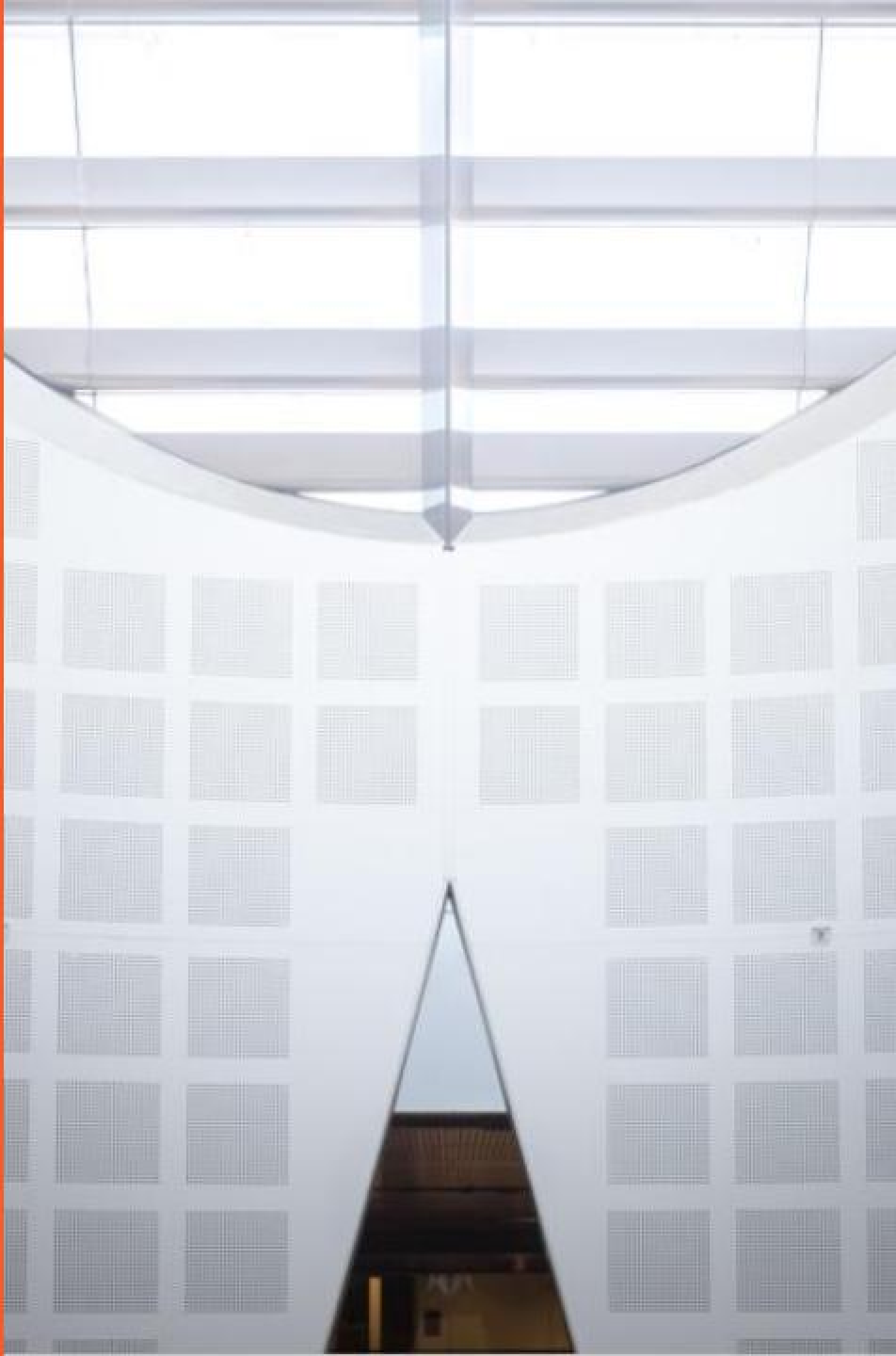
Biomedical Signal Processing Package Usage

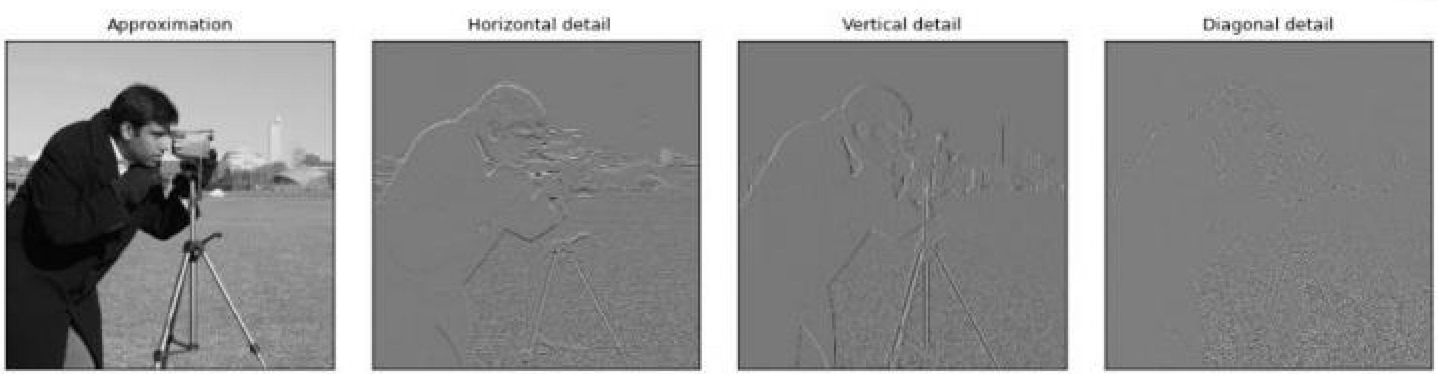
Reference: Healthcare Data Analytics

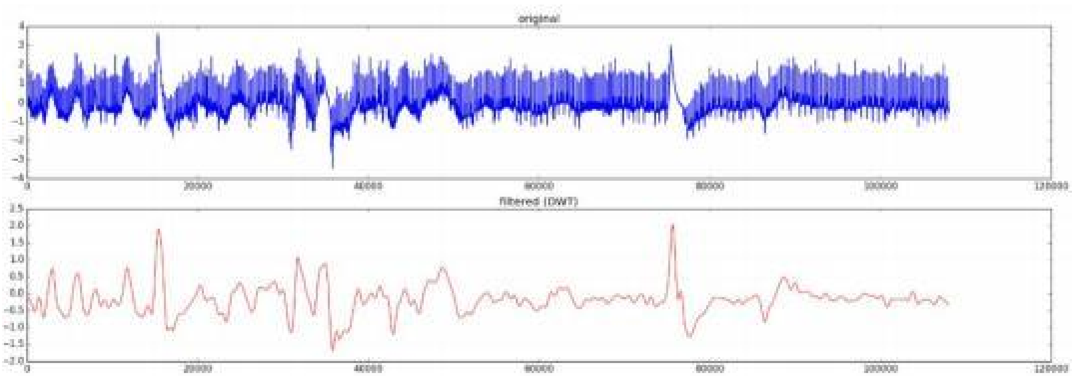


Discrete Wavelet Transform (DWT) for Filtering

Discrete Wavelet Transform for Filtering

- PyWavelets (`pywt`) is a python package for performing Wavelet Transforms
  - The documentations can be found at [PyWavelets - Wavelet Transforms in Python — PyWavelets Documentation]()
  - With the package, various 1-D, 2-D DWT can be performed





Discrete Wavelet Transform for Filtering

- PyWavelets (pywt) is a python package for Wavelet Transforms
- The code to import PyWavelets package in python is

```
import pywt
```

- The two key functions in the package are
  - pywt .wavedec
    - For performing the 1-D DWT Transform
  - pywt .waverec
    - For performing the 1-D I-DWT Transform

Discrete Wavelet Transform for Filtering

pywt.**wavedec**(*data, wavelet, mode='symmetric', level=None, axis=-1*)

Multilevel 1D Discrete Wavelet Transform of data.

Parameters:

data: array_like

Input data

- Numpy array
  - Import numpy as np
  - Array=np.asarray(python_list)

wavelet : *Wavelet object or name string*

Wavelet to use

mode : *str, optional*

Signal extension mode, see Modes.

level : *int, optional*

Decomposition level (must be >= 0). If level is None (default) then it will be calculated using the `dwt_max_level` function.

axis: int, optional

Axis over which to compute the DWT. If not given, the last axis is used.

Returns:

[cA_n, cD_n, cD_n-1, ..., cD2, cD1] : *list*

Ordered list of coefficients arrays where `n` denotes the level of decomposition. The first element (`cA_n`) of the result is approximation coefficients array and the following elements (`cD_n` - `cD_1`) are details coefficients arrays.

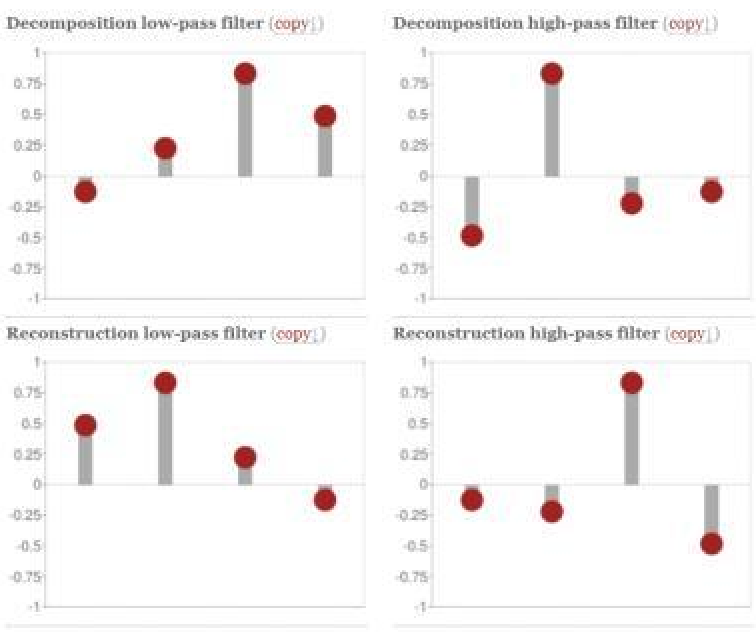
Discrete Wavelet Transform for Filtering

wavelet : *Wavelet object or name string*

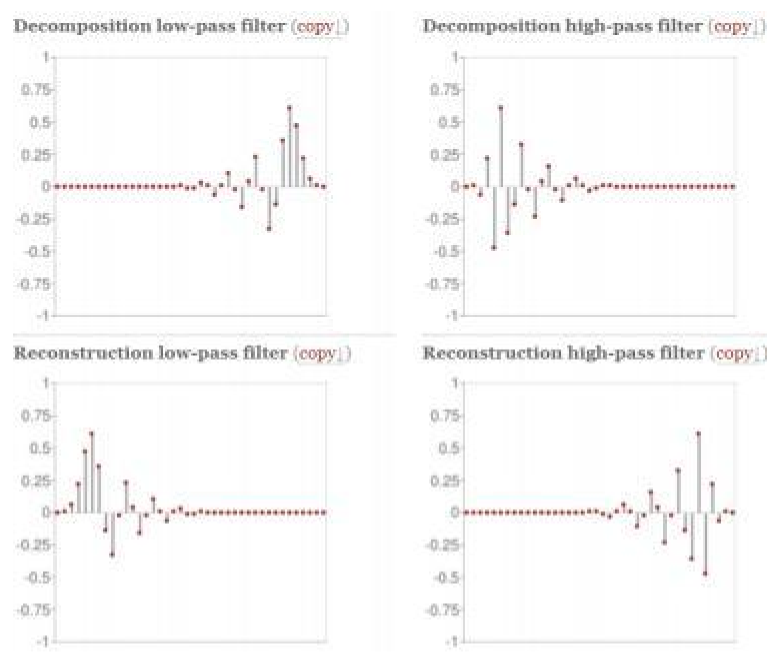
Wavelet to use

Wavelet filter type includes: ‘db1’,’db2’,’db3’··· ‘db20’
Others can be seen by:

```
print(pywt.wavelist())
```



Daubechies 2 filter values (db2)



Daubechies 20 filter values (db20)

Discrete Wavelet Transform for Filtering

mode : *str, optional*

Signal extension mode, see Modes.

- Padding types for the input data
  - ['zero', 'constant', 'symmetric', 'periodic', 'smooth' , 'periodization', 'reflect', 'antisymmetric', 'antireflect']

- `zero` - **zero-padding** - signal is extended by adding zero samples:

```
... 0  0 | x1 x2 ... xn | 0  0 ...
```

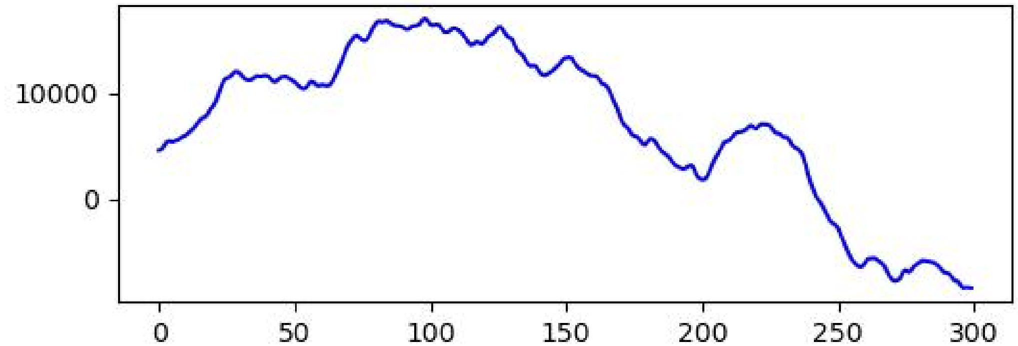
- `constant` - **constant-padding** - border values are replicated:

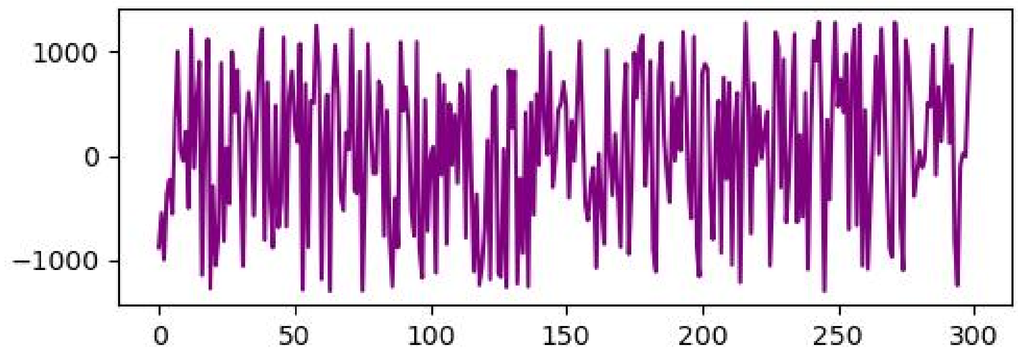
```
... x1 x1 | x1 x2 ... xn | xn xn ...
```

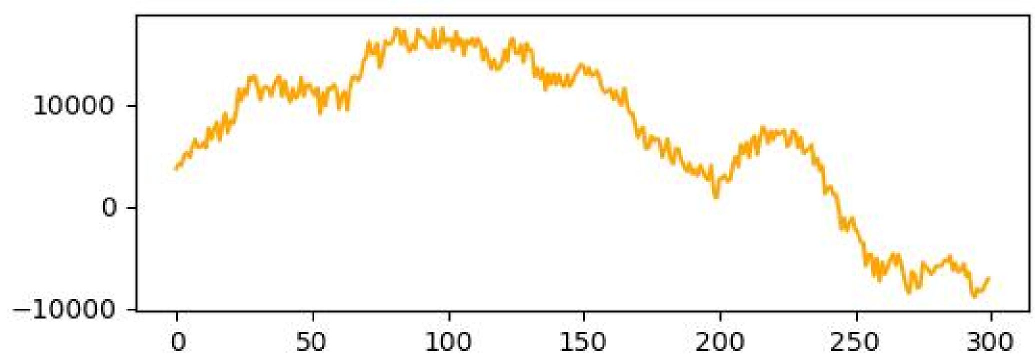
- `symmetric` - **symmetric-padding** - signal is extended by *mirroring* samples. This mode is also known as half-sample symmetric.:

```
... x2 x1 | x1 x2 ... xn | xn xn-1 ...
```

Discrete Wavelet Transform for Filtering



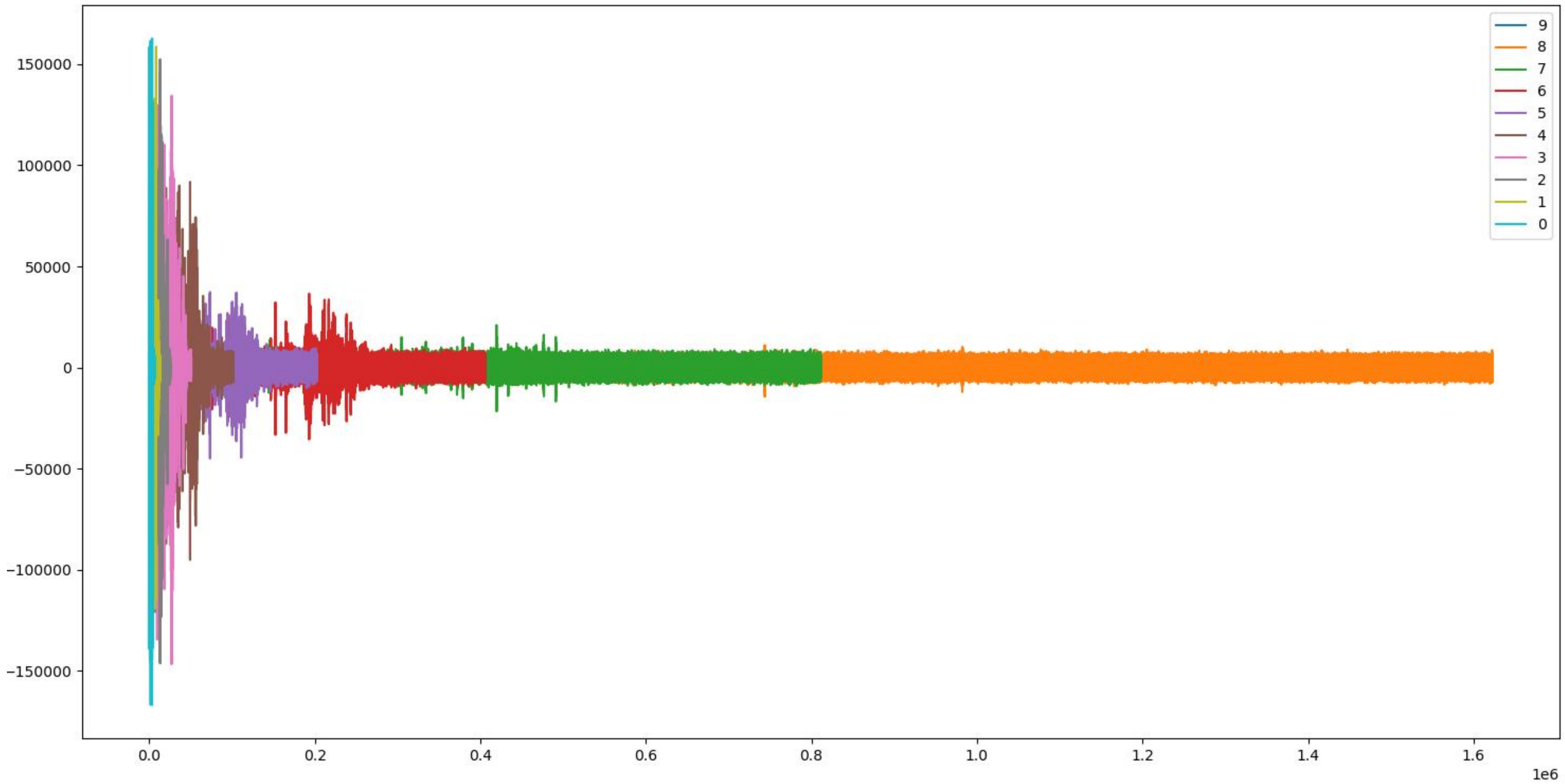




Discrete Wavelet Transform for Filtering

- By running the following code, DWT is performed

```
DWTcoeffs = pywt .wavedec (data,wavelet_type,mode=mode
 _used, level=9, axis=-1)
```

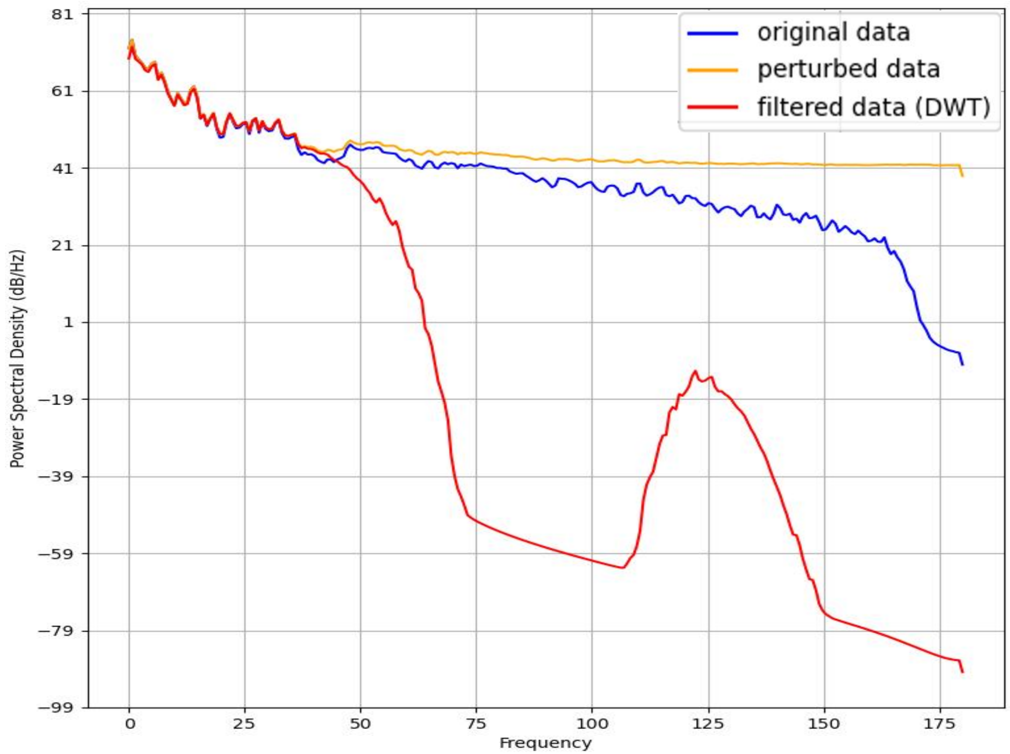


Discrete Wavelet Transform for Filtering

- To perform thresholding

```
for i in range(4):
    DWTcoeffs [-i] = np.zeros_like (DWTcoeffs [-i])
```

  - This sets the last 4 coefficient to zero



Discrete Wavelet Transform for Filtering

pywt.**waverec**(*coeffs*, *wavelet*, *mode='symmetric'*, *axis=-1*)

Multilevel 1D Inverse Discrete Wavelet Transform.

Parameters: **coeffs** : *array_like*

Coefficients list [cAn, cDn, cDn-1, …, cD2, cD1]

wavelet : *Wavelet object or name string*

Wavelet to use

mode : *str, optional*

Signal extension mode, see Modes.

axis: int, optional

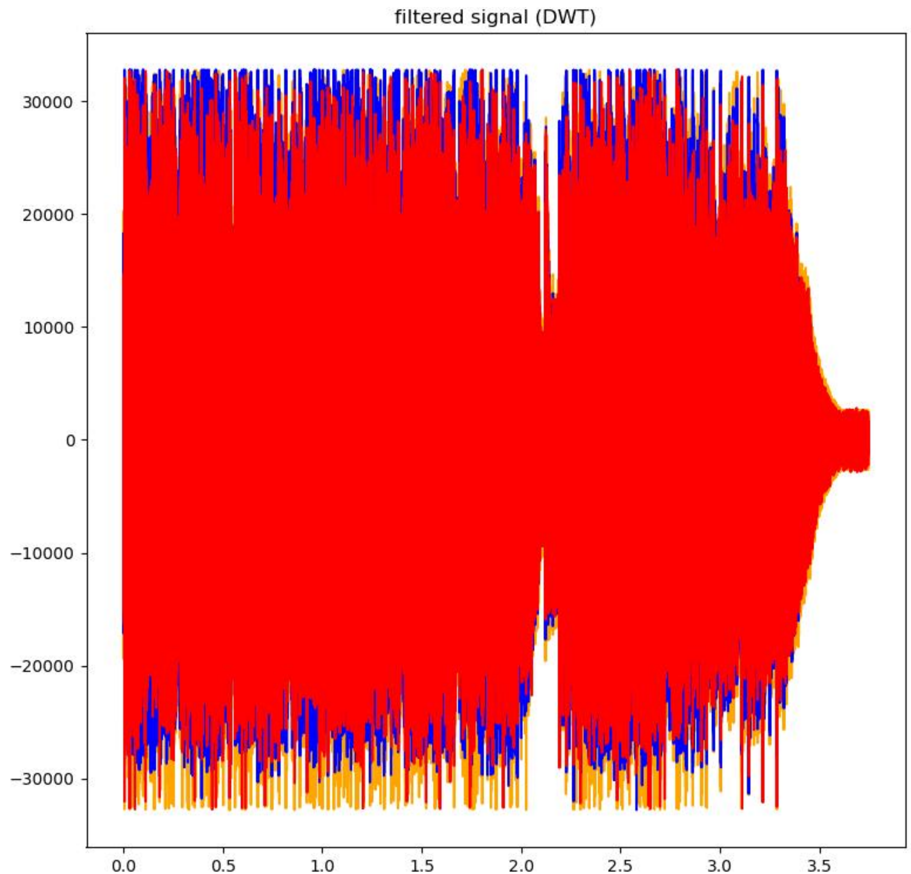
Axis over which to compute the inverse DWT. If not given, the last axis is used.

Same as the pywt.wavedec

Discrete Wavelet Transform for Filtering

- To perform the I -DWT:

```
filtered_data_dwt=pywt .waverec (DWTcoeffs,wavelet_typ
 e,mode=mode_used,axis=-1)
```

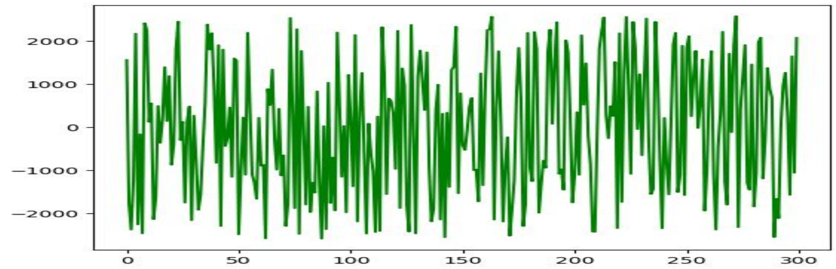


Discrete Wavelet Transform for Filtering

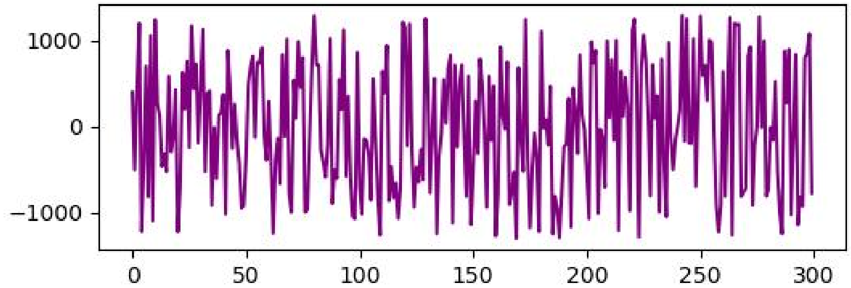
- To calculate the separated noise:

```
separated noise = data - filtered_data_dwt
```

Separated noise



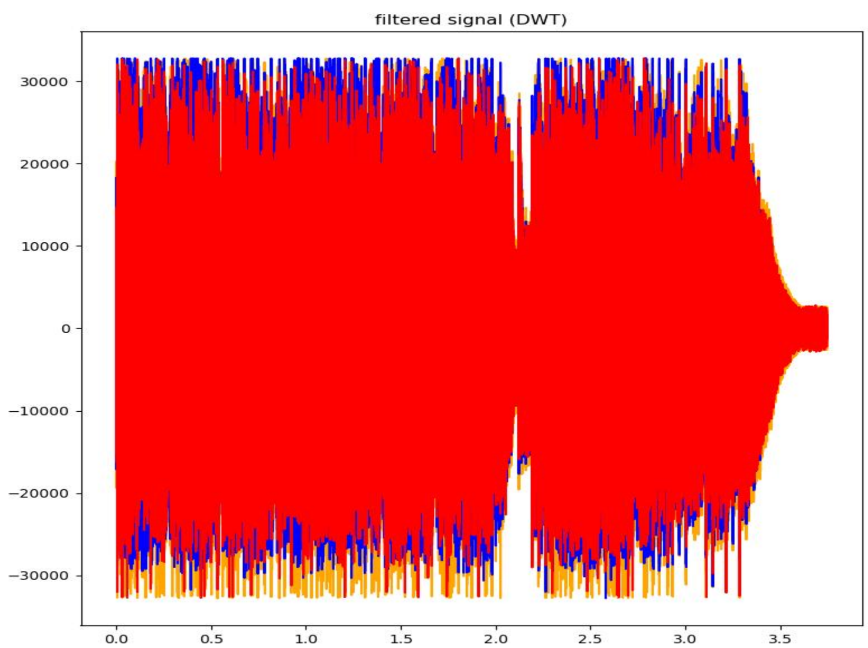
Original noise



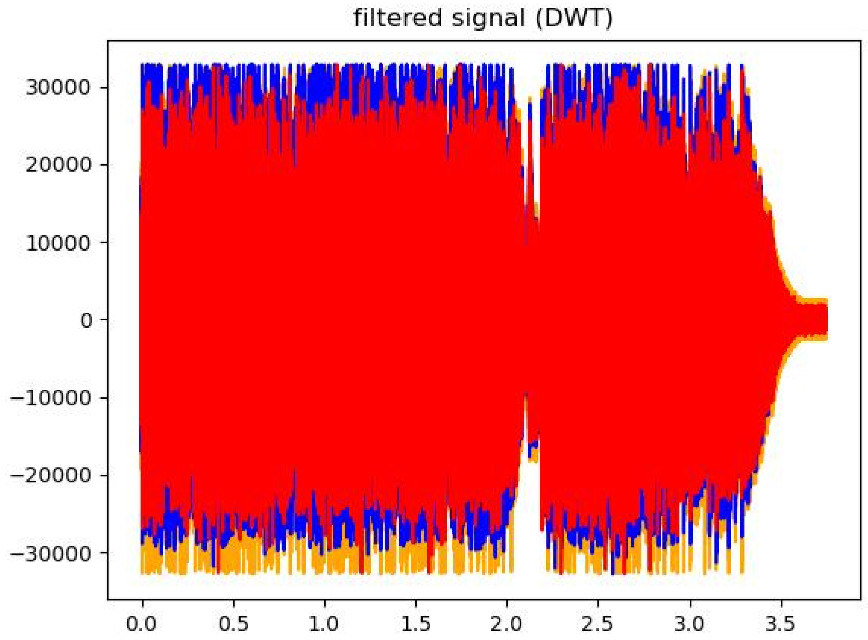
Discrete Wavelet Transform for Filtering

- Hyperparameters adjustable:
  - `(1) wavelet type`
  - `(2) mode used`
  - `(3) hard thresholding configuration`
  - `(4) soft thresholding configuration`

Mode='zero'



Mode='symmetric'



Least Mean Square Adaptive Filter

Least Mean Square Adaptive Filter

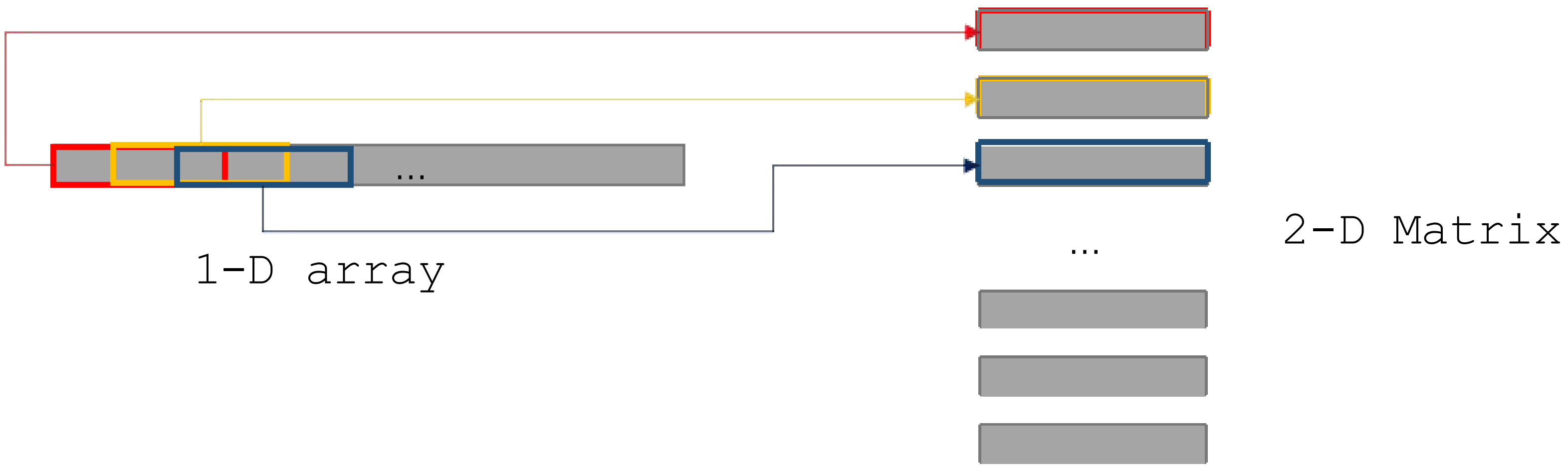
- To perform LMS-filtering:
  - Install the package by “pip install padasip”
  - Before using, make sure to import the package “`import padasip as pa`”
  - The Padasip library is designed to simplify adaptive signal processing tasks within python (filtering, prediction, reconstruction, classification)
  - Documentations available at Padasip — Padasip 1.2.1 documentation (matousc89.github.io)

Least Mean Square Adaptive Filter

- To initialize a LMS-filter:

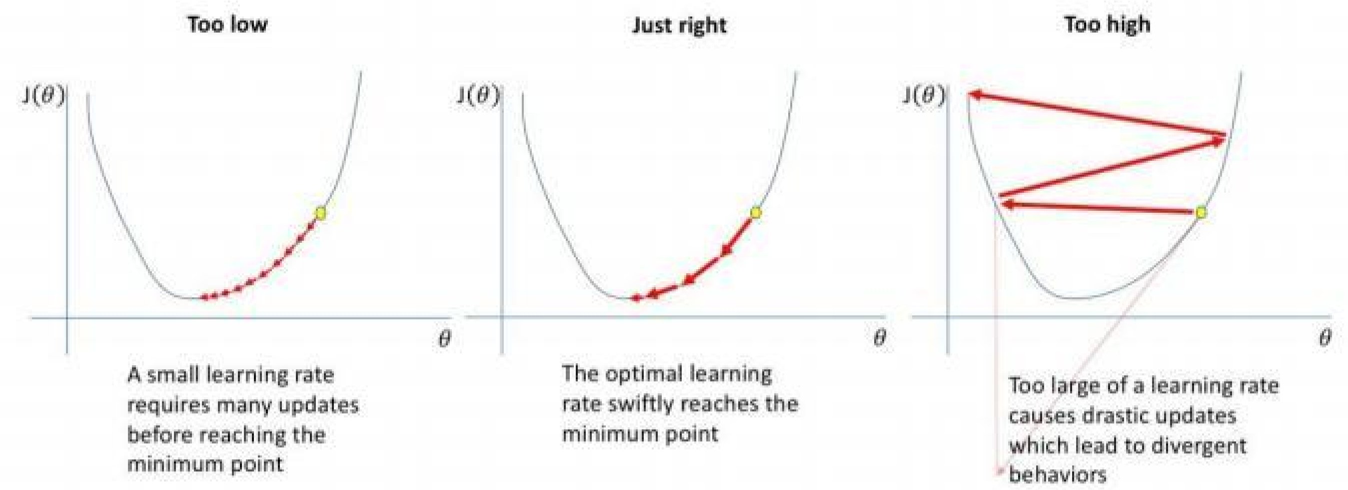
  `filter = pa.filters .FilterLMS (n=…, mu=… )`

  - n: the size of the filter
  - The input data to the LMS -filter need to be adjusted from a 1 -D array into a 2 -D matrix by using a sliding window
  - Each row should be of size n.



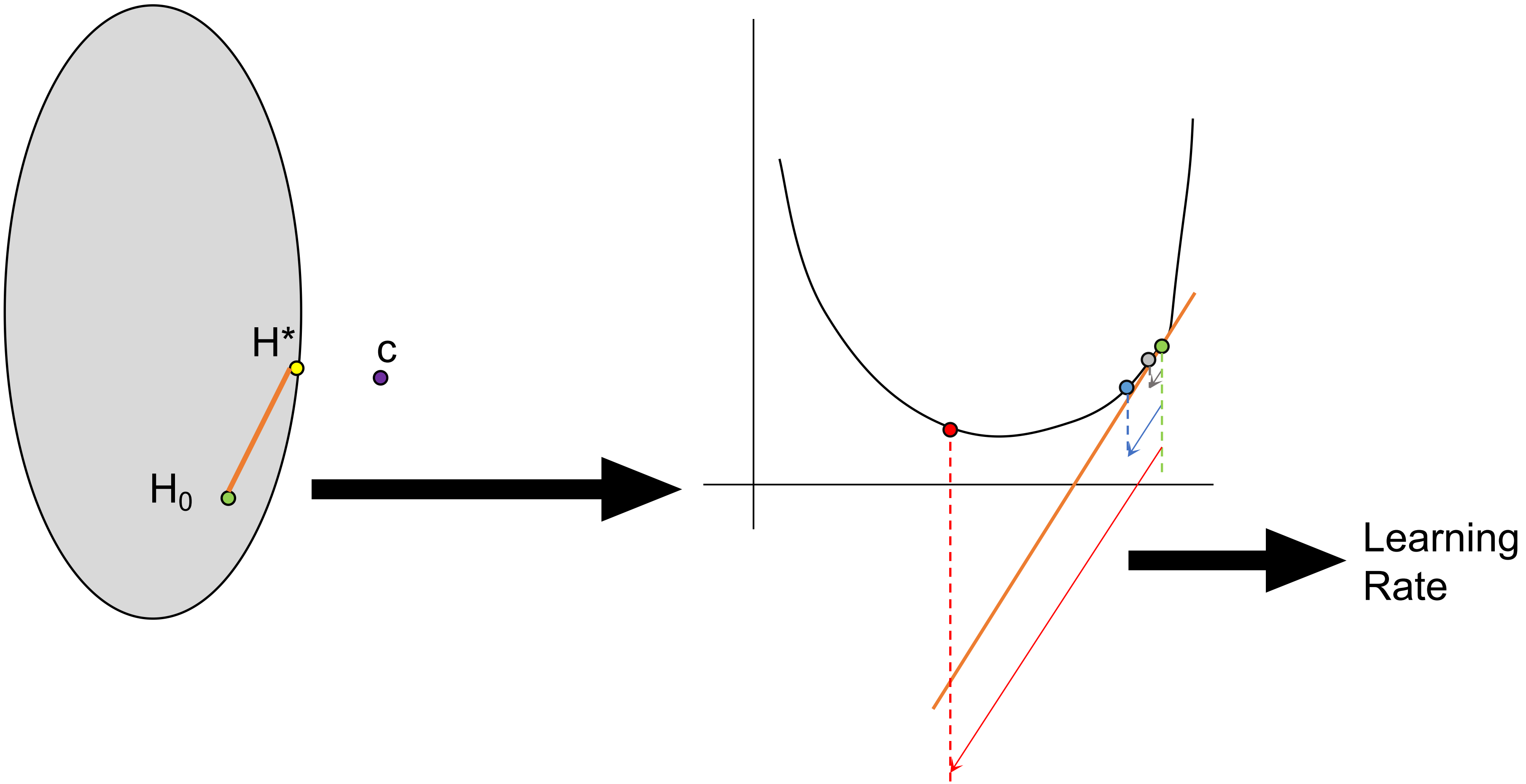
Least Mean Square Adaptive Filter

- To initialize a LMS-filter:
  `filter = pa.filters .FilterLMS (n=…, mu=… )`
  - Mu: The learning rate of the Least Mean Sqaure algorithm
  - Mu being too large will cause the filter not to train properly
  - Mu being too small will cause training to be slow and not reaching the best value after training



Least Mean Square Adaptive Filter

- Learning rate and optimization



Least Mean Square Adaptive Filter

- To initialize a LMS-filter:

```
filter = pa.filters.FilterLMS (n=…, mu=…)
```

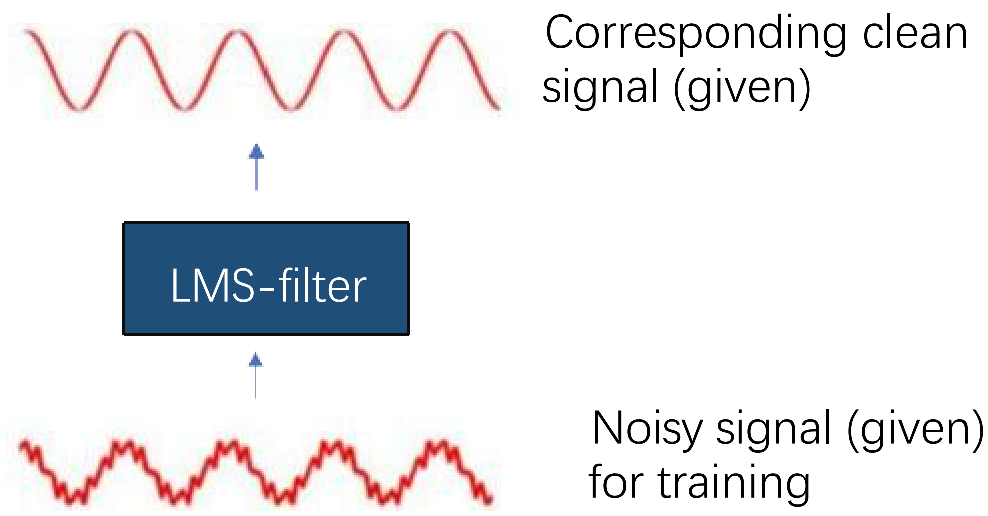
  - The output of the above line is an object for the filter
  - With this filter object, training and predictions can be performed.

Least Mean Square Adaptive Filter

- To train the filter object:

```
y, e, w = filter .run (Target_signal,Corresponding_nois
 y_signal )
```

  - With the above line, the filter value is automatically adjusted based on the Noisy_signal and Corresponding_clean_signal.



Least Mean Square Adaptive Filter

- To train the filter object:

```
y, e, w = filter .run (Target_signal,Corresponding_nois
 y_signal
```

  - Target_signal: 1 -D signal
  - Corresponding_noisy_signal: 2 -D matrix
  - y: filtered signal with the trained filter (1 -D signal)

Least Mean Square Adaptive Filter

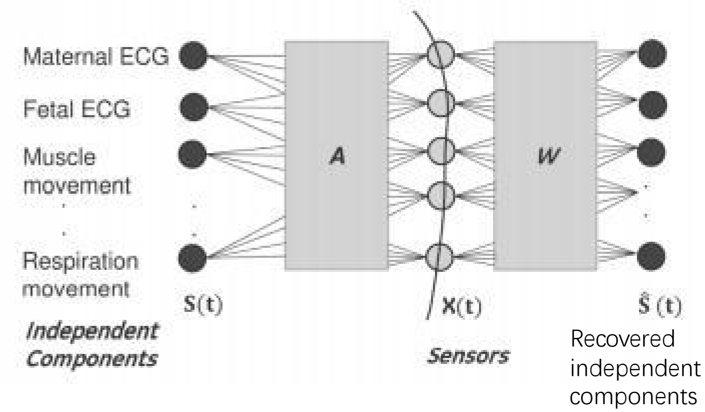
- To make a new prediction with the filter object based on new inputs:

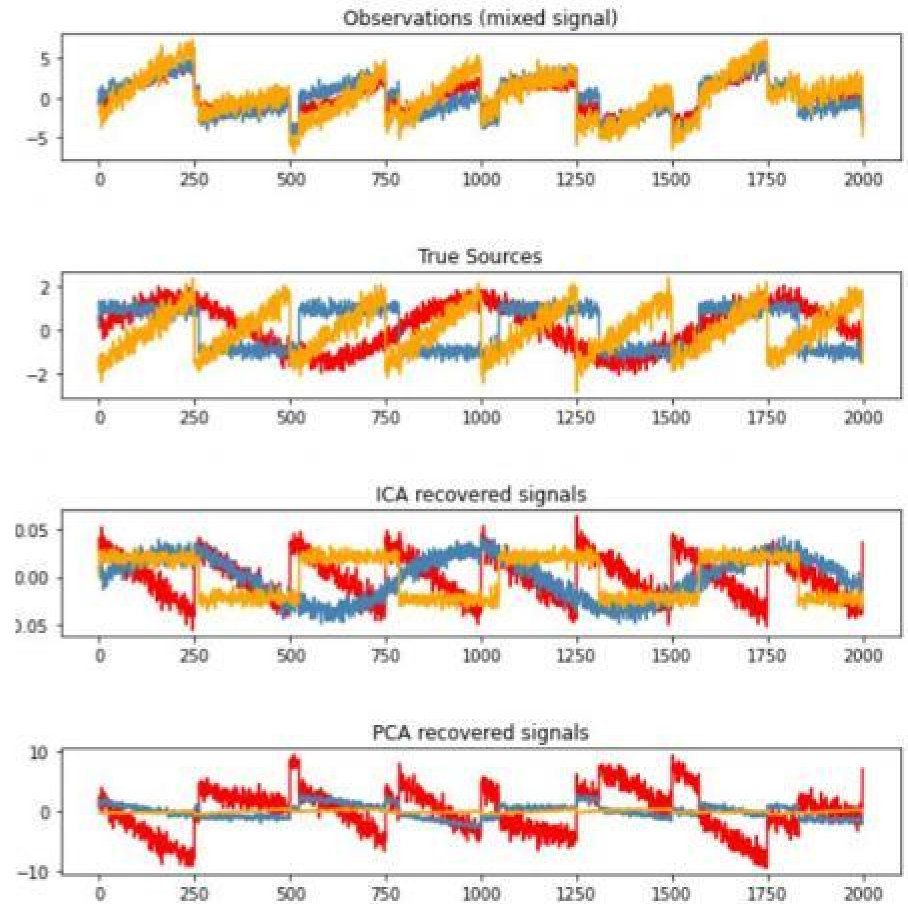
```
y1=[]
for i in range(len(x)):
    temp = filter.predict(x[i])
    y1.append(temp)
```

  - The new input needs to be converted into a 2 -D matrix using the sliding window (stride = 1)
  - Values in each sliding window need to be fed into “filter.predict()” to gain “temp”, which is a single value.
- The hyperparameters adjustable are:
  - (1) filter size (n)
  - (2) learning rate (mu)

Independent Component Analysis (ICA)

Independent Component Analysis (ICA)





ICA can better disentangle signal than PCA

Independent Component Analysis (ICA)

- Before using ICA, it need to be imported from sklearn
  - `from sklearn.decomposition import FastICA`
  - Documentations given at [sklearn.decomposition.FastICA — scikit - learn 1.1.1 documentation]()

- Build a toy dataset:
  - We start with 3 independent signals (s1, s2, s3) and a mixing matrix A
  - We then mix the three signals using the mixing matrix A

Independent Component Analysis (ICA)

- Steps to perform the mixing (for the toy example):

```
s1 = np.sin(2 * time)   # Signal 1 : sinusoidal signal
s2 = np.sign(np.sin(3 * time))   # Signal 2 : square signal
s3 = signal.sawtooth(2 * np.pi * time)   # Signal 3: saw tooth signal

S = np.c_[s1, s2, s3]
S += 0.2 * np.random.normal(size=S.shape)   # Add noise

S /= S.std(axis=0)   # Standardize data
# Mix data
A = np.array([[1, 1, 1], [0.5, 2, 1.0], [1.5, 1.0, 2.0]])   # Mixing matrix
X = np.dot(S, A.T)   # Generate observations
```

Independent Component Analysis (ICA)

- To perform the disentangling with ICA:

```
ica = FastICA(n_components=3)
S_ = ica.fit_transform(X)
```

  - Parameter:
    n_component: it indicates the amount of signal that we aim to disentangle

  - X: the mixed input in the form of a matrix

  - Method:
    fit_transform(X): calculate the disentangled result

Independent Component Analysis (ICA)