complex(cos(-2.*PI*k*((L - 1)/2.) / (double)fs)
                , sin(-2. * PI*k*((L - 1) / 2.) / (double)fs))*lim;
        }
        else {
            upper = complex(sin(2.* PI*k*L / (double)(2.*fs)), 0.0);
            H[k] = upper / bottom * complex(cos((double)(-2.*PI*k*((L - 1) / 2.)) / (double)fs)
                , sin(-2. * PI*k*((L - 1) / 2.) / (double)fs));
        }

    }

    for (int k = 0; k < fs; k++) {
        out_k << k << endl;
        out_mag << H[k].mag() << endl;

    }
}
```

$$D_L(e^{j\hat{w}}) = \frac{\boxed{\sin\left(\frac{\hat{w}L}{2}\right)}}{\boxed{\sin\left(\frac{\hat{w}}{2}\right)}} e^{-j\hat{w}\frac{(L-1)}{2}}$$

예제) y[n] $=\sum_{k=0}^{L-1} x[n-k]$ 의 L-포인트 이동 합 시스템(디리클레 형)의 Magnitude를 구하시오.

L = 11, fs = 1000

```
#include<iostream>
#include<fstream>
#include"complex.h"
```

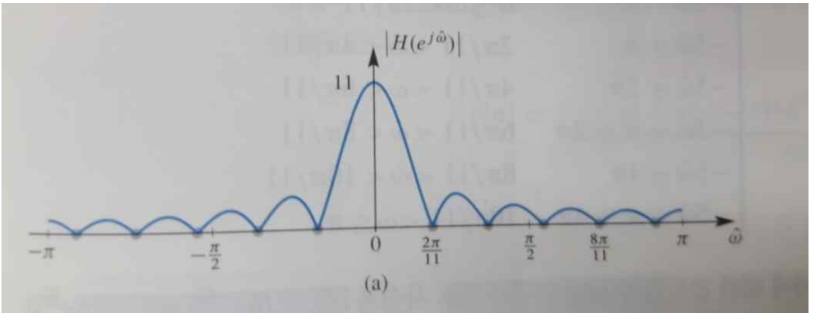
```
void Dirichlet(int L, int fs, ofstream& out_k, ofstream& out_mag)
{
    complex *H = new complex[fs];
    complex upper, bottom;
    double lim = 11.;

    for (int k = 0; k < fs; k++) {
        bottom = complex(sin(2.*PI*k / (double)(2. * fs)), 0.);
        if (bottom.mag()==0.0) {
            H[k] = complex(cos(-2.*PI*k*((L - 1)/2.) / (double)fs)
                , sin(-2. * PI*k*((L - 1) / 2.) / (double)fs))*lim;
        }
        else {
            upper = complex(sin(2.* PI*k*L / (double)(2.*fs)), 0.0);
            H[k] = upper / bottom * complex(cos((double)(-2.*PI*k*((L - 1) / 2.)) / (double)fs)
                , sin(-2. * PI*k*((L - 1) / 2.) / (double)fs));
        }

    }

    for (int k = 0; k < fs; k++) {
        out_k << k << endl;
        out_mag << H[k].mag() << endl;

    }
}
```

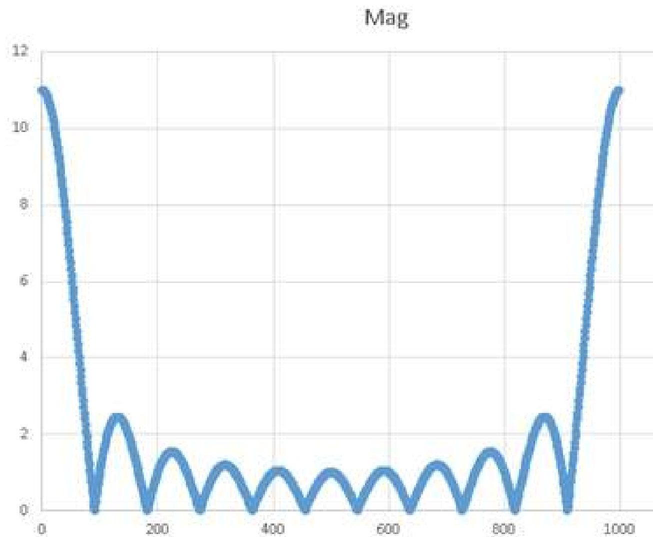
$$D_L(e^{j\hat{w}}) = \frac{\sin\left(\frac{\hat{w}L}{2}\right)}{\sin(\frac{\hat{w}}{2})} e^{-j\hat{w}\frac{(L-1)}{2}}$$

예제) y[n] =$\sum_{k=0}^{L-1} x[n-k]$ 의 L-포인트 이동 합 시스템(디리클레 형)의 Magnitude를 구하시오.
L = 11, fs = 1000

$$D_L(e^{j\hat{w}}) = \frac{\sin\left(\frac{\hat{w}L}{2}\right)}{\sin(\frac{\hat{w}}{2})} e^{-j\hat{w}\frac{(L-1)}{2}}$$

예제 결과 비교 (DSP First vs C++ code)





연습문제

Feedback 성분

$$y[n] = x[n] - 0.64x[n-2] + \boxed{0.4y[n-1] - 0.68y[n-2]}$$

$$y[n]-0.4y[n-1]+0.68y[n-2] = x[n] - 0.64x[n-2]$$

$$\updownarrow$$

$$Y(e^{j\Omega})(1-0.4e^{-j\Omega}+0.68e^{-j2\Omega}) = X(e^{j\Omega})(1-0.64e^{-j2\Omega})$$

$$H(e^{j\Omega}) = \frac{Y(e^{j\Omega})}{X(e^{j\Omega})} = \frac{(1-0.64e^{-j2\Omega})}{(1-0.4e^{-j\Omega}+0.68e^{-j2\Omega})}$$

Ex1 - code

```
#include<iostream>
#include<fstream>
#include"complex.h"
using namespace std;

#define PI 3.141592
#define N 64

void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag);

void main()
{
    double a1, b1, c1, a2, b2, c2;

    ofstream out_k, out_mag;
    out_k.open("k.txt");
    out_mag.open("mag.txt");

    a1 = 1;
    b1 = 0;
    c1 = -0.64;

    a2 = 1;
    b2 = -0.4;
    c2 = 0.68;

    Z_function(a1, b1, c1, a2, b2, c2, out_k, out_mag);


    system("pause");
    return;

}
```

File 저장

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{(1-0.64Z^{-2})}{(1-0.4Z^{-1}+0.68Z^{-2})}$$

Ex1 - code

```
#include<iostream>
#include<fstream>
#include"complex.h"
using namespace std;

#define PI 3.141592
#define N 64

void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag);

void main()
{
    double a1, b1, c1, a2, b2, c2;

    ofstream out_k, out_mag;
    out_k.open("k.txt");
    out_mag.open("mag.txt");

    a1 = 1;
    b1 = 0;
    c1 = -0.64;

    a2 = 1;
    b2 = -0.4;
    c2 = 0.68;

    Z_function(a1, b1, c1, a2, b2, c2, out_k, out_mag);


    system("pause");
    return;

}
```

계수 변수

$$H(Z) = \frac{a_1 + b_1 Z^{-1} + c_1 Z^{-2}}{a_2 + b_2 Z^{-1} + c_2 Z^{-2}}$$

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{(1 - 0.64Z^{-2})}{(1 - 0.4Z^{-1} + 0.68Z^{-2})}$$

Ex1 - code

```
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

    if (Zeros > 0) { // 서로 다른 두 실근
        zeros1 = complex((-x1 - sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
        zeros2 = complex((-x1 + sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
    }
    else if (Zeros == 0) { // 중근
        zeros1 = zeros2 = complex((-x1) / (2 * x0), 0);
    }
    else {//허근
        zeros1 = complex((-x1) / (2 * x0), sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
        zeros2 = complex((-x1) / (2 * x0), -sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2))/(2 * y0), 0);
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)) / (2 * y0), 0);
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k] - poles1)*(Z[k] - poles2));
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{\boxed{(1-0.64Z^{-2})}}{\boxed{(1-0.4Z^{-1}+0.68Z^{-2})}}$$

Ex1 - code

```
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

                                                       , 0);
                                                       , 0);



                                                       x2)) / (2 * x0));
                                                       *x2)) / (2 * x0));
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2))/(2 * y0), 0);
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)) / (2 * y0), 0);
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k] - poles1)*(Z[k] - poles2));
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

이차방정식 $ax^2 + bx + c = 0$

$D = b^2 - 4ac > 0$: 서로 다른 두 근

$D = b^2 - 4ac = 0$: 중근

$D = b^2 - 4ac < 0$: 서로 다른 두 허근

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{\boxed{(1 - 0.64Z^{-2})}}{\boxed{(1 - 0.4Z^{-1} + 0.68Z^{-2})}}$$

Ex1 - code

```
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

    if (Zeros > 0) { // 서로 다른 두 실근
        zeros1 = complex((-x1 - sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
        zeros2 = complex((-x1 + sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
    }
    else if (Zeros == 0) { // 중근
        zeros1 = zeros2 = complex((-x1) / (2 * x0), 0);
    }
    else {//허근
        zeros1 = complex((-x1) / (2 * x0), sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
        zeros2 = complex((-x1) / (2 * x0), -sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2))/(2 * y0), 0);
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)) / (2 * y0), 0);
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k] - poles1)*(Z[k] - poles2));
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{\boxed{(1-0.64Z^{-2})}}{(1-0.4Z^{-1}+0.68Z^{-2})}$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

근의 공식

Ex1 - code

```
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

    if (Zeros > 0) { // 서로 다른 두 실근
        zeros1 = complex((-x1 - sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
        zeros2 = complex((-x1 + sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
    }
    else if (Zeros == 0) { // 중근
        zeros1 = zeros2 = complex((-x1) / (2 * x0), 0);
    }
    else {//허근
        zeros1 = complex((-x1) / (2 * x0), sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
        zeros2 = complex((-x1) / (2 * x0), -sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2))/(2 * y0), 0);
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)) / (2 * y0), 0);
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k] - poles1)*(Z[k] - poles2));
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{\boxed{(1-0.64Z^{-2})}}{(1-0.4Z^{-1}+0.68Z^{-2})}$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

근의 공식

Ex1 - code

```
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

    if (Zeros > 0) { // 서로 다른 두 실근
        zeros1 = complex((-x1 - sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
        zeros2 = complex((-x1 + sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
    }
    else if (Zeros == 0) { // 중근
        zeros1 = zeros2 = complex((-x1) / (2 * x0), 0);
    }
    else {//허근
        zeros1 = complex((-x1) / (2 * x0), sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
        zeros2 = complex((-x1) / (2 * x0), -sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2))/(2 * y0), 0);
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)) / (2 * y0), 0);
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k] - poles1)*(Z[k] - poles2));
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{\boxed{(1-0.64Z^{-2})}}{(1-0.4Z^{-1}+0.68Z^{-2})}$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

근의 공식

Ex1 - code

```
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

    if (Zeros > 0) { // 서로 다른 두 실근
        zeros1 = complex((-x1 - sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
        zeros2 = complex((-x1 + sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
    }
    else if (Zeros == 0) { // 중근
        zeros1 = zeros2 = complex((-x1) / (2 * x0), 0);
    }
    else {//허근
        zeros1 = complex((-x1) / (2 * x0), sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
        zeros2 = complex((-x1) / (2 * x0), -sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2))/(2 * y0), 0);
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)) / (2 * y0), 0);
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k] - poles1)*(Z[k] - poles2));
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{\boxed{(1 - 0.64Z^{-2})}}{(1 - 0.4Z^{-1} + 0.68Z^{-2})}$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

근의 공식

Ex1 - code

```cpp
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

    if (Zeros > 0) { // 서로 다른 두 실근
        zeros1 = complex((-x1 - sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
        zeros2 = complex((-x1 + sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
    }
    else if (Zeros == 0) { // 중근
        zeros1 = zeros2 = complex((-x1) / (2 * x0), 0);
    }
    else {//허근
        zeros1 = complex((-x1) / (2 * x0), sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
        zeros2 = complex((-x1) / (2 * x0), -sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2))/(2 * y0), 0);
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)) / (2 * y0), 0);
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k] - poles1)*(Z[k] - poles2));
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{(1-0.64Z^{-2})}{\boxed{(1-0.4Z^{-1}+0.68Z^{-2})}}$$

$$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

근의 공식

Ex1 - code

```
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

    if (Zeros > 0) { // 서로 다른 두 실근
        zeros1 = complex((-x1 - sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
        zeros2 = complex((-x1 + sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
    }
    else if (Zeros == 0) { // 중근
        zeros1 = zeros2 = complex((-x1) / (2 * x0), 0);
    }
    else {//허근
        zeros1 = complex((-x1) / (2 * x0), sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
        zeros2 = complex((-x1) / (2 * x0), -sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2))/(2 * y0), 0);
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)) / (2 * y0), 0);
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k] - poles1)*(Z[k] - poles2));
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

$$H(Z)=\frac{Y(Z)}{X(Z)}=\frac{(1-0.64Z^{-2})}{\boxed{(1-0.4Z^{-1}+0.68Z^{-2})}}$$

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

근의 공식

Ex1 - code

```
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

    if (Zeros > 0) { // 서로 다른 두 실근
        zeros1 = complex((-x1 - sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
        zeros2 = complex((-x1 + sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
    }
    else if (Zeros == 0) { // 중근
        zeros1 = zeros2 = complex((-x1) / (2 * x0), 0);
    }
    else {//허근
        zeros1 = complex((-x1) / (2 * x0), sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
        zeros2 = complex((-x1) / (2 * x0), -sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2))/(2 * y0), 0);
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)) / (2 * y0), 0);
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k] - poles1)*(Z[k] - poles2));
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{(1-0.64Z^{-2})}{\boxed{(1-0.4Z^{-1}+0.68Z^{-2})}}$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

근의 공식

Ex1 - code

```
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

    if (Zeros > 0) { // 서로 다른 두 실근
        zeros1 = complex((-x1 - sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
        zeros2 = complex((-x1 + sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
    }
    else if (Zeros == 0) { // 중근
        zeros1 = zeros2 = complex((-x1) / (2 * x0), 0);
    }
    else {//허근
        zeros1 = complex((-x1) / (2 * x0), sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
        zeros2 = complex((-x1) / (2 * x0), -sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2))/(2 * y0), 0);
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)) / (2 * y0), 0);
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k] - poles1)*(Z[k] - poles2));
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{(1-0.64Z^{-2})}{\boxed{(1-0.4Z^{-1}+0.68Z^{-2})}}$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

근의 공식

Ex1 - code

```
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

    if (Zeros > 0) { // 서로 다른 두 실근
        zeros1 = complex((-x1 - sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
        zeros2 = complex((-x1 + sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
    }
    else if (Zeros == 0) { // 중근
        zeros1 = zeros2 = complex((-x1) / (2 * x0), 0);
    }
    else {//허근
        zeros1 = complex((-x1) / (2 * x0), sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
        zeros2 = complex((-x1) / (2 * x0), -sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2))/(2 * y0), 0);
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)) / (2 * y0), 0);
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k] - poles1)*(Z[k] - poles2));
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{(1-0.64Z^{-2})}{(1-0.4Z^{-1}+0.68Z^{-2})}$$

```
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

    if (Zeros > 0) { // 서로 다른 두 실근
        zeros1 = complex((-x1 - sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
        zeros2 = complex((-x1 + sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
    }
    else if (Zeros == 0) { // 중근
        zeros1 = zeros2 = complex((-x1) / (2 * x0), 0);
    }
    else {//허근
        zeros1 = complex((-x1) / (2 * x0), sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
        zeros2 = complex((-x1) / (2 * x0), -sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2))/(2 * y0), 0);
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)) / (2 * y0), 0);
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k] - poles1)*(Z[k] - poles2));
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{(1-0.64\boxed{Z}^{-2})}{(1-0.4\boxed{Z}^{-1}+0.68\boxed{Z}^{-2})}$$

Ex1 - code

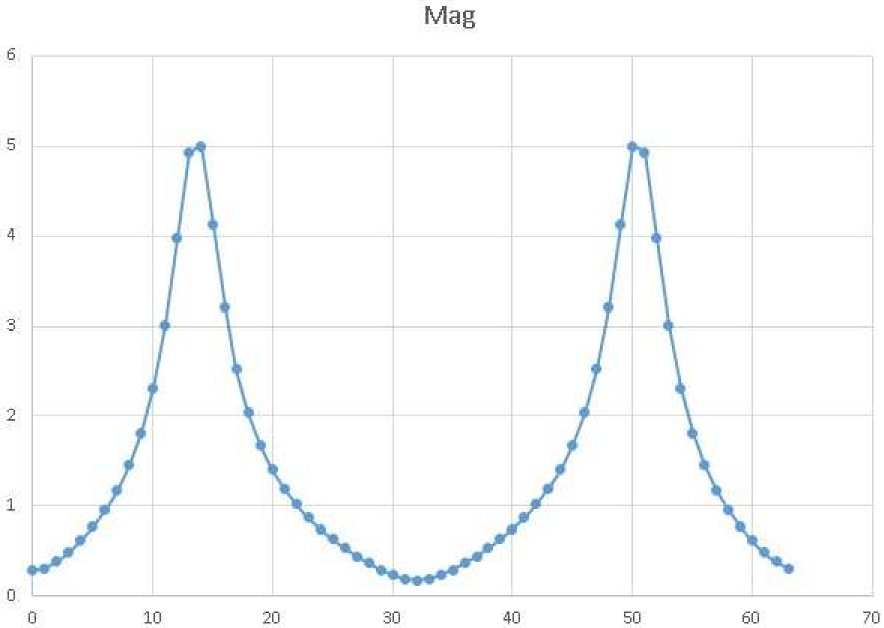
```
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

    if (Zeros > 0) { // 서로 다른 두 실근
        zeros1 = complex((-x1 - sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
        zeros2 = complex((-x1 + sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
    }
    else if (Zeros == 0) { // 중근
        zeros1 = zeros2 = complex((-x1) / (2 * x0), 0);
    }
    else {//허근
        zeros1 = complex((-x1) / (2 * x0), sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
        zeros2 = complex((-x1) / (2 * x0), -sqrt(abs(x1*x1 - 4 * x0*x2)) / (2 * x0));
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2))/(2 * y0), 0);
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)) / (2 * y0), 0);
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1*y1 - 4 * y0*y2)) / (2 * y0));
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k] - poles1)*(Z[k] - poles2));
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

$$H(Z) = \frac{Y(Z)}{X(Z)} = \boxed{\frac{(1-0.64Z^{-2})}{(1-0.4Z^{-1}+0.68Z^{-2})}}$$

Ex1 - code

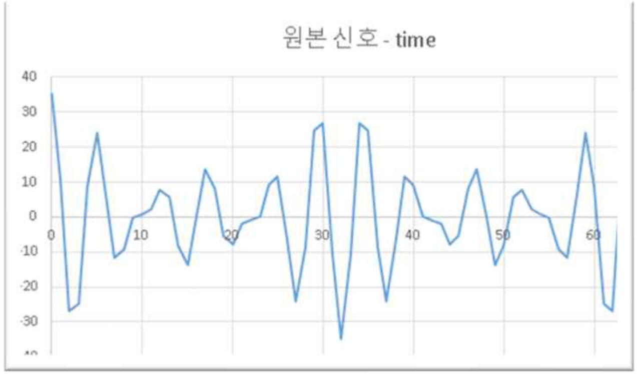
```
void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag)
{
    complex zeros1, zeros2, poles1, poles2;
    double Zeros, Poles;

    Zeros = x1 * x1 - 4 * x0*x2;
    // 판별식

    if (Zeros > 0) { // 서로 다른 두 실근
        zeros1 = complex((-x1 - sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
        zeros2 = complex((-x1 + sqrt(x1*x1 - 4 * x0*x2)) / (2 * x0), 0);
    }
    else if (Zeros == 0) { // 중근
        zeros1 = zeros2 = complex((-x1) / (2 * x0), 0);
    }
    else {//허근
        zeros1 = complex((-x1) / (2 * x0), sqrt(abs(x1*
        zeros2 = complex((-x1) / (2 * x0), -sqrt(abs(x1
    }

    Poles = y1 * y1 - 4 * y0*y2;
    //판별식

    if (Poles > 0) {
        poles1 = complex((-y1 - sqrt(y1*y1 - 4 * y0*y2)
        poles2 = complex((-y1 + sqrt(y1*y1 - 4 * y0*y2)
    }
    else if (Poles == 0) {
        poles1 = poles2 = complex((-y1) / (2 * y0), 0);
    }
    else {
        poles1 = complex((-y1) / (2 * y0), sqrt(abs(y1*
        poles2 = complex((-y1) / (2 * y0), -sqrt(abs(y1
    }

    complex* H = new complex[N];
    complex* Z = new complex[N];

    for (int k = 0; k < N; k++) {
        Z[k] = complex(2 * PI*k / (double)N);
        H[k] = (Z[k] - zeros1)*(Z[k] - zeros2) / ((Z[k]
        out_k << k << endl;
        out_mag << H[k].mag() << endl;
    }
}
```

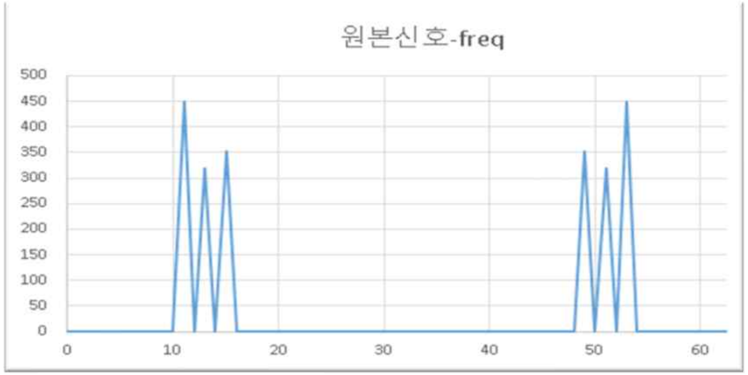
$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{(1-0.64Z^{-2})}{(1-0.4Z^{-1}+0.68Z^{-2})}$$



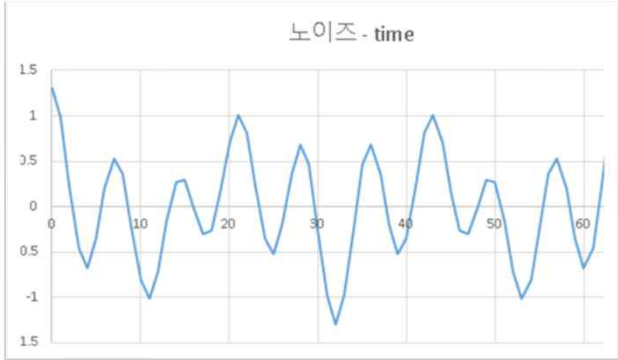
Ex1 - code

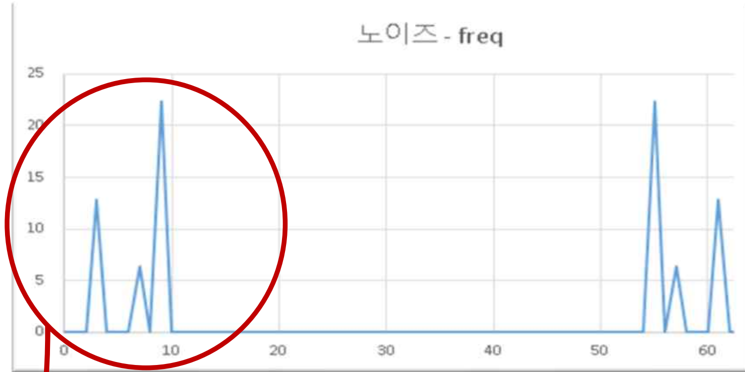
원본 신호



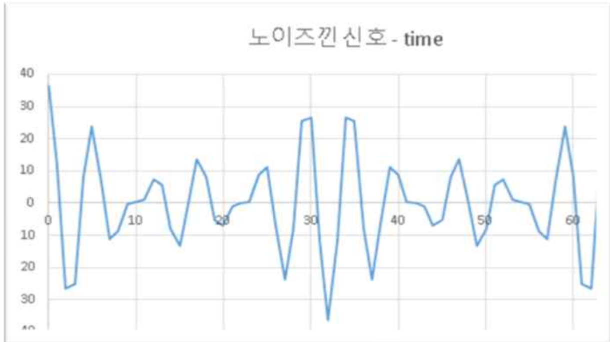


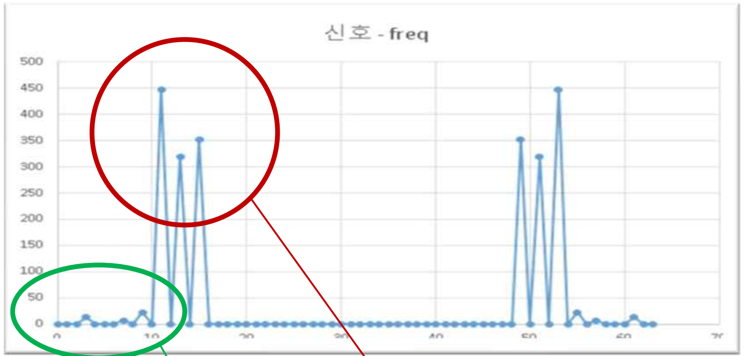
노이즈 신호





원본 + 노이즈 신호

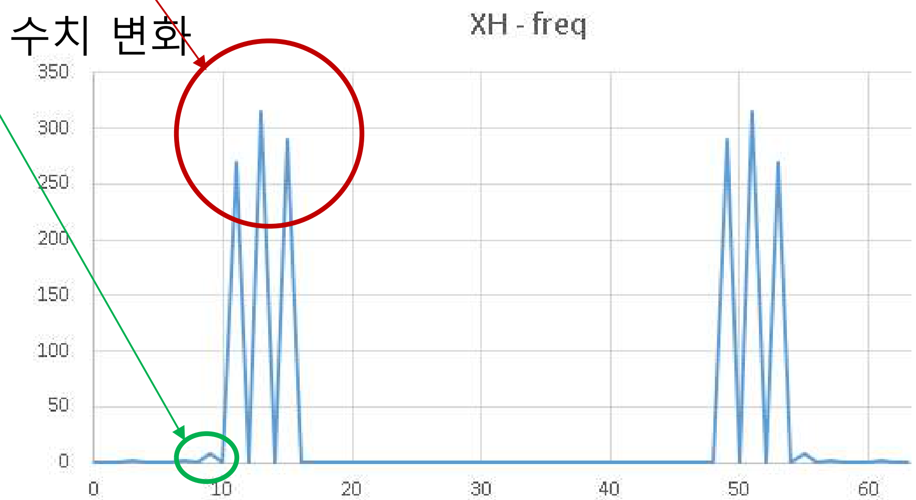


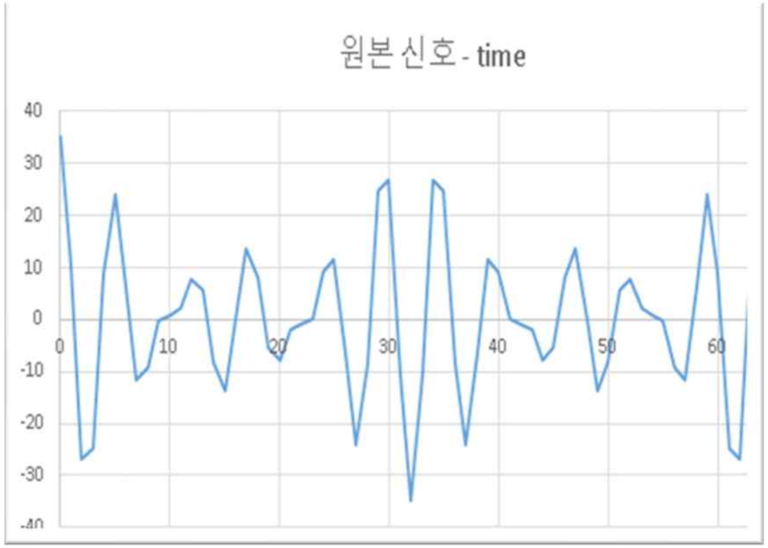


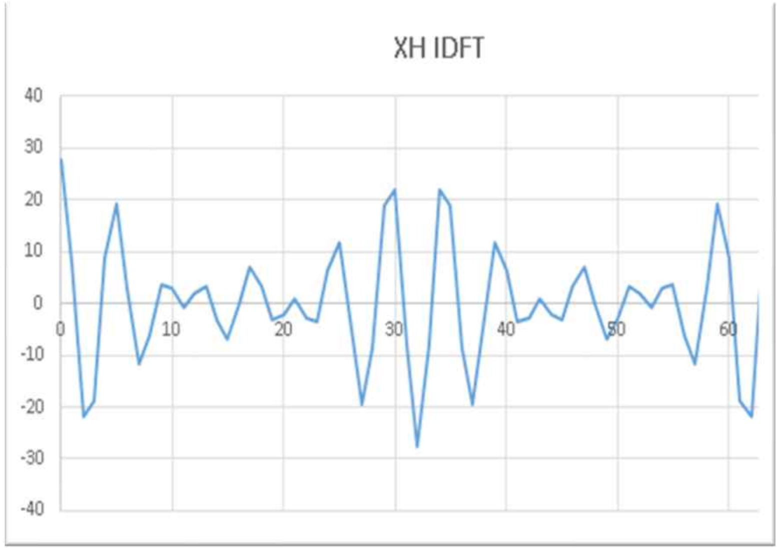
X

주파수 영역의 신호 X

주파수 영역 (정규화된) 필터







- IIR Filter(Infinite Impulse Response Filter)
  - IIR Filter는 디지털 필터의 한 종류로 입력신호의 값과 출력 신호 값이 재귀적으로(recursive)적용 되어 filtering 수행.
  - 특성 함수 Impulse response는 무한한 길이를 가지게 된다.
  - IIR Filter의 식의 형태에서 보면 feedback 성분을 가진다.
  - FIR Filter에 비해 위상 변이가 크기 때문에 입력 파형과 출력 파형이 유사한 파형을 갖지 않는다.

필터 실습

```cpp
#include<iostream>
#include<fstream>
#include"complex.h"
using namespace std;

#define PI 3.141592
#define N 64

void Z_function(double x0, double x1, double x2, double y0, double y1, double y2,
    ofstream& out_k, ofstream& out_mag);

void main()
{
    double a1, b1, c1, a2, b2, c2;

    ofstream out_k, out_mag;
    out_k.open("k.txt");
    out_mag.open("mag.txt");

    a1 = 1;
    b1 = 0;
    c1 = -0.64;

    a2 = 1;
    b2 = -0.4;
    c2 = 0.68;

    Z_function(a1, b1, c1, a2, b2, c2, out_k, out_mag);


    system("pause");
    return;

}
```

실습 코드를 추가할 부분

Assignment (1)

$$D_L(e^{j\hat{w}}) = \frac{\sin\left(\frac{\hat{w}L}{2}\right)}{\sin(\frac{\hat{w}}{2})} e^{-j\hat{w}\frac{(L-1)}{2}}$$

y[n] $=\sum_{k=0}^{L-1} x[n-k]$ 의 L-포인트 이동 합 시스템(디리클레 형)을 이용하여

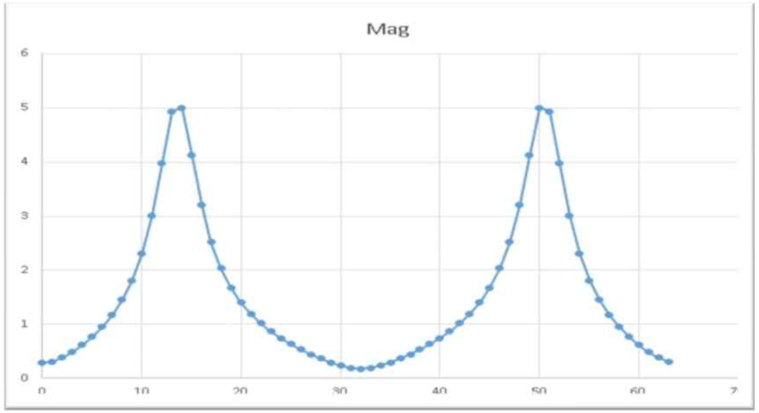
제공된 음악파일을 Filtering하시오. 차단 주파수: 3000Hz

제출 파일
- .cpp 파일, 결과 보고서(입력 음악의 주파수 영역 모습과 결과 음악의 주파수 영역 사진 첨부)
- Filtering한 음악 wav 파일

Assignment (2)

Poles , Zeros 점을 설정하여, 아래와 같은 Band pass Filter를 구현 하세요.

통과 주파수 대역은 1.5k ~ 2.5 k Hz가 되도록 설정한다.



제공된 음악 파일을 필터링한 후 결과 음악을 생성하라.

과제 제출 파일
- .cpp 파일, 보고서 파일 (입력 영상과 필터링된 음악의 주파수 영역 모습 사진을 반드시 포함하라)
- 결과 음악 파일

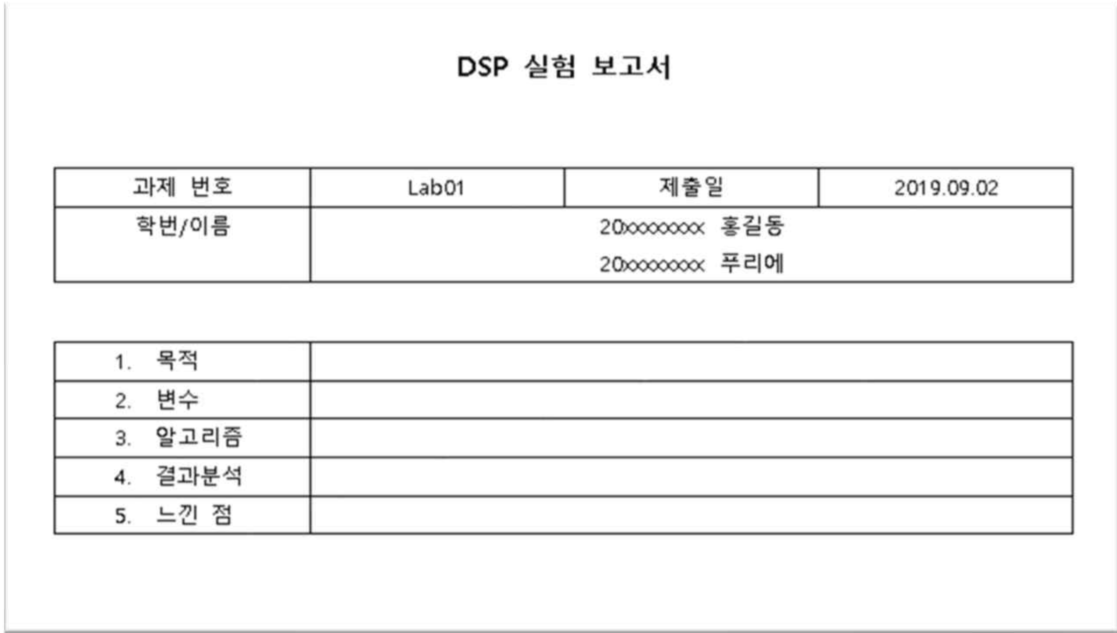
Assignment Rule

"KLAS에 제출할 때 다음 사항을 꼭 지켜주세요"

1. 파일명 : "Lab00_요일_대표자이름.zip"

Ex) Lab01_목_홍길동.zip **(압축 툴은 자유롭게 사용)**

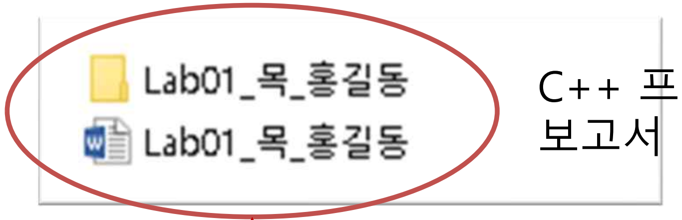
2. 제출 파일 **(보고서와 프로그램을 압축해서 제출)**
   - 보고서 파일 (hwp, word): 이름, 학번, 목적, 변수, 알고리즘(순서), 결과 분석, 느낀 점
   - 프로그램



DSP 실험 보고서

과제 번호	Lab01	제출일	2019.09.02
학번/이름	20xxxxxxxx 홍길동 20xxxxxxxx 푸리에		

1. 목적	
2. 변수	
3. 알고리즘	
4. 결과분석	
5. 느낀 점	



C++ 프로그램 파일
보고서

압축하여 제출