1. 개요

- 이산신호의 컨볼루션 연산 과정 이해.
- 이산신호의 컨볼루션 연산을 구현하기 위한 알고리즘에 대한 이해.
- 이산신호 컨볼루션 알고리즘의 결과 및 교환법칙과 결합법칙 확인.

2. 본문

(1) 컨볼루션 연산 과정 이해.

다른 말로는 합성곱 연산이라고 불리며, 아래의 복잡해 보이는 식이 컨볼루션의 연산 수식이다.

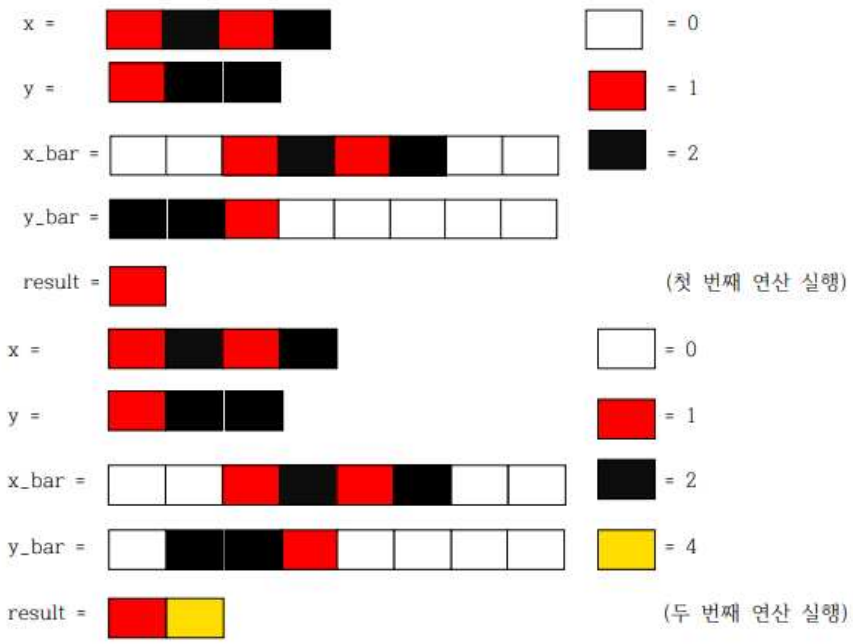
$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = \sum_{k=-\infty}^{\infty} x[k]h[-(k-n)]$$

수식 (1)

하나의 함수(x[n])와 또 다른 함수(h[n])를 반전 한뒤, n만큼의 시간구간 동안 차례대로 h[-n]을 이동시키며 곱의 합을 구하는 연산이다.
주파수 축에서 곱하기 연산과 관련이 있는 연산이다.

(2) 이산신호의 컨볼루션 연산을 구현하기 위한 알고리즘에 대한 이해.

컨볼루션 연산을 구현하기 위해서는 미끄럼 방식(sliding method) 알고리즘을 이해 해야한다. 아래의 사진은 예비보고서에서 설명에 사용했던 두 장의 사진이다.
그림을 기준으로 설명을 하려고 한다.



슬라이딩 대상 신호(y_bar)는 두 번째 입력 신호(y)가 y축 반전된 신호라는 것을 볼 수 있다. 수식(1)을 보면 k의 값이 -∞ ~ ∞ 이지만, 유효값이 없는 부분은 '0'이기 때문에 y_bar 신호와 x의 신호가 겹치는 부분부터 이동연산을 실행하며, 겹치는

부분이 없어지면 연산을 종료한다. 이처럼 유효한 부분만 연산하게 되면, 미끄럼 연산의 횟수는 [x신호의 길이 + y신호의 길이 -1]로 줄어든다.

이러한 유효값의 특성을 이해하면, 컨볼루션 결과(후에 result라고 줄여서 표현한다.)의 범위는 쉽게 알 수 있다. x신호가 시작되는 시간을 n1, y신호가 시작되는 시간을 n2라고 하면, result의 유효값이 시작하는 시간축 좌표는 n1+n2 가 된다. 위 그림에서 y_bar신호를 x신호의 시작점에 맞추기 위해 실제 시간 축에서 좌측으로 n2만큼 이동 한 후 슬라이딩 연산이 시작된다고 생각하면 이해하기 쉽다. 비슷한 과정에 의해 result의 유효값이 끝나는 시간축 좌표는 'x신호가 끝나는시간 축 좌표 + y신호가 끝나는 시간 축 좌표' 로 정의된다.

알고리즘을 구현할 때도 이러한 컨볼루션 과정의 특성들을 모두 고려했다. 매트랩 컨볼루션 알고리즘에 대해서는 이어지는 장에서 하려고 한다.

(3)이산신호 컨볼루션 알고리즘의 결과 코드 & 그래프 (실습결과)

알고리즘을 구현하면서 가장 중요했던 부분은 입력으로 x,y 두 신호를 받았을 때 항상 y 신호 (두번째 입력요소)만 슬라이딩을 하게 한 것이다. 교환법칙이나 결합법칙을 증명하기 위해서는 두 신호의 입력 순서가 반대로 됐을 때 슬라이딩 대상 신호도 바뀌어야 하기 때문이다. 아래의 사진은 실습에 사용 할 컨볼루션 연산함수의 소스코드이다.

```
function [t_conv, result] = conv_3_jo(x, y, N_x, N_y)
x_1 = [zeros(1,length(y)-1), x, zeros(1,length(y)-1)];      (A)
y_bar = [fliplr(y), zeros(1,length(x_1)- length(y))];

result = zeros(1,length(x) + length(y) - 1);                (B)
for i = 1: length(x) + length(y) - 1                        (C)
    result(i) = sum(prod([x_1;y_bar]));
    y_bar = [0, y_bar(1:length(x_1)-1)];
end

t_conv = N_x(1) + N_y(1) : N_x(end) + N_y(end);             (D)
```

(A) 구간은 슬라이딩 연산을 위하여 제로패딩을 해주는 부분이다. x신호 양 옆에 'y신호의 길이 - 1'만큼의 0을 추가해주고, y신호는 값을 반전 후(fliplr 함수 사용) x_1 이라는 변수와 길이가 일치할 때 까지 0을 추가해주는 구간이다. **(B)** 구간에서는 총 연산횟수만큼 '0'으로 가득 찬 result 배열을 만들어준다. **(C)** 구간에서는 x_1 과 y_bar 두 배열의 곱의 합을 구해서 result 배열에 차례대로 값을 넣어준다. 길이가 일치하는 두 행렬의 곱연산에서 prod함수를 사용하였다. **(D)** 구간에서는 컨볼루션 결과의 시간축 좌표를 위에서 설명했던 정의대로 구현해준 뒤 반환해준다.

(2) 실습 DEMO 다음 두 신호의 컨볼루션 연산 결과를 구하시오.

- $x_1[n] = \{0,0,0,0,0,1,1,1,0,0,0\}, n = -5,-4,\ldots,4,5$
- $x_2[n] = \{0,0,0,1,1,1,1,1,0,0,0\}, n = -5,-4,\ldots,4,5$

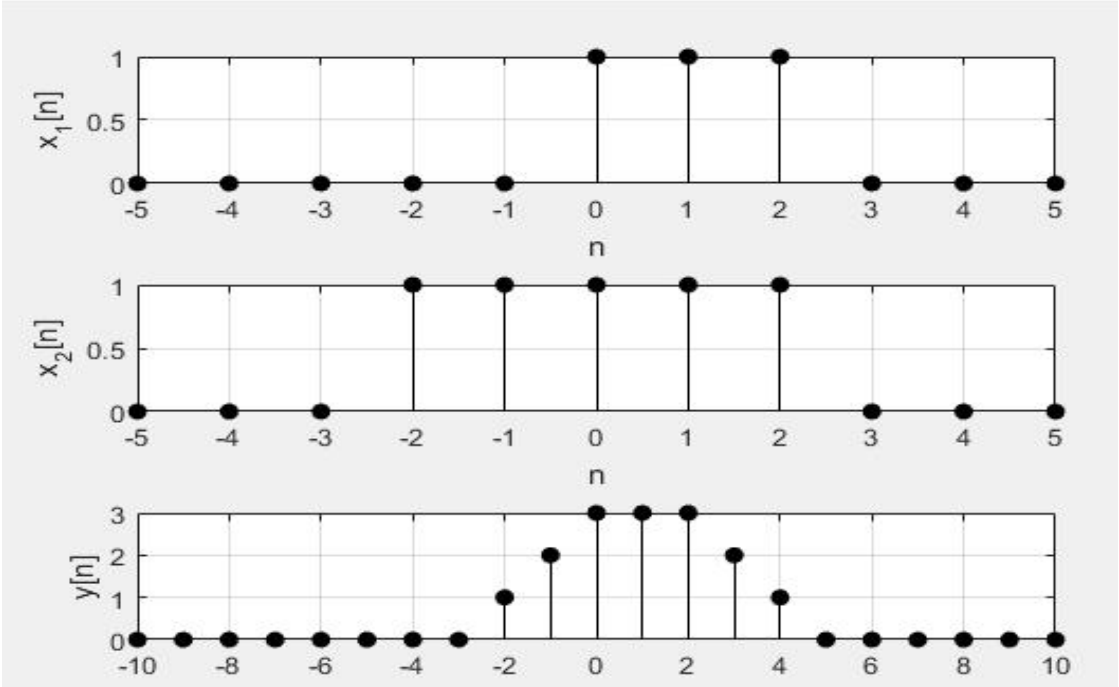
```
clc;
clear;

x = [0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0];
y = [0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0] ;
n_x = -5:5;
n_y = -5:5;

[t_conv,result] = conv_3_jo(x, y,n_x,n_y)

%y1
figure(1)
subplot(311)
stem(n_x,x,'k','MarkerFaceColor','k');
grid on
xlabel('n');
ylabel('x_1[n]');
```

(코드에서 subplot(311) 이후의 나머지 부분은 생략하였습니다.)

손으로 풀어본 결과와 비교해보았습니다.

y[n] = [1,2,3,3,3,2,1], n = -2:4

각각 계수와 시간축 값이며, 아래 실습 결과 그래프와 일치하는 것을 볼 수 있다.



(3) 실습 DEMO 다음 두 신호의 컨볼루션 연산 결과를 구하시오.

- $x_1[n] = A\cos(2\pi \hat{f} n + \theta),\ A = 1, \hat{f} = 0.1, \theta = 0,\ n = 0, 1, \ldots, 14, 15$
- $x_2[n] = \delta[n],\ n = -5, -4, \ldots, 4, 5$

```
clc;
clear;
n_x = 0 : 15;
n_y = -5 : 5;

x = cos(2*pi*0.1*n_x);
y = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0];

[t_conv,result] = conv_3_jo(x, y, n_x, n

%y1
figure(3)
subplot(311)
stem(n_x,x,'k','MarkerFaceColor','k');
grid on
xlabel('n');
ylabel('x_1[n]');
```

(코드에서 subplot(311) 이후의 나머지 부분은 생략하였습니다.)

손으로 풀어본 결과와 비교해보았습니다.

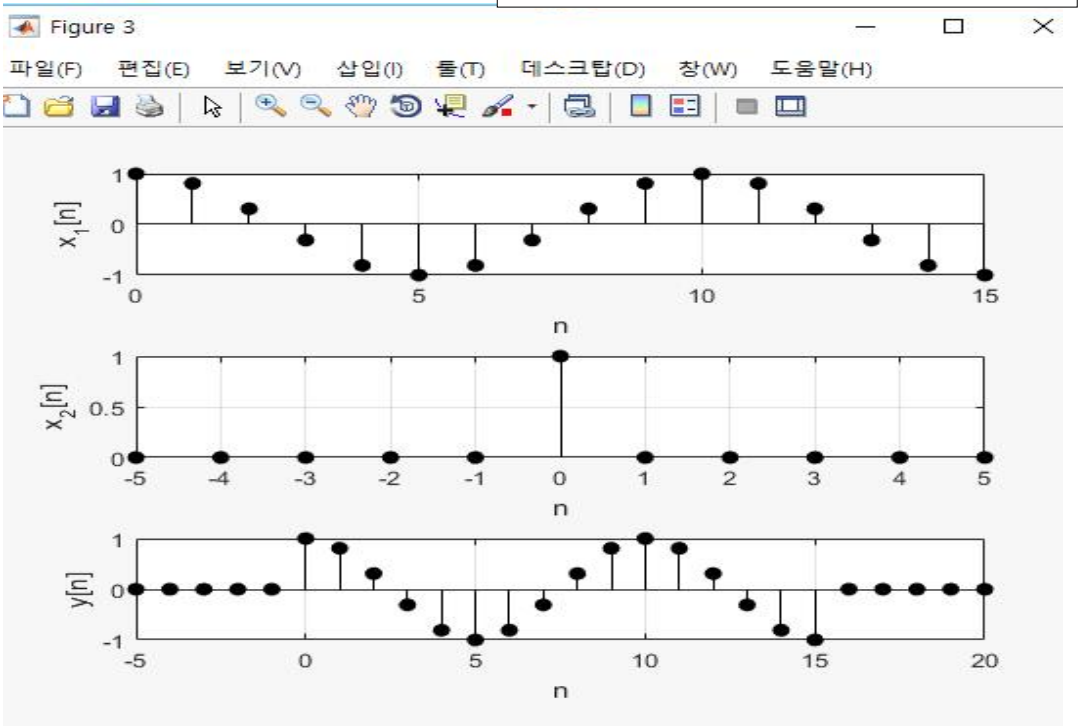
(유횻값만 계산)

y[n] = [1, 0.8090, 0.3090, -0.3090, -0.8090, -1, -0.8090, -0.3090, 3090, 0.8090, 1, 0.8090, 0.3090, -0.3090, -0.8090, -1]

(소수점 5번째 자리에서 반올림)

n = 0:15

각각 계수와 시간축 값이며, 계산 결과에 0을 앞, 뒤로 5개씩 추가해준다면, 결과 그래프와 똑같은 것을 볼 수 있습니다.



(4) 실습 DEMO 다음 두 신호의 컨볼루션 연산 결과를 구하시오.

- $x_1[n] = A\cos(2\pi\hat{f}n + \theta),\ A = 1, \hat{f} = 0.1, \theta = 0,\ n = 0, 1, \ldots, 14, 15$
- $x_2[n] = A\cos(2\pi\hat{f}n + \theta),\ A = 1, \hat{f} = 0.05, \theta = 0,\ n = -5, -4, \ldots, 4, 5$

```
clc;
clear;
n_x = 0 : 15;
n_y = -5 : 5;
x = cos(2*pi*0.1*n_x);
y = cos(2*pi*0.05*n_y);

[t_conv,result] = conv_3_jo(x, y, n_x, n_y);

%y1
figure(1)
subplot(311)
stem(n_x,x,'k','MarkerFaceColor','k');
grid on
xlabel('n');
ylabel('x_1[n]');
```

(코드에서 subplot(311) 이후의 나머지 부분은 생략하였습니다.)

손으로 풀어본 결과와 비교해보았습니다.

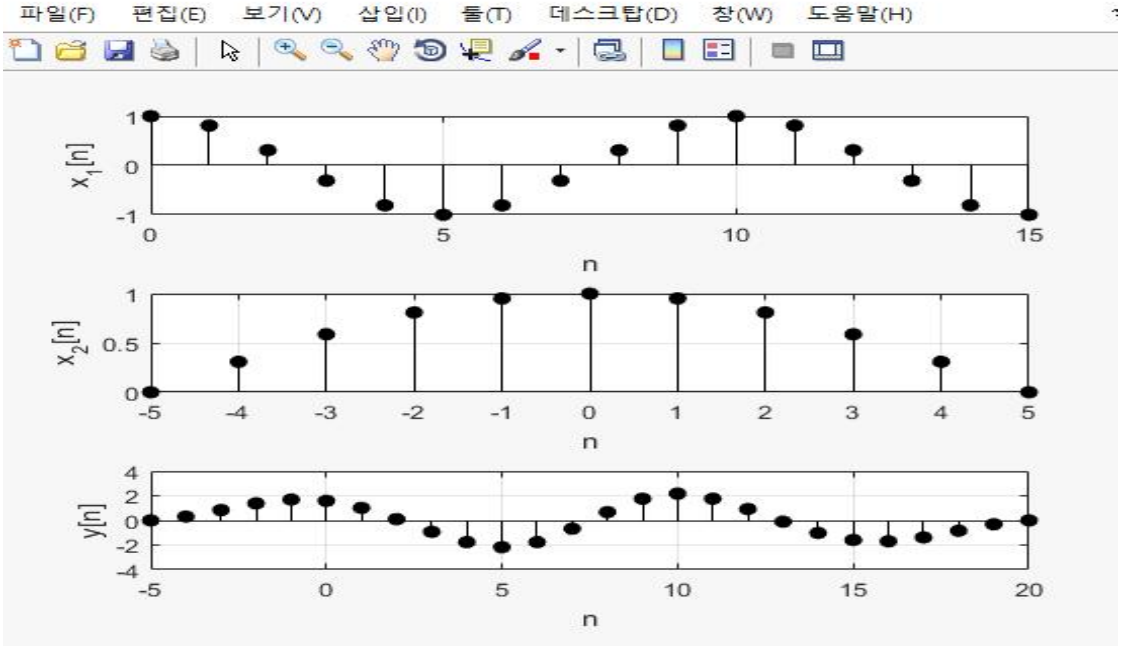
(유횻값만 계산)

y[n] = [0.3090, 0.8378, 1.3800, 1.6917, 1.5878, 1.0194, 0.1013, -0.9223, -1.7601, -2.1756, -1.7601, -0.6723, 0.6723, 1.7601, 2.1756, 1.7601, 0.9223, -0.1013, -1.0194, -1.5878, -1.6917, -1.3800, -0.8378, -0.3090]

(소수점 5번째 자리에서 반올림)

n = -4:19

각각 계수와 시간축 값이며, 계산 결과에 0을 앞, 뒤로 추가해준다면, 결과그래프와 똑같은 것을 볼 수 있습니다.

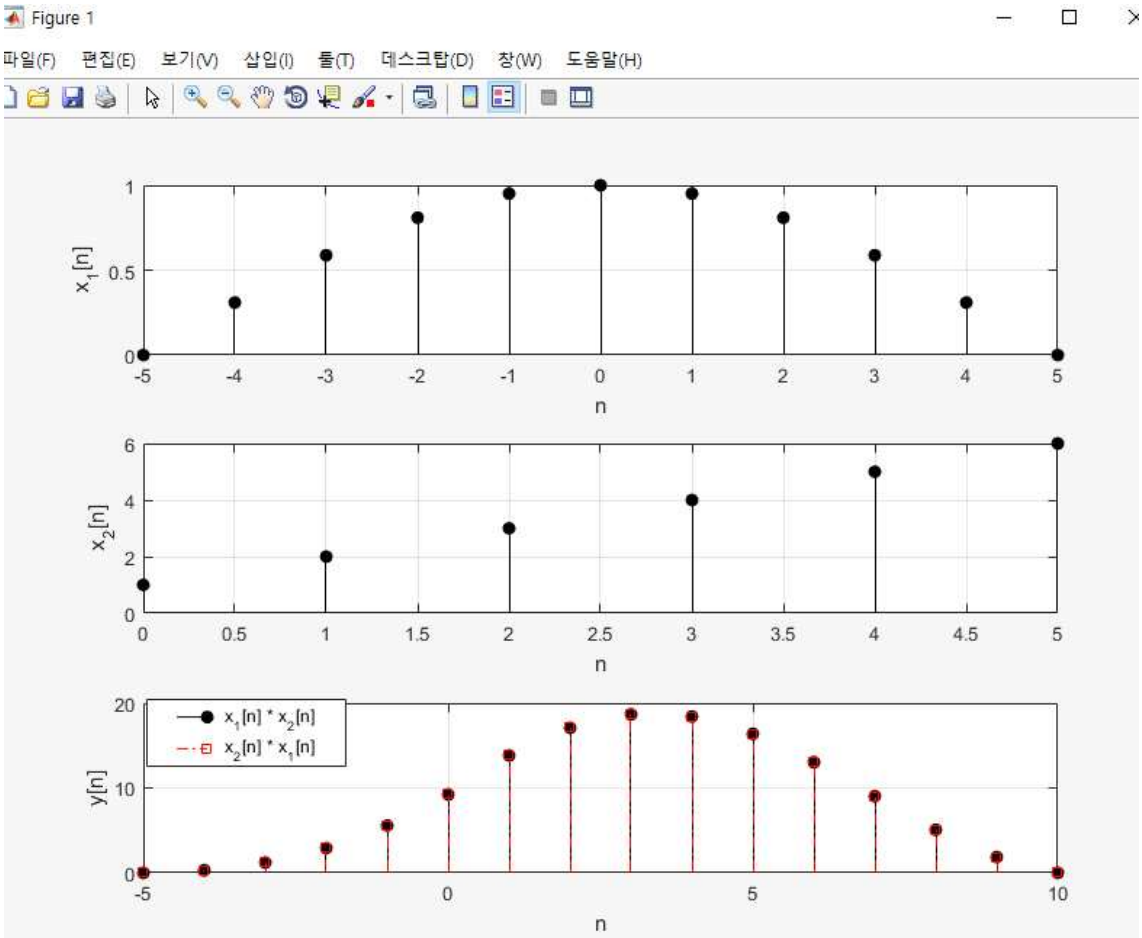


3.2 컨볼루션 연산의 특성

(1) 실습 DEMO 다음 두 이산신호의 컨볼루션에 대해서 교환법칙이 성립함을 보여라.

- $x_1[n] = A\cos(2\pi\hat{f}n + \theta),\ A = 1, \hat{f} = 0.05, \theta = 0,\ n = -5, -4, \ldots, 4, 5$
- $x_2[n] = \{1, 2, 3, 4, 5, 6\},\ n = 0, 1, 2, 3, 4, 5$

```
clc;
clear;

n_x = -5 : 5;
n_y = 0 : 5;
x = cos(2*pi*0.05*n_x);
y = [1,2,3,4,5,6];

[t_conv,result] = conv_3_jo(x, y, n_x, n_y);
[t_conv_1,result_1] = conv_3_jo(y, x, n_y, n_x);

%y1
figure(1)
subplot(311)
stem(n_x,x,'k','MarkerFaceColor','k');
grid on
xlabel('n');
ylabel('x_1[n]');

%y2
subplot(312)
stem(n_y,y,'k','MarkerFaceColor','k');
grid on
xlabel('n');
ylabel('x_2[n]');

%y3
subplot(313)
stem(t_conv,result,'k','MarkerFaceColor','k');
grid on
hold on
xlabel('n');
ylabel('y[n]');
stem(t_conv_1,result_1,'-.sr');
legend('x_1[n] * x_2[n]', 'x_2[n] * x_1[n]','Location','NorthWest' );
```



y[n] = [0.3090, 1.2058, 2.9116, 5.5685, 9.2254, 13.8333, 17.0872, 18.6684, 18.4222, 16.3727, 13.0296, 9.0291, 5.0718, 1.8541]

n = -4 : 9

각각 계수와 시간축 값이며, 계산 결과에 0을 앞, 뒤로 추가해준다면, 결과그래프와 똑같은 것을 볼 수 있습니다.

저희 조의 컨볼루션 함수는 신호의 형태나 길이에 상관없이 두 번째로 들어온 신호가 슬라이딩의 대상 신호가 됩니다. 신호의 순서를 바꿔서 입력해준 뒤, 결과가 일치하기 때문에, 교환법칙이 성립하는 것을 확인할 수 있습니다.

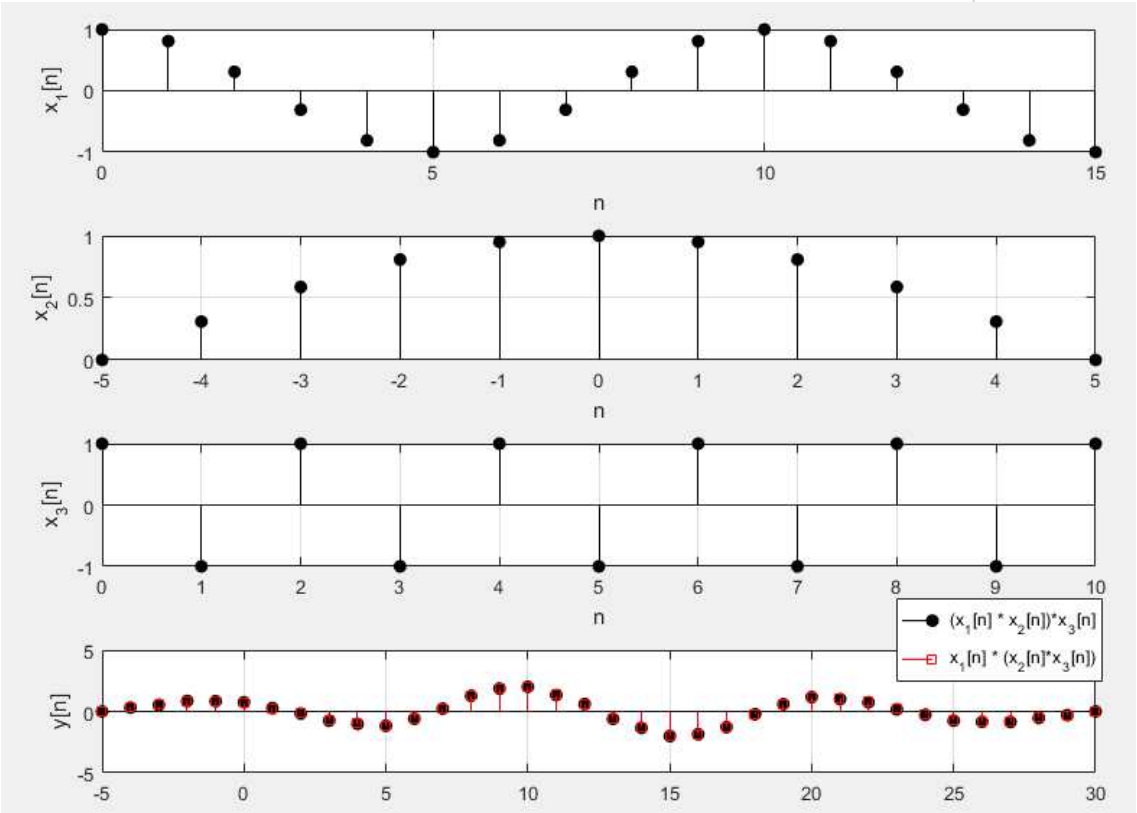
※ 검정색 -> x1*x2 컨볼루션한 값 빨간색 -> x2*x1 컨볼루션한 값

(2) 실습 DEMO 다음 세 이산신호의 컨볼루션에 대해서 결합법칙이 성립함을 보여라.

- $x_1[n] = A\cos(2\pi \hat{f} n + \theta)$, $A = 1, \hat{f} = 0.1, \theta = 0$, $n = 0, 1, \ldots, 14, 15$
- $x_2[n] = A\cos(2\pi \hat{f} n + \theta)$, $A = 1, \hat{f} = 0.05, \theta = 0$, $n = -5, -4, \ldots, 4, 5$
- $x_3[n] = (-1)^n$, $n_3 = 0, 1, \ldots, 9, 10$

```
clc;
clear;
n1 = 0:15;
n2 = -5 : 5;
n3 = 0 : 10;
x1 = cos(2*pi*0.1*n1);
x2 = cos(2*pi*0.05*n2);
x3 = (-1).^n3;
%(x1*x2)*x3
[t_conv,result] = conv_3_jo(x1, x2, n1, n2);
[t_conv_1,result_1] = conv_3_jo(result, x3, t_conv, n3);
%x1*(x2*x3)
[t_conv,result] = conv_3_jo(x2, x3, n2, n3);
[t_conv_2,result_2] = conv_3_jo(x1,result, n1, t_conv);
%x1
figure(1)
subplot(411)
stem(n1,x1,'k','MarkerFaceColor','k');
grid on
xlabel('n');
ylabel('x_1[n]');
%x2
subplot(412)
stem(n2,x2,'k','MarkerFaceColor','k');
grid on
xlabel('n');
ylabel('x_2[n]');
%x3
subplot(413)
stem(n3,x3,'k','MarkerFaceColor','k');
grid on
xlabel('n');
ylabel('x_3[n]');
```

```
%conv
subplot(414)
stem(t_conv_1,result_1,'k','MarkerFaceColor','k');
grid on
hold on
stem(t_conv_2,result_2,'sr');
legend('(x_1[n] * x_2[n])*x_3[n]', 'x_1[n] * (x_2[n]*x_3[n])','Location','NorthEast' );
xlabel('n');
ylabel('y[n]');
```



y[n] = [0.3090, 0.5288, 0.8513, 0.8404, 0.7473, 0.2721, -0.1708, -0.7515, -1.0086, -1.1670, -0.5931, 0.2298, 1.2802, 1.8599, 2.0074, 1.3404, 0.6013, -0.6013, -1.3404, -2.0074, -1.8599, -1.2802, -0.2298, 0.5931, 1.1670, 1.0086, 0.7515, 0.1708, -0.2721, -0.7473, -0.8404, -0.8513, -0.5288, -0.3090]

n = -4 : 29

각각 계수와 시간축 값이며, 계산 결과에 0을 앞, 뒤로 추가해준다면, 결과그래프와 똑같은 것을 볼 수 있습니다.

앞서 설명했던 저희조의 함수를 이용하면, 입력의 순서만 변경하여 x1*(x2*x3) 와 (x1*x2)*x3를 구현했고 두 파형이 일치기 때문에, 결합법칙을 증명했습니다.

※ 검정색 -> (x1*x2)*x3 컨볼루션한 값 빨간색 -> x1*(x2*x3) 컨볼루션한 값

(3) 실습 DEMO 다음 이산신호 입력 $x[n]$와 임펄스응답 $h[n]$으로 이동성질 $y[n-n_0] = x[n-n_0] * h[n]$ 이 만족함을 보여라.

- $x[n] = A\cos(2\pi \hat{f} n + \theta)$, $A = 1$, $\hat{f} = 0.1$, $\theta = 0$, $n = 0, 1, \ldots, 14, 15$
- $h[n] = (0.5)^n$, $n = 0, 1, \ldots, 4, 5$

```
clc;
clear;
n1 = 0 : 15;
n2 = 0 : 5;
x1 = cos(2*pi*0.1*n1);
x2 = (0.5).^n2;

[t_conv,result] = conv_3_jo(x1, x2, n1, n2);

%x1
figure(1)
title('y[n] = x1[n] + x2[n]')
subplot(311)
stem(n1,x1,'k','MarkerFaceColor','k');
grid on
xlabel('n');
ylabel('x_1[n]');

%x2
subplot(312)
stem(n2,x2,'k','MarkerFaceColor','k');
grid on
xlabel('n');
ylabel('x_2[n]');

%y
subplot(313)
stem(t_conv,result,'k','MarkerFaceColor','k');
grid on
xlabel('n');
ylabel('y[n]');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
delay = 5;
n1= 0 : 15;
n2 = 0 : 5;
x1 = cos(2*pi*0.1*n1);
x1 = [x1 , zeros(1,5)]; % n' = n1 - 5 구현
n1 = 0 : 15 + delay ;
x2 = (0.5).^n2

[t_conv,result] = conv_3_jo(x1, x2, n1, n2);
t1 = -5 : 15;
t2 = -5 : 20;
%x1
figure(2)
title('y[n-5] = x1[n-5] + x2[n]')
subplot(311)
stem(t1,x1,'r','MarkerFaceColor','r');
axis([0,15,-1,1]);
grid on
xlabel('n');
ylabel('x_1[n]');

%x2
subplot(312)
stem(n2,x2,'k','MarkerFaceColor','k');
grid on
xlabel('n');
ylabel('x_2[n]');

%y
subplot(313)
stem(t2,result,'r','MarkerFaceColor','r');
axis([0,20,-2,2]);
grid on
xlabel('n');
ylabel('y[n]');
```

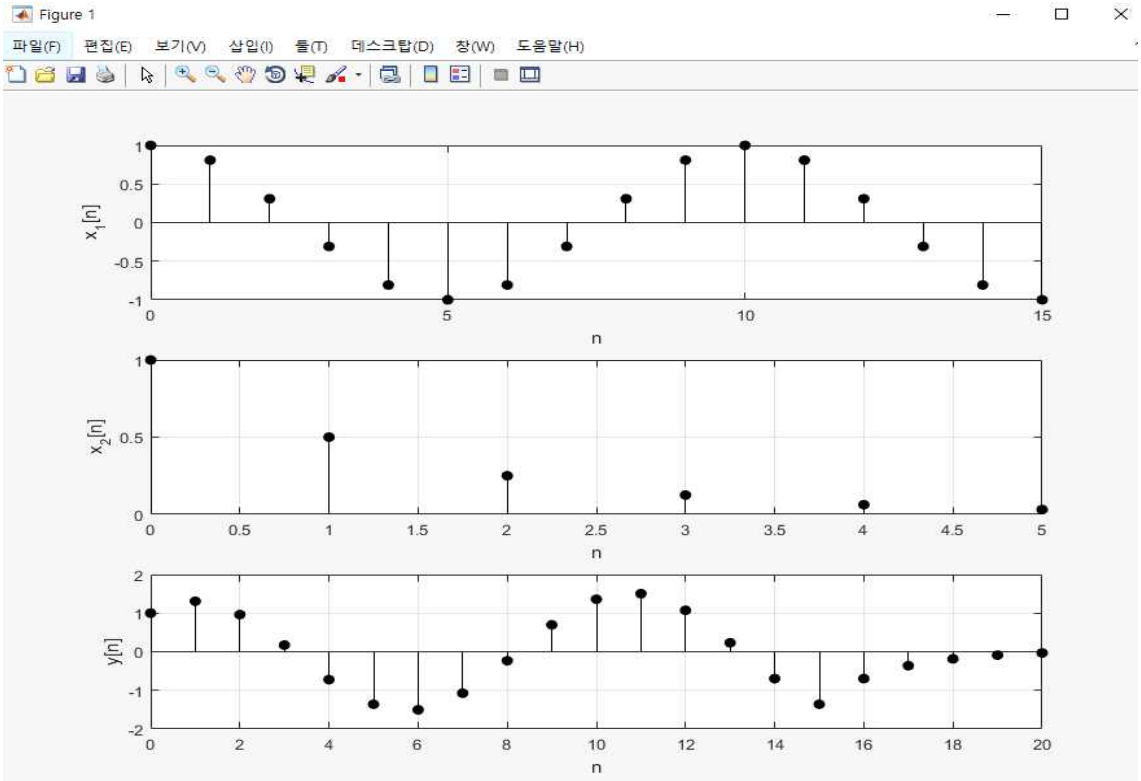


Figure 1
파일(F) 편집(E) 보기(V) 삽입(I) 툴(T) 데스크탑(D) 창(W) 도움말(H)
x1[n]
x2[n]
y[n]
n

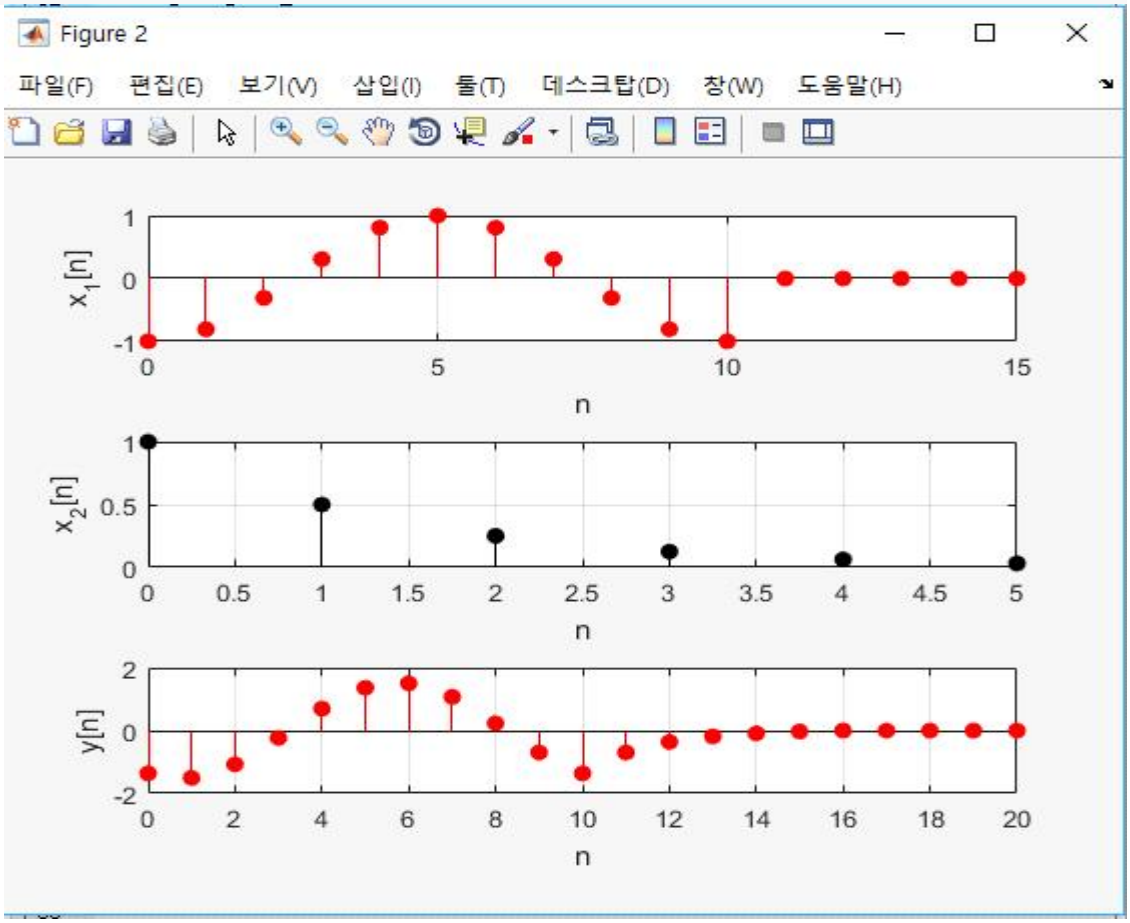


Figure 2
파일(F) 편집(E) 보기(V) 삽입(I) 툴(T) 데스크탑(D) 창(W) 도움말(H)
x1[n]
x2[n]
y[n]
n

-시간 이동 전의 컨볼루션 연산 결과.

y[n] = [1.0000, 1.3090, 0.9635, 0.1727, -0.7226, -1.3613, -1.5053, -1.0743, -0.2330, 0.6974, 1.3613, 1.5053, 1.0743, 0.2330, -0.6974, -1.3613, -0.6963, -0.3608, -0.1852, -0.0878, -0.0313]

n = 0 : 20;

-시간 이동 후의 컨볼루션 연산 결과.

y[n] = [-1.3613, -1.5053, -1.0743, -0.2330, 0.6974, 1.3613, 1.5053, 1.0743, 0.2330, -0.6974, -1.3613, -0.6963, -0.3608, -0.1852, -0.0878, -0.0313, 0, 0, 0, 0, 0]

n’ = 0: 20;

교수님이 말씀해주신 대로 “ n' = n1 - 5”라고 가정하고 시간축을 표현했습니다.
시간지연이 일어나도 신호의 파형이 변하지 않았다면 컨볼루션 연산 결과는 정확하게 일치하는 것을 볼 수 있습니다. 결과 그래프를 통해 컨볼루션의 시간 지연 성질까지 확인하고 성공적으로 실습을 끝냈습니다.

※ Figure 1-> y[n]=x[n]*h[n] 값 Figure 2 -> y[n-5]=x[n-5]*h[n] 값

3. 느낀점 및 소감

지금까지 컨볼루션 연산을 직접 해본 적이 없고, 필요하다면 매트랩 함수를 사용했었습니다. 실제로는 함수 한 줄이면 됐었는데, 생각보다 많은 계산과 고급 알고리즘이 들어가기 때문에, 구현하기 위한 소모된 시간은 많았습니다.

최대한 프로그램 코드를 짧고 단순하게 짜보려고 했고 나름 성공적으로 수행하였다고 생각합니다. 이번 실습으로 컨볼루션 연산에 대해 깊이 있게 알 수 있는 한 주 였습니다.

목 차