Anomaly Detection (이상치 탐지)

1. 데이터셋: NASA Bearing dataset 4번

(https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/)

(1) 설명:

- 4개의 베어링
- 데이터셋 1: 1개 베어링 당 2개의 엑셀로미터(x,y 축)
- 데이터셋 2,3: 1개의 베어링 당 1개의 엑셀로미터

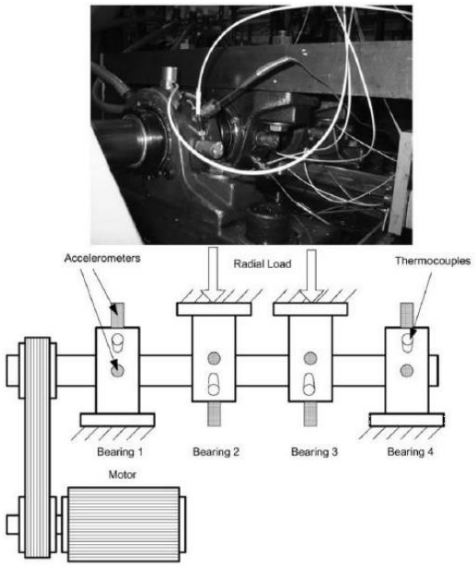


Figure 1 – Bearing test rig and sensor placement illustration [Qiu et al., 2006]

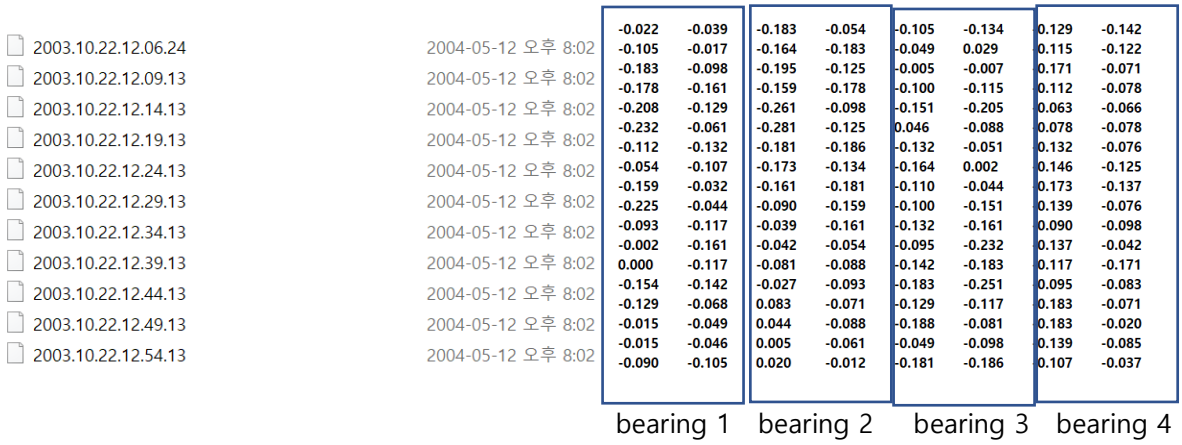
(2) Data Structure:

데이터셋 1:

Set No. 1:

Recording Duration:	October 22, 2003 12:06:24 to November 25, 2003 23:39:56
No. of Files:	2,156
No. of Channels:	8
Channel Arrangement:	Bearing 1 – Ch 1&2; Bearing 2 – Ch 3&4; Bearing 3 – Ch 5&6; Bearing 4 – Ch 7&8.
File Recording Interval:	Every 10 minutes (except the first 43 files were taken every 5 minutes)
File Format:	ASCII
Description:	At the end of the test-to-failure experiment, inner race defect occurred in bearing 3 and roller element defect in bearing 4.

- 03.10.22~03.11.25
- 2156개의 파일
- 8개의 채널(4개의 베어링, 각 베어링 당 2개의 가속도계)
- 베어링 1: 1,2채널 베어링 2: 3,4채널 베어링 3: 5,6채널 베어링 4: 7,8채널
- 10분마다 최신화
- 마지막에는 베어링 3, 베어링 4 가 완전히 고장나는 구조

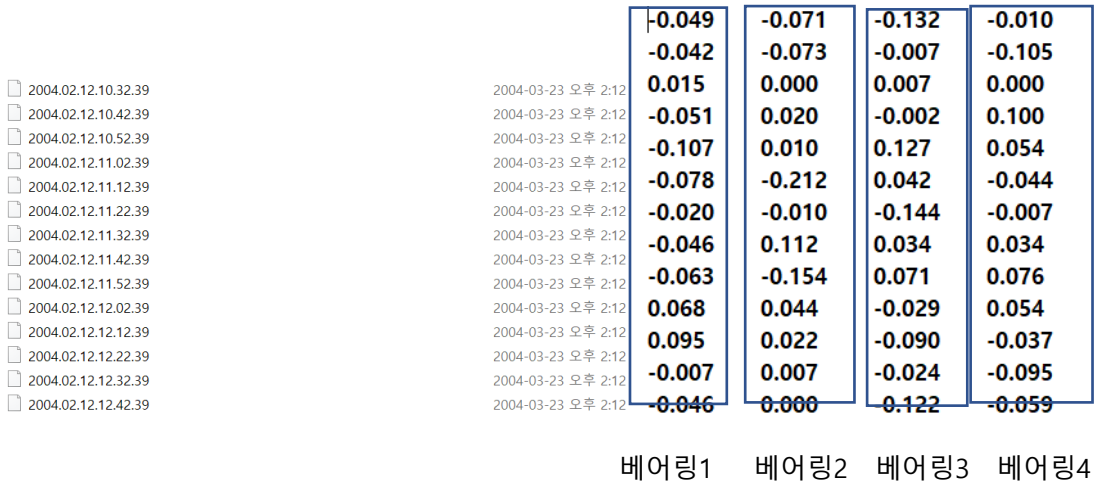


		bearing 1		bearing 2		bearing 3		bearing 4	
2003.10.22.12.06.24	2004-05-12 오후 8:02	-0.022	-0.039	-0.183	-0.054	-0.105	-0.134	0.129	-0.142
2003.10.22.12.09.13	2004-05-12 오후 8:02	-0.105	-0.017	-0.164	-0.183	-0.049	0.029	0.115	-0.122
2003.10.22.12.14.13	2004-05-12 오후 8:02	-0.183	-0.098	-0.195	-0.125	-0.005	-0.007	0.171	-0.071
2003.10.22.12.19.13	2004-05-12 오후 8:02	-0.178	-0.161	-0.159	-0.178	-0.100	-0.115	0.112	-0.078
2003.10.22.12.24.13	2004-05-12 오후 8:02	-0.208	-0.129	-0.261	-0.098	-0.151	-0.205	0.063	-0.066
2003.10.22.12.29.13	2004-05-12 오후 8:02	-0.232	-0.061	-0.281	-0.125	0.046	-0.088	0.078	-0.078
2003.10.22.12.34.13	2004-05-12 오후 8:02	-0.112	-0.132	-0.181	-0.186	-0.132	-0.051	0.132	-0.076
2003.10.22.12.39.13	2004-05-12 오후 8:02	-0.054	-0.107	-0.173	-0.134	-0.164	0.002	0.146	-0.125
2003.10.22.12.44.13	2004-05-12 오후 8:02	-0.159	-0.032	-0.161	-0.181	-0.110	-0.044	0.173	-0.137
2003.10.22.12.49.13	2004-05-12 오후 8:02	-0.225	-0.044	-0.090	-0.159	-0.100	-0.151	0.139	-0.076
2003.10.22.12.54.13	2004-05-12 오후 8:02	-0.093	-0.117	-0.039	-0.161	-0.132	-0.161	0.090	-0.098
		-0.002	-0.161	-0.042	-0.054	-0.095	-0.232	0.137	-0.042
		0.000	-0.117	-0.081	-0.088	-0.142	-0.183	0.117	-0.171
		-0.154	-0.142	-0.027	-0.093	-0.183	-0.251	0.095	-0.083
		-0.129	-0.068	0.083	-0.071	-0.129	-0.117	0.183	-0.071
		-0.015	-0.049	0.044	-0.088	-0.188	-0.081	0.183	-0.020
		-0.015	-0.046	0.005	-0.061	-0.049	-0.098	0.139	-0.085
		-0.090	-0.105	0.020	-0.012	-0.181	-0.186	0.107	-0.037

데이터셋 2:

- 04.02.12~04.02.19
- 984개 파일
- 4개 채널(4개의 베어링, 각 1개의 가속도계)
- 베어링 1: 채널 1, 베어링 2: 채널 2, 베어링 3: 채널 3, 베어링 4: 채널 4

- 10분마다 기록
- 결국 베어링 1에서 완전히 고장나는 데이터셋



(3) 데이터 시각화:

데이터셋 1:

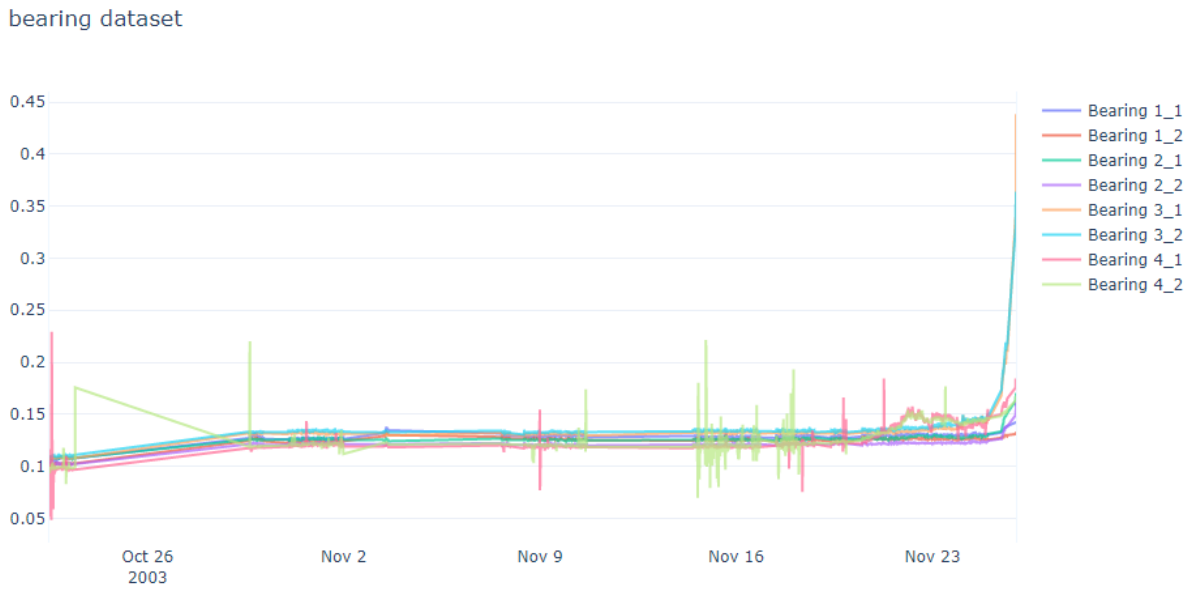
```
data_dir = "C:₩₩Users₩₩정해웅₩₩OneDrive₩₩Desktop₩₩SKTAIfellowship₩₩Bearing Dataset₩₩IMS₩₩1st_test₩₩1st_test"
data = pd.DataFrame()

for filename in os.listdir(data_dir):
    dataset = pd.read_csv(os.path.join(data_dir, filename), sep='₩t')
    dataset_mean_abs = np.array(dataset.abs().mean())
    dataset_mean_abs = pd.DataFrame(dataset_mean_abs.reshape(1,8))
    dataset_mean_abs.index = [filename]
    data = data.append(dataset_mean_abs)

data.columns = ['Bearing_1_1','Bearing_1_2', 'Bearing_2_1','Bearing_2_2', 'Bearing_3_1','Bearing_3_2', 'Bearing_4_1','Bearing_4_2']
```

```
data
```

	Bearing_1_1	Bearing_1_2	Bearing_2_1	Bearing_2_2	Bearing_3_1	Bearing_3_2	Bearing_4_1	Bearing_4_2
2003.10.22.12.06.24	0.104148	0.100253	0.107147	0.102004	0.106149	0.108150	0.094803	0.099513
2003.10.22.12.09.13	0.103651	0.099854	0.108189	0.102920	0.106661	0.108458	0.095070	0.093587
2003.10.22.12.14.13	0.105039	0.101543	0.108543	0.104042	0.108740	0.109875	0.096158	0.098299
2003.10.22.12.19.13	0.104900	0.101573	0.108152	0.103378	0.108068	0.110010	0.096814	0.098602
2003.10.22.12.24.13	0.104779	0.102181	0.107943	0.102629	0.108454	0.109350	0.096358	0.098471
...	...	...	...	...	...	...	...	...
2003.11.25.16.07.32	0.138784	0.130001	0.151358	0.138561	0.209843	0.217346	0.165271	0.154710
2003.11.25.23.13.21	0.142147	0.131042	0.162772	0.149214	0.339327	0.330122	0.175640	0.163224
2003.11.25.23.19.56	0.142098	0.131348	0.167258	0.154300	0.357823	0.351640	0.184356	0.165720
2003.11.25.23.29.56	0.141889	0.132988	0.158432	0.153226	0.331172	0.335432	0.174025	0.161537
2003.11.25.23.39.56	0.142014	0.131698	0.170618	0.162325	0.438427	0.364037	0.181261	0.168055



데이터셋 2:

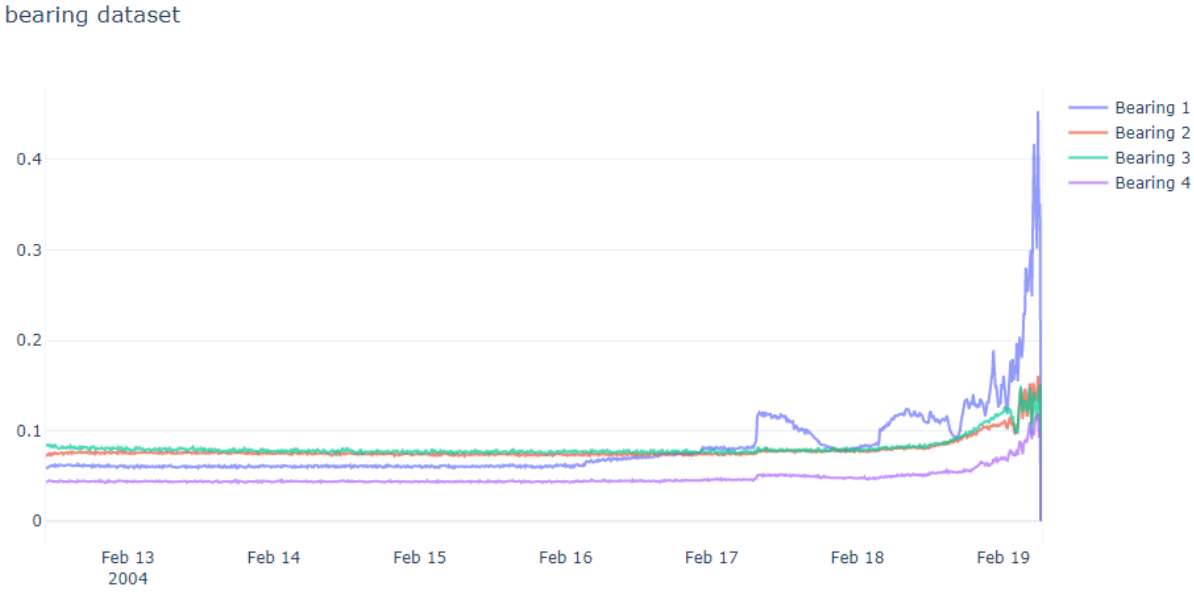
```
data_dir = "C:₩₩Users₩₩정해웅₩₩OneDrive₩₩Desktop₩₩SKTAIfellowship₩₩Bearing Dataset₩₩IMS₩₩2nd_test₩₩2nd_test"
data = pd.DataFrame()

for filename in os.listdir(data_dir):
    dataset = pd.read_csv(os.path.join(data_dir, filename), sep='₩t')
    dataset_mean_abs = np.array(dataset.abs().mean())
    dataset_mean_abs = pd.DataFrame(dataset_mean_abs.reshape(1,4))
    dataset_mean_abs.index = [filename]
    data = data.append(dataset_mean_abs)

data.columns = ['Bearing_1', 'Bearing_2', 'Bearing_3', 'Bearing_4']
```

```
data
```

	Bearing_1	Bearing_2	Bearing_3	Bearing_4
2004.02.12.10.32.39	0.058333	0.071832	0.083242	0.043067
2004.02.12.10.42.39	0.058995	0.074006	0.084435	0.044541
2004.02.12.10.52.39	0.060236	0.074227	0.083926	0.044443
2004.02.12.11.02.39	0.061455	0.073844	0.084457	0.045081
2004.02.12.11.12.39	0.061361	0.075609	0.082837	0.045118
...	...	...	...	...
2004.02.19.05.42.39	0.453335	0.161016	0.137440	0.119047
2004.02.19.05.52.39	0.337583	0.132400	0.144992	0.092125
2004.02.19.06.02.39	0.351111	0.152266	0.151299	0.100817
2004.02.19.06.12.39	0.001857	0.003732	0.003656	0.001786
2004.02.19.06.22.39	0.001168	0.000767	0.000716	0.001699


bearing dataset
Bearing 1
Bearing 2
Bearing 3
Bearing 4
0.4
0.3
0.2
0.1
0
Feb 13
2004
Feb 14
Feb 15
Feb 16
Feb 17
Feb 18
Feb 19

2. 데이터 전처리 1: 주파수 domain으로 변환

(1) 주파수 domain으로 변환

- 시간 t축에서 존재하는 신호를 주파수 f(or w) 축으로 옮기는 것

- 대부분의 신호는 진폭과 위상이 다른 여러 개의 sin 곡선의 합으로 이루어짐

- 신호를 sin 곡선으로 바꾸는 이유는 LTI 적용 시 모양이 변하지 않는 유일한 파형이라서

- 시계열 데이터를 주파수 영역으로 바꿈으로써 제어 시스템을 디자인

- 주파수로 변환 시 데이터의 양 줄어듬

(2) FFT vs DWT

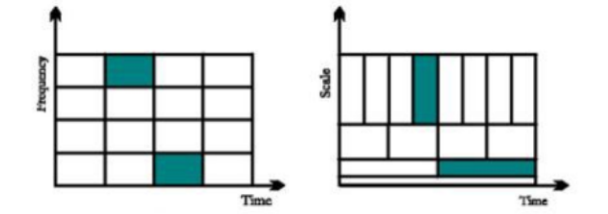
- 대표적인 주파수 변환 기법(Fast Fourier Transform vs Discrete Wavelet Transform)

	Fourier Transform	Wavelet
필요성	Time -> Frequency 통한 신호 분석 용이함	
차이점	시간 영역에 대한 정보 확인 어려움	시간 영역 정보 포함
	고정 크기 Window 사용	고/저주파에 가변적 Window 사용 ->시간/주파수 해상도 정확히 검출
	모터 가동 시 운전 전류만 취득 -> 모터 극수와 에어덕트 개수 같을 시 오진단 -> 고장 아닌데 공장 중단	운전 전류/기동전류 모두 취득 -> 전류 피크/극수 개수 양호하다고 판단 -> 정확 진단
단점	Wavelet에 비해 떨어지는 성능	고주파 신호 영역 분해 불가

Fourier Transform
FT은 시간 영역에 대한 신호를 주파수 영역의 성분으로 분해하는 변환 방식입니다. 이 변환 방식은 시간 영역에 대한 정보의 확인이 어렵다는 단점을 갖고 있습니다.

STFT
STFT은 FT의 시간 영역에 대한 정보 확인이 어렵다는 단점을 보완하며, 고정 크기의 Window를 사용하여서 시간 해상도와 주파수 해상도를 최적화하는 신호 변환 방식입니다. 이 STFT은 Non-Stationary 신호에 대한 검출이 어렵다는 단점을 갖고 있습니다. Non-Stationary 신호는 시간별로 주파수 대역의 신호가 달라진다는 것을 나타냅니다.

Wavelet Transform
FT과 STFT 각각의 단점을 보완하는 방식이 WT입니다. WT과 STFT의 가장 큰 차이점은 STFT에서는 고정 크기의 Window를 사용하였다면, WT에서는 가변적인 크기의 Window를 이용하며, 고주파와 저주파에 대하여 Window 크기를 달리 두어 신호를 검출한다는 것이 특징입니다. WT은 가변적인 Window 크기를 두는 방식을 통하여 고주파와 저주파에 대한 시간 해상도와 주파수 해상도를 STFT에 비하여 상대적으로 정확하게 검출할 수 있습니다.

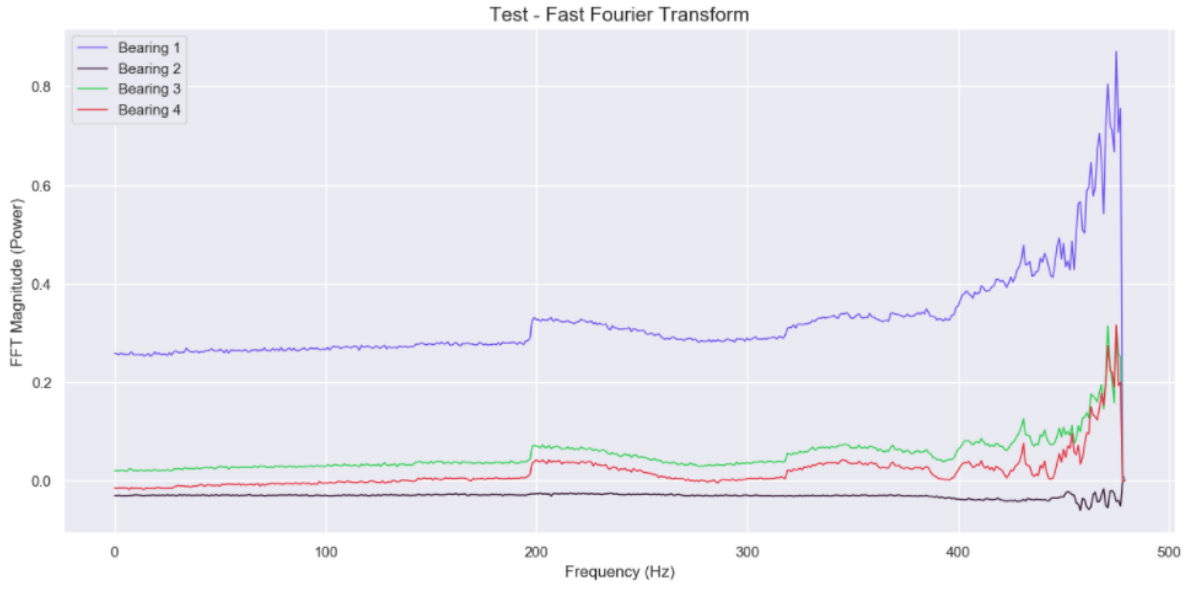


Short-time Fourier transform Discrete wavelet transform

(3) 데이터셋 2 FFT 적용

```
train_fft = np.fft.fft(train)
test_fft = np.fft.fft(test)
```

```
fig, ax = plt.subplots(figsize=(15, 7), dpi=100)
ax.plot(test_fft[:,0].real, label='Bearing 1', color='#6949ed', animated = True, linewidth=1)
ax.plot(test_fft[:,1].imag, label='Bearing 2', color='#250525', animated = True, linewidth=1)
ax.plot(test_fft[:,2].real, label='Bearing 3', color='#25cd49', animated = True, linewidth=1)
ax.plot(test_fft[:,3].real, label='Bearing 4', color='#e42535', animated = True, linewidth=1)
plt.legend()
plt.xlabel('Frequency (Hz)')
plt.ylabel('FFT Magnitude (Power)')
ax.set_title('Test - Fast Fourier Transform', fontsize=15)
plt.show()
```



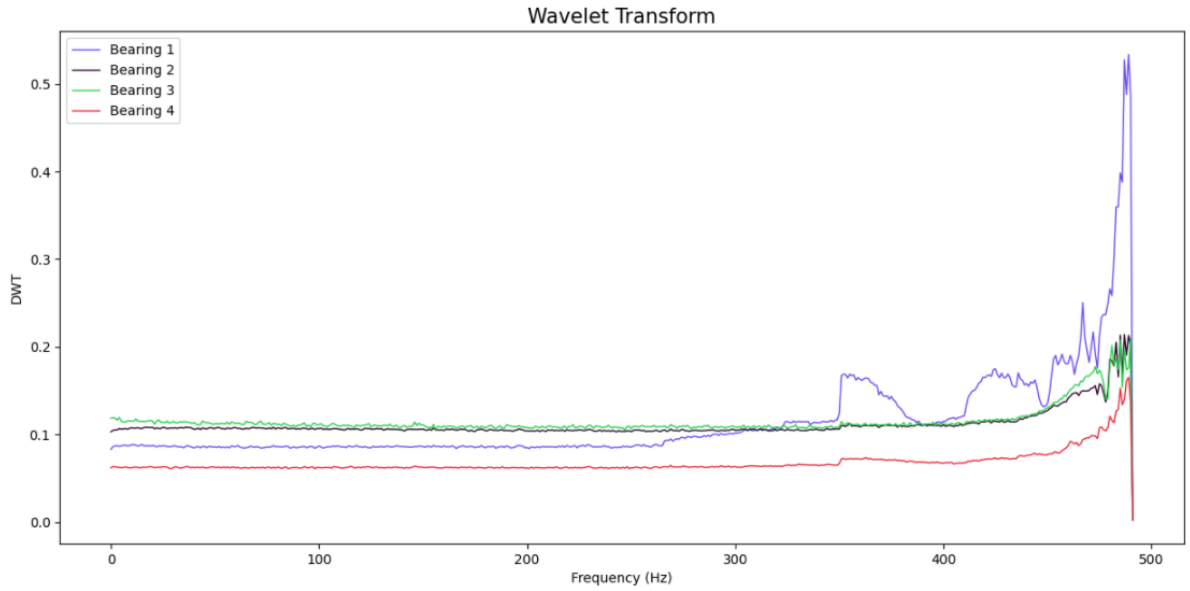
(4) 데이터셋 2 DWT 적용

```
import pywt
import matplotlib.pyplot as plt



a1,b1 = pywt.dwt(temp['Bearing 1'],'db1')
a2,b2 = pywt.dwt(temp['Bearing 2'],'db1')
a3,b3 = pywt.dwt(temp['Bearing 3'],'db1')
a4,b4 = pywt.dwt(temp['Bearing 4'],'db1')
```

```
fig, ax = plt.subplots(figsize=(15, 7), dpi=100)
ax.plot(a1, label='Bearing 1', color='#6949ed', animated = True, linewidth=1)
ax.plot(a2, label='Bearing 2', color='#250525', animated = True, linewidth=1)
ax.plot(a3, label='Bearing 3', color='#25cd49', animated = True, linewidth=1)
ax.plot(a4, label='Bearing 4', color='#e42535', animated = True, linewidth=1)
plt.legend()
plt.xlabel('Frequency (Hz)')
plt.ylabel('DWT')
ax.set_title('Wavelet Transform', fontsize=15)
plt.show()
```



3. 데이터 전처리 2: LPF 적용

(1) Low Pass Filter 필요성

- 원하는 주파수 대역의 신호만 통과시키기 위해

- 고주파 Noise 없애기 위해

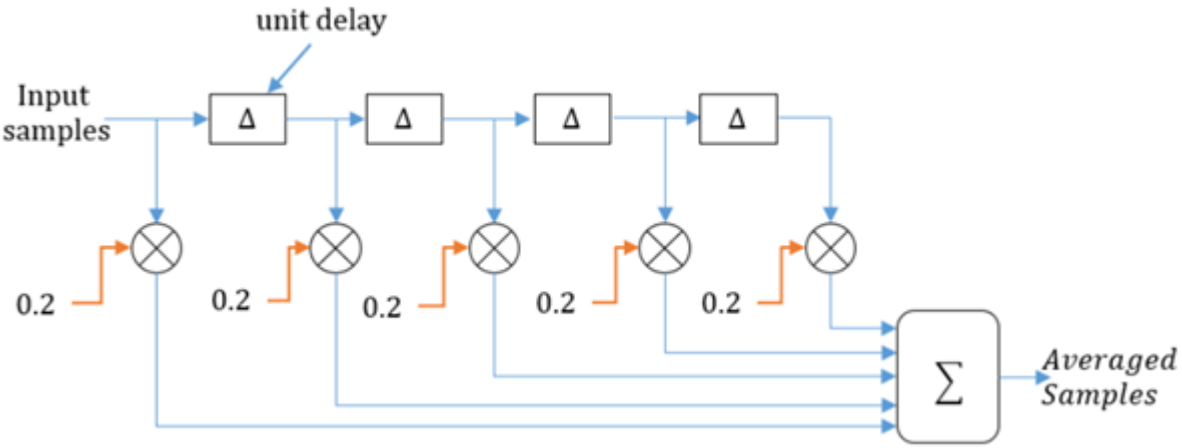
- 전체적인 진동 효과 완화 -> smoothing

(2) MA Filter

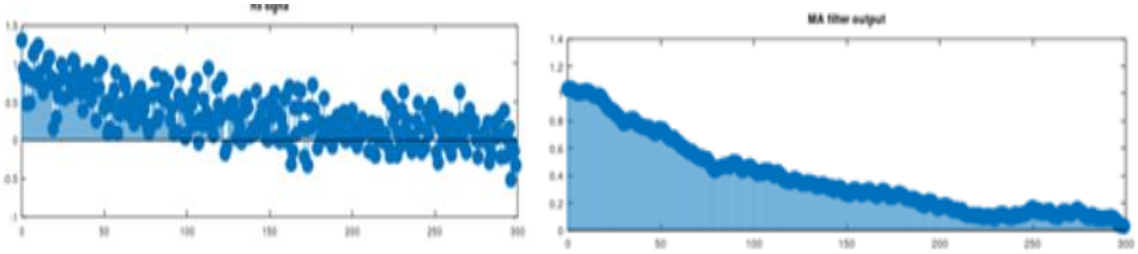
- MA(Moving Average) 필터는 간단한 Low Pass FIR(Finiter Impulse Response) 필터로써 샘플화된 데이터나 시그널의 배열을 smoothing하게 하기 위해 주로 쓰이는 필터

- 시간에 대한 L개의 입력 샘플과 이들의 평균값을 가지고, 하나의 출력 값을 가진다. 이는 매우 간단한 LPF(Low Pass Filter) 구조

- 필터의 길이(L)이 늘어남에 따라 출력값의 smoothness도 증가한다. 반면, 데이터의 sharp transition은 무뎌지게 된다. 이것은 이 필터가 시간영역에서는 훌륭한 반응을 보이지만 주파수 영역에서는 poor response를 보임



- 적용 예시:



기존 잡음 섞인 신호 **MA 필터 적용 시**

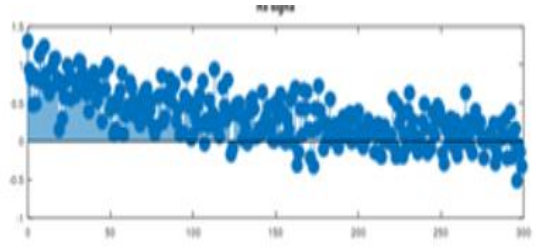
(3) AR Filter

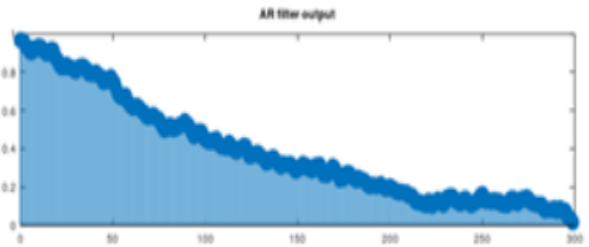
- AR(Autoregressive) 필터는 랜덤 프로세스의 대표 중 하나

- 이는 출력이 바로 전 값에 linearly 의존한다는 것을 특정

- 따라서 모델은 차분 방정식의 형태

- MA 모델과 합쳐 ARMA나 ARIMA와 같은 모델도 존재

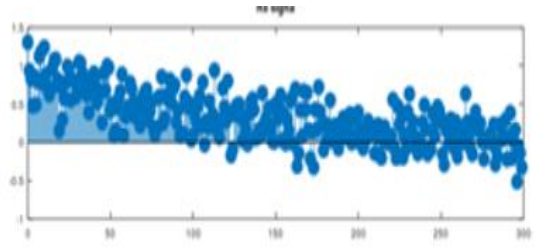


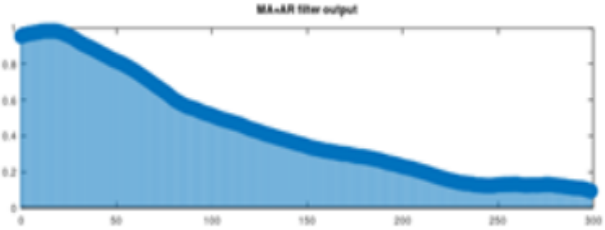


기존 잡음 신호 AR 필터 적용 시

(3) ARMA Filter

- AR+MA 동시 적용한 모델





기존 잡음 신호 ARMA 필터 적용 시