DSP REPORT

과목명	디지털신호처리
담당교수	유 훈 교수님
학과	융합전자공학과
학년	2학년
학번	201910906
이름	이학민
제출일	2020.12.19

상명대학교
SANGMYUNG UNIVERSITY

1. Lab01_ADC

1.1 Code

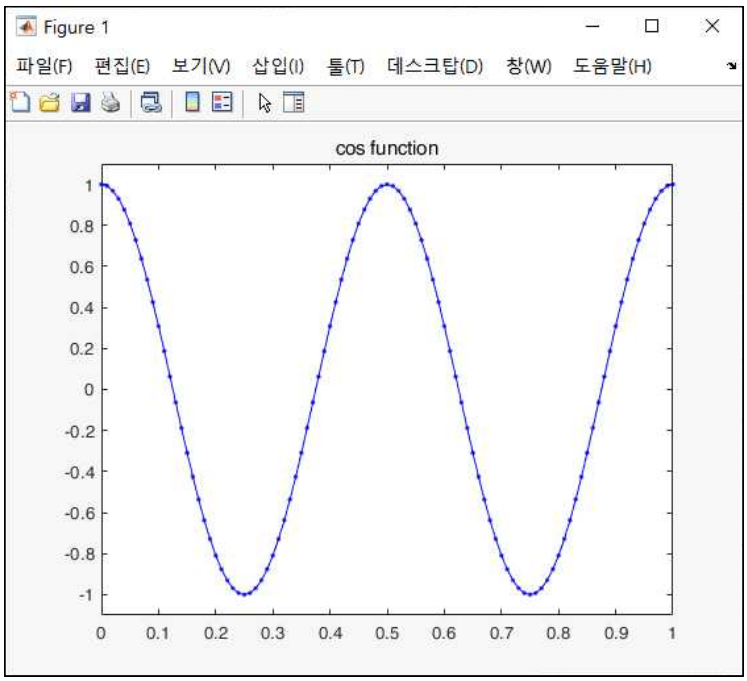
```
% Lab01 Analog-to-digital converting

% Sampling
fs = 100; % 샘플링 주파수
fi = 2; % 아날로그 신호의 최대 주파수 설정
delta = 1/fs; % 샘플링 주기

x = 0:delta:1; % x 영역의 범위
fx = cos(2*pi*fi*x); % 최소 = -1, 최대 = +1

figure(1);
plot(x, fx, 'b.-');
axis([0,1,-1.1, 1.1]);
title('cos function');

% Quantization
qbit = 8; % 양자화 비트수
maxval = 2^(qbit-1)-1; % -31 ~ 31
fxq = round(maxval*fx);

figure(2);
stem(x,fxq,'.');

% inverse Quantization
fxr = fxq/maxval;
fxe = fx- fxr;
errpow = mean(fxe.^2);

figure(3);
plot(x, fx, 'b-', x, fxr, 'r-');
axis([0,1,-1.1, 1.1]);
title(['cos function, frequency=', num2str(fi)]);

figure(4);
stem(x,fxe,'.');
title(['Error function, Power=', num2str(errpow*1000)]);
```

1.2 Figure

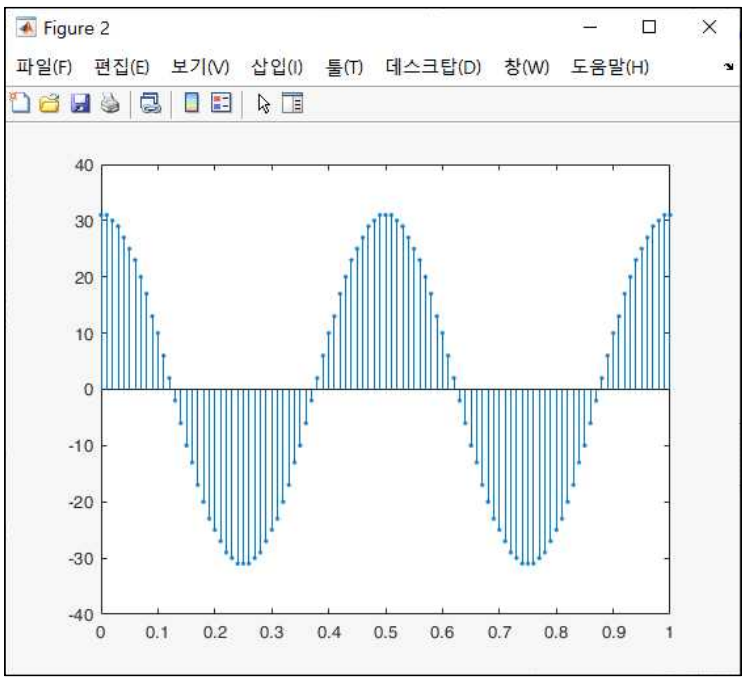
아날로그 신호 $f(x) = \cos(2\pi f x)$의 ADC converting 과정에 대한 figure이다. 신호와 ADC converting을 진행할 때 설정한 값은 다음과 같다.

샘플링 주파수	100
최대 주파수	2
샘플링 주기	1/100
양자화 비트 수	6

<조건 1>

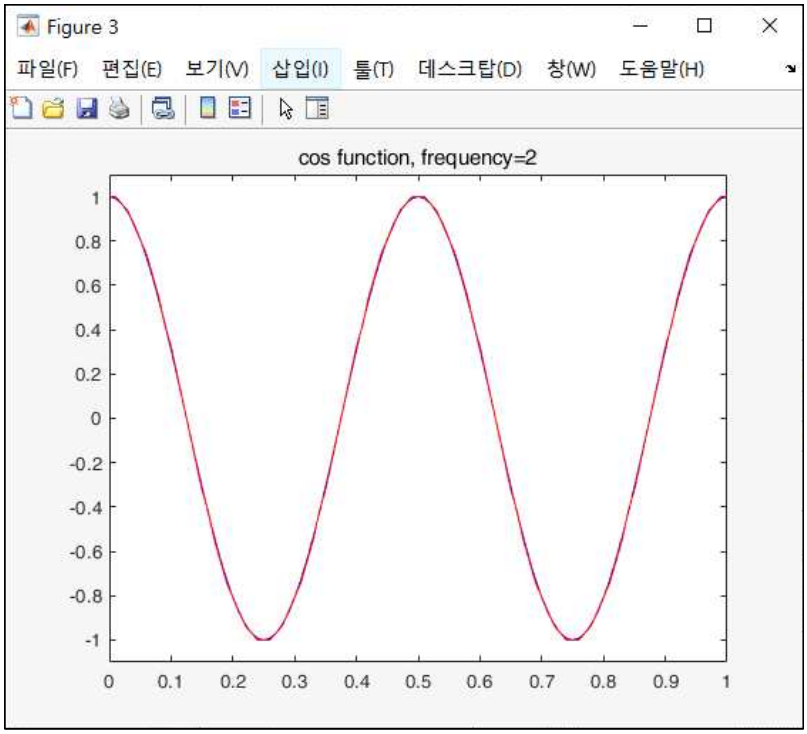


[그림 1] 샘플링



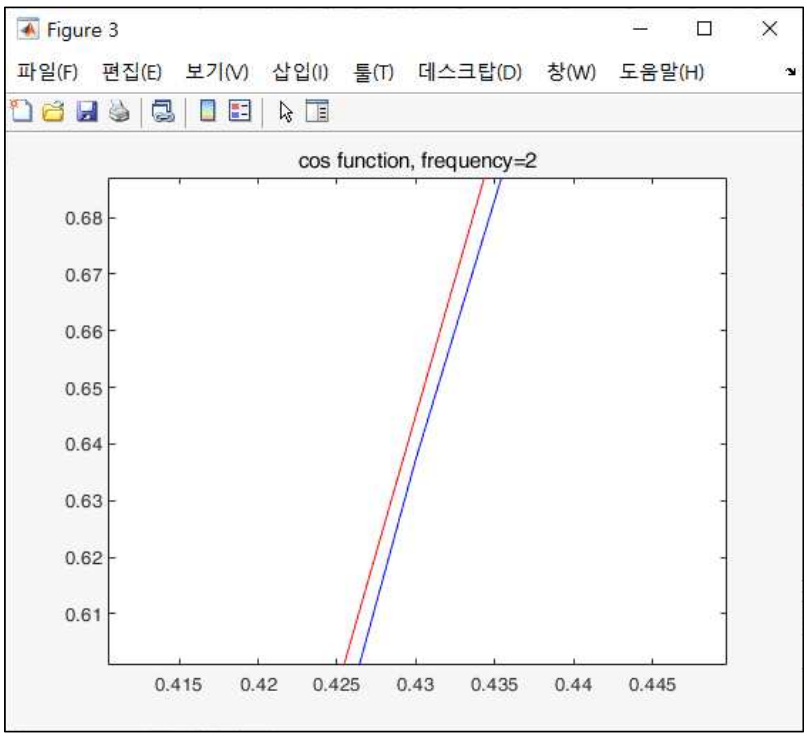
[그림 2] 양자화

ADC 변환 처리된 함수를 얻기 위해서 역 양자화를 진행한다.

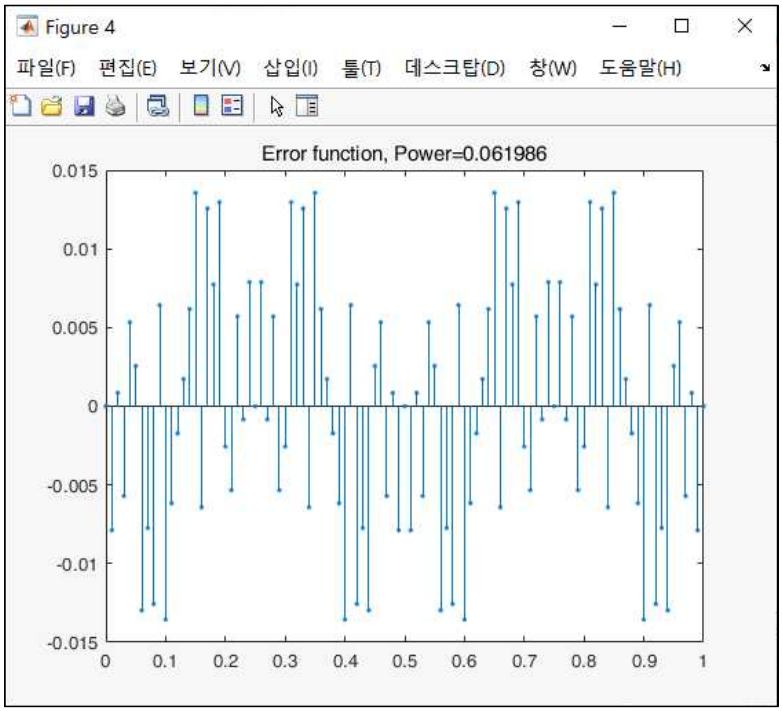


[그림 3] 역 양자화

샘플링 후 양자화를 거친 신호를 역 양자화하면 오차가 발생한다. 그래프를 확대해 보면 ADC 변환을 거친 신호가 초기의 아날로그 신호인 $f(x) = \cos(2\pi f x)$와 오차가 생김을 확연히 알 수 있다.

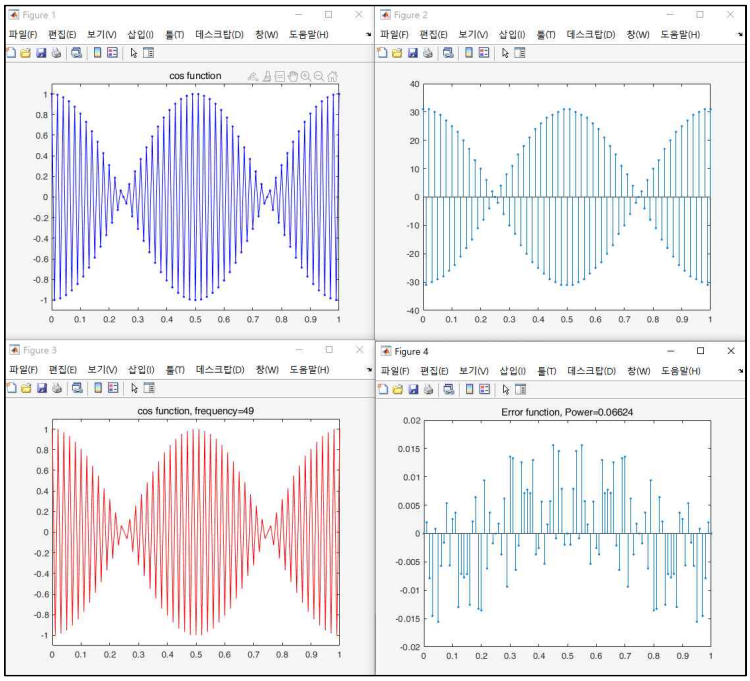


[그림 4] 오차



[그림 5] 오차 함수

오차값(fxe)은 초기 신호의 값(fx)에서 변환된 신호의 값(fxr)을 빼준 값이고 오차의 power를 통해 신호를 분석한다. 오차의 power는 오차^{2}의 평균과 같다.

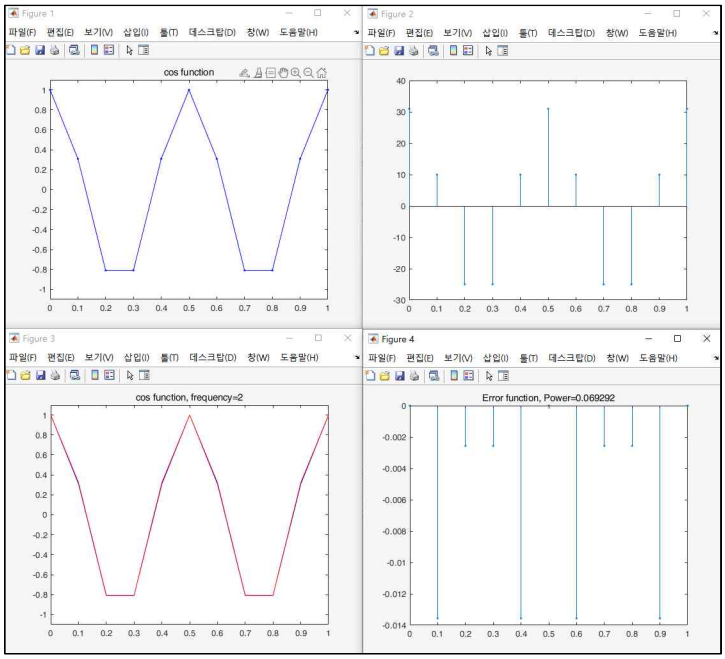


[그림 6]

샘플링 주파수	100
최대 주파수	49
샘플링 주기	1/100
양자화 비트 수	6

<조건 2>

[그림 6]은 다음 조건에서 코드를 수행하였을 때 모습이다.

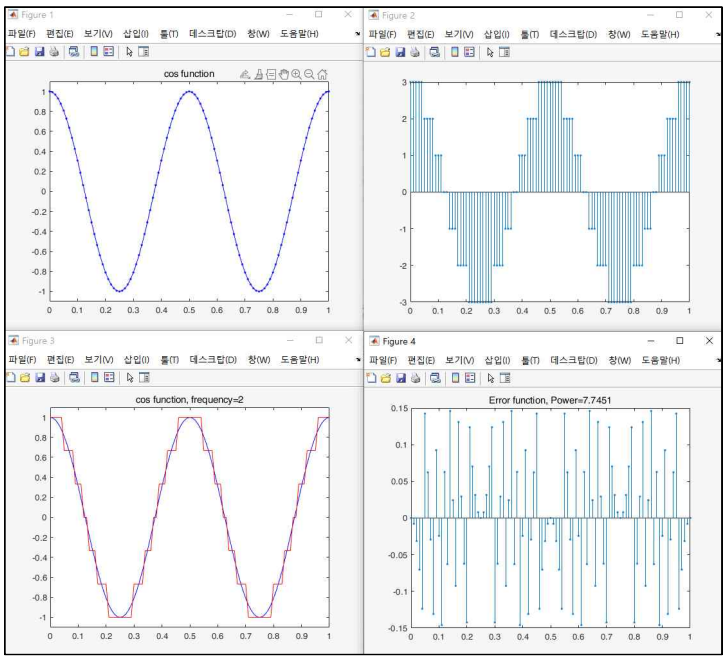


[그림 7]

샘플링 주파수	10
최대 주파수	2
샘플링 주기	1/10
양자화 비트 수	6

<조건 3>

[그림 7]은 다음 조건에서 코드를 수행하였을 때 모습이다.



[그림 8]

샘플링 주파수	100
최대 주파수	2
샘플링 주기	1/100
양자화 비트 수	3

<조건 4>

[그림 8]은 다음 조건에서 코드를 수행하였을 때 모습이다.

[그림 6], [그림 7]에서 알 수 있듯이 샘플링 주파수와 최대 주파수의 차이가 너무 적게 나면 원하는 결과를 도출할 수 없다. 특히 [그림 8]을 보면 ADC는 샘플링 주파수보다 양자화 비트 수에 민감하게 반응하는 것을 알 수 있다. 실습의 <조건 1>과 비교했을 때 양자화 비트 수만 3으로 줄어들은 <조건 4>의 경우 오차의 power값이 <조건 1>의 값보다 약 125배 큼을 알 수 있다.

이론에서 배운 샘플링 주파수 설정에 따르면 최대 주파수의 2배만 되어도 되지만 실제로 실습을 통해 프로그램을 만들어보니 원하고자 하는 값을 얻지 못하였다. 따라서 실습에 따르면 올바른 ADC converting을 위해서는 최소 샘플링 주파수가 최대 주파수의 약 20배 정도 되어야 하고 그에 걸맞는 충분한 양자화 비트 수를 설정해 주어야함을 알 수 있다.

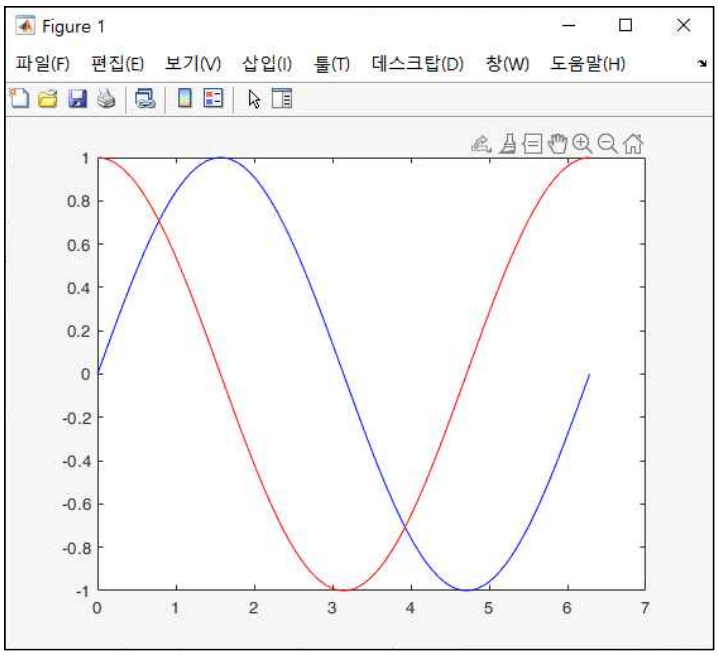
2. Lab02_integral

2.1 Code

```
% 디지털 영역에서의 미적분
clear all;

% Sampling cos sin
fs = 100;
delta = 2*pi/fs;
x = 0:delta:2*pi;

ys = sin(x);
yc = cos(x);

figure(1);
plot(x,ys,'b', x, yc, 'r');

% 이산 신호 미분
dys = diff([0,ys])/(delta); % (f(x+h)-f(x))/h
figure(2);
plot(x,ys, 'b', x, dys, 'r');


% 이산 신호 적분

iyc = cumsum(yc)*delta;
figure(3);
plot(x,yc, 'b', x, iyc, 'r');
```

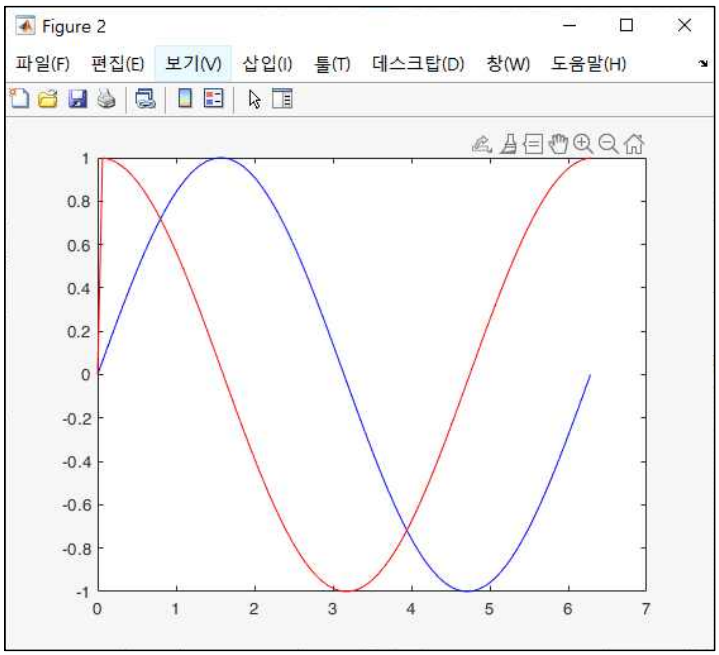
2.2 Figure

$f(x) = \cos(2\pi f x)$, $f = 100$와 $g(x) = \sin(2\pi f x)$, $f = 100$



[그림 9]

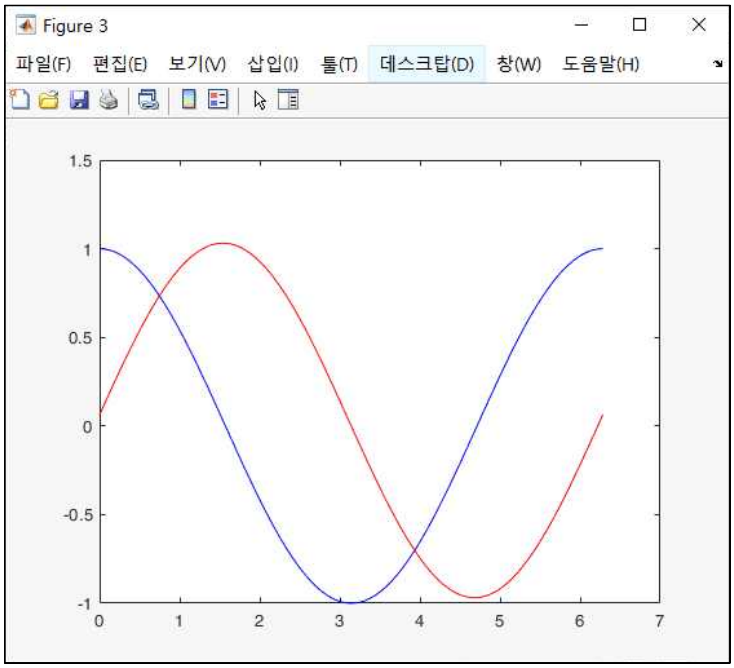
파란색 그래프는 주파수가 100인 sin그래프이고 빨간색 그래프는 주파수가 100인 cos그래프이다.



[그림 10]

미분의 수학적 정의에 따라 sin함수의 미분 그래프를 그리면 다음과 같다.

파란색 그래프가 sin그래프이고 빨간색 그래프가 sin을 미분한 그래프이다. sin의 미분은 cos으로 [그림 9]의 cos그래프와 비슷한 그래프로 그려진다. 초기값이 0으로 설정되어 있어 x가 0인 근처 부근에서 미분 그래프의 오차값이 발생한다.



[그림 11]

미분의 수학적 정의에 따라 cos함수의 적분 그래프를 그리면 다음과 같다.
파란색 그래프가 cos그래프이고 빨간색 그래프가 cos을 미분한 그래프이다. cos의 적분은 sin으로 [그림 9]의 sin그래프와 비슷한 그래프로 그려진다. 초기값이 0이 아닌 값으로 설정되어 있어 x가 0인 근처 부근에서 적분 그래프의 오차값이 발생한다.

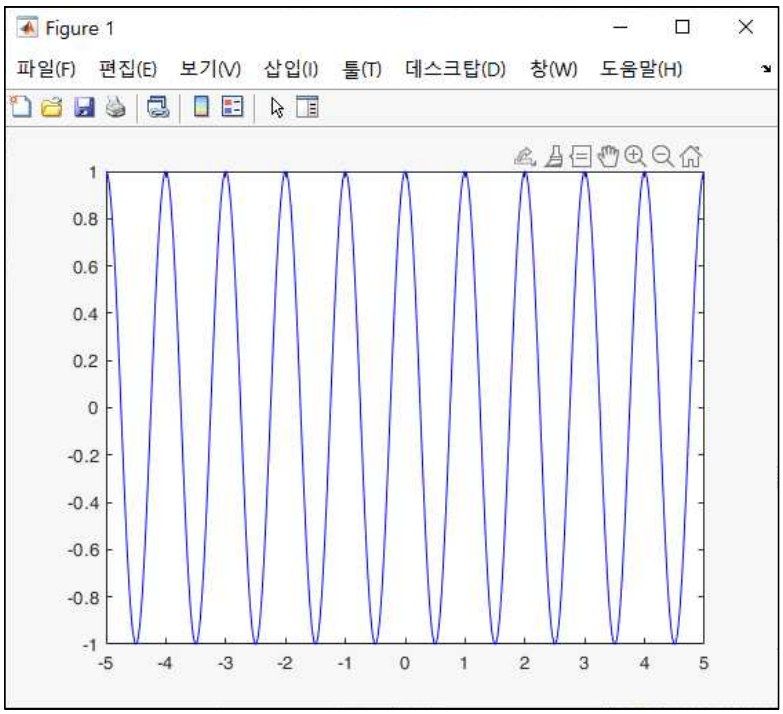
3. Lab03_period

3.1 Code

```
% 주기함수를 생성하는 실습

% sampling
delta = 0.01; % fs = 1/delta
range = 5;
f0 = 1; % 기본주기 = 1

% x 영역
x = -range:delta:range-delta;

fc = (cos(2*pi*x)+1)/2;

% 임펄스꼴의 주기함수
w = 0.99999;
fr = fc >= w;
figure(3);
plot(x,fr,'b');

fc = cos(2*pi*x);
figure(1);
plot(x, fc, 'b');

% 사각펄스 주기함수
fr = sign(fc);
figure(2);
plot(x,fr,'b');

% 삼각파 함수
delta = 0.01;
xu = -0.5:delta:0.5-delta;
fxu = 1-2*abs(xu); % 1-2|x|
figure(4);
plot(xu, fxu, 'b-');

% 삼각파 주기함수
ftri = [];
for n=1:10
    ftri = [ftri, fxu];
```

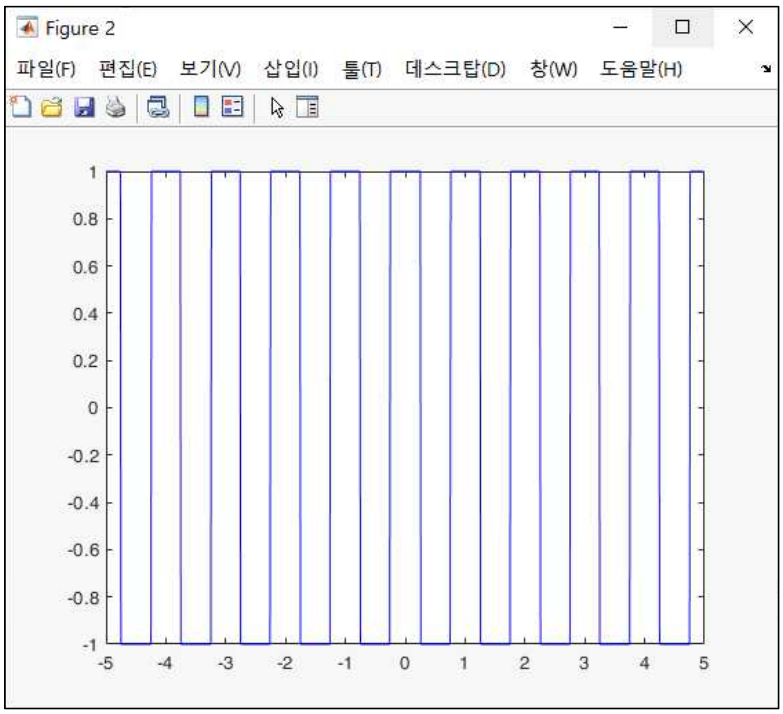
```
end
figure(5);
stem(ftri, '.');

% sigHatTrain을 이용한 삼각파 주기함수
[ftri,x] = sigHatTrain(5, 0.01);
figure(6);
stem(x,ftri, '.');
```

3.2 Figure



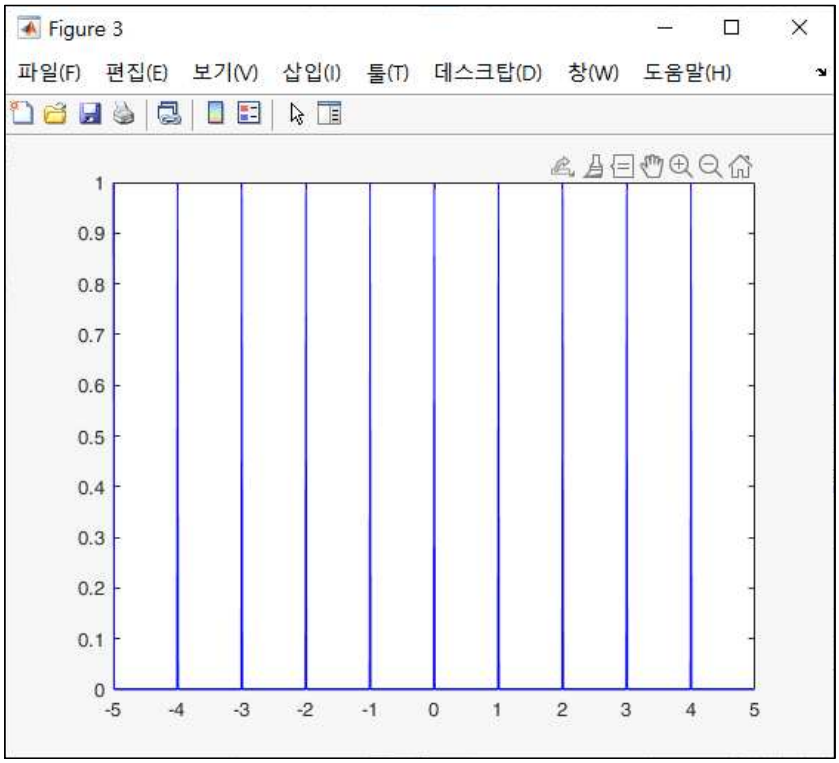
[그림 12]

[그림 12]는 cos 주기함수이다. cos 주기함수의 부호변화를 응용하여 사각펄스 주기함수를 구현할 수 있다. 이 때 x값의 범위는 -5 ~ 5 사이로 지정하였다.



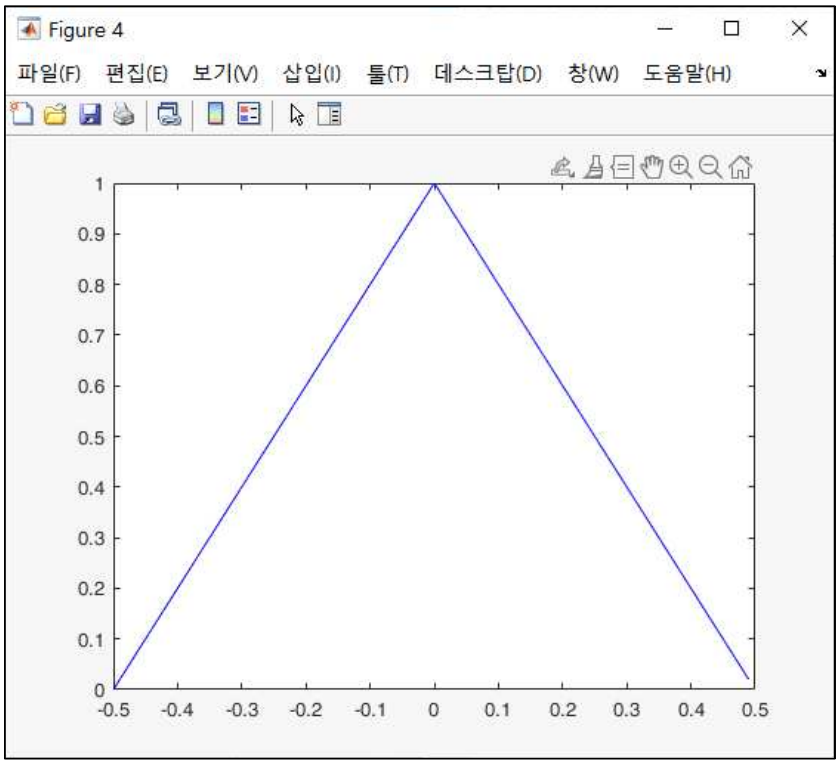
[그림 13]

[그림 13]은 사각펄스 주기함수이다. cos 주기함수의 부호가 양수일 때는 1, 음수일 때는 -1의 함수값을 가지도록 하면 된다. 사각펄스 주기함수의 폭을 수정하여 임펄스꼴의 주기함수(Dirac-delta function)를 구현할 수 있다.



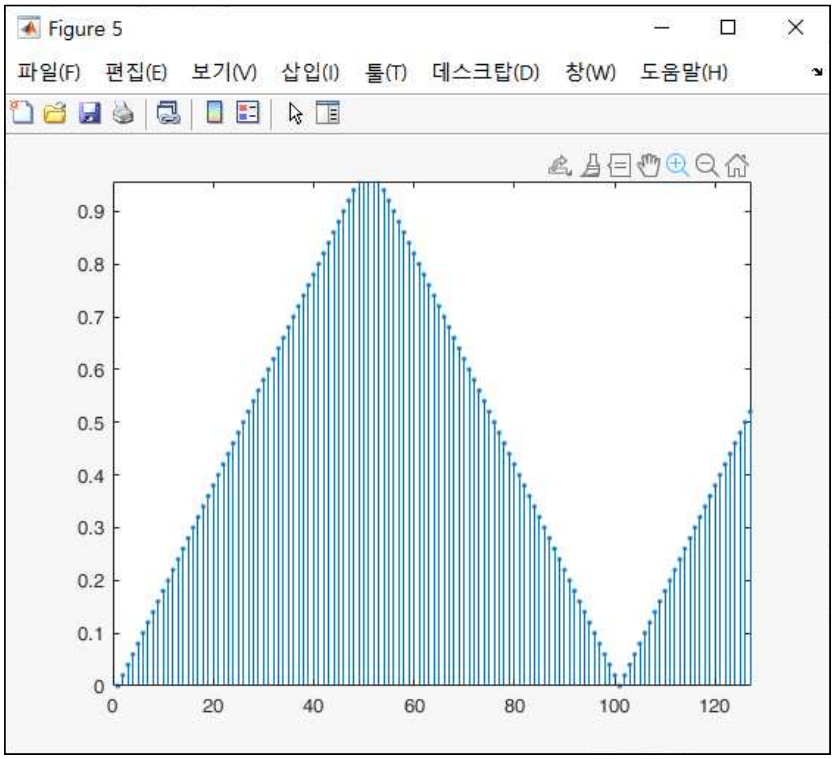
[그림 14]

[그림 14]는 임펄스꼴의 주기함수이다. 사각펄스 주기함수의 폭을 0에 수렴하도록 만들어주면 구현할 수 있다. 함수값이 무한대로 발산하면 임펄스 함수 형태가 된다.



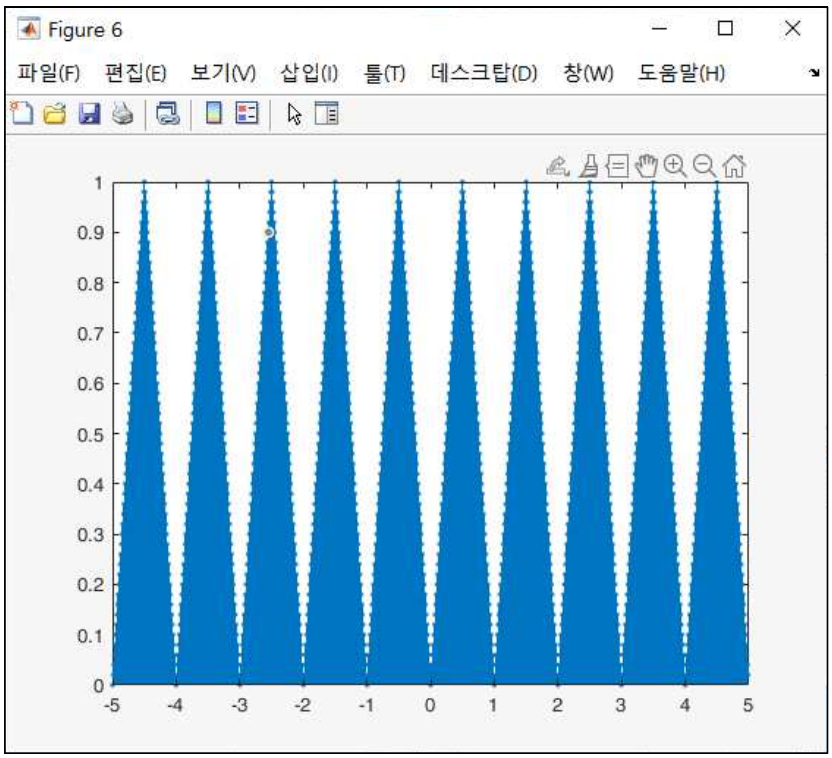
[그림 15]

[그림 15]는 삼각파 함수이다. 이 함수를 반복적으로 사용함으로써 삼각파 주기함수를 구현할 수 있다.



[그림 16]

[그림 16]은 삼각파 주기함수이다. [그림 15]의 삼각파를 반복문을 통해 여러 번 사용하여 구현하였다. 화면을 최대로 축소하면 [그림 17]과 같은 모양의 그래프를 볼 수 있다.



[그림 17]

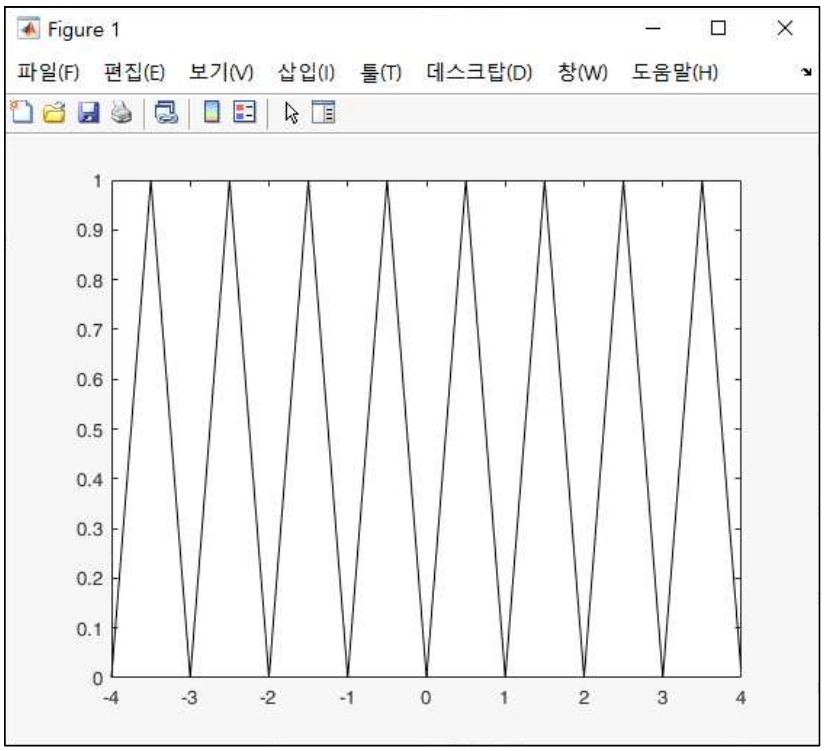
sigHatTrain 함수를 사용하여 구현한 삼각파 주기함수이다. 그래프 확대 시 [그림 16]과 같은 양상을 보인다. sigHatTrain 함수는 4번에서 다루도록 한다.

4. Lab04_sigHatTrain

4.1 Code

```
function [fx, x] = sigHatTrain(range,delta)
% 이 함수는 삼각파 형태의 신호를 생성합니다.
% [fx, x] = sighatTrain(range,delta)

if nargin == 1; delta = 0.01; end
if nargin == 0; delta = 0.01; range=4; end

xu = -0.5:delta:0.5-delta; % 한 주기에 해당하는 정의역
fxu = 1-2*abs(xu); % 1-2|x| % 한 주기에 해당하는 함수

x = -range:delta:range-delta;
len = length(x);

lenxu = length(xu); % 한 주기의 샘플수, 크기
% len = lenxu*range*2;

ftri = zeros(1,len);
for n=1:range*2]
    ftri(1+(n-1)*lenxu:n*lenxu) = fxu;
    %ftri = [ftri, fxu];
end
fx = ftri

if nargout == 0
    plot(x,fx,'k-');
end
```

4.2 Figure



[그림 17]

[그림 17]에서 볼 수 있듯이 sigHatTrain은 [그림 15]와 같은 삼각파 형태의 신호를 생성한다.

삼각파 형태의 주기함수를 반복적으로 사용할 때 편의성을 위하여 3번의 Code 중 삼각파 주기함수를 만드는 부분만 별도의 함수를 만들었다. 이 함수의 이름을 sigHatTrain이라 하고 함수는 x의 범위와 샘플링 주기를 사용할 때마다 입력받는다. 따라서 sigHatTrain 함수를 사용하면 x의 범위와 샘플링 주기을 쉽게 바꾸면서 여러 가지의 삼각파 주기함수를 생성할 수 있다.
sigHatTrain 함수의 경우 입력된 argument 중 샘플링 주기가 없을 경우 default 값으로 0.01을 사용하며 argument의 x범위와 샘플링 주기 모두 없을 경우 default 값으로 x의 범위는 -4 ~ 4를, 샘플링 주기는 0.01을 사용하는 것으로 설정되어 있다.
또한 출력이 아무것도 없을 경우에는 sigHatTrain 내의 fx의 그래프를 plot을 통해 그리도록 기본 설정하였다.

끝.