DSP PROJECT

- 신호의 왜곡을 최소화하는 시스템 설계 -

디지털신호처리 정원국 교수님

전자정보공학부 전자전공

20180474 남아리

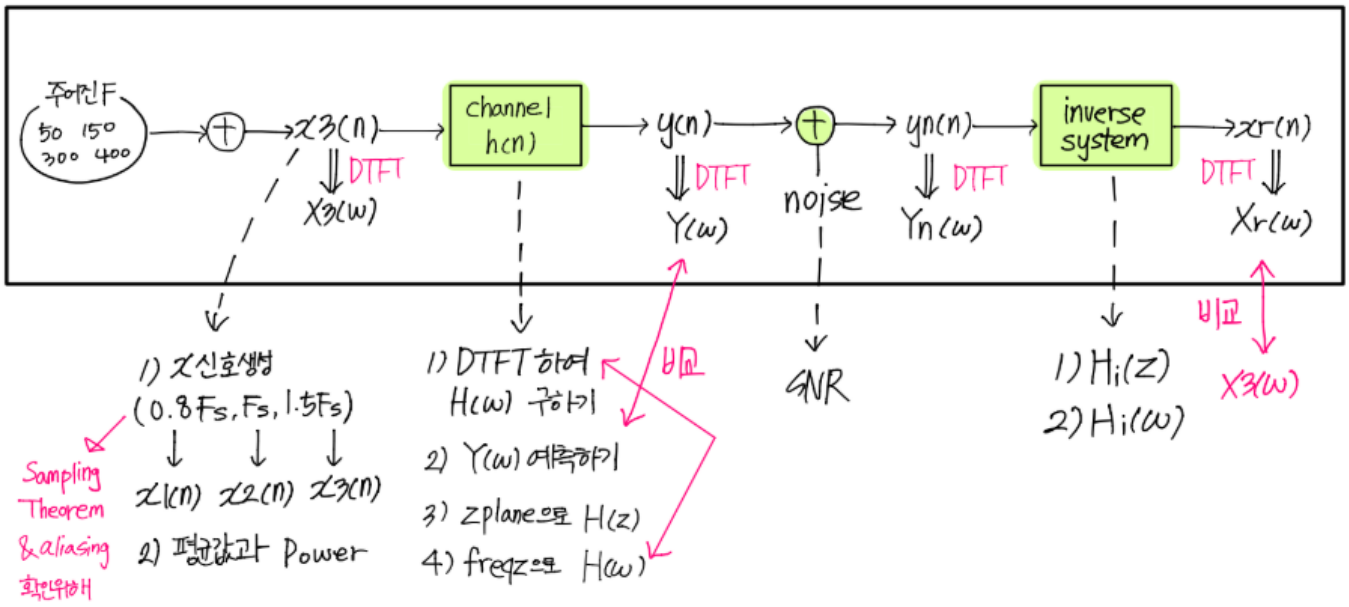
목차

1. 설계 목적 및 동기

1) Sampling Theorem을 바탕으로 Fs를 구하고, 그에 따라 발생하는 aliasing에 대해 이해한다.
2) 신호가 채널을 통과하며 생기는 왜곡과 그 왜곡을 Inverse system으로 처리하는 과정을 이해한다.
3) 직접 MATLAB을 이용하여 시스템을 설계하고 시뮬레이션해봄으로써 지금까지 이론으로 배운 DTFT와 Z-Tr을 실제로 적용해본다.
4) 이론으로만 공부했던 내용을 MATLAB의 PLOT을 통해 시각적으로 확인한다.
5) 이론 수업에서 배웠던 내용을 바탕으로 MATLAB의 결과를 예측하고, 예측 결과와 실제 결과를 비교해봄으로써 본인이 어떤 개념이 부족한지 파악하고, 책으로만 학습하여 헷갈렸던 개념들을 몸소 체화하며 익힌다.
6) MATLAB CODE를 작성하며 기본적인 MATLAB 사용법을 익힌다.
7) 프로젝트를 진행하면서 신호처리에 대한 기본 개념들을 이해하고 전체적인 틀을 스스로 정립한다.

2. 프로젝트 설계

1) 전체적인 설계



2) 단계별 설계 및 각 figure에 해당하는 그래프 설명

- 주어진 4개의 Tone을 바탕으로 Fs (Sampling Frequency)를 설정하고, 샘플 수 N을 설정한다.
- 주어진 4개의 Tone을 더한 신호 x(n)을 생성하되, Fs값을 0.8Fs, Fs, 1.5Fs로 각각 다르게 하여 x1(n), x2(n), x3(n)의 신호를 생성한다.
- 각 x(n)의 신호를 DTFT하여 X1(w), X2(w), X3(w)를 생성하고 X(w) 신호들을 figure1에 plot한다.
- x3(n)을 channel h(n)에 넣어, y(n)을 생성하고 figure2에 stem한다.
- h(n)을 직접 DTFT하여 구하고, 구한 H(w)를 figure3에 plot한다.
- y(n)을 DTFT하여 Y(w)를 생성하고 figure4에 plot한다.
- zplane을 이용하여 H(z)의 pole-zero diagram을 figure5에 그리고, freqz를 이

용하여 H(w)를 figure6에 plot한다.

- y(n)에 백색잡음을 더한 yn(n)값을 생성한다.
- yn(n)을 DTFT하여 Yn(w)를 생성하고 figure7에 Y(w)와 Yn(w)를 plot한다.
- Inverse System을 설계하고 위에서와 마찬가지로 pole-zero diagram과 주파수 응답을 figure8에 그린다.
- Inverse System을 통과한 xr(n)신호를 생성하고, 제대로 복구되었는지 확인하기 위해 x3(n)과 xr(n)을 figure9에 stem한다.
- xr(n)을 DTFT하여 Xr(w)를 생성하고 Xr(w)와 X3(w)를 figure10에 plot한다.

3. 프로젝트 구현

1) Fs와 N

```
Fs = 900;

N = 9000;

n = [0:N-1];
```

Sampling Theorem을 이용하였다. Sampling Theorem에 의하면, Fs > 2Fmax 이므로 주어진 50,150,300,400Hz 중에서 가장 큰 400Hz를 2배한 것 보다 커야 한다. 그러므로 Fs > 800Hz 이어야 aliasing이 일어나지 않고 모든 신호를 정보 손실없이 복원할 수 있다. 그래서 이 프로젝트에서는 800보다 큰 900Hz를 Sampling Frequency로 설정하였다.

많은 숫자 중 900으로 설정한 이유는, N값의 설정과 주파수 해상도를 고려하기 위함이다. 주파수해상도는 Fs/N으로 이것의 배수들을 잘 잡아낼 수 있다. 그래서 N값이 클수록 좋겠지만, 실제적으로는 메모리의 문제가 발생하기 때문에 적당한 값을 설정해야 한다. 프로젝트를 실행한 이 노트북으로는 N이 10000을 넘어가면 문제가 발생하지는 않으나 속도가 매우 느려지는 것을 볼 수 있었다. 그래서 Fs/N = 0.1을 만들기위해 N을 9000, Fs를 900으로 설정하였다.

2) x(n) 신호의 생성

```
fsc = [0.8 1.0 1.5];

x3=zeros(size(n));

for f=F

   x3=x3+cos(2*pi*f/(fsc(3)*Fs)*n+0);

end
```

Sampling Theorem에 의해, 디지털에서는 $\pi > 2\pi f/Fs$ 를 만족해야 한다. 그렇지 않으면 겹치는 부분이 발생한다. 그래서 $x(n) = \cos\left(\frac{2\pi fn}{Fs}\right)$으로 x(n)의 신호를 생

생하였다.

또한 Sampling Theorem을 만족하지 않는 경우, aliasing이 발생하는지 확인하기 위해 Fs를 상수배하여 각각 다르게 한, 3가지의 신호를 생성하였다. 그리고 4가지의 tone을 다 더하기 위해 for-end문을 이용하였으며, 더하기 전에 모든 값이 0으로 초기화된 1X9000의 행렬을 만들기위해 zero문을 이용하였다.

3) 평균값과 power

```
Mx3 = sum(x3)/N;

Px3 = sum(x3.^2)/N;
```

위와 같이 mean과 power를 구현하였으며 mean의 값은 0, power의 값은 2로 계산했다. 그 이유는 cos함수는 -1~1을 왔다갔다하는데, 한 주기를 잘라서 그 값을 다 더하면 0이 나오므로 mean의 값은 0이다. 그리고 power는 제곱을 해서 구하는데, 제곱을 하면 양수가 되므로 한 주기를 다 더하면 2가 된다. 약간의 오차는 발생했지만 실제 결과값과 계산한 값이 동일하게 출력되었다.

Mx1	double	1x1	-1.7635e-15
Mx2	double	1x1	6.3392e-15
Mx3	double	1x1	0.00054248
N	double	1x1	9000
Pn	double	1x1	1.0054
Px1	double	1x1	2.0000
Px2	double	1x1	2.0000
Px3	double	1x1	2.0003

4) DTFT

다음과 같이, DTFT의 기본식을 이용하여 n단위의 신호를 DTFT한 주파수 영역신호를 생성했다.

```
Xw3 = x3*exp(-j*n'*w);
```

5) H(z)와 H(w) – zplane & freqz

zplane을 이용하여 H(z)의 pole-zero diagram을 그렸고, freqz로 H(w)를 생성한 뒤 plot으로 그래프를 그렸다.

```
zplane(B,A);                    Hw = freqz(B,A,K);
```

위와 같이 구현하였으며 형식과 괄호안에 들어가는 값은 다음과 같다.

zplane(분자의 계수, 분모의 계수)

freqz(분자의 계수, 분모의 계수, 샘플의 수)

6) Inverse System

Inverse system은 원래의 입력신호가 채널을 통과하면서 생긴 왜곡을 없애고 다시 원래의 입력신호를 출력하는 시스템이다.

입력 신호가 x(n)이고 이것을 DTFT한 것을 X(w), 시스템 H(w)와 Hi(w)를 통과한 신호를 Xr(w)이라고 한다면, Xr(w) = X(w)H(w)Hi(w)이다. 그런데 이 때, Inverse System을 함으로써 원하는 결과는 Xr(w) = X(w)가 되는 것이다. 그러므로, H(w)Hi(w) = 1이어야하고, 따라서 H(w)와 Hi(w)는 역수 관계이다.

그래서 분모와 분자의 계수를 바꿔 다음과 같이 구현하였다.

```
Hz = zplane(A,B);               Hiw = freqz(A,B,K);
```

7) noise를 섞어서 yn(n) 생성

SNR	double	1x1	100
SNR_dB	double	1x1	20

기존의 y에 잡음을 섞을 것이기 때문에 난수로 이루어진 y크기의 1x9000행렬을

만들었다.

```
noise = randn(size(y));
```

그리고 SNR은 신호 대 잡음 비를 의미하며 위에 보이는 것과 같이 SNR_dB = 20, SNR = 10^(20/10) = 100으로 설정하였다

8) xr(n)과 xr(n)의 DTFT

filter를 통해서 구현하였다. 형식과 괄호안의 값은 다음과 같다.

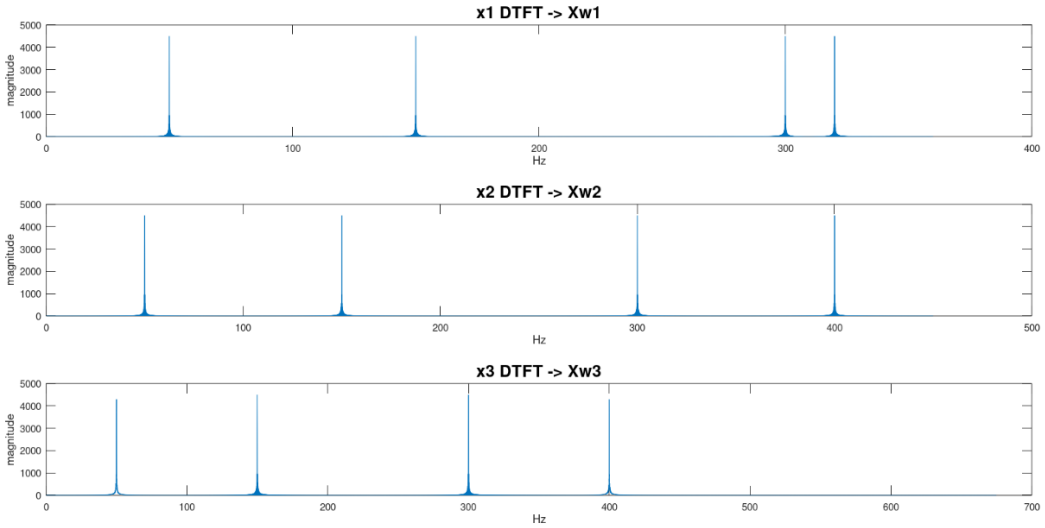
filter(분자의 계수, 분모의 계수, 필터링할 입력 값)

그리고 코드는 아래와 같이 구현하였으며 Inverse System으로 필터링을 하는 것이기 때문에 분자의 계수를 A, 분모의 계수를 B라고 작성하였다.

```
xr = filter(A,B,yn);                    xrw = xr*exp(-j*n'*w);
```

4. 시뮬레이션 결과 및 분석

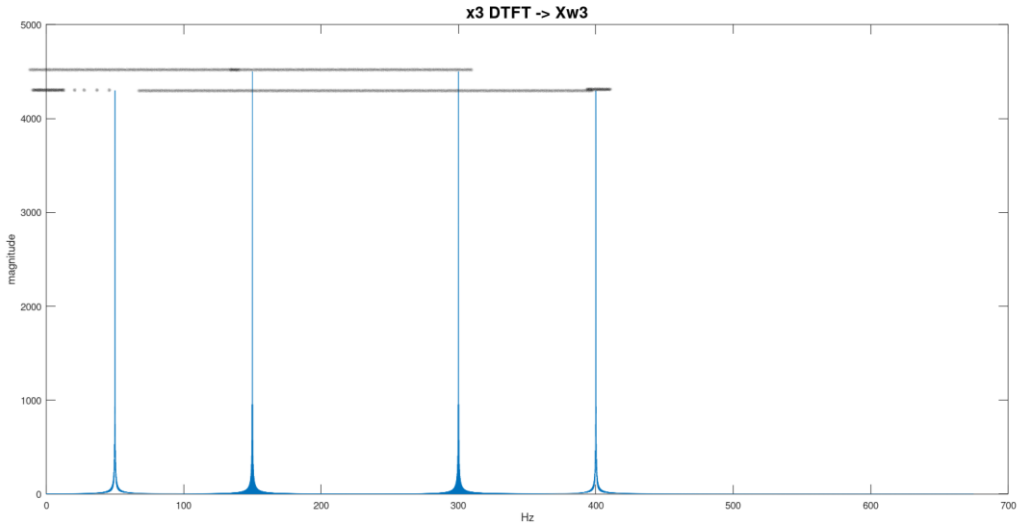
1) x(n)을 DTFT하여 생성된 X(w) 결과 및 분석

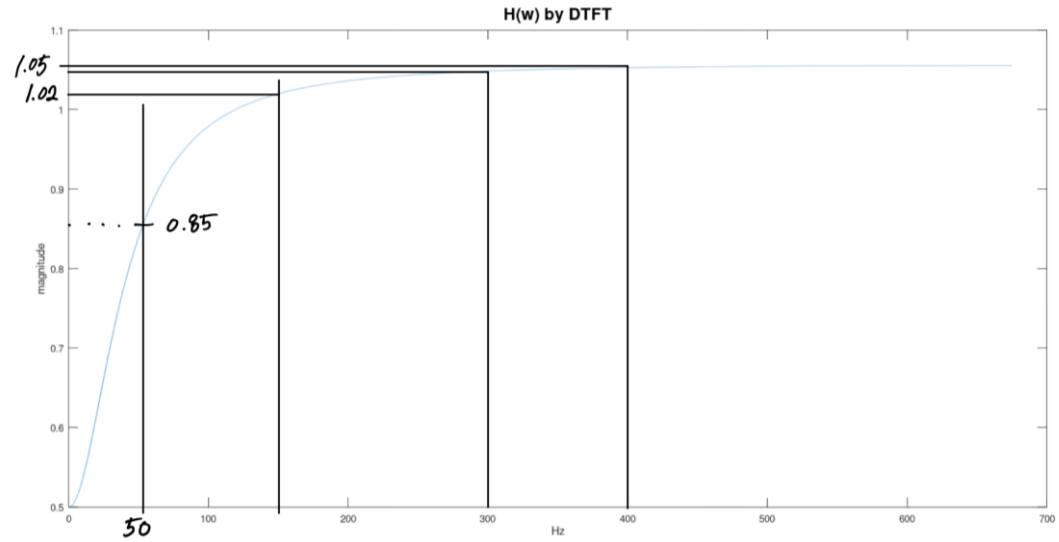


x1(n)은 0.8Fs인 720Hz로 Sampling을 하였고, x2(n)은 900, x3(n)은 1350으로 Sampling하였다. Sampling Theorem에 의하면, Fs > 2Fmax 이므로 주어진 50, 150, 300, 400Hz 중에서 가장 큰 400Hz를 2배한 것 보다 커야 한다. 그러므로 Fs > 800Hz 이어야 aliasing이 일어나지 않고 모든 신호를 정보 손실없이 복원할 수 있다. 그런데 x1(n)은 800보다 작은 720Hz로 Sampling 되었기 때문에, X1(w)에서 aliasing이 일어난 것을 확인할 수 있다.

2) H(w) 구하고 Y(w) 예측

눈대중으로, X(w)에서 50과 400Hz에서의 크기는 약 4300정도라고 하고, 150과 300Hz에서의 크기는 4500정도로 하였다.





그리고 각각의 주파수에서의 |H(w)|는 각각 0.85, 1.02, 1.05, 1.055라고 어림잡는다. 마지막 것은 1.05와 비슷하나 아주 약간의 차이가 있어서 1.055라고 어림잡았다.

그렇게해서 X(w)H(w)를 계산하면 다음과 같은 값이 나온다.

50Hz에서의 크기 : 4300*0.85 = 3655

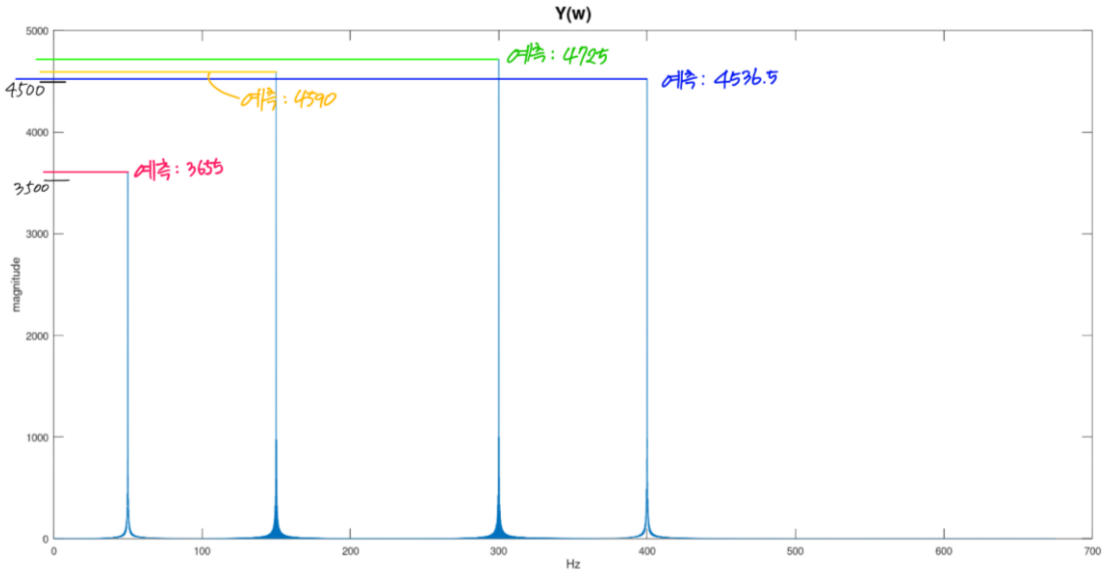
150Hz에서의 크기 : 4500*1.02 = 4590

300Hz에서의 크기 : 4500*1.05 = 4725

400Hz에서의 크기 : 4300*1.055 = 4536.5

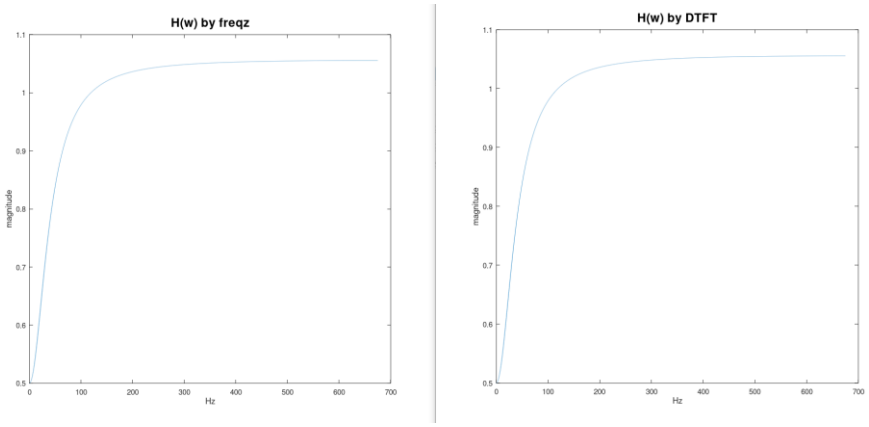
그래서 Y(w)는 각각의 주파수에서, 위의 값과 같은 크기를 갖는 그림이 띄워질 것이라 예측한다.

3) Y(w)와 위의 예측 비교



마우스 커서로 좌표 값을 확인한 결과, 각각의 주파수에서 3615, 4602, 4726, 4532의 크기를 가졌다. 위에 2번에서의 예측을 눈대중으로 하였고, 3번에서 Y(w)의 크기를 마우스 커서로 확인한 값도 정확하게 자로 잰 것이 아니기에 오차가 발생하였다. 하지만 큰 오차 없이 예측했던 대로 결과가 나왔음을 확인할 수 있다.

4) 직접 수식으로 구한 H(w)와 freqz로 구한 H(w)의 결과 비교

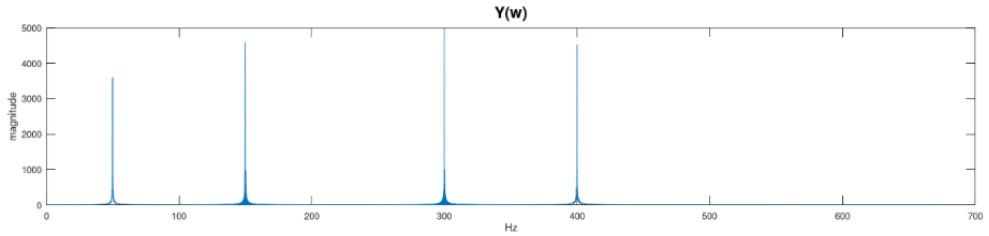


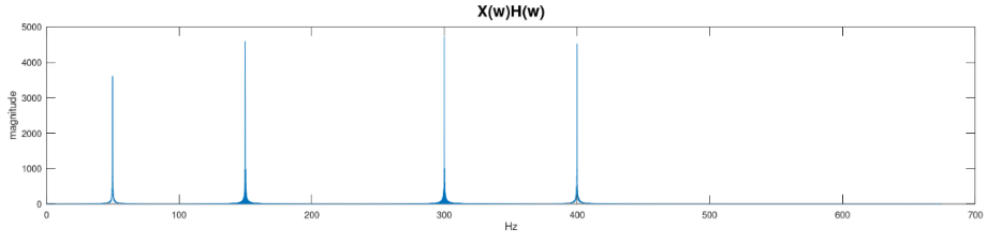
위의 왼쪽은 freqz로 구한 H(w)이고, 오른쪽은 프로젝트의 7번과정에서 h(n)을 직접 DTFT하여 구한 H(w)이다. 직접 계산해서 그린 것과 함수를 이용하여 그린 것이 동일함을 확인할 수 있다. 또한 이론에서 배운 DTFT를 실제로 적용해보고, 그것이 실제로 성립함을 눈으로 확인할 수 있었다.

또한, $H(w) = {1 - 0.9 \times e^{-jw}}/{1 - 0.8 \times e^{-jw}}$ 에 w=0을 대입했을 때, 0.1/0.2 = 0.5의 값이 나왔는데, 계산한 것과 동일하게 그래프에 나타났음을 확인할 수 있다.

밑에의 그래프는 '프로젝트의 수행절차 7번'에서 DTFT로 H(w)를 잘 구했는지 확인하기 위해 추가적으로 수행한 작업이다. DTFT로 구한 H(w)에 X(w)를 곱하여 X3(w)H(w)의 그래프를 생생하였고, x3(n)을 channel에 직접 넣어서 생성한 y(n)을 DTFT하여 만든 Y(w)의 그래프를 생성하였다. 그래서 두 그래프가 동일한지 비교하는 작업을 하였다. 물론, 위에서도 볼 수 있듯이 H(w)가 제대로 출력되었지만 다시 한번 제대로 확인하기 위해 이와 같은 작업을 거쳤다.

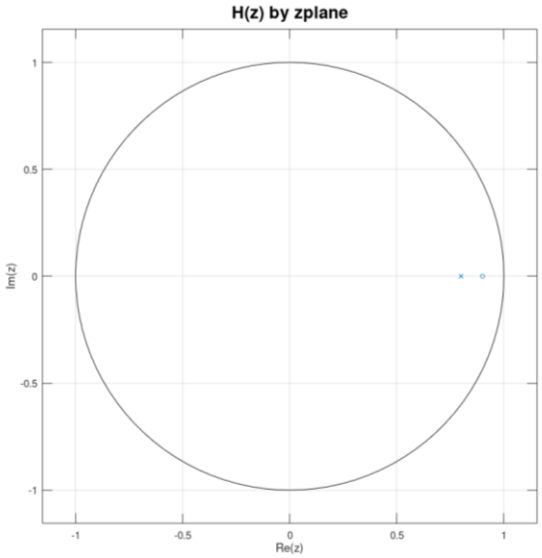
```
Hw = (1+b*exp(-j*w))./(1-a*exp(-j*w));

plot(w*(fsc(3)*Fs)/(2*pi),   abs(Xw3.*Hw'));
```





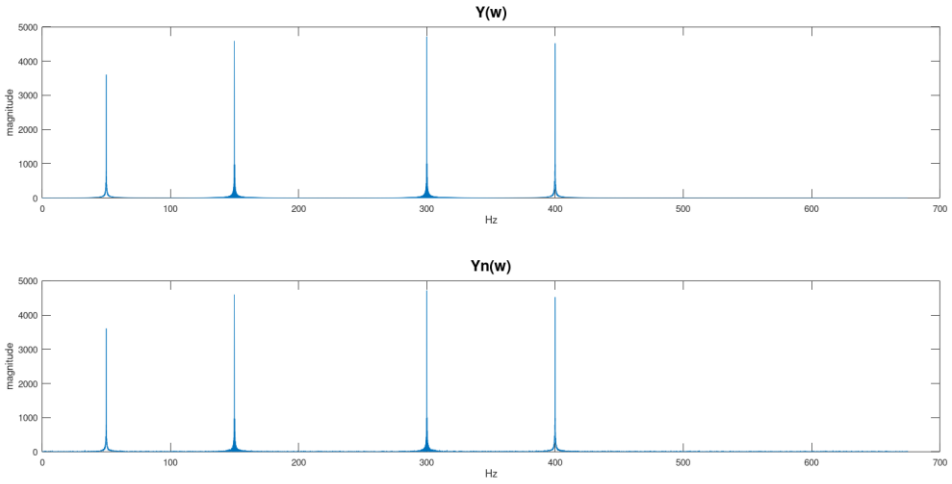
오른쪽은 H(z)의 pole-zero diagram를 그린 것이다.

$H(w) = {1 - 0.9 \times e^{-jw}}/{1 - 0.8 \times e^{-jw}}$ 로 계산되었다. 그래서 ploe이 0.8, zero가 0.9가 나왔으며, 계산 결과와 동일하게 zplane의 pole-zero diagram이 그려진 것을 볼 수 있다.

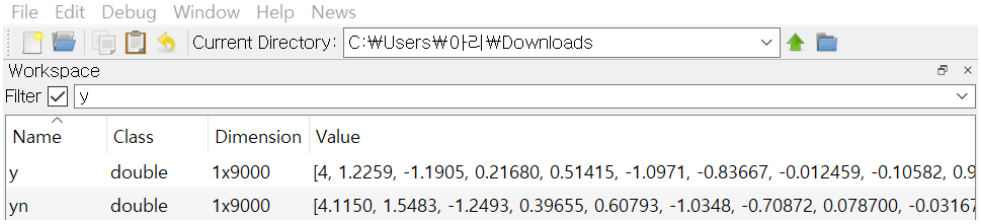


5) noise를 섞은 Yn(w)

SNR을 20으로 설정하여 Yn(n)을 생성하였으며 그 결과는 다음과 같다.

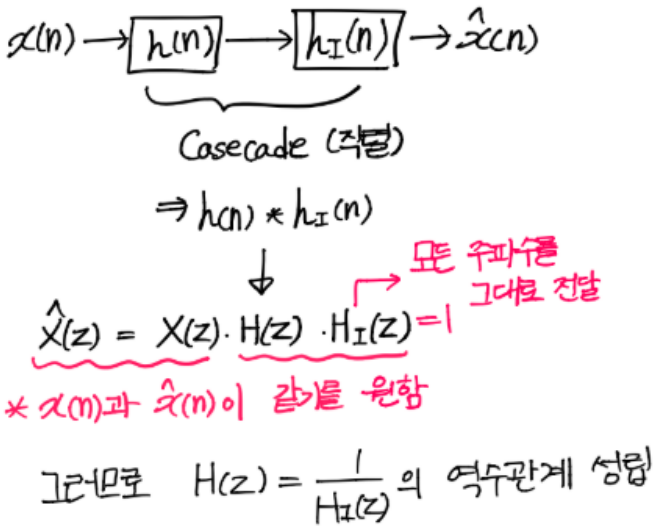


위에서 볼 수 있듯이, Yn(w)의 신호에서는 Y(n)과 달리 밑바닥에 자글자글하게 잡음에 섞인 것을 확인할 수 있다.



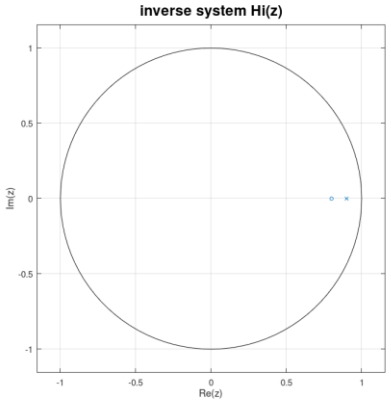
그리고 n단위의 신호에서 확인해 본 결과, 원래의 y신호에서 잡음이 섞여 yn의 값이 y와 달라진 것을 확인할 수 있다.

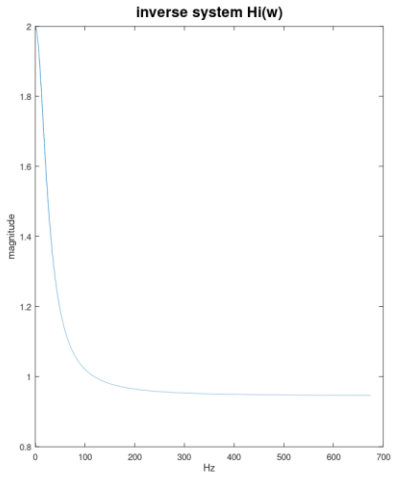
6) Inverse System



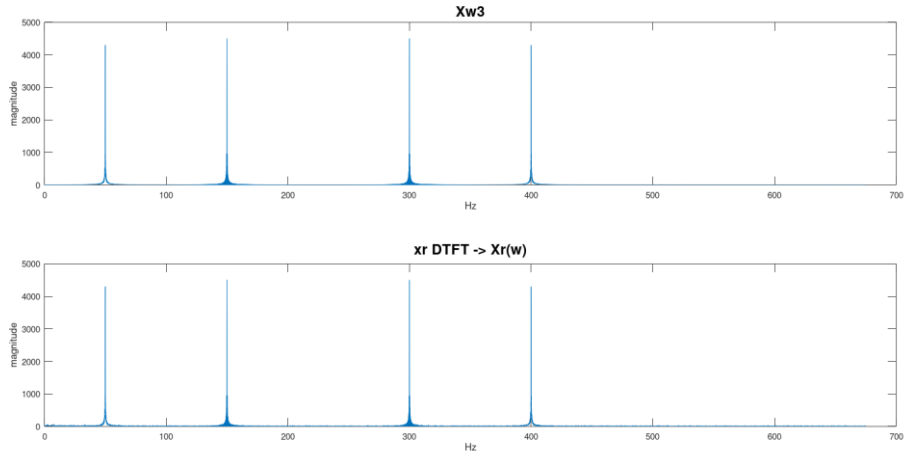
Inverse System은 입력 신호가 채널을 통과하면서 발생한 왜곡을 없애기 위한 것이다. 그러므로 위와 같이, H(z)와 Hi(z)는 물론 H(w)와 Hi(w)도 역수관계가 성립해야 한다.

그래서 처음에 H(w)를 freqz(B,A,K)로 설정하였으므로, Hi(w)를 freqz(A,B,K)로 설계하였다. 역수 관계이므로, pole과 zero가 서로 바뀌어 inverse system에서는 pole이 0.9, zero가 0.8로 나타난 것을 아래의 pole-zero diagram을 통해 확인할 수 있다. 또한, $Hi(w) = {1 - 0.8 \times e^{-jw}}/{1 - 0.9 \times e^{-jw}}$ 에 w=0을 대입했을 때, 0.2/0.1 = 2의 값이 나왔는데, 계산한 것과 동일하게 그래프에 나타났음을 확인할 수 있다.





7) xr(n)에 대한 DTFT spectrum

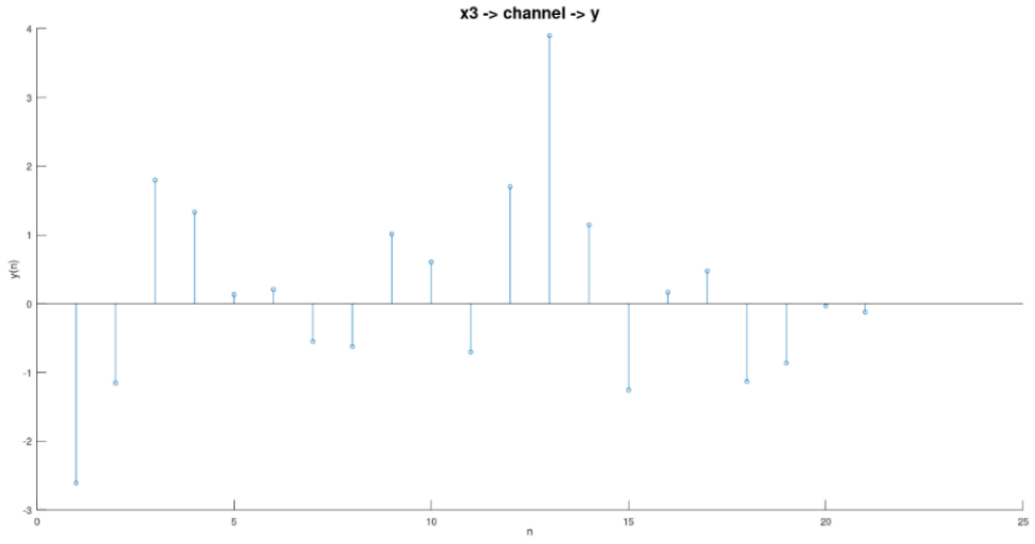


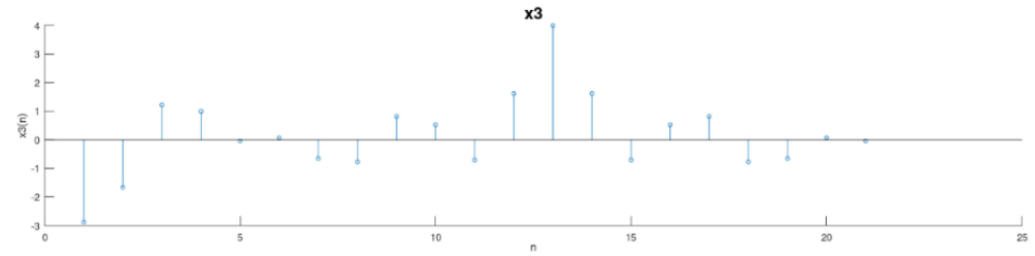
Inverse system에 yn을 넣고 DTFT한 Xr(w)가 원래 입력 신호였던 X3(w)와 동일하게 그려지는 것을 확인할 수 있다.

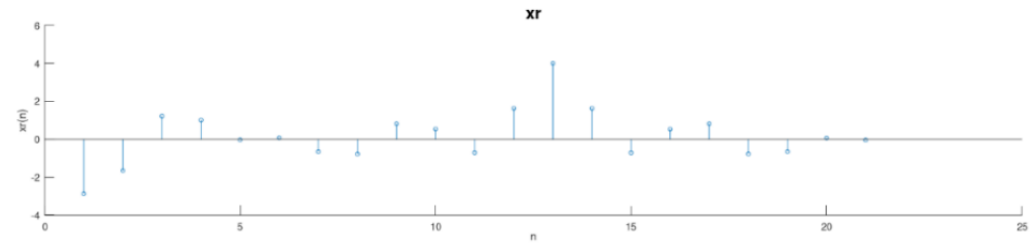
그러나 랜덤으로 noise를 넣었기 때문에 noise를 완전히 제거할 수는 없었다.

5. 추가수행 및 시행착오

1) 신호들의 n단위에서의 그래프를 확인하고 싶어서 y(n)과 x3(n), xr(n)을 stem하여 그래프를 그렸다.

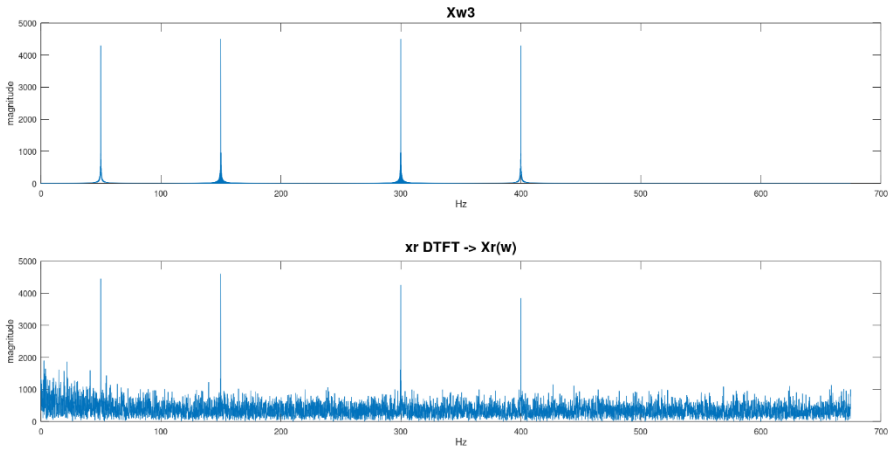






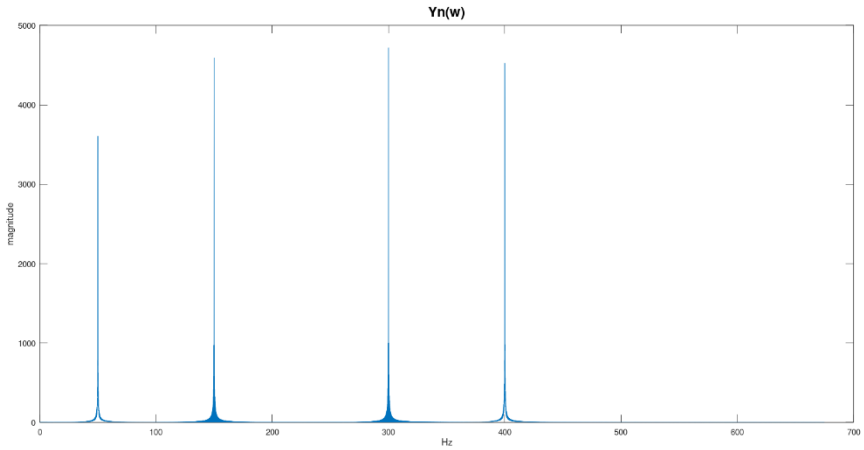
2) SNR_dB = -10으로 설정하여 수행

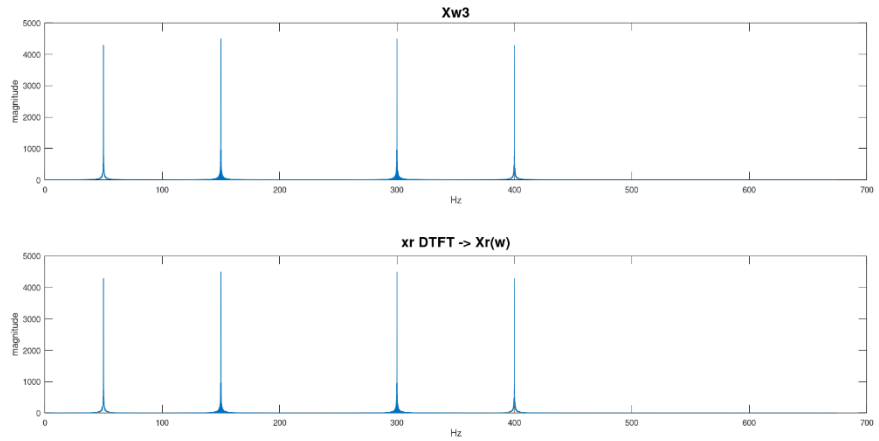
잡음이 너무 커져서 복구가 제대로 되지 않은 것을 볼 수 있었다. 아래의 그래프는 잡음을 더한 뒤 복구하여 DTFT한 Xr(w)와 원본인 X3(w) 신호이다.



3) SNR_dB = 50으로 수행

신호 대 잡음비가 너무 작아져서 잡음을 더한 yn(n)에 큰 변화가 없고, Yn(w)에서도 잡음을 그래프에서 볼 수 없었다. 아래는 잡음을 더하고 DTFT한 Yn(w)와 Yn(w)를 Inverse System에 넣어서 복구한 Xr(w)와 원본 신호인 X3(w)이다.





4) 라이브러리 사용하지 않기

라이브러리를 사용하지 말라는 조건 때문에 X(w)등의 신호를 plot할 때, 많은 고민을 했다. 그래서 abs를 사용하지 않고 다음과 같이 sqrt와 conj를 사용하여 X(w)의 magnitude를 그렸다. 그러나, sqrt와 conj 같은 라이브러리를 사용하지 않고 생성하는 방법은 찾지 못했다.

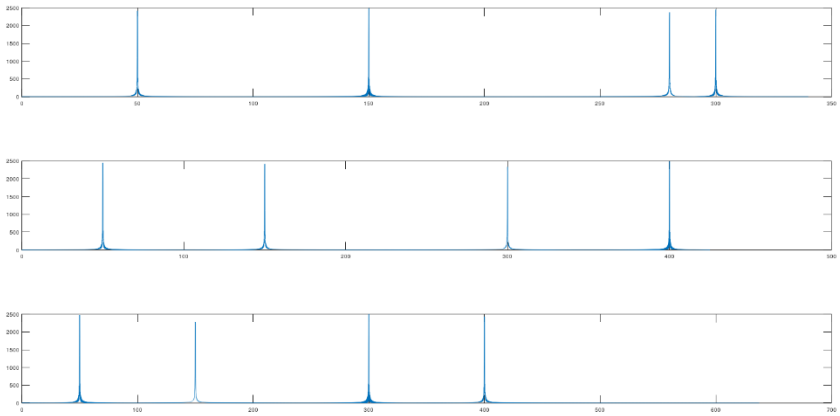
```
plot(w*(fsc(1)*Fs)/(2*pi),sqrt(Xw1.*conj(Xw1)));   %abs(Xw1)
```

5) N 값 변경

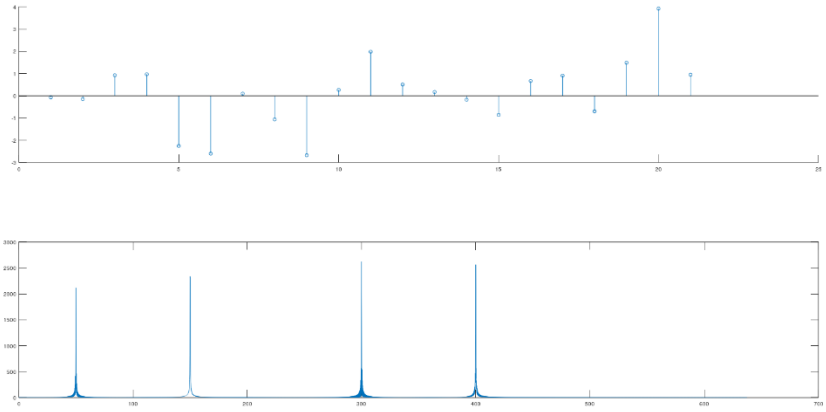
N값을 변경했을 때, 달라지는 점을 확인하기 위해 추가적으로 수행하였다.

- N = 5000

  아래의 첫번째 그래프는 위에서부터 순서대로 X1(w), X2(w), X3(w)이다. 두번째 그래프는 위에서부터 순서대로 y(n)과 Y(w)의 그래프이다.

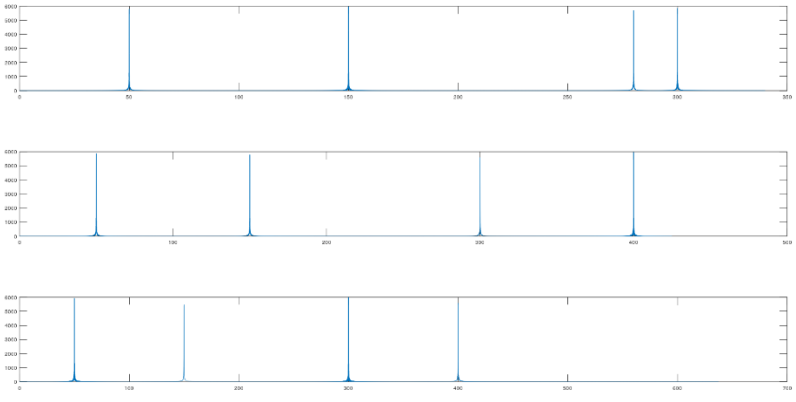


N=5000일 때 X(w)



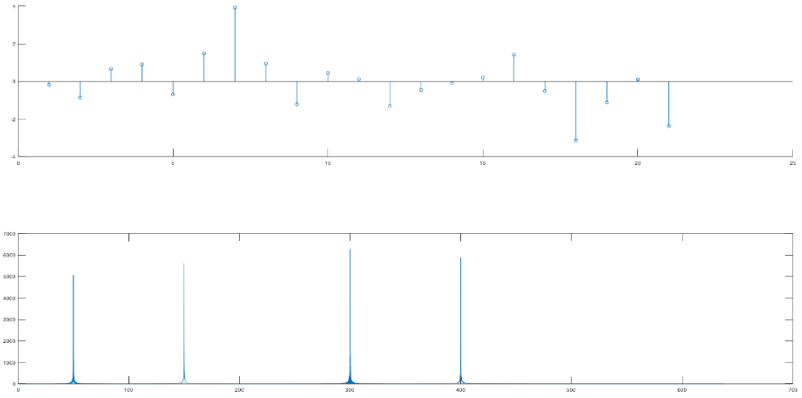
N=5000일 때 y(n)과 Y(w)

- N = 12000



위는 N=12000일 때 위에서부터 순서대로 X1(w), X2(w), X3(w)이다. 아래는 위

에서부터 순서대로 y(n)과 Y(w)의 그래프이다.



- 한 주기당 샘플 수가 달라짐으로써 촘촘한 정도가 달라지고, 그에 따라 y신호는 동일하더라도, end-20부터 end까지 출력되는 부분이 달라짐을 확인할 수 있었다. 또한 N의 값이 커짐에 따라 주파수 영역에서 나타나는 sidelobe가 줄어드는 것을 확인할 수 있었다.

6) 프로젝트 수행 중 시행착오– H(w) by DTFT

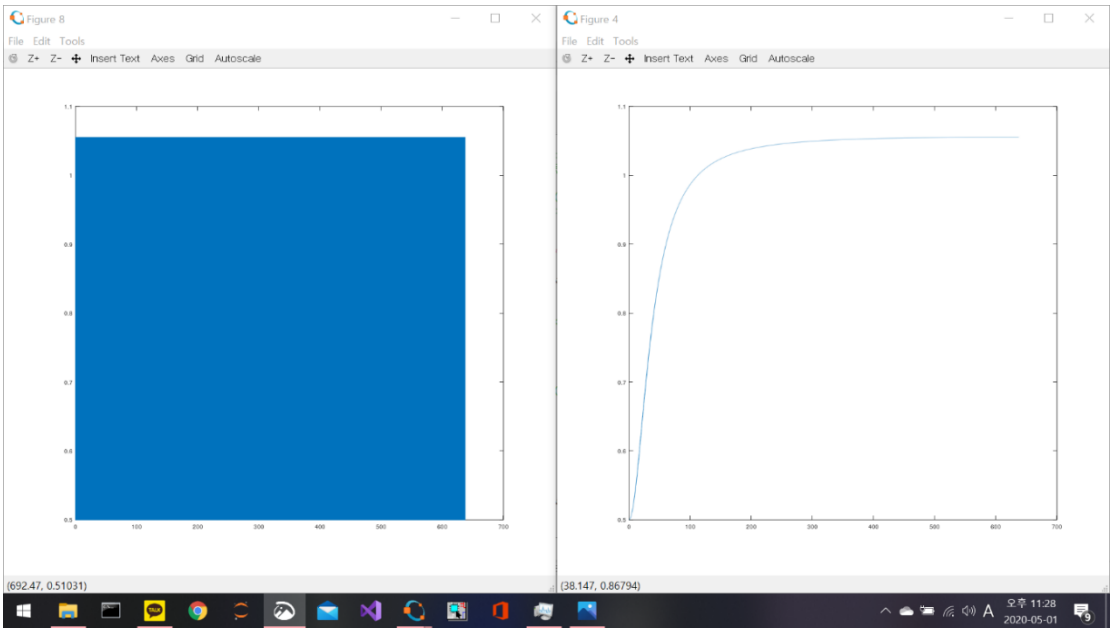
수행절차 7번에서 h(n)을 직접 수식으로 DTFT하여 H(w)를 구할 때, 7번의 의미를 잘못 이해하여, 직접 계산하라는 것이 아니라, h(n)에 *exp(-jwn)을 하여 DTFT한 값을 그리라는 것으로 잘못 이해하였다.

그래서 처음에는 h(n)을 구하려고 했었다. 그 과정에서 위에서 구한 y(n)을 Y(w)이라고 순간적으로 착각하여 h(n) = y(n)/x3(n)이라고 생각했었다. 그러나 x3(n)과 h(n)을 컨볼루션한 것이 y(n)이므로 단순히 나눈다고 해서 h(n)을 구할 수 없다는 것을 알고, 오류를 범했다는 것을 깨달았다.

그러다가 직접 DTFT하여 H(w)를 구한 뒤, 그 수식을 그대로 matlab에 써야한다는 것을 알고 다시 진행하였다. 그런데 n을 전체로 해야한다고 잘못 생각하여

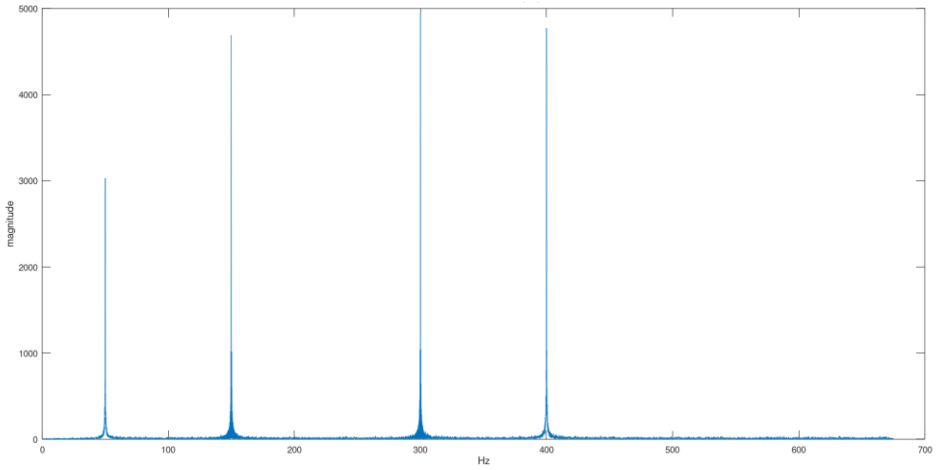
```
for d=0:N
   Hw = (1-b*exp(-j*d*w))./ (1+a*exp(-j*d*w));
end
plot(w*(fsc(3)*Fs)/(2*pi),abs(Hw));
```

위와 같이 입력을 하는 오류를 범했고 아래와 같이 이상한 그래프가 그려졌었다. 그리고 n=0과 n=1일 때만 a,b 값이 있다는 것을 깨닫고 Hw = (1+b*exp(-j*w))./(1-a*exp(-j*w));로 dot operation을 사용한 수식으로 수정하니, freqz에서 나온 high-pass filter 형태가 나왔다.

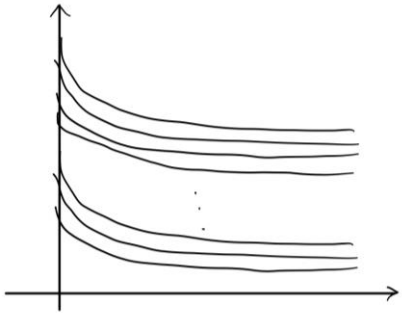


7) filter 없이 xr생성 – 시도했으나 실패

- 필터말고 다른 방법은 없을까 궁금하여 여러가지 방식으로 정말 맨땅에 헤딩을 반복했다. 먼저 filter를 사용하지 않고 앞에서 x3의 신호를 직접 channel에 넣었듯이, yn의 신호를 직접 넣어봤다. 그러나 inverse system을 잘못 구하여 X가 복구되는 것이 아니라 Yn이 그대로 시스템에 들어가서 Yn이 그려졌었다.



- 그러다가 어차피 주파수 영역에서 channel을 통과하는 것은 곱하기인데 Yn(w)와 Hi(w)를 곱해도 나오지 않을까라는 생각이 들어서 수행해보았으나, 각각의 Yn(w) 신호가 low-pass에 곱해진 다음과 같은 형태로 그려졌었다. 실행시키다가 n의 개수가 너무 많아서 다 그릴 수가 없었는지, 그리다가 '응답 없음'이 뜨면서 창이 꺼져서 캡쳐를 못했다.



6. 고찰

이번 프로젝트에서는 채널을 통과한 출력신호 Y에 잡음을 섞고 Inverse System을 설계하여, 채널을 통과하면서 왜곡되고 잡음이 섞인 원래의 신호 X를 복구하는 작업을 진행하였다.

먼저 4개의 tone을 다 더하고 Fs를 이용해, 신호를 Sampling 하여 n단위의 discrete 신호 x를 생성하였다. 그리고 이 신호를 채널에 넣고 y를 생성한 뒤, 잡음을 섞어 yn을 만들었다. 그 뒤, 이 yn을 inverse system에 통과시켜 원래의 신호 x를 복구시키는 작업을 하였다. 또한 프로젝트를 진행하면서 DTFT를 이용하여 n 단위의 x, h, y, yn, xr 신호를 주파수 영역으로 변환하는 과정도 진행하였다.

프로젝트의 결과는 잘 나왔다고 생각한다. 먼저 aliasing을 확인하기 위해 일부러 Fs에 상수배를 한 것도 Sampling Theorem에 따라, 0.8Fs에서만 aliasing이 나타난 것을 볼 수 있었다. Y(w)도 예측했던 것과 유사하게 나왔으며, inverse system에 의해 원본 신호인 x도 잘 복구되었다.

다만, Y(w)의 값과 예측한 것의 값이 아주 미세한 오차가 발생하였는데, 이는 눈대중으로 눈금을 보고 값을 계산하여 오차가 발생한 것이라고 생각한다. 또한 원본 신호를 복구할 때, Xr(w)와 X3(w)가 유사했지만 여전히 완전히 동일하게 복구되지는 않았다. 이는 잡음을 랜덤으로 생성하여 더한 것이기 때문에 완벽하게 없앨 수 없었던 것이라고 생각한다.

프로젝트를 진행하면서 Sampling Theorem, DTFT, Z-TR, Inverse, ROC 등 책으로만 배웠던 주요 개념을 실제로 적용하고 그 결과를 시각적으로 직접 볼 수 있었다. 그래서 이론에 대해 더 쉽게 이해할 수 있었다.

또한 직접 실습함으로써 주요 개념들을 몸소 익히고 체화할 수 있었다. 사실 이론 수업에서 DTFT가 나온 순간부터 이해는커녕 필기하기에 바빴었다. 그렇게 DTFT, Z-Tr 등 여러가지 개념이 서로 섞이고, ROC라는 개념이 새로 등장하면서 어려움을 겪었으며, 쉽사리 책을 펼쳐 파고들 엄두가 나질 않았었다. 그러나 프로젝트를 통해 이론 내용을 공부하고 그 내용을 matlab에 실제로 적용하면서, 책으로만 봐서 잘 와닿지 않아 어려움을 느꼈던 개념들을 이해할 수 있었다.

무엇보다 이론을 바탕으로 matlab 결과를 예측해보고 matlab의 결과와 비교하는 과정을 거치면서 스스로가 어떤 부분이 부족하고, 어떤 개념을 잘 모르는지 파악하여 보완할 수 있었다. 그리고 결과들을 분석하기위해 '이 부분이 왜 이러한 결과가 나왔는가?'에 대

한 물음에 스스로 답을 찾으면서 지금까지 배운 내용을 다시 공부하고 전체적인 개념들을 큰 틀로 정리하는 시간을 가질 수 있었다. 물론 아직 온전하게 이해한 것은 아니지만 프로젝트를 통해 이전보다 나아졌다고 말할 수 있다.

그리고 matlab을 실험시간에 아주 조금 다뤄본 것이 전부였고, 그마저도 진도를 너무 빨리 나가느라 2차례 정도 배운 것이어서 matlab에 대한 걱정이 많았다. 그러나 프로젝트를 진행하면서 matlab을 다루는 방법을 익힐 수 있었다. 특히, edit이라는 창이 있는 줄도 몰라서 지금까지 과제나 퀴즈를 수행할 때 command 창에서 했었는데, edit 창이라는 것을 새롭게 알게 되었다. 정말 matlab에 대한 기본이 전혀 없던 학생이었지만, 프로젝트를 통해 기본적인 사용법을 익히고 matlab을 사용하는 것에 조금 익숙해질 수 있었다. 특히 pkg load signal과 plot, filter 같은 것들은 앞으로도 절대 잊어버리지 않을 것 같다.

7. 참고자료

1. randn

https://kr.mathworks.com/help/matlab/ref/randn.html

2. freqz

https://kr.mathworks.com/help/signal/ref/freqz.html

3. SNR

https://kr.mathworks.com/help/signal/ref/snr.html

4. filter

https://kr.mathworks.com/help/matlab/ref/filter.html