

# Digital Signal Processing Tutorial

Stephen Coshatt<sup>1</sup> and Dr. WenZhan Song<sup>1</sup>

The University of Georgia<sup>1</sup>

Sensor Data Science and AI  
Spring 2025



UNIVERSITY OF  
**GEORGIA**

# Outline

1 Signals and Noise

2 Filters

3 Decomposition

4 Time and Frequency Domain

5 Contact Information

# Table of Contents

1 Signals and Noise

2 Filters

3 Decomposition

4 Time and Frequency Domain

5 Contact Information

# Basic Waves

## ① Sine Wave

```
sine_wave(duration=10, sampling_rate=100, amplitude=1.5, frequency=0.3,  
show=True)
```

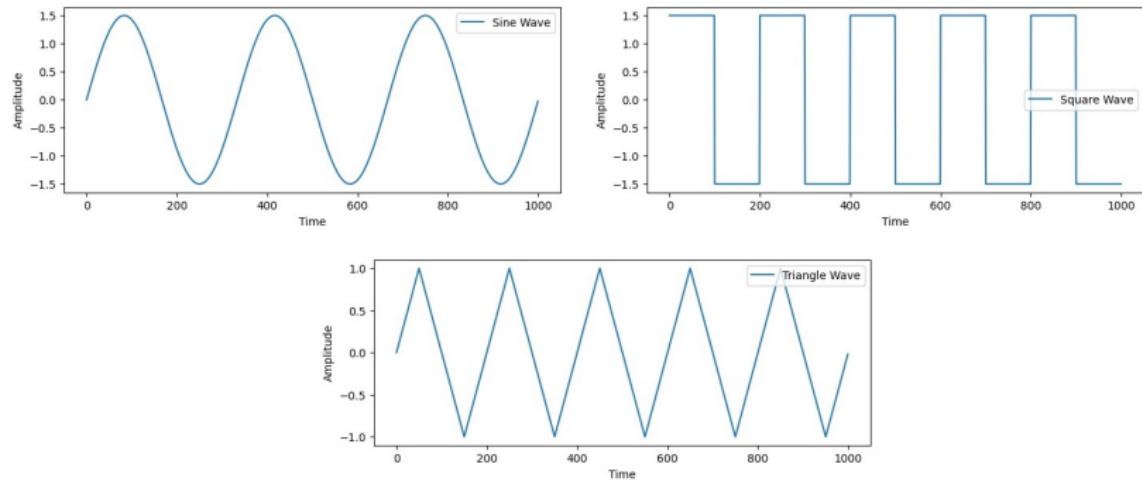
## ② Square Wave

```
square_wave(duration=10, sampling_rate=100, amplitude=1.5, frequency=.5,  
show=True)
```

## ③ Triangle Wave

```
triangle_wave(duration=10, sampling_rate=100, amplitude=1,  
frequency=0.5, show=True)
```

# Basic Waves



**Figure:** Sine, Square, and Triangle Waves

# Chirps and Pulses

## ① Chirps:

Linear: `chirp_wave_linear(f0=0.1, c=0.25, show=True)`

Exponential: `chirp_wave_exponential(f0=0.1, k=1.5, show=True)`

Hyperbolic: `chirp_wave_hyperbolic(f0=0.1, f1=2.6, show=True)`

## ② Pulses:

`pulse_wave(duration=10, sampling_rate=100, amplitude=1, d=0.5, frequency=0.3, expansion=3, show=True)`

# Chirps and Pulses

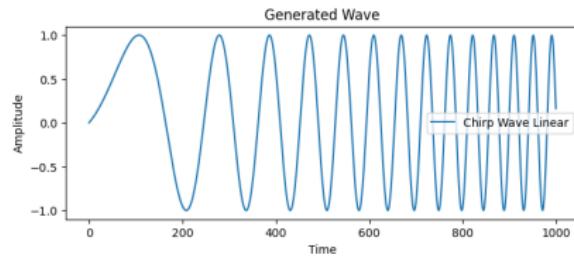


Figure: Linear Chirp

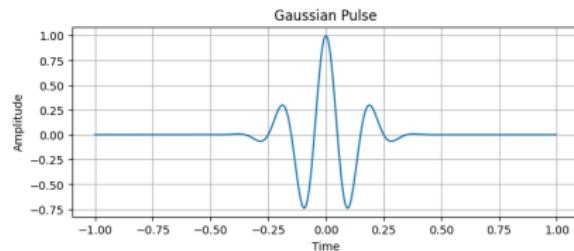


Figure: Pulse

# SCG Signal

```
import DSP
```

## ① Daubechies Wavelet SCG Signal

```
signal_db = dsp.scg_simulate(num_rows=1, duration=10,  
    sampling_rate=100, pulse_type="db", heart_rate=(70,71),  
    respiratory_rate=(15,16), systolic=(120,121), diastolic=(90,91))
```

## ② Morlet Wavelet SCG Signal

```
signal_mor = dsp.scg_simulate(num_rows=1, duration=10,  
    sampling_rate=100, pulse_type="mor", heart_rate=(70,71),  
    respiratory_rate=(15,16), systolic=(120,121), diastolic=(90,91))
```

# SCG Signal

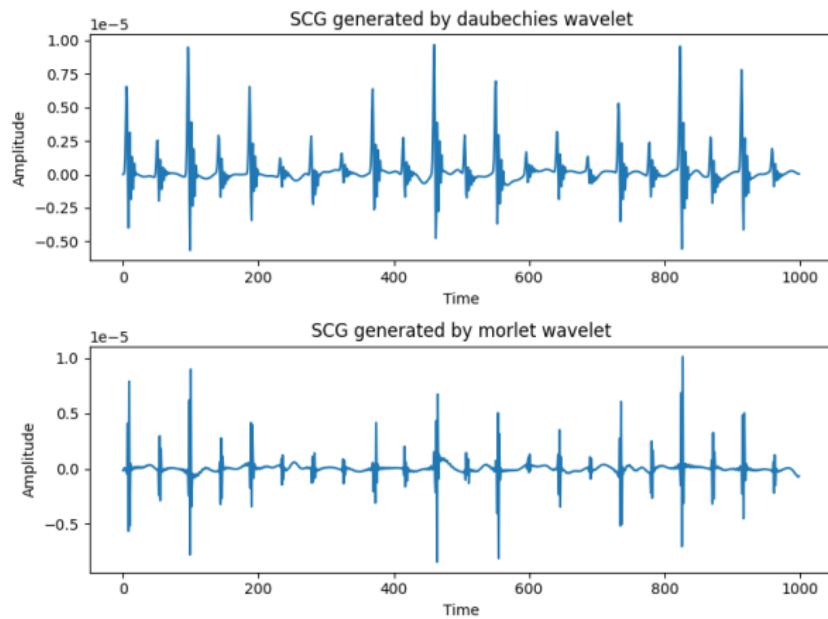


Figure: Two types of SCG generation

# SCG Signal

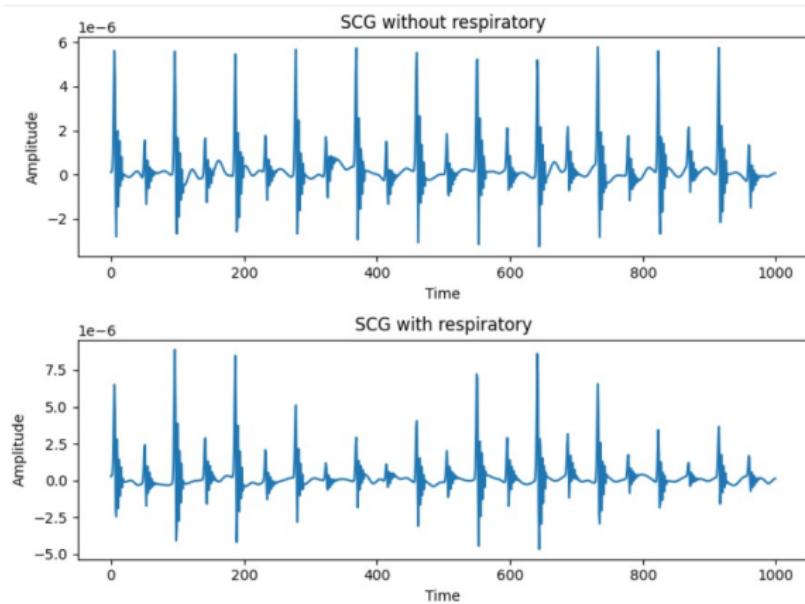
## ① SCG without Respiratory

```
dsp.scg_simulate(num_rows=1, duration=10, sampling_rate=100,  
pulse_type="db", heart_rate=(70,71), add_respiratory = False,  
systolic=(120,121), diastolic=(90,91))
```

## ② SCG with Respiratory

```
dsp.scg_simulate(num_rows=1, duration=10, sampling_rate=100,  
pulse_type="db", heart_rate=(70,71), add_respiratory=True,  
systolic=(120,121), diastolic=(90,91))
```

# SCG Signal



**Figure:** Adding Respiratory to SCG

# SCG Signal

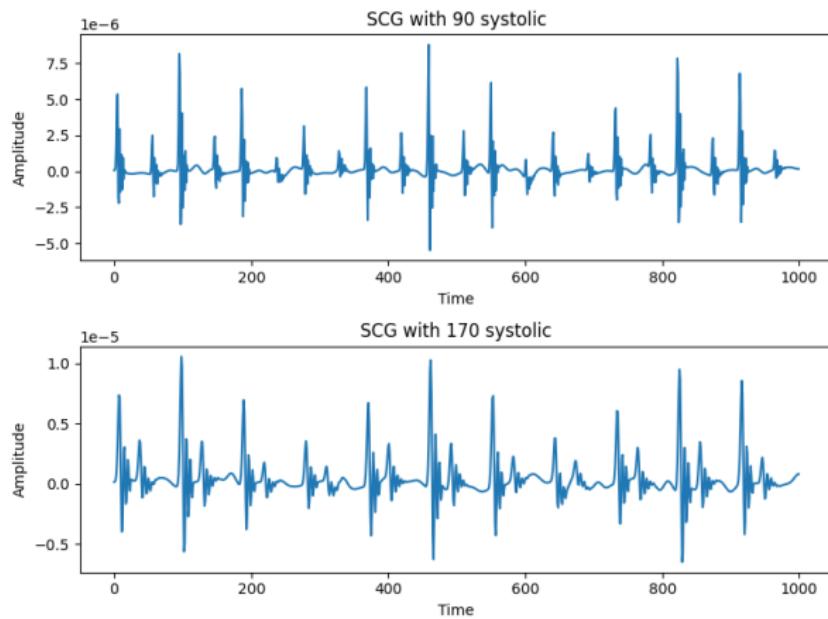
## ① SCG with Systolic of 90

```
dsp.scg_simulate(num_rows=1, duration=10, sampling_rate=100,  
pulse_type="db", heart_rate=(70,71), respiratory_rate=(15,16),  
systolic=(90,91), diastolic=(90,91))
```

## ② SCG with Systolic of 170

```
dsp.scg_simulate(num_rows=1, duration=10, sampling_rate=100,  
pulse_type="db", heart_rate=(70,71), respiratory_rate=(15,16),  
systolic=(170,171), diastolic=(90,91))
```

# SCG Signal



**Figure:** SCG with Systolic pressures of 90 and 170

# Noise

## ① White

(Gaussian, Laplacian, and Band-Limited options) - Random signal w/ equal intensity at different frequencies

## ② Impulse

sudden spike in signal

## ③ Burst

sudden intermittent burst of spikes

## ④ Brown

power spectral density inversely proportional to square of the frequency

# Noise

## ① Pink

Also known as Flicker noise. equal power in each octave

## ② Blue

PSD proportional to the frequency

## ③ Power line

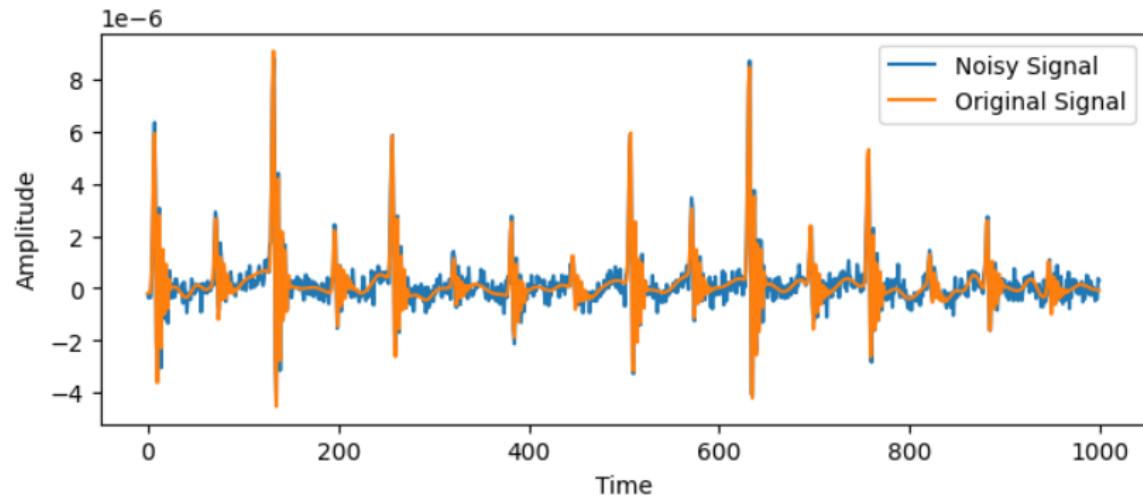
electrical interference generated by power lines

## ④ Echo

occurs when portion of a signal reflects back to source

# Noise

```
add_white_noise(signal, noise_amplitude=0.3, model=0, show=True)
```



**Figure:** Signal with Gaussian White Noise

# Table of Contents

1 Signals and Noise

2 Filters

3 Decomposition

4 Time and Frequency Domain

5 Contact Information

# Filters

- ① **Butterworth** - high pass, low pass, band pass, band stop
- ② **Moving Average**
- ③ **Savitzky-Golay**
- ④ **Wiener**
- ⑤ **Notch**
- ⑥ **Matched**
- ⑦ **Wavelet Denoising**
- ⑧ **Adaptive**
- ⑨ **Kalman**
- ⑩ **Dynamic Time Warping Averaging**

The Butterworth filter is a popular linear filter designed to pass signals within a specified frequency range while attenuating frequencies outside that range.

- ① Characterized by a **smooth frequency response**
- ② Commonly used in signal processing and communications

Mathematically, the filter transfer function is given by

$$H(s) = \frac{1}{1 + \left(\frac{s}{\omega_c}\right)^{2n}}$$

where  $s$  is the complex frequency variable,  $\omega_c$  is the cutoff frequency, and  $n$  is the filter order.

## General Linear Filter Types:

- ① **High Pass** - Allows signals above a specified frequency pass
- ② **Low Pass** - Allows signals below a specified frequency pass
- ③ **Band Pass** - Allows signals in a specified frequency band to pass
- ④ **Band Stop** - Allows signals to pass except those in a specified frequency band

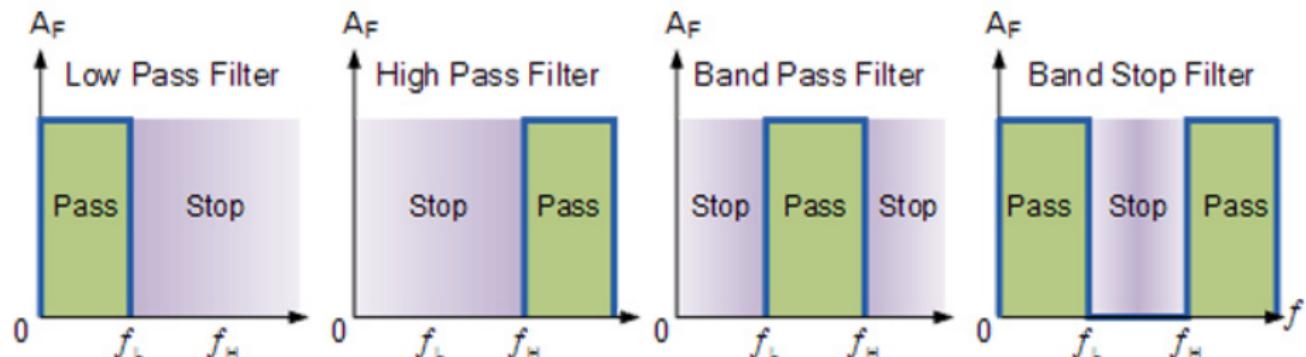
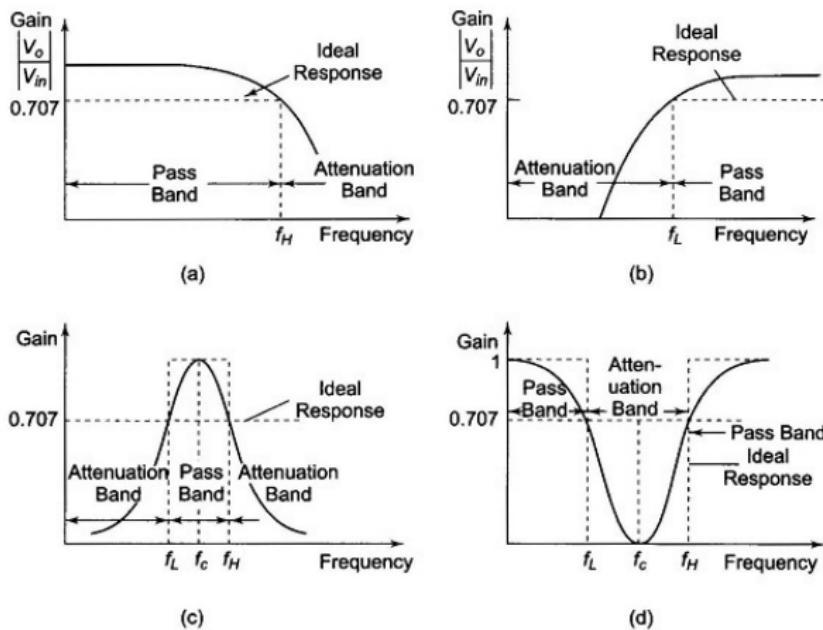


Figure: Linear Filters

Image from Frequency Filter



**Figure:** Butterworth Filter Responses - (a) Low Pass, (b) High Pass, (c) Band Pass, (d) Band Stop

Image from Classification of Active Filters

The **order** parameter in a Butterworth filter specifies the degree of the filter. It determines the frequency response characteristics of the filter, affecting its cutoff frequency and slope.

- ① Higher orders result in steeper cutoffs, effectively filtering out frequencies beyond the desired range.
- ② Lower orders have wider transition regions but lower phase distortion.
- ③ The choice of order balances filter performance and computational complexity based on specific application requirements.

## ① Make an SCG signal

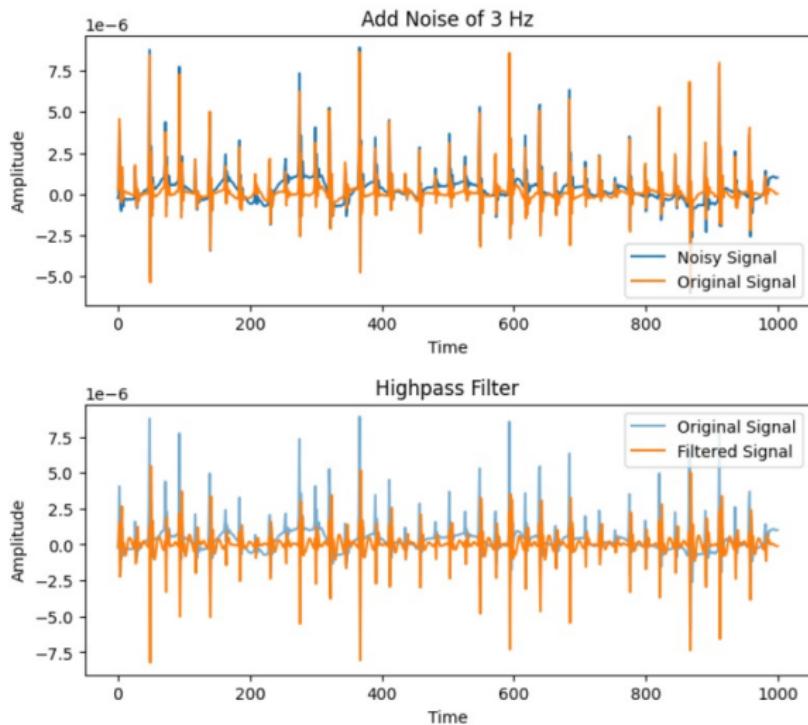
```
signal = scg_simulate():1000]
```

## ② Add noise

```
signal_with_3Hz_Noise = add_distort_noise(signal, n_samples=1000,  
sampling_rate=100,noise_frequency=3, noise_amplitude=0.5, show=True)
```

## ③ Filter Signal

```
filtered_signal = butter_highpass_filter(signal_with_3Hz_Noise, cutoff=5,  
order=10, show=True)
```



**Figure:** Filter 3 Hz Noise with Butterworth

# Simple Moving Average Filter (SMA)

The SMA filter is a basic time-domain filter that computes the average of a specified number of consecutive data points. It provides a simple means of smoothing a time series to reveal underlying trends.

Mathematically, the SMA for a window size  $N$  is given by

$$y(t) = \frac{1}{N} \sum_{i=1}^N x(t - i + 1).$$

Essentially, the SMA filter is a type of **low-pass filter**.

# Exponential Moving Average (EMA) Filter

The EMA filter assigns exponentially decreasing weights to past data points, **giving more importance to recent observations**. It is widely used for trend analysis and noise reduction.

Mathematically, the EMA is defined by  $y(t) = \alpha x(t) + (1 - \alpha)y(t - 1)$ , where  $\alpha$  is the smoothing factor.

Essentially, the EMA filter is a type of **low-pass filter** too.

# Filters

## Moving Average Filters

### ① Simple

```
simple_moving_average_filter(signal_with_40Hz_Noise, length=25,  
show=True)
```

### ② Exponential

```
exponential_moving_average_filter(signal_with_40Hz_Noise, length=25,  
alpha=0.2, show=True)
```

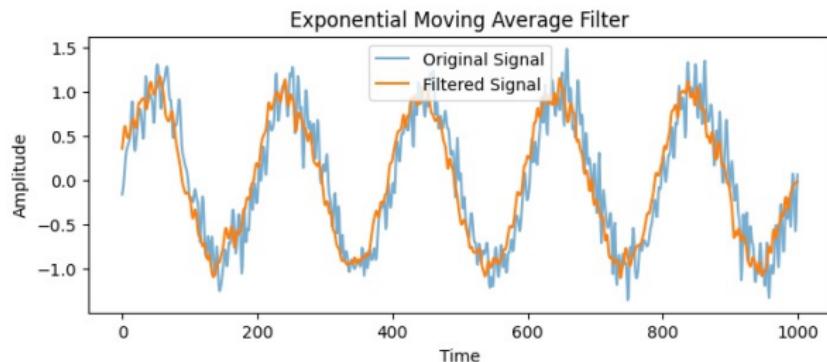
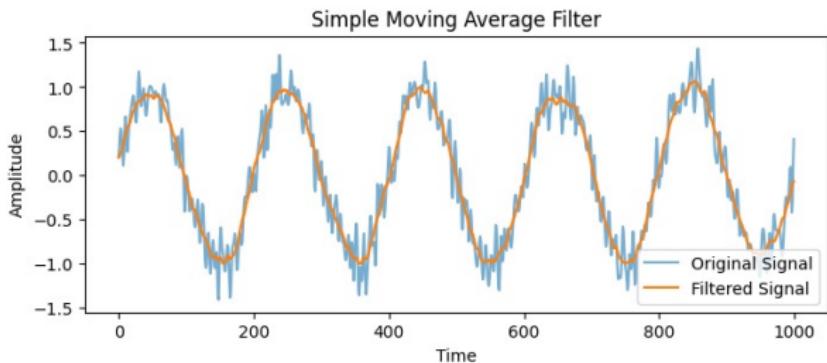


Figure: Moving Average Filters

# Savitzky-Golay Filter (savgol\_filter)

The **Savitzky-Golay filter** is a smoothing algorithm that preserves important features of a signal while reducing noise. It employs polynomial fitting within a sliding window to smooth the data.

Mathematically, the filter coefficients are determined by least squares fitting, providing a balance between noise reduction and signal preservation.

$$\text{Minimize } \sum_{i=-m}^m (y_{k+i} - \sum_{j=0}^p a_j x_{k+i}^j)^2$$

**Example:**

If polynomial is  $y = a_0 + a_1x + a_2x^2$

$$\text{then Minimize } \sum_{i=-2}^2 (y_{k+i} - (a_0 + a_1x_{k+i} + a_2x_{k+i}^2))^2$$

**Reference:** [Introduction to the Savitzky-Golay Filter: A Comprehensive Guide](#)

# Filters

## ① Savitzky-Golay Filter

```
savgol_filter(signal_with_5Hz_noise, window_length=15, show=False)
```

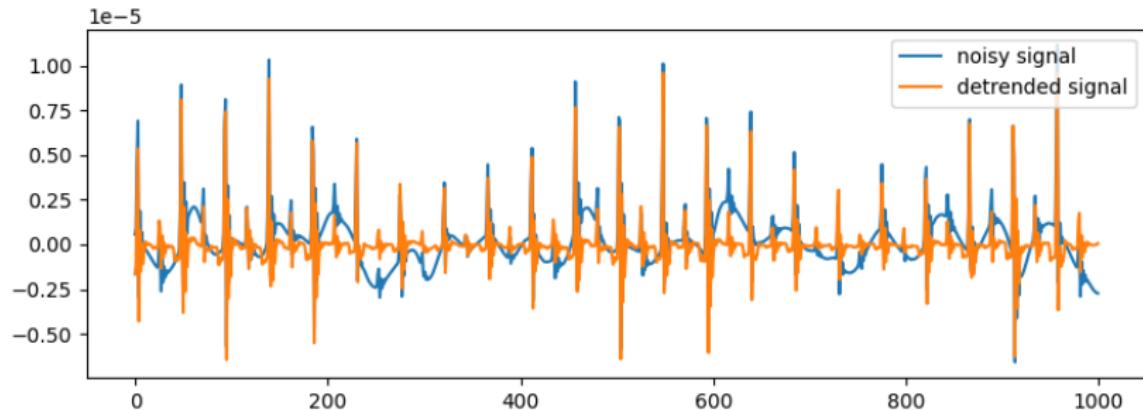


Figure: Savgol Filter

# Wiener Filter

The Wiener filter is an optimal linear filter used for signal deconvolution and noise reduction. It minimizes mean-squared error between the estimated signal and the true signal, enhancing signal-to-noise ratio.

Mathematically, the Wiener filter in the frequency domain is given by

$H(f) = \frac{S_x(f)}{S_x(f) + S_n(f)}$  , where  $S_x(f)$  is the signal power spectrum,  $S_n(f)$  is the noise power spectrum.

Wiener filters are characterized by the following:

- ① **Assumption:** signal and (additive) noise are stationary linear stochastic processes with known spectral characteristics or known autocorrelation and cross-correlation
- ② **Requirement:** the filter must be physically realizable/causal (this requirement can be dropped, resulting in a non-causal solution)
- ③ **Performance criterion:** minimum mean-square error (MMSE).

# Filters

## ① Wiener Filter

```
wiener_filter(signal + noise, noise, show=True)
```

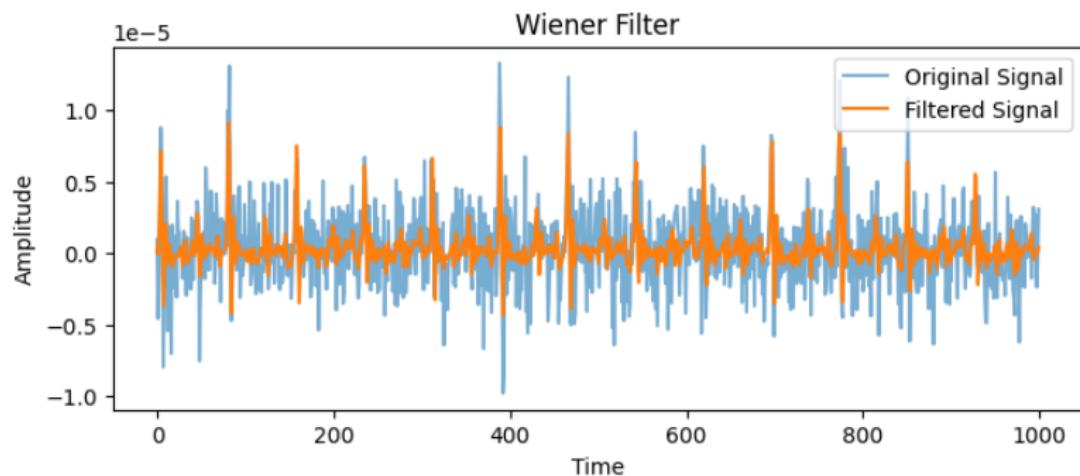


Figure: Wiener Filter

# Notch Filter

The notch filter is designed to suppress specific frequencies, often used to eliminate unwanted interference or noise at a particular frequency.

It creates a notch or a dip in the frequency response centered around the target frequency.

Mathematically, the transfer function of a notch filter can be represented as  $H(f) = \frac{1}{1 + \frac{(f/f_0)^2}{Q}}$ , where  $f_0$  is the center frequency, and  $Q$  is the quality factor.

A Notch filter is essentially a very narrow band stop filter.

# Filters

## ① Notch

```
notch_filter(signal_with_5Hz_Noise, cutoff=5, q=3, fs=100, show=True)
```

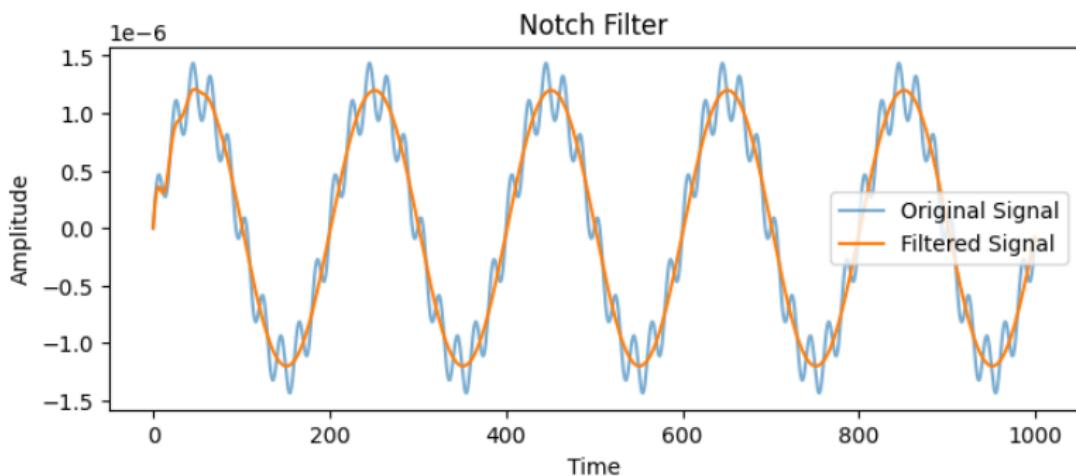


Figure: Notch Filter

# Matched Filter

The matched filter is a signal processing filter that maximizes the signal-to-noise ratio for a known signal when embedded in noise.

It is particularly effective in detecting signals with a known template.

Mathematically, the matched filter output is the convolution of the received signal and the time-reversed conjugate of the template signal.

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

**Where:**  $x[k]$  is the input function of  $k$ , and  $y[n]$  is the filtered output

# Filters

## ① Matched

```
matched_filter(signal_with_40Hz_Noise_, clean_template, show=False)
```

```
matched_filter(signal_with_40Hz_Noise_, noisy_template, show=False)
```

# Filters

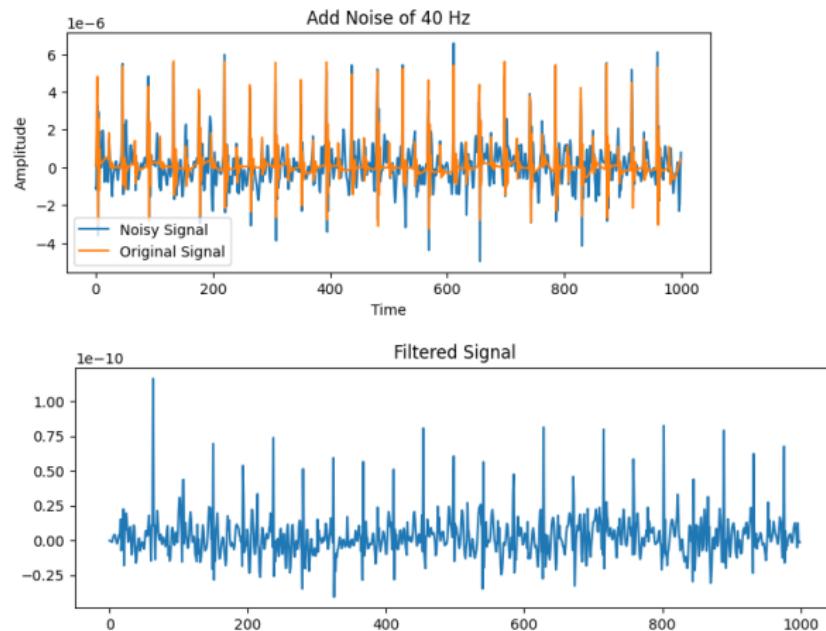


Figure: Matched Filter

# Kalman Filter

A Kalman filter is a mathematical algorithm that uses a series of noisy measurements over time to estimate the true state of a system, providing a more accurate representation of the underlying variables by combining a prediction based on a system model with new measurements, effectively "filtering out" noise and uncertainties in the data; it's particularly useful when you can't directly measure the desired variables but have access to related, imperfect measurements.

The filter performs two main operations:

- ① **Prediction:** Uses the system model to predict the next state based on the previous estimate.
- ② **Update:** Incorporates new measurements to refine the predicted state, weighting the new information based on its reliability

Assumes noise is Gaussian

# Kalman Filter Continued

## **State equation:**

$$x_{k+1} = Ax_k + Bu_k + w_k$$

## **Output equation:**

$$y_k = Cx_k + z_k$$

## **Where:**

$A$ ,  $B$ , and  $C$  are matrices

$k$  is the time index

$x$  is the state of the system

$w$  is the process noise

$z$  is the measurement noise

# Kalman Filter Continued

**Assumptions:**  $k$ ,  $w_k$ , and  $z_k$  are independent random variables

**Process Noise Covariance:**

$$S_w = E(w_k w_k^T)$$

**Measurement Noise Covariance:**

$$S_z = E(z_k z_k^T)$$

**Where:**

$w_T$  is the transpose of  $w$  random noise vector

$z_T$  is the transpose of  $z$  random noise vector

$E(\cdot)$  is the expected value

# Kalman Filter Equations

$$K_k = AP_k C^T (CP_K C^T + S_z)^{-1}$$

$$\hat{x}_{k+1} = (A\hat{x}_k + Bu_k) + K_k(y_{k+1} - C\hat{x}_k)$$

$$P_{k+1} = AP_k A^T + S_w - AP_k C^T S_z^{-1} CP_k A^T$$

**Where:**

$K$  is the Kalman gain

$P$  is the estimation error covariance

$\hat{x}$  is the state estimate. The first term is the estimate at time  $k + 1$  and the second term is the *correction* term

$-1$  superscript indicates matrix inversion

$T$  superscript indicates matrix transposition

**Equations from [Kalman Filter Explained \(with Equations\)](#)**

# Filters

## ① Kalman Filter

```
kalman_filter(x, x_last, p_last, Q, R)
```

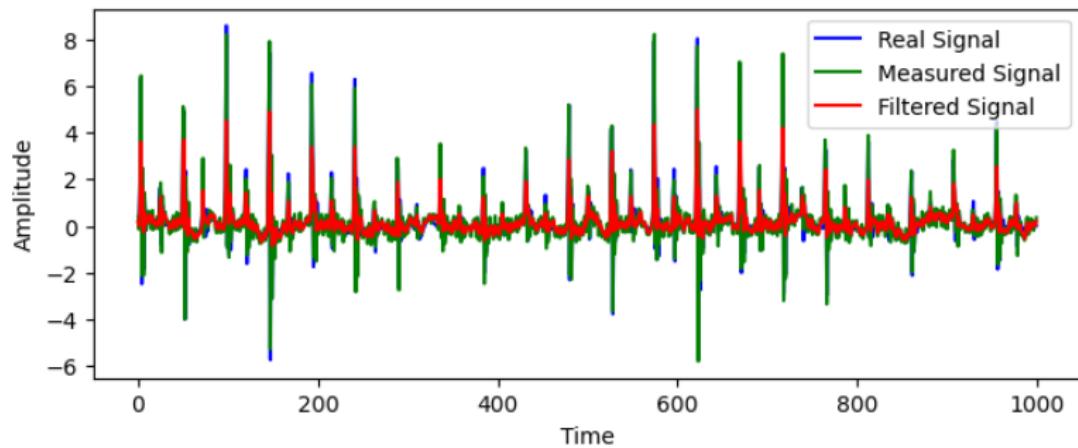


Figure: Kalman Filter

# Adaptive Filters

An adaptive filter is a filter with non-constant coefficients. The filter coefficients are adjusted based on a criterium which is often defined to optimize the performance of the filter in its ability to estimate an unknown quantity in an input signal.

In this tutorial, we will limit ourselves to adaptive finite impulse response (FIR) filters.

The filter accepts an input signal  $\mathbf{x}$  and produces an output signal  $\mathbf{y}$ . The FIR coefficients are adjustable, meaning that at every new sample of  $\mathbf{x}$ , the coefficients can take on a new value. The new value of filter coefficients is determined using a coefficient update algorithm, which computes an adjustment for each filter coefficient based on an error signal  $\mathbf{e}$ . The error signal  $\mathbf{e}$  is typically computed as the difference between the actual output signal  $\mathbf{y}$  and a desired output signal  $\mathbf{d}$ .

# Adaptive Filters

The desired output signal  $d$  depends on the specific application of the adaptive filter. However, the adaptive algorithm will change the coefficients so as to minimize the mean squared value of the error signal  $e$ . That is, given that the filter output is defined by filter coefficients  $w(n) = [w_0(n), w_1(n), \dots, w_{N-1}(n)]^T$ , we try to minimize the expected square error:

$$\min_{w(n)} E [e^2(n)]$$

There are generally four different configurations common for an adaptive filter:

- ① System Identification
- ② **Noise Cancellation**
- ③ Equalization
- ④ Adaptive Prediction

# Basic Adaptive Filter

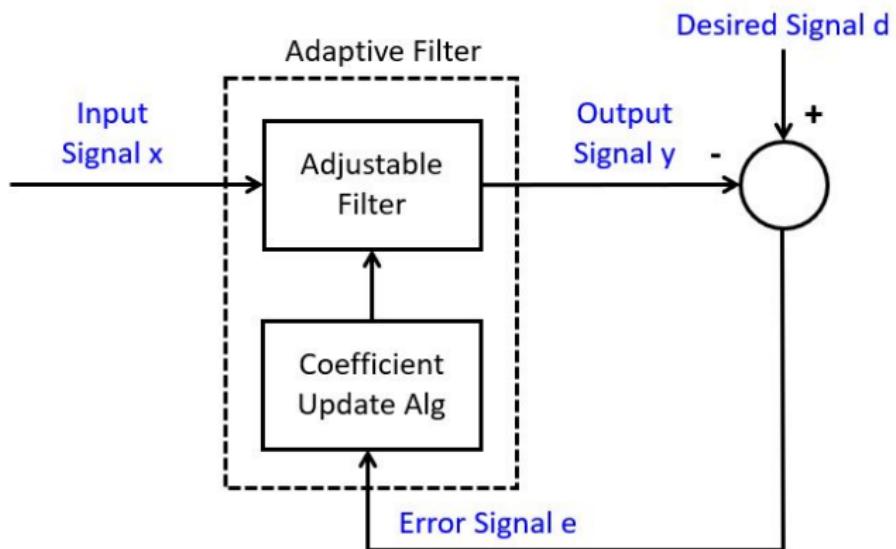


Figure: Adaptive Filter

Image from [Adaptive Filters](#)

# Adaptive Noise Cancellation

In Noise Cancellation, we are interested in removing a known disturbance  $n_1$  from a signal.

The disturbance is affected by the system dynamics  $H(z)$  into  $n_0$ , so that we are unsure how much the input signal is affected by the disturbance.

Using an adaptive filter, we estimate the system dynamics, and we remove the filtered disturbance from the output signal.

The output signal, in the case of noise cancellation, is created out of the error signal, which in this case will closely resemble the input signal  $x$ .

# Adaptive Filter for Noise Cancellation

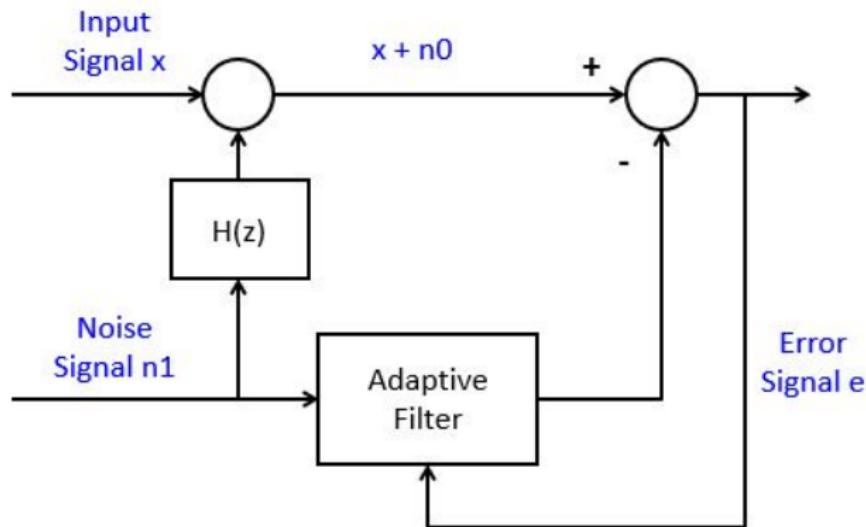


Figure: Adaptive Filter for Noise Cancellation

Image from Adaptive Filters

```
lms_filter(input, desired_signal, n=1, mu=0.015, show=False)
```

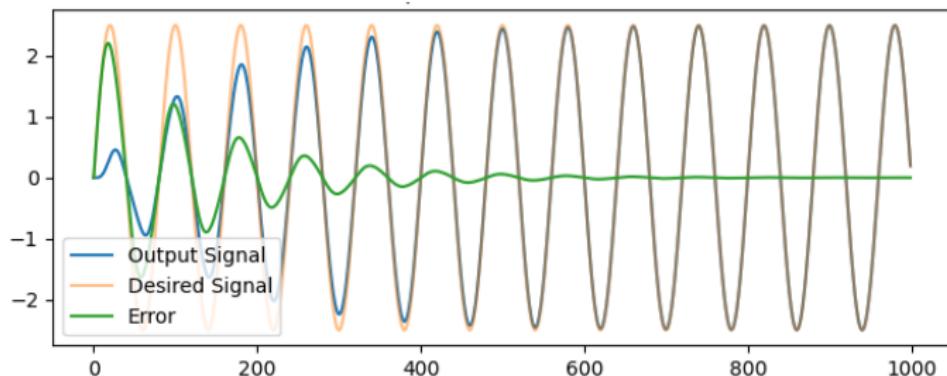


Figure: FIR Filter

# Signal Averaging

Signal Averaging is a signal processing technique that improves the signal-to-noise ratio (SNR) by averaging multiple measurements of a signal.

# Signal Averaging Steps

- ① Take multiple samples of the signal. Each sample may have some degree of noise or error added to the true signal, so they are not identical.
- ② **Align the Signals:** If the signal is periodic or repeatable, align measurements in time (e.g., using time synchronization or triggering).
- ③ **Average the Values:** Average all signal values point-by-point across all the samples. This is done for each corresponding time or measurement point. Mathematically, for  $N$  measurements,  $x_1(t), x_2(t), \dots, x_N(t)$  of a signal, the averaged signal  $x(t)$  is calculated as:

$$x(t) = \left(\frac{1}{N}\right) \sum_{i=1}^N x_i(t)$$

- ④ **Reduced Noise:** The noise is expected to have a random nature, thus averaging over many samples should cancel the noise, leaving a clearer representation of the signal. The more measurements averaged, the lower the noise level becomes (in the ideal case), and the clearer the true signal is.

# Averaging Pros & Cons

## Pros:

- ① **Improved Signal-to-Noise Ratio (SNR):** Signal averaging reduces random noise by averaging it out, which increases the SNR. This makes the underlying signal easier to analyze or measure.
- ② **Enhancing Low-Level Signals:** When dealing with low-amplitude signals that are buried in noise, averaging can make these signals more visible by reducing the noise that tends to mask them.
- ③ **Simple and Effective:** Signal averaging is a straightforward and computationally simple method to improve data quality, especially when the signal is repetitive or periodic.

## Cons:

- ① **Non-Stationary Signals:** If the signal changes over time or is non-stationary (i.e., it evolves in an unpredictable way), signal averaging may distort or lose important details. This technique works best for signals that are repetitive or periodic.
- ② **Slow Process:** Signal averaging can be time-consuming because it requires multiple samples to be taken over time, especially if many samples are needed to effectively reduce noise.

# Dynamic Time Warping

**Dynamic Time Warping (DTW)** is a similarity measure between two temporal sequences, accommodating for variations in their alignment and speed. It finds an optimal warping path by minimizing the accumulated distance between corresponding points.

Mathematically, given sequences  $A = [a_1, a_2, \dots, a_n]$  and  $B = [b_1, b_2, \dots, b_m]$ , DTW constructs a cost matrix  $C$  where  $C(i, j)$  represents the local distance between  $a_i$  and  $b_j$ . It then calculates an optimal path from  $C(1, 1)$  to  $C(n, m)$  through dynamic programming. DTW is widely used in pattern recognition, speech processing, and signal analysis for aligning time series data.

# DTW Concepts

- ① **Non-linear Alignment:** DTW can align sequences that have varying speeds. For example, it can compare two sequences where one sequence is "stretched" or "compressed" relative to the other (e.g., a faster and slower version of the same movement or event).
- ② **Time Warping:** The idea of warping comes from adjusting the time axis in such a way that corresponding points in the two sequences match as well as possible. This is done by considering all possible alignments of the two sequences, even if they are of different lengths.
- ③ **Distance Measure:** DTW computes the distance between two time series by considering each point in one series and finding the optimal match (with a possible shift) in the other series. The total distance is the sum of these optimal matches.

# DTW Steps

- ① **Cost Matrix:** DTW uses a cost matrix to evaluate the alignment between the two sequences. This matrix measures the "cost" of aligning different points from each sequence. It is calculated by comparing each pair of points from the two sequences and computing the difference between them (often squared differences).
- ② **Recursive Calculation:** The cost matrix is filled recursively, where each cell contains the minimal cumulative distance required to reach that point from the start of the sequences. The cumulative cost is the sum of the cost to reach the current point and the minimum of the previous possible alignments.
- ③ **Optimal Path:** The final DTW distance is obtained by tracing back from the end of the cost matrix to the start. This path represents the optimal alignment between the two sequences.
- ④ **Warping Path:** The warping path describes how the points in one sequence correspond to the points in the other sequence. It allows one sequence to be "warped" (or stretched/compressed) to align with the other.

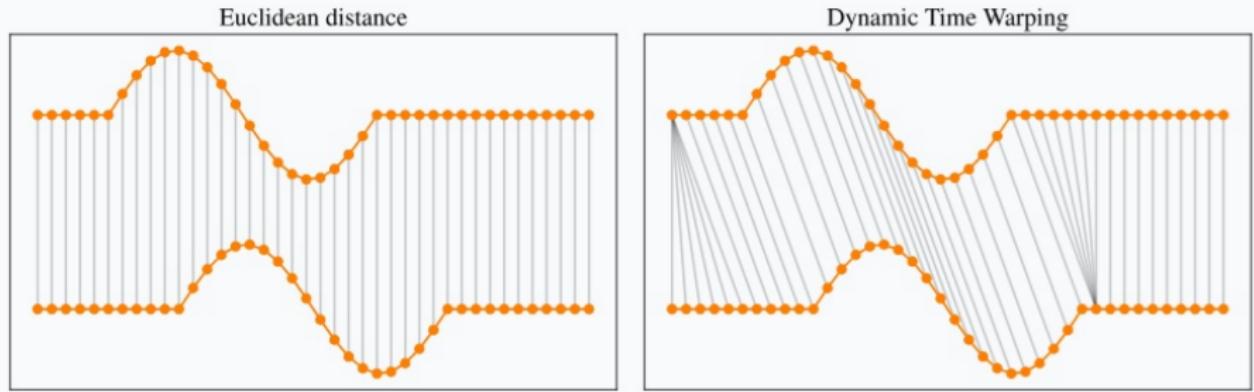


Figure: DTW and Euclidean Distance Comparison.

Image from [An Introduction to Dynamic Time Warping by Romain Tavenard](#)

# DTW Pros

- ① **Handles Temporal Shifts:** DTW is robust to shifts in time. It can align time series even when they are stretched or compressed along the time axis.
- ② **Flexible Similarity Measure:** DTW provides a more flexible and accurate measure of similarity for time series compared to simple methods like Euclidean distance, especially when there is temporal misalignment.
- ③ **General Purpose:** DTW can be applied to a wide range of problems involving time series, from speech and audio processing to medical signal analysis.

# DTW Cons

- ➊ **Computationally Expensive:** The basic DTW algorithm has a time complexity of  $O(n^2)$ , where  $n$  is the length of the time series. This can be slow for long sequences.
- ➋ **Overfitting:** If not properly tuned or constrained, DTW can overfit the alignment to the noise in the data, leading to inaccurate comparisons.
- ➌ **No Consideration for Global Shape:** DTW optimizes local alignment, but this might lead to overemphasis on small local features rather than global patterns in the time series.
- ➍ **Distance Matrix Size:** Since DTW requires constructing a full  $n \times n$  distance matrix, it can be memory-intensive for large datasets.

# Non-Linear Adaptive Averaging Functions

The Non-Linear Adaptive Averaging Function (NLAAF) is a signal processing technique designed to improve the quality of a signal by adaptively adjusting how averaging is applied based on the characteristics of the signal itself. Unlike simple linear averaging methods, which apply the same weight or averaging function to all data points, NLAAF adjusts the averaging process dynamically according to the local behavior or properties of the signal.

# NLAAF Continued

The idea behind NLAAF is to use local signal characteristics to determine how much averaging should be applied at each point in the signal. For example, at regions where the signal changes rapidly (edges or transitions), the averaging might be reduced or avoided, so the filter does not blur important features. In regions where the signal is more uniform or contains noise, stronger smoothing or averaging is applied.

The non-linear nature of the filter allows it to better handle complex signals with noise and varying levels of smoothness, making it more adaptable and capable of preserving important features compared to traditional linear filters.

# NLAAF Concepts

- ① **Non-Linear:** NLAAF is a non-linear filter, meaning that it doesn't apply a linear weighting to each data point in the signal. Instead, the function may apply a more complex, non-linear operation to smooth the signal while preserving important features like edges or rapid transitions that a linear filter might blur or distort.
- ② **Adaptive:** The term "adaptive" means that the filter adjusts itself to the characteristics of the data. For example, it might apply stronger smoothing where the signal is relatively stable and less smoothing where the signal changes rapidly. This is important when dealing with signals that contain both noise and sharp transitions, as it can help preserve the important parts of the signal while reducing noise.
- ③ **Averaging Function:** Averaging refers to combining multiple data points in some way to produce a smoothed value. In NLAAF, this function is not fixed or uniform across the entire signal, but changes depending on the local context of the signal.

# NLAAF Tutorial Functions

**Note:** There are two NLAAF function in this tutorial.

- ① **NLAAF 1** - Applies DTW to two small windows (pieces) and determines the average of the pieces. These averages are used to build a center piece.
- ② **NLAAF 2** - same as NLAAF 1, except a specified number of pieces greater than 2 are used to build the center piece.

# Iterative Constrained Dynamic Time Warping

Iterative Constrained Dynamic Time Warping (ICDTW) is an enhanced DTW algorithm, designed to address some of the limitations of DTW, especially in terms of computational efficiency and accuracy when comparing time series with similar patterns but differing temporal alignments.

## STEPS:

- ① **Initial Alignment:** Start by performing a basic alignment between the two time series using standard DTW.
- ② **Apply Constraints:** During each iteration, impose constraints that restrict the allowable warping paths between the two time series. These constraints might include things like limiting the range of movement in the warping path or ensuring monotonicity.
- ③ **Refinement:** After applying constraints, refine the alignment by recalculating the DTW distance with the constrained path.
- ④ **Repeat:** The iterative process is repeated until the alignment converges to an optimal or sufficiently accurate solution.

# ICDTW Benefits

- ① **Handling Large Variations:** By constraining the warping path and iterating over the alignment, ICDTW can more effectively handle large variations in the time series (e.g., different lengths, tempo variations, or large misalignments) compared to traditional DTW.
- ② **Preserving Important Features:** The application of constraints during the iterative process helps to prevent the alignment from distorting important features in the time series, such as sharp transitions or edges. This is especially useful in time series that contain noise or irregularities.
- ③ **Improved Efficiency:** The iterative refinement approach makes ICDTW more computationally efficient, particularly when dealing with long time series, as it limits the warping search space through constraints.
- ④ **Flexibility:** ICDTW can be customized with various constraints, making it more adaptable to specific types of time series comparisons.

# DTW Barycenter Averaging

**DTW Barycenter Averaging (DBA)** is a technique used to compute an average of multiple time series using Dynamic Time Warping (DTW). Unlike traditional methods of averaging time series, which simply compute the mean of corresponding points, DBAs use DTW to align the time series before averaging. This allows for the averaging of time series that may have temporal misalignments, speed variations, or shifts. The primary goal of DBA is to find a central tendency or "average" time series that best represents a set of input time series, while accounting for non-linear alignments in the data.

The term "**barycenter**" refers to the center of mass or average of a set of data points. In the context of DBA, the barycenter represents an average time series, which is computed by iteratively aligning and averaging multiple time series.

# DTWBA Steps

- ➊ **Initialization:** Start with an initial guess for the barycenter, which can be any of the input time series or some other starting point.
- ➋ **Iterative Process:** In each iteration, the algorithm performs the following steps:
  1. **Alignment using DTW:** For each time series in the dataset, use DTW to align that series with the current barycenter.
  2. **Averaging:** The average of all the aligned time series is computed at each time point, creating a new candidate barycenter.
- ➌ **Convergence:** The process is repeated iteratively. In each iteration, the barycenter is updated, and the DTW alignment is recomputed. The algorithm converges when the barycenter stops changing significantly between iterations or after a pre-defined number of iterations.
- ➍ **Final Barycenter:** The final output is a time series that represents the average (or barycenter) of the input time series. This series is aligned in a way that best captures the common underlying pattern of the input sequences.

# DTWBA Pros

- ① **Handling Temporal Misalignment:** DBA excels in scenarios where the input time series are misaligned or have different lengths. By using DTW to align the series first, DBA can compute a meaningful average despite these misalignments.
- ② **Robustness to Noise:** Since DBA aligns the time series before averaging, it tends to be more robust to noise than traditional averaging methods. By aligning the time series, DBA reduces the impact of small fluctuations that may not represent the true underlying patterns.
- ③ **Flexible for Variable Length Time Series:** Has ability to handle time series of different lengths.
- ④ **Improved Pattern Recognition:** Can produce more accurate representations of a group of time series, making it useful for pattern recognition tasks.

# DTWBA Cons

- ① **Computational Complexity:** DTW itself requires  $O(N^2)$  time complexity for a pair of time series of length  $N$ , and performing this for many series can be quite slow.
- ② **Sensitive to Initialization:** DTWBA can be sensitive to its initial guess.
- ③ **Pairwise DTW Computation:** DTWBA involves computing the pairwise alignment of all time series using DTW, which can be very resource-intensive for large datasets or long time series.
- ④ **Struggle with Extremely Noisy Data:** While DTWBA can handle some noise, extreme noise or outliers can still affect the result.
- ⑤ **Overfitting to Local Variations:** Can overfit to local variations and fluctuations within each time series.
- ⑥ **Difficulty with Large Variations in Scale:** If the time series differ greatly in scale (amplitude) in addition to time shifts, DTWBA might not always produce a meaningful average.

# DTW Averaging Methods

- ① Non-Linear Adaptive Averaging Function (NLAAF) 1 - Applies DTW to two small windows (pieces) and determines the average of the pieces. These averages is used to build a center piece.
- ② NLAAF 2 - same as NLAAF 1, except a specified number of pieces greater than 2 are used to build the center piece.
- ③ Iterative Constrained Dynamic Time Warping (ICDTW) - Repeated DTW comparison over pieces with constraints added over each iteration. Becomes more refined at each step.
- ④ DTW Barycenter Averaging (DBA) - Randomly determines a "Barycenter". Iteratively uses DTW to compare to each piece and update the Barycenter.

# Averageing Code

- ① performNLAFF1(list(series), show=True)
- ② performNLAFF2(list(series), show=True)
- ③ performICDTW(list(series), show=True)
- ④ performDBA(series, show=True)

# Averaging Code & Results

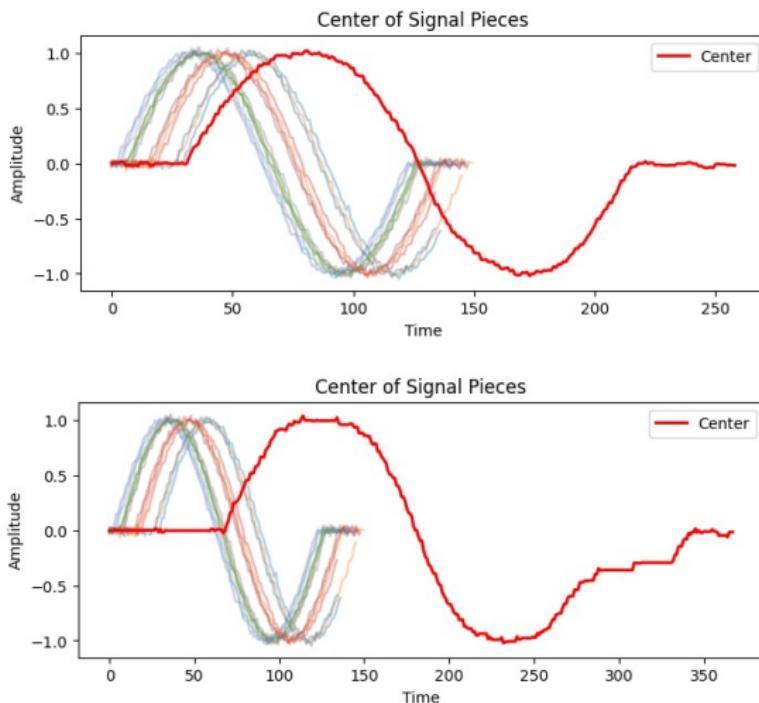


Figure: NLAAF1 and NLAAF2

# Averaging Continued

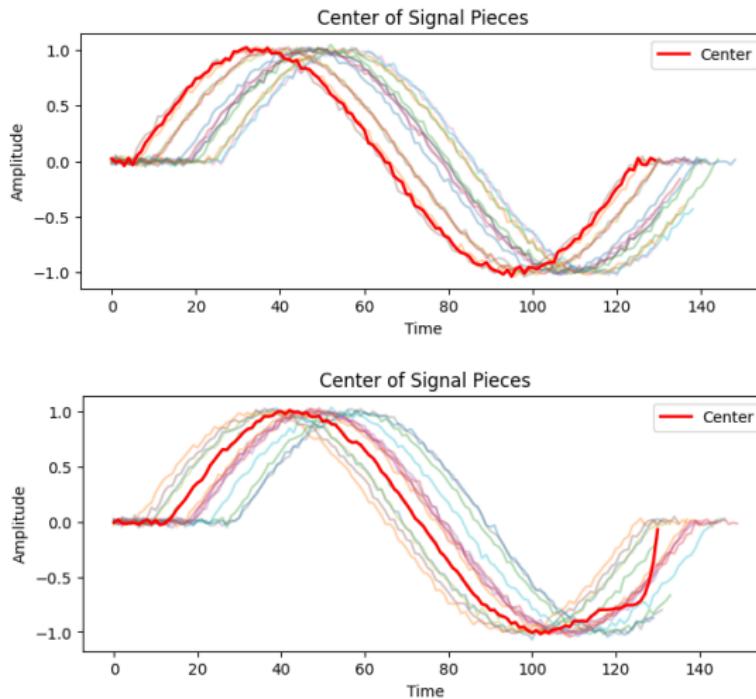


Figure: ICDTW and DBA

# Table of Contents

1 Signals and Noise

2 Filters

3 Decomposition

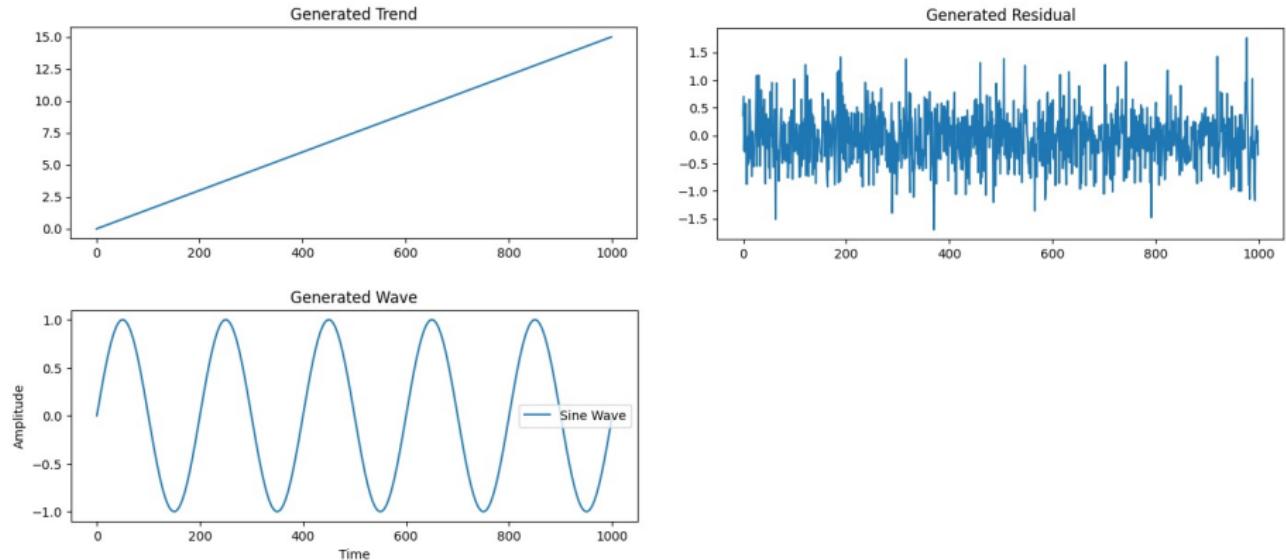
4 Time and Frequency Domain

5 Contact Information

# Decomposition

- ④ Decomposition in signal processing is the breakdown of a complex signal into simpler components.

# Generated Signals



**Figure:** Generated Signals to Combine for Decomposition

# Seasonal Decomposition

**Seasonal decomposition** is a technique used in time series analysis to break down a time series into its individual components: trend, seasonality, and residual (or noise). This helps to better understand the underlying structure of the data and is often used to detect patterns, forecast future values, and improve the accuracy of predictive models.

The primary goal of seasonal decomposition is to separate the time series into components that represent different underlying behaviors.

# Seasonal Decomposition Continued

The model can be expressed as  $y(t) = T(t) + S(t) + R(t)$ , where  $T(t)$  is the trend,  $S(t)$  is the seasonal component, and  $R(t)$  is the residual:

- ① **Trend:** The long-term movement or direction in the data. It represents the general tendency of the data to increase or decrease over time, ignoring short-term fluctuations.
- ② **Seasonality:** The repeating and predictable patterns or cycles that occur at regular intervals, such as daily, weekly, monthly, or yearly. This component captures fluctuations that recur over a fixed period.
- ③ **Residual:** The "leftover" noise or random fluctuations in the data that can't be explained by the trend or seasonal components. This component represents irregular variations, outliers, or random events.

# Seasonal Decomposition Code

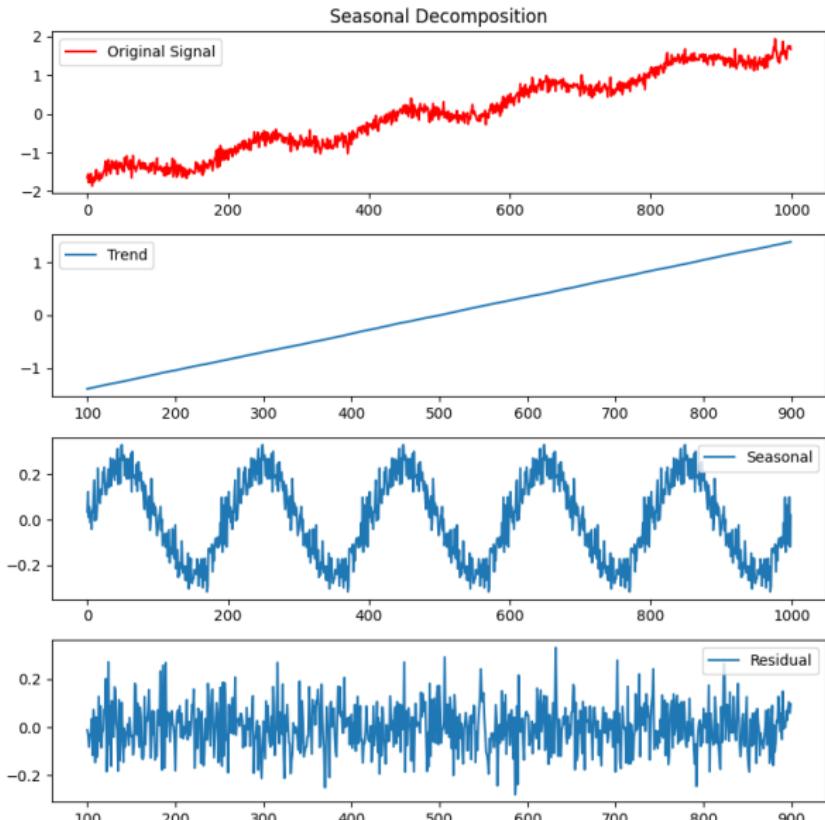
```
Trend = 1.5 * np.linspace(0, 10, 1000)
```

```
Seasonal = sinewave(duration = 10, samplingrate = 100, frequency = 0.5)
```

```
Residual = np.random.normal(0, 0.5, 10 * 100)
```

```
signal = Seasonal + Trend + Residual
```

```
_ = seasonal_decomposition(signal, period = 200, show = True)
```



**Figure:** Seasonal Decomposition: Original, Trend, Seasonal, and Residual

# Empirical Mode Decomposition (EMD)

Empirical Mode Decomposition (EMD) is a data-driven method for analyzing non-linear and non-stationary time series data. It decomposes a time series into a set of intrinsic mode functions (IMFs), which represent different oscillatory modes of the data. These IMFs are derived from the data itself, without relying on a pre-defined basis, making EMD particularly useful for analyzing complex, real-world signals that do not follow simple trends or seasonality.

EMD decomposes a signal into intrinsic mode functions (IMFs) based on local extrema. Each IMF represents a specific oscillatory mode. Mathematically, a signal  $x(t)$  is decomposed as  $x(t) = \sum_{i=1}^n C_i(t) + R(t)$ , where  $C_i(t)$  is the  $i$ -th IMF and  $R(t)$  is the residual.

# EMD Steps

- ➊ **Sifting Process:** The signal is first "sifted" to extract the high-frequency oscillations (IMFs) using the local maxima and minima. This involves:
  1. Identifying the local maxima and minima.
  2. Interpolating between the maxima and minima to create upper and lower envelopes.
  3. Subtracting the mean of the upper and lower envelopes from the signal to obtain the IMF.
  4. Repeating the process on the residual signal until the stopping criterion is met.
- ➋ **Iterative Extraction:** This process continues for several iterations. Each iteration extracts an IMF, and what remains after extracting each IMF is a progressively lower-frequency component of the signal. It stops when the residual signal is a monotonic function (or when other convergence criteria are met).
- ➌ **Final Residual:** The final residual is often considered the trend of the original time series, which has no oscillatory modes left.

# EMD Pros

- ① **Data-Driven:** EMD does not assume a pre-defined model or basis (like Fourier transforms or wavelets), making it highly flexible for complex, real-world signals.
- ② **Adaptability:** It works well for non-stationary, non-linear signals, making it suitable for a wide variety of applications, including those where other methods may struggle.
- ③ **No Assumptions:** Unlike methods such as Fourier or wavelet transforms, EMD does not require assumptions about the signal's behavior or frequency content.
- ④ **Multiscale Analysis:** It allows for multiscale decomposition, which helps to capture different frequency components of the signal.

# EMD Cons

- ① **Mode Mixing:** EMD can sometimes suffer from mode mixing, where different frequency components are incorrectly mixed together in the same IMF. This can occur if the signal is highly irregular or contains sharp discontinuities.
- ② **Sensitivity to Noise:** EMD can be sensitive to noise, especially for signals with small-scale variations.
- ③ **Computational Cost:** The iterative nature of the method can be computationally expensive for large datasets.

# EMD Code

```
emd_decomposition(signal, show=True)
```

# Generated Signals

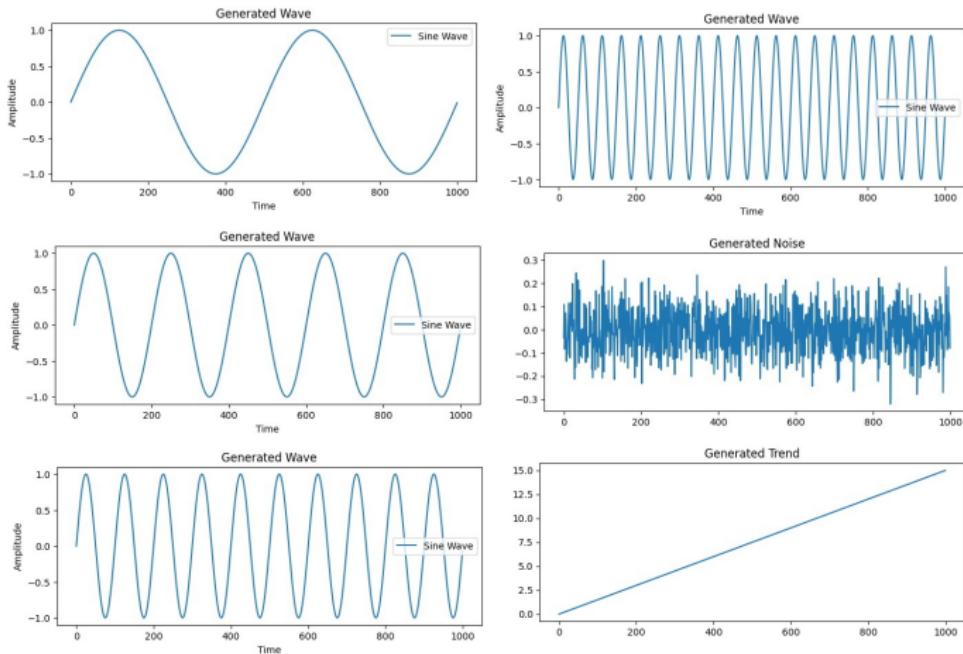


Figure: Generated Waves

# EMD Continued

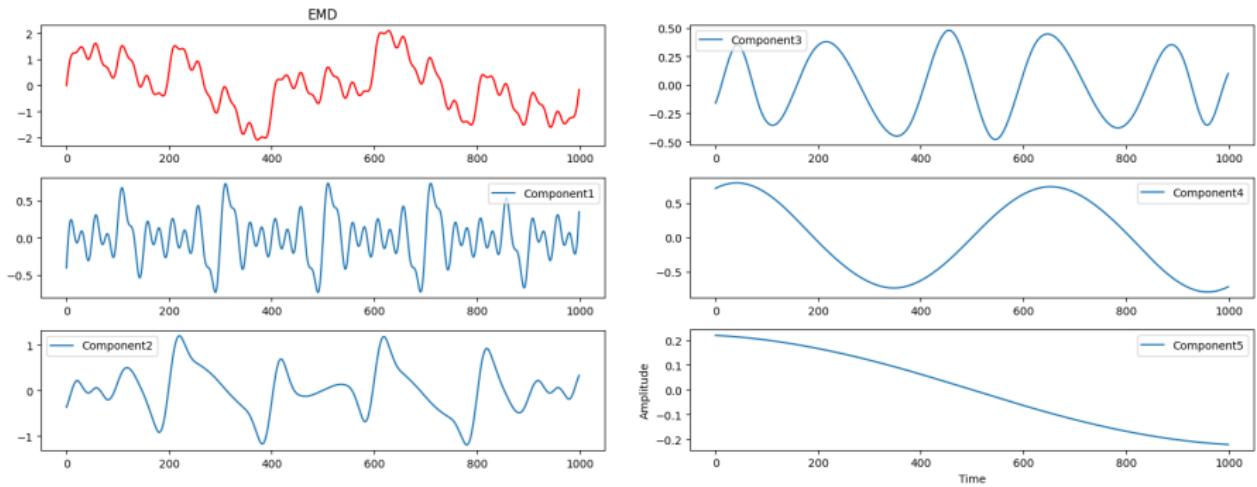


Figure: EMD

# Ensemble EMD

EEMD is an extension of EMD that addresses mode mixing by adding noise and performing multiple decompositions. It aims to improve the decomposition results by providing a more stable representation of the signal's components.

## Code:

```
eemd_decomposition(signal, show=True)
```

# EEMD Steps

- ① **Add White Noise:** White noise is added to the original signal. This introduces randomness into the signal, helping to avoid the bias caused by mode mixing and improving the decomposition process.
- ② **Decompose with EMD:** Perform Empirical Mode Decomposition on the noisy signal. This step results in a set of IMFs that are influenced by the added noise.
- ③ **Repeat the Process:** The process is repeated several times, each time adding a different realization of white noise to the signal. For each noisy signal, a separate EMD decomposition is performed, resulting in a different set of IMFs.
- ④ **Ensemble Averaging:** After completing the decompositions for all white noise realizations, the IMFs from each decomposition are averaged. The averaging process helps to cancel out the effects of the white noise, leaving behind a more accurate representation of the true intrinsic oscillations in the original signal.
- ⑤ **Final Decomposition:** The resulting IMFs, averaged over the ensemble of noisy realizations, form the final set of IMFs that are less influenced by noise and mode mixing.

# Advantages of EEMD Over EMD

- ① **Reduced Mode Mixing:** By adding white noise and performing the decomposition multiple times, EEMD mitigates the problem of mode mixing that can occur in traditional EMD. The averaging process helps to separate components more clearly into different frequency bands.
- ② **Noise Robustness:** The introduction of white noise in EEMD helps to "drown out" the effects of real-world noise, making the decomposition more stable and robust, especially in the presence of noise or irregularities in the data.
- ③ **More Reliable IMFs:** The ensemble averaging step leads to more stable and meaningful IMFs, as the noise and instability from individual decompositions are averaged out, resulting in IMFs that better represent the underlying dynamics of the signal.
- ④ **Better Handling of Complex Signals:** EEMD is better suited for dealing with complex, non-linear, and non-stationary signals, such as those found in biomedical data, mechanical systems, and financial data, where EMD alone may struggle.

# EEMD Continued

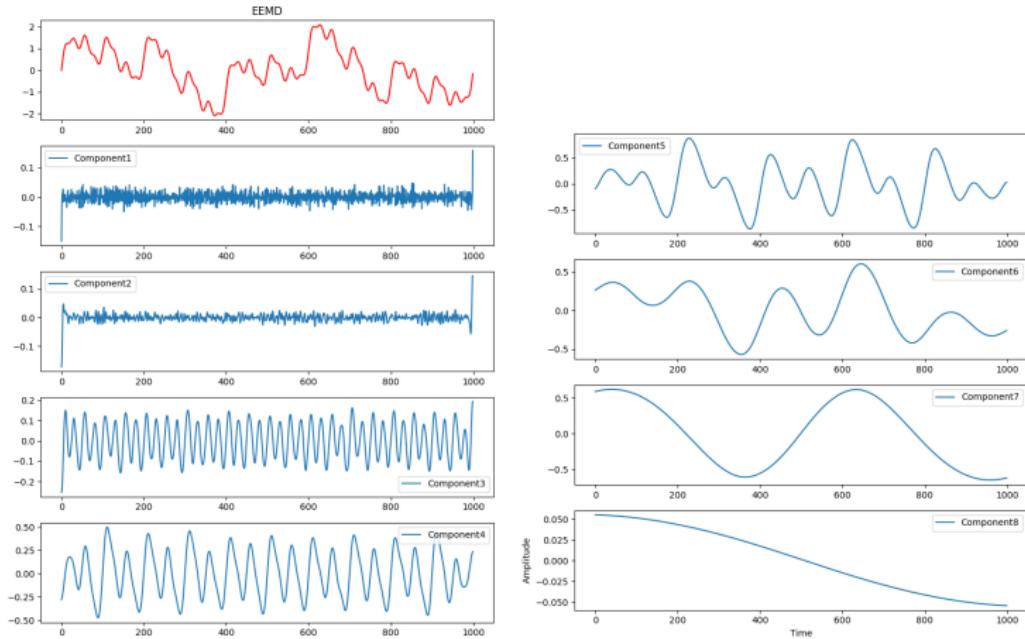


Figure: EEMD

# Complete EEMD

CEEMD is a further enhancement of EEMD that uses an ensemble of EMD decompositions with different white noise added at each iteration. It improves upon EEMD by reducing the residual error and enhancing the decomposition's accuracy.

**The steps for CEEMD are the same as EEMD, except for Add Noise:**  
Instead of adding just positive white noise (as in EEMD), both positive and negative white noise of equal amplitude are added to the signal. This makes the process more symmetric and helps to cancel out noise artifacts more effectively.

**Code:** `ceemd_decomposition(signal, show=True)`

# CEEMD Continued

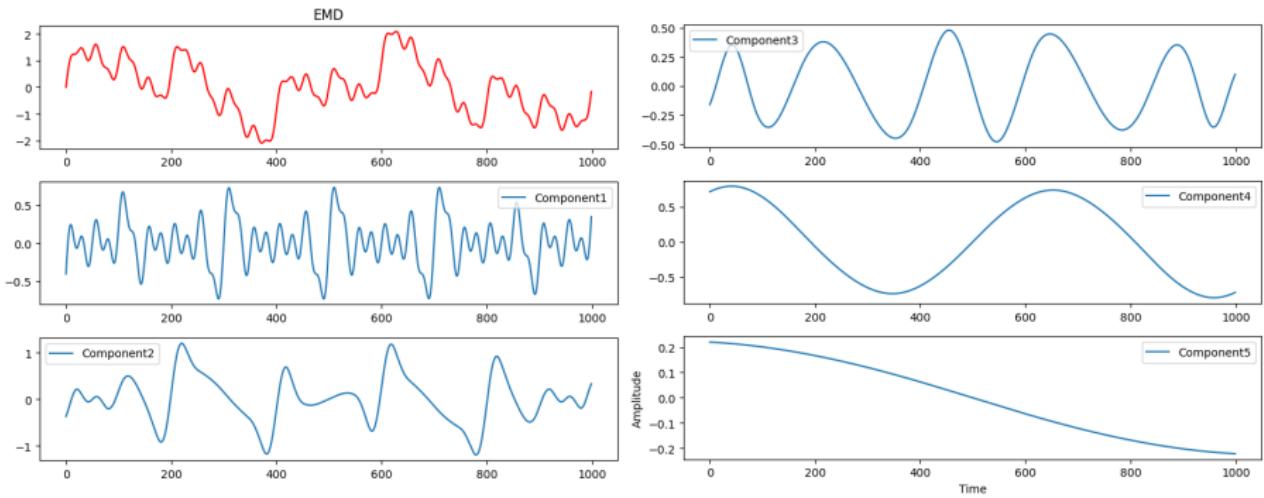


Figure: CEEMD

# Variational Mode Decomposition (VMD)

Variational Mode Decomposition (VMD) is a signal processing technique that is used to decompose a complex signal into a set of intrinsic modes or modes of oscillation. It is particularly useful for analyzing non-stationary and non-linear signals, and it has gained popularity as an alternative to methods like Empirical Mode Decomposition (EMD) and its variations (such as CEEMD).

VMD decomposes a signal into a set of oscillatory modes with varying frequencies. It minimizes the mode mixing problem by solving a variational optimization problem. The decomposition can be expressed as  $x(t) = \sum_{i=1}^n u_i(t) + R(t)$ , where  $u_i(t)$  are the modes and  $R(t)$  is the residual.

**Code:** `vmd_decomposition(signal, show=True)`

# VMD Steps

VMD is based on an optimization framework where the objective is to find a set of modes such that their sum approximates the original signal. Here's a step-by-step breakdown of how VMD works:

- 1. Signal Representation:** VMD takes a time-domain signal  $x(t)$  and aims to represent it as the **sum** of a set of modes  $u_k(t)$ , each of which has a specific frequency component. The modes are represented as functions with limited frequency bandwidths.
- 2. Frequency Localization:** Each mode is associated with a center frequency  $F_k$ , and the objective is to make the **frequency bandwidth** of each mode as narrow as possible. This is done by limiting the frequency range of each mode, which helps in extracting specific frequency components from the signal.

## VMD Steps Continued

**3. Variational Optimization:** VMD solves an optimization problem in the form of a variational principle to find the modes and their associated frequencies. The optimization problem seeks to minimize the following cost function:

$$\min_{u_k(t), f_k} \sum_{k=1}^K \left( \left| \frac{\delta}{\delta t} u_k(t) \right|^2 + \alpha |f_k - \hat{f}_k|^2 \right)$$

**Where:**

$u_k(t)$  is the  $k$ -th mode.

$f_k$  is the central frequency of the  $k$ -th mode.

$\hat{f}$  is the reference frequency.

$\alpha$  is a regularization parameter that controls the balance between bandwidth and frequency accuracy.

The first term in the cost function is the **bandwidth** of the mode, while the second term penalizes deviations from the desired central frequency.

# VMD Steps Continued

4. **Iterative Update:** The algorithm uses an iterative process to update the modes and their associated frequencies in order to minimize the cost function. This is done by alternating between updating the modes and adjusting the central frequencies until convergence is achieved.
5. **Reconstruction:** After obtaining the modes, the original signal can be reconstructed by summing all the extracted modes. The result is a signal representation in terms of different oscillatory components, each with a well-defined frequency range.

# VMD Pros

- ① **Precise Frequency Separation:** VMD ensures that the extracted modes correspond to distinct frequency bands, reducing mode mixing compared to traditional methods like EMD.
- ② **Noise Robustness:** VMD is more robust to noise and is less sensitive to high-frequency noise than methods like EMD, which can struggle in noisy environments.
- ③ **Flexible and Generalizable:** VMD works well for a wide range of non-linear and non-stationary signals, making it applicable across many domains like biomedical, financial, and geophysical analysis.

# VMD Cons

- ① **Parameter Sensitivity:** The performance of VMD depends on selecting the right parameters, such as the number of modes  $K$ , the regularization parameter  $\alpha$ , and the convergence tolerance. Incorrect parameter settings can lead to poor results.
- ② **Computational Complexity:** VMD requires solving an optimization problem for each mode, which can be computationally intensive for large datasets or real-time applications.

# Generated Waves

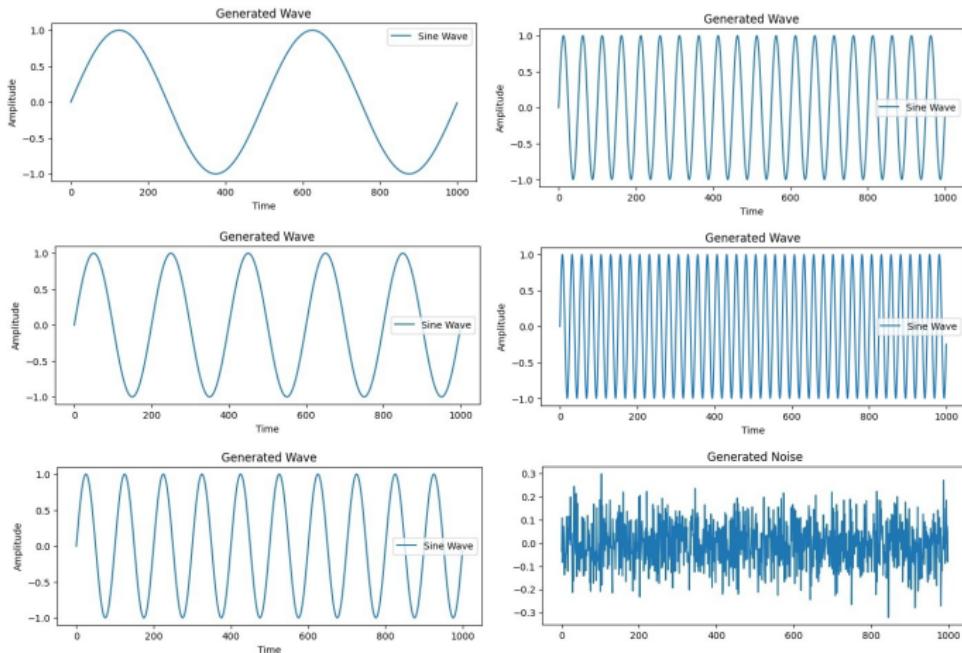


Figure: Generated Waves

# VMD Continued

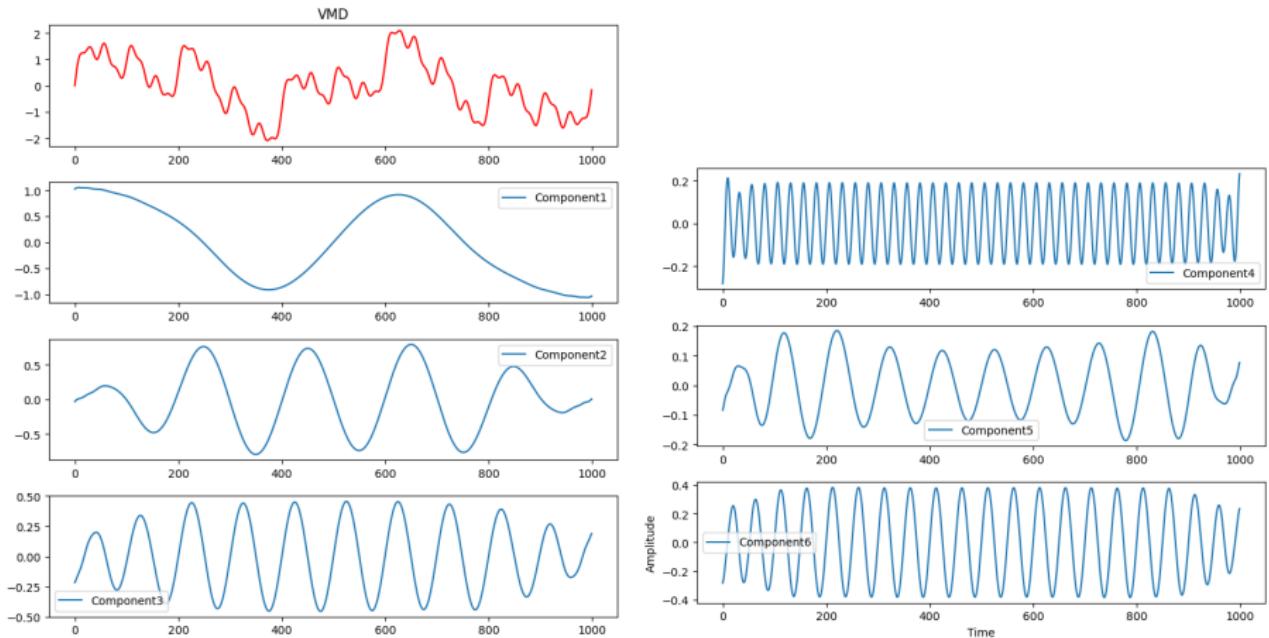


Figure: VMD

# Blind Source Separation (BSS)

BSS is a signal processing technique that aims to extract independent source signals from their observed mixtures. Two widely used methods for BSS are Principal Component Analysis (PCA) and Independent Component Analysis (ICA).

**PCA-Based Blind Source Separation** - PCA is employed to transform the observed mixed signals into a new set of uncorrelated variables called principal components. In the context of BSS, PCA can be applied to the covariance matrix of the observed signals. The principal components are ordered in terms of their variances, and by selecting a subset of these components, one can achieve a decorrelated representation of the mixed signals. However, PCA does not guarantee independence.

$C * v = \lambda * v$  where  $C$  is the covariance matrix,  $v$  is an eigenvector, and  $\lambda$  is the corresponding eigenvalue.

# Independent Component Analysis BSS

ICA aims to find a linear transformation of the observed signals such that the resulting components are statistically independent. The key assumption is that the sources are statistically independent.

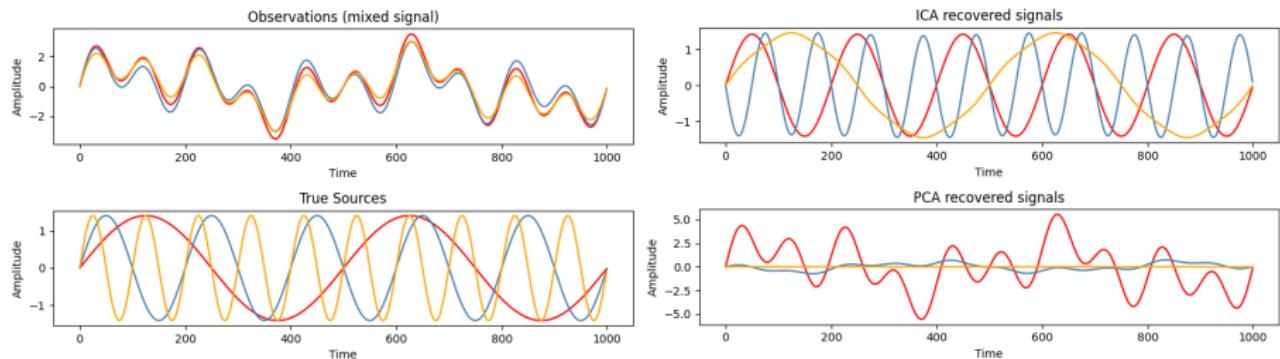
Mathematically, given a matrix representing the observed mixtures, ICA seeks a demixing matrix  $W$  such that  $S = WX$ , where  $S$  contains the estimated source signals.

The optimization problem in ICA is often formulated as maximizing the non-Gaussianity of the estimated sources. Using negentropy as a measure of non-Gaussianity is a common approach, leading to objective functions like:

$$J(W) = \sum_{i=1}^n [E\{G(u_i)\} - E\{G(v_i)\}]$$

where  $u_i$  is the  $i$ -th estimated source,  $v_i$  is the  $i$ -th component of the observed mixtures,  $G(\cdot)$  is a nonlinear function, and  $E\{\cdot\}$  denotes expectation.

# BSS Continued



**Figure:** BSS: Observations, True Sources, ICA Recovery, PCA Recovery

# Table of Contents

1 Signals and Noise

2 Filters

3 Decomposition

4 Time and Frequency Domain

5 Contact Information

# Time and Frequency Domain

- ④ Discussion of Time Domain Features
- ② Discussion of Frequency Domain Features
- ③ Discussion of Time-Frequency Domain Features

# Peak Detection & Envelope Extraction

**Peak detection** in signal processing refers to the process of identifying local maxima (peaks) or minima (troughs) in a signal. These peaks are important features in many applications, as they often correspond to significant events, changes, or patterns in the data.

**Envelope extraction** is a signal processing technique used to isolate the "envelope" of a signal, which represents the smooth curve that outlines the peaks of a modulated waveform. It's often applied to modulated signals, such as amplitude modulation (AM) in radio signals, to extract the slower-varying component of the signal (the envelope), while ignoring the higher-frequency oscillations.

# Definitions

- ① **Peak:** A data point is considered a peak if it is higher than its immediate neighbors. For instance, in a time series, a peak at time  $t$  satisfies:

$$x(t) > x(t-1) \text{ and } x(t) > x(t+1)$$

where  $x(t)$  is the value of the signal at time  $t$  and  $t-1$  and  $t+1$  are its neighboring values

- ② **Trough:** Similarly, a trough is a local minimum, where:

$$x(t) < x(t-1) \text{ and } x(t) < x(t+1)$$

- ③ **Local Peaks:** These are peaks within a small neighborhood of data points.

- ④ **Global Peaks:** These refer to the overall highest values within the entire signal.

- ⑤ **Thresholding:** A peak might only be considered significant if it exceeds a certain threshold value, which helps filter out smaller, less relevant peaks from noisy data.

# Peaks, Valleys, & Envelopes

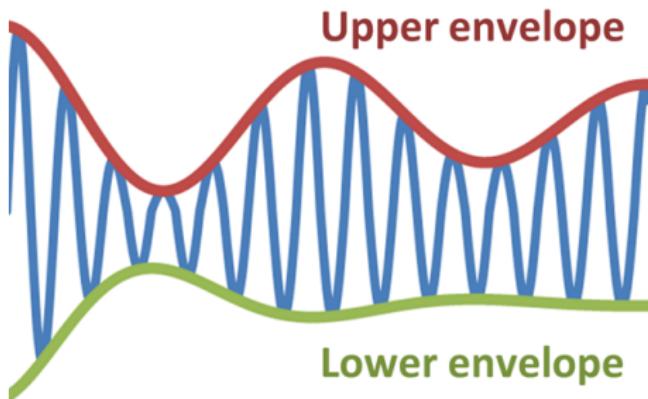


Figure: Peaks, Valleys & Envelopes

Image from Wikipedia

# Peaks

get\_peaks(signal)

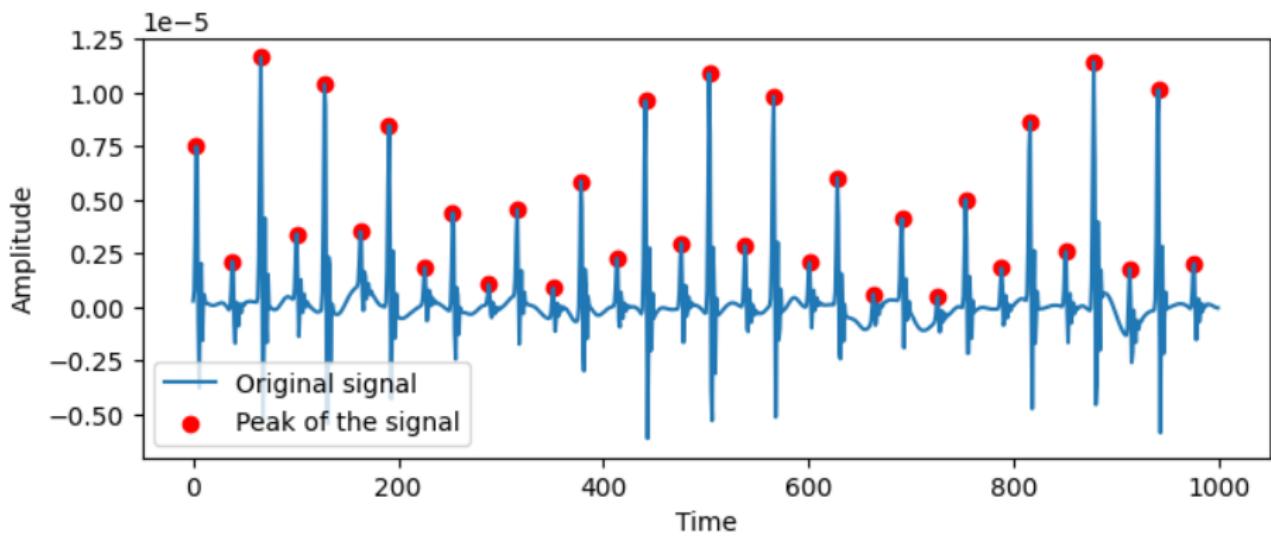


Figure: Get Peaks

# Envelope Detection

```
envelope_from_peaks(signal)
```

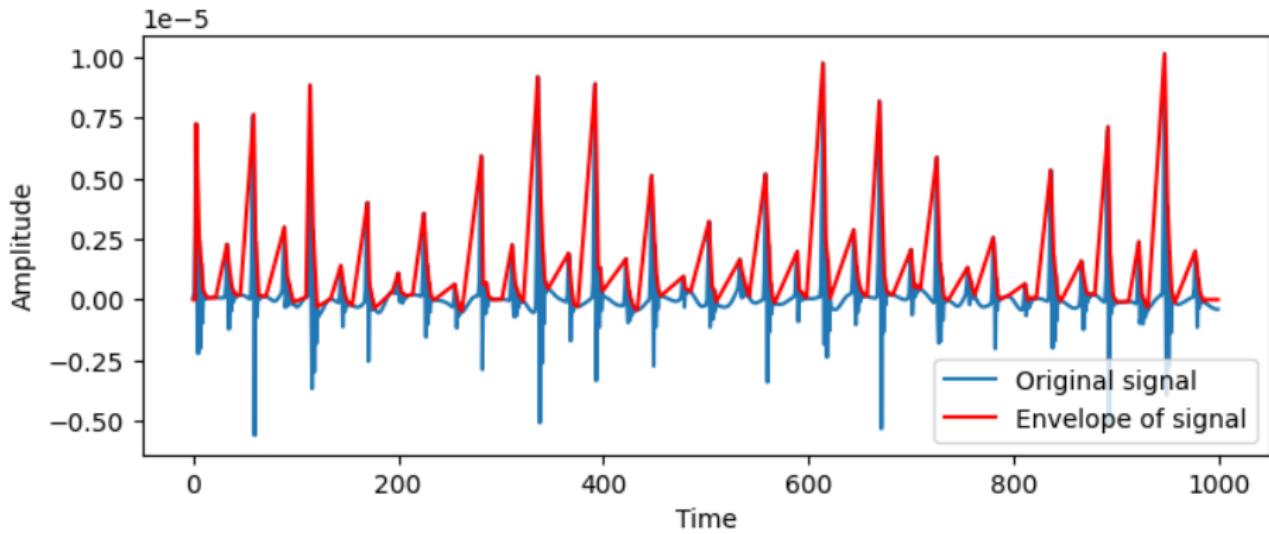


Figure: Get Envelope from Peaks

## Average Envelope

We can also use the average window to get the envelope. The average window will calculate the average value of a region to replace the middle timestamp in the region. The longer the window, the smoother the envelope, but the more serious it will change the shape of the original signal.

$$\begin{aligned} envelope[i] = & \ signal[i - \frac{window\_length}{2}] + \dots + signal[i] + \\ & \dots + signal[i + \frac{window\_length}{2}] \end{aligned}$$

**Usage Scenario:** It is suitable for signals with relatively stable periodicity and slow amplitude variations. This is because the moving average window can eliminate high-frequency noise while preserving the low-frequency characteristics of the signal. However, this envelope will lose the information of peaks' vertical location.

# Average Envelope Continued

average\_envelope(signal, 3)

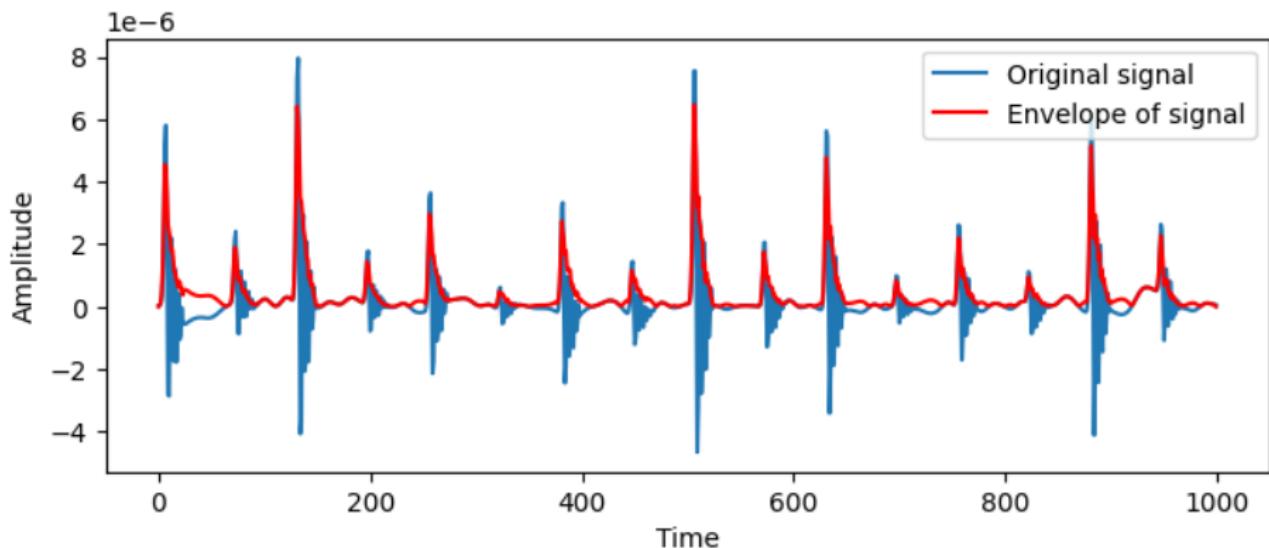


Figure: Get Average Envelope

# Hilbert Transformation - Envelope & Phase Extraction

- ① If a modulated signal is expressed as  $x(t) = a(t)\cos[\phi(t)]$
- ② The instantaneous amplitude or the envelope of the signal is given by  $a(t)$
- ③ The instantaneous phase is given by  $\phi(t)$
- ④ The instantaneous angular frequency is derived as  $\omega(t) = \frac{d}{dt}\phi(t)$
- ⑤ The instantaneous temporal frequency is derived as  $f(t) = \frac{1}{2\pi} \frac{d}{dt}\phi(t)$

We note that the modulated signal is a real-valued signal. We also take note of the fact that amplitude/phase and frequency can be easily computed if the signal is expressed in complex form. So we can use Hilbert transformation to transform the real-valued signal to a complex version.

# Analytic Signal

An **analytic signal** is a complex-valued signal that is derived from a real-valued signal through the Hilbert transform. It provides a convenient way to represent both the original signal and its instantaneous amplitude and phase. The analytic signal is particularly useful in signal processing for analyzing modulated signals, extracting instantaneous frequency, and performing various types of signal demodulation.

# Hilbert Transformation Continued

The analytic signal is

$$z(t) = z_r(t) + z_i(t) = x(t) + jHT\{x(t)\}$$

So

$$a(t) = |z(t)| = \sqrt{z_r^2(t) + z_i^2(t)}$$

$$\phi(t) = \angle z(t) = \arctan \left[ \frac{z_i(t)}{z_r(t)} \right]$$

$$f(t) = \frac{1}{2\pi} \frac{d}{dt} \phi(t)$$

where  $z(t)$  denotes the analytic signal, the subscripts  $i$  and  $r$  mean the imaginary and real,  $j$  is an imaginary unit, and  $HT\{\}$  denotes Hilbert transform.

# Envelope Extraction with Hilbert, Steps: Transform

- ➊ **Hilbert Transform:** Given a real-valued signal  $x(t)$ , you first compute its Hilbert transform  $\hat{x}(t)$ . This transform produces a signal that is the **quadrature (90-degree phase-shifted)** version of the original signal.
- ➋ **Construct the Analytic Signal:** The analytic signal  $z(t)$  is formed by combining the original signal  $x(t)$  and the Hilbert transform  $\hat{x}(t)$  into a complex signal:

$$z(t) = x(t) + j\hat{x}(t)$$

where  $j$  is the imaginary unit.

- ➌ **Envelope Extraction:** The envelope of the original signal  $x(t)$  is the magnitude of the analytic signal  $z(t)$ . This is calculated as:

$$\text{Envelope}(t) = |z(t)| = \sqrt{x(t)^2 + \hat{x}(t)^2}$$

This envelope provides a smooth curve that represents the signal's varying amplitude.

# Hilbert Transformation Continued

inst\_amplitude(signal)

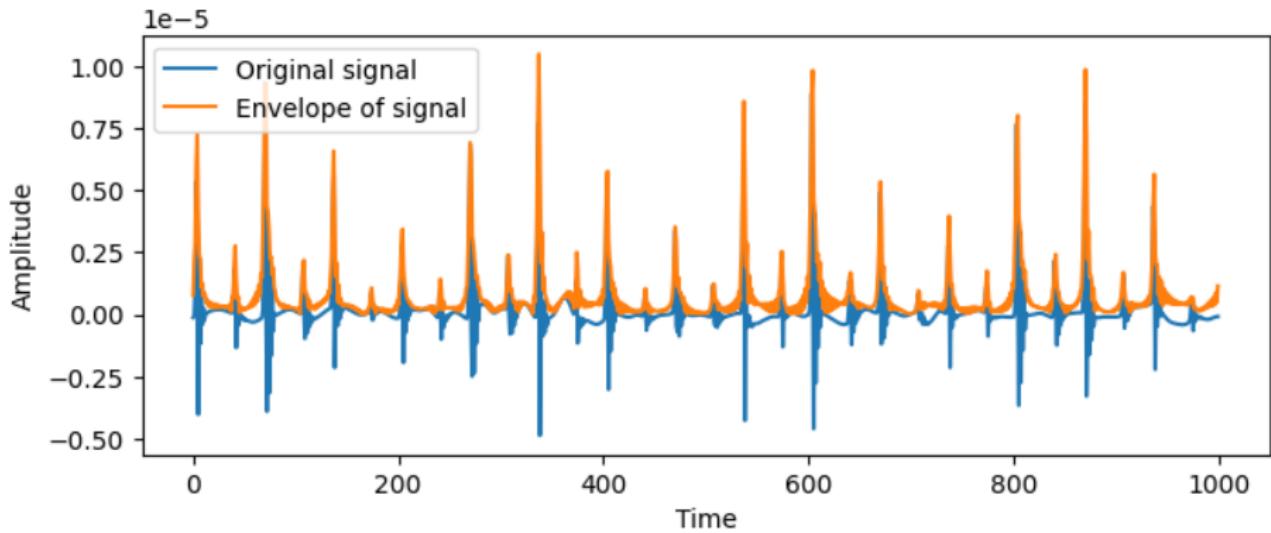


Figure: Envelope from Hilbert Transform

# SSA and SST

**Singular Spectrum Analysis (SSA)** and **Singular Spectrum Transform (SST)** are both time series analysis techniques that utilize concepts from **Singular Value Decomposition (SVD)** and matrix factorization. While they share some similarities in their foundational concepts, there are key differences in their implementation, goals, and applications.

# SSA: Singular-Spectrum Analysis

**SSA** is a method used to analyze time series data by reconstructing the time series in a higher-dimensional space. SSA focuses on decomposing the time series into a set of additive components, such as trends, oscillations, and noise. The goal is often to extract useful patterns from the data, such as periodic trends, or to denoise the signal.

# SSA Steps

- ① **Embedding:** The time series is embedded into a trajectory matrix which involves creating a matrix by sliding a window across the time series, where each row of the matrix corresponds to a segment of the time series.
- ② **Singular Value Decomposition (SVD):** Perform SVD on the trajectory matrix. The decomposition produces singular values and singular vectors. The singular values indicate the importance of each component in explaining the variance of the data.
- ③ **Component Extraction:** The singular vectors corresponding to large singular values are assumed to represent meaningful features (like trends or oscillations). The components with smaller singular values are considered noise.
- ④ **Reconstruction:** A key feature is the ability to reconstruct the original signal from a subset of the components (i.e., by keeping the largest singular values or specific singular vectors).

# Singular Value Decomposition

The SVD of a matrix  $X$  (of size  $m \times n$ ):

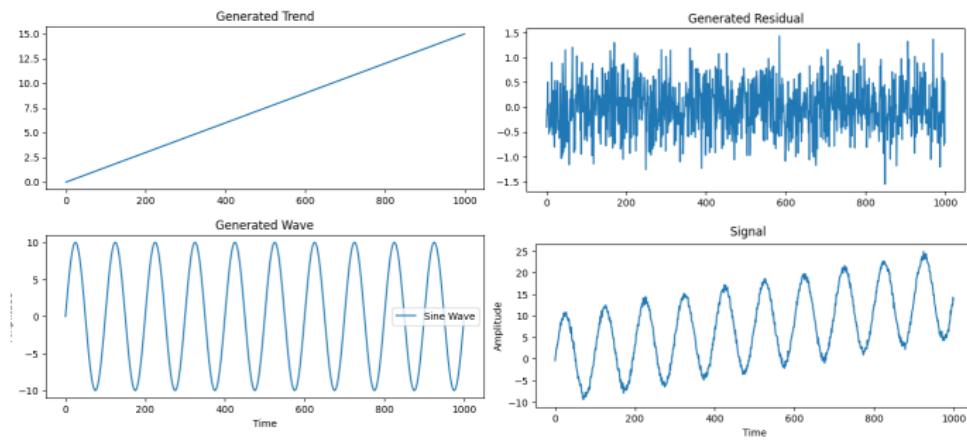
$$X = U\Sigma V^T$$

Where:

- ①  $U$ : An  $m \times m$  orthogonal matrix whose columns are the left singular vectors of  $X$ .
- ②  $\Sigma$ : A diagonal  $m \times n$  containing the singular values of  $X$  in descending order.
- ③  $V^T$ : The transpose of an  $n \times n$  orthogonal matrix, where the columns are the right singular vectors of  $X$ .

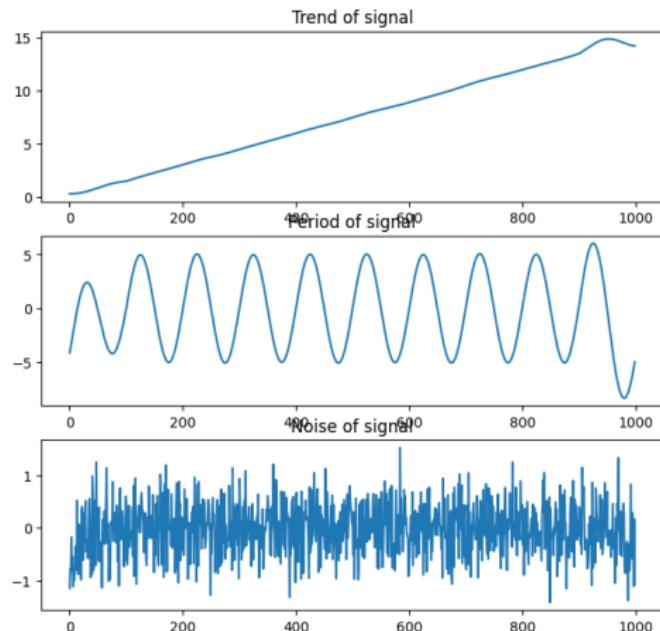
**Reference:** [Singular Value Decomposition \(SVD\)](#)

# SSA Continued



**Figure:** SSA: Generated Trend, Period, Noise, & Combined signals

# SSA Continued



**Figure:** SSA: Trend, Period, Noise

# SST: Singular Spectrum Transform

**Singular Spectrum Transform (SST)** is an extension or variation of SSA. SST also involves the creation of a trajectory matrix and performs SVD, but it typically focuses more on transforming the data to extract features and reconstructing it in a manner that emphasizes certain components of the time series. SST is particularly associated with nonlinear transformations of the time series and is often employed for feature extraction or dynamics analysis.

# SST Steps

- ① **Trajectory Matrix Construction:** Similar to SSA, SST starts with the creation of a trajectory matrix by embedding the time series using a sliding window approach.
- ② **SVD:** As with SSA, SVD is applied to the trajectory matrix to obtain the singular values and singular vectors.
- ③ **Feature Extraction:** The singular vectors corresponding to the largest singular values (i.e., the dominant components) are often kept for reconstruction. In SST, the focus might be on emphasizing certain features (such as the largest singular values) or transforming the signal in a way that highlights those features.
- ④ **Reconstruction:** SST involves reconstructing the signal by keeping only certain singular vectors and values, often with the intention of emphasizing certain periodic or structural components. This is similar to SSA, but SST might also involve additional transformations or modifications to the components.

# SST Steps

- ① **Embedding:** Convert the original signal into matrix form. Usually via a Hankel matrix where rows and columns consist of subsequences of the original signal..
- ② **Singular Value Decomposition (SVD):** Perform singular value decomposition on the embedded matrix, breaking it down into three matrices:  $U$ ,  $\Sigma$ , and  $V^T$ . Here,  $U$  contains the left singular vectors,  $\Sigma$  contains the singular values in a diagonal matrix, and  $V^T$  contains the right singular vectors.
- ③ **Grouping and Reconstruction:** Group the singular values based on their magnitudes into several subsequences. These subsequences correspond to different frequency and trend components of the signal. By selecting relevant combinations of singular values, different components of the original signal can be reconstructed.
- ④ **Back-Transformation:** Perform the inverse transformation on the decomposed subsequences to obtain an estimate of the original signal. This step involves reversing the embedding operation on the reconstructed subsequences.

# Fast SST

**fastsst** is a Python package that provides a fast implementation of the SST algorithm, which is used for change point detection in time series data. It leverages efficient numerical methods to speed up the computation of the SST, making it suitable for analyzing large datasets.

**Note:** This tutorial uses fastsst in the examples.

## Reference

Change Point Detection Fast SST uses change point detection to generate an anomaly score, which can be used for anomaly detection.

Change point detection is a method used to identify points in a time series where the underlying behavior or characteristics of the data have changed significantly, often indicating a shift in the data-generating process.

In Fast SST, the anomaly score is calculated by measuring the deviation of the signal at each time step from the expected behavior of the singular values (SVs).

Using Fast SST's anomaly scores, a threshold could be set to detect anomalies.

# SST Results

Aspect	Singular Spectrum Analysis (SSA)	Singular Spectrum Transform (SST)
<b>Goal</b>	Decompose and extract components of the time series (trends, oscillations, noise)	Emphasize or transform certain features of the time series, typically focusing on dominant components
<b>Focus</b>	Primarily on <b>decomposition</b> and <b>reconstruction</b> to isolate trends and noise	Focus on <b>feature extraction</b> and often <b>nonlinear transformations</b> for dynamic system analysis
<b>Signal Reconstruction</b>	Reconstructs the signal by selecting important components based on SVD	Reconstructs the signal, often with an emphasis on dominant features or dynamics
<b>Application Areas</b>	Denoising, trend extraction, time series forecasting	Nonlinear signal processing, feature extraction, chaotic systems
<b>Complexity</b>	Often simpler and more focused on basic time series features	May involve more complex transformations for extracting dynamic or nonlinear features
<b>Use of Singular Values</b>	Singular values indicate the relative importance of components in the time series	Singular values are used similarly but often with more emphasis on transforming the data
<b>Signal Components</b>	Extracts <b>linear</b> components such as trends and periodicities	Can extract both <b>linear</b> and <b>nonlinear</b> components (especially for dynamic systems)

Figure: SSA vs. SST

# SSA vs. SST

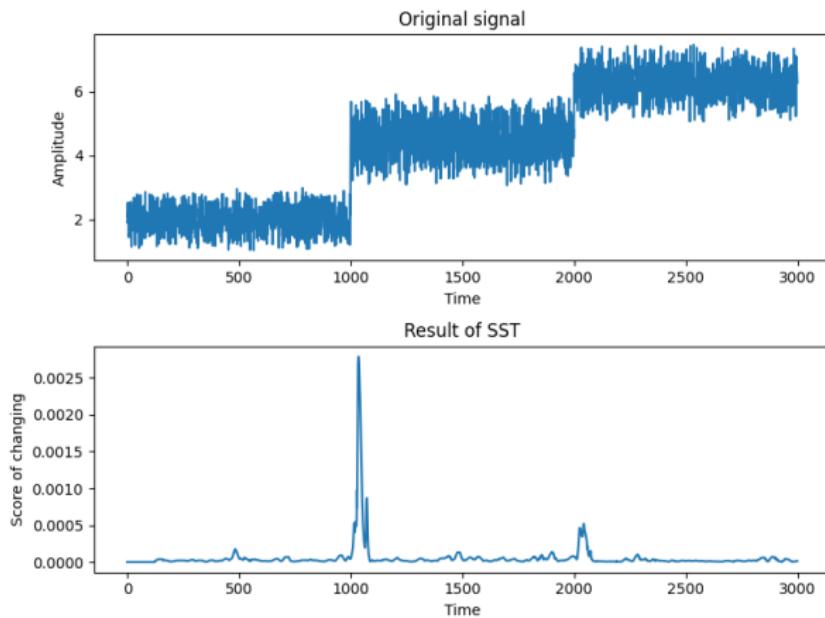


Figure: SST Results

# Petrosian Fractal Dimension (PFD)

The **Petrosian Fractal Dimension (PFD)** is a measure used to quantify the complexity or irregularity of a fractal or a signal. It specifically assesses the fractal dimension of a set or a signal, particularly in the context of time-series data or image analysis.

The fractal dimension is a mathematical concept used to describe the complexity of a fractal object—essentially, it measures how much detail the object has at different scales. For instance, a smooth curve has a fractal dimension of **1**, while a more jagged and intricate curve may have a dimension between **1** and **2**.

## Petrosian Fractal Dimension (PFD) Overview:

- ① PFD is often used in the analysis of complex signals or time-series data where the complexity or roughness of the data is of interest.
- ② Unlike other fractal dimension techniques (like the box-counting method), PFD is designed to be computationally efficient and works well even for data sets with limited resolution.

# PFD continued

## Characteristics:

- ① The Petrosian Fractal Dimension provides a value between **0** and **2** for most datasets. A value closer to **1** suggests a signal with moderate complexity, while a value closer to **2** indicates a more intricate and rough signal or pattern.
- ② It is less sensitive to scale changes and resolution compared to other methods like the box-counting method, making it advantageous for certain applications where computational efficiency is needed.

# PFD continued

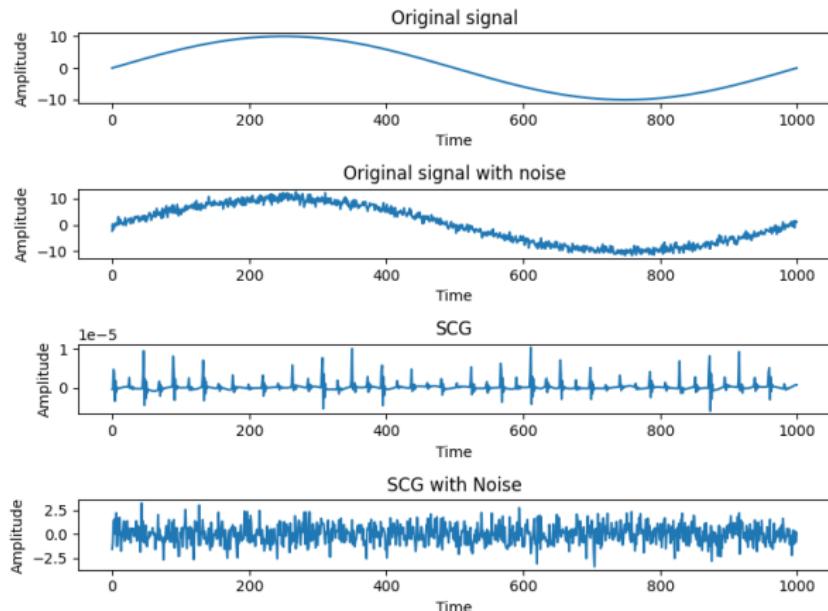
**PFD is defined as follows:**

$$PFD = \frac{\log_{10}(N)}{\log_{10}(N) + \log_{10}\left(\frac{N}{N+0.4N_{zc}}\right)}$$

where  $N$  is the length of the signal and  $N_{zc}$  is the number of zero crossings in the signal derivative.

# PFD Continued

`pfd(signal1)); pfd(signal2)); pfd(signal3)); pfd(signal4))`



**Figure: PFD Scores**

# PFD Scores

- ① Wave Signal: **1.0001157789650048**
- ② Wave Signal with noise: **1.0361347362709525**
- ③ SCG: **1.022166220796765**
- ④ SCG with Noise: **1.0349099215682838**

# Statistical Moments

In statistics, **moments** are quantitative measures used to describe the shape and characteristics of a distribution or dataset. The concept of moments is fundamental in probability theory, where each moment provides insights into different aspects of a probability distribution. The most common moments are the **mean**, **variance**, **skewness**, and **kurtosis**, but moments can also be higher-order. They provide a concise way to summarize the distribution of data.

# Statistical Moments continued

Moments are mathematical expressions derived from the distribution or dataset. The **n-th moment** of a random variable (or dataset) is defined as the expected value of the **n-th power of the deviation from the mean**. The general formula for the n-th moment  $\mu_n$  about the mean is:

$$\mu_n = E[(X - \mu)^n]$$

**Where:**

- ①  $\mu$  is the mean (or expected value) of the random variable
- ②  $E$  denotes the expected value (or the mean of the distribution),
- ③  $X$  is the random variable,
- ④  $n$  is the order of the moment (e.g., 1st moment, 2nd moment, etc.).

The **n-th moment** captures various properties of the distribution's shape.

# Mean: the First Moment

The mean or expected value ( $\mu$ ) of a random variable is the first moment about the origin. It provides a measure of the "center" or "average" value of a distribution. Mathematically:

$$\mu_1 = E[X] = \sum_i p(x_i) \cdot x_i$$

**Interpretation:** The mean gives the central location of the data or distribution. For a dataset, it's the average value.

# Variance: the Second Moment

The **second moment** about the mean is the **variance** ( $\sigma^2$ ). It describes the spread or dispersion of the data around the mean. The formula for variance is:

$$\mu_2 = E[(X - \mu)^2] = Var(X)$$

**Interpretation:** Variance measures how far data points typically deviate from the mean. A larger variance indicates a wider spread in the data.

The **standard deviation** is simply the square root of the variance and provides a more interpretable measure of spread:

$$\sigma = \sqrt{Var(X)}$$

# Skewness: the Third Moment

The third moment is related to **skewness**, which measures the asymmetry of the distribution around the mean. If the skewness is:

- ① **Positive:** The distribution has a long right tail (right-skewed).
- ② **Negative:** The distribution has a long left tail (left-skewed).

Mathematically, skewness is defined as:

$$\mu_3 = E[(X - \mu)^3]$$

The skewness is normalized by dividing by the cube of the standard deviation, so it's dimensionless:

$$Skewness = \frac{\mu_3}{\sigma^3}$$

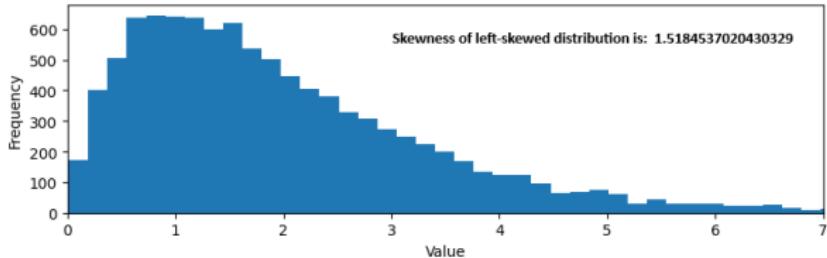
**Interpretation:** Skewness indicates whether the data is symmetrically distributed. A skewness close to zero suggests the distribution is nearly symmetric.

# Skewness Continued

`skew(array1); skew(array2)`

Example of Skewness

The left-skewed distribution



The right-skewed distribution

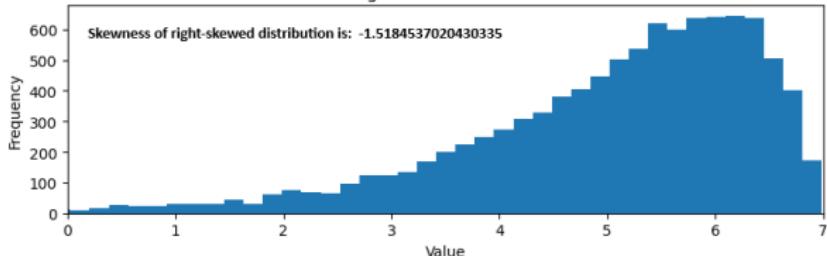


Figure: Skewness: Left Skewed & Right Skewed

# Kurtosis: the Fourth Moment

The fourth moment is related to **kurtosis**, which measures the "tailedness" of the distribution. It tells us whether the data has heavy tails (more extreme values) or light tails (less extreme values) compared to a normal distribution.

The formula for kurtosis involves the fourth central moment:

$$\mu_4 = E[(X - \mu)^4]$$

The kurtosis is normalized by dividing by the fourth power of the standard deviation, so it's dimensionless:

$$Kurtosis = \frac{\mu_4}{\sigma^4}$$

# Kurtosis Continued

## Interpretation:

- ① **Leptokurtic (Kurtosis > 3):** Distribution has heavy tails, meaning extreme values (outliers) are more likely.
- ② **platykurtic (Kurtosis < 3):** Distribution has light tails, meaning extreme values are less likely.
- ③ **Mesokurtic (Kurtosis = 3):** Distribution has the same tail behavior as the normal distribution.

For excess kurtosis, we subtract 3 from the kurtosis value:

$$\text{Excess Kurtosis} = \text{Kurtosis} - 3$$

**Note:** The reason we subtract 3 is that the normal distribution has a kurtosis of 3, so excess kurtosis normalizes this value to 0 for a normal distribution. This makes the interpretation of kurtosis easier.

# Kurtosis Continued

`kurtosis(array3); kurtosis(array4)`

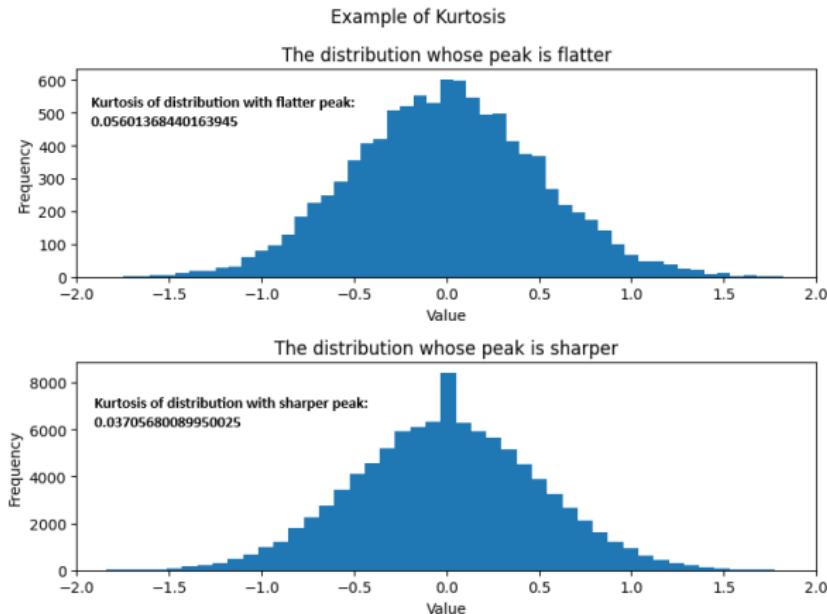


Figure: Kurtosis: Flat & Sharp

# Higher Order Moments

Moments higher than the fourth are also possible, though they are less commonly used. The **n-th moment** for  $n > 4$  will describe even more subtle characteristics of the distribution, though their interpretation can become increasingly abstract.

# Moments Overview

Moment	Name	Description	Formula for nth Moment (about the mean)
$\mu_1$	Mean (Expected Value)	The average or "center" of the distribution.	$\mu_1 = \mathbb{E}[X]$
$\mu_2$	Variance	Measure of the spread of the distribution around the mean.	$\mu_2 = \mathbb{E}[(X - \mu)^2] = \text{Var}(X)$
$\mu_3$	Skewness	Measure of the asymmetry of the distribution around the mean.	$\mu_3 = \mathbb{E}[(X - \mu)^3]$
$\mu_4$	Kurtosis	Measure of the "tailedness" of the distribution.	$\mu_4 = \mathbb{E}[(X - \mu)^4]$
Higher $\mu_n$	Higher Moments	Moments beyond the fourth capture more detailed features, but interpretation becomes less intuitive.	$\mu_n = \mathbb{E}[(X - \mu)^n]$

Figure: Kurtosis: Statistical Moments Overview

# Fast Fourier Transform (FFT)

The **Fast Fourier Transform (FFT)** is an efficient algorithm used to compute the **Discrete Fourier Transform (DFT)** of a signal, or its inverse. The DFT is a mathematical technique used to analyze the frequency content of discrete signals.

The **Fourier Transform (FT)**, is a mathematical operation that decomposes a signal into a sum of sinusoidal components. The Fourier Transform converts the signal from the time domain to the frequency domain, revealing the frequencies present in the signal.

- ① The continuous Fourier transform is used for continuous signals, but in practice, discrete signals are used.
- ② The Discrete Fourier Transform (DFT) is used for discrete, finite-length sequences of data.

# FFT Mathematics

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N}$$

$$x[n] = \frac{1}{2\pi} \sum_{k=0}^{N-1} X[k]e^{j2\pi kn/N}$$

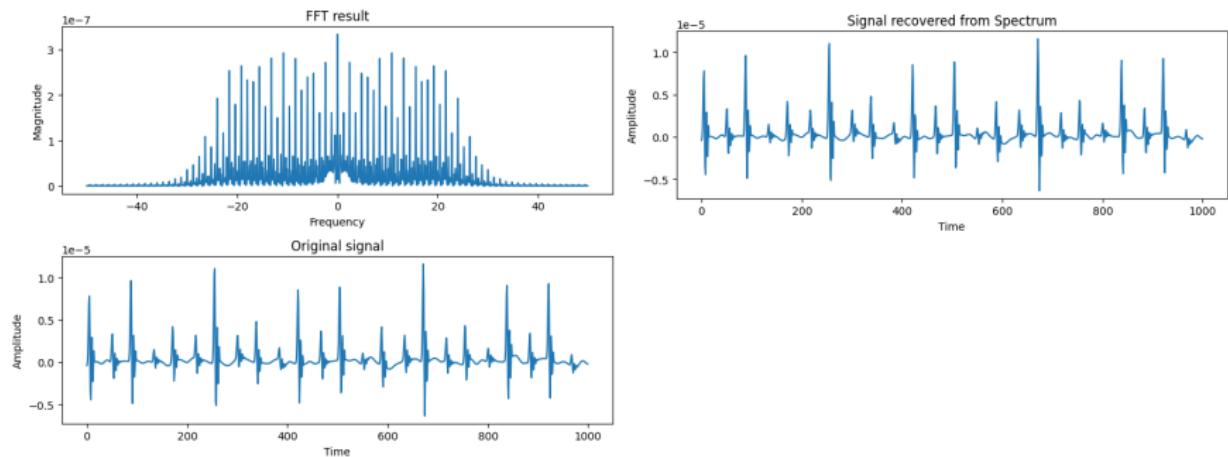
## Where:

- ①  $x[n]$  is the input sequence (time-domain signal)
- ②  $X[k]$  is the frequency-domain representation (Fourier coefficients)
- ③  $N$  is the length of the sequence (number of samples)
- ④  $k$  represents the frequency bin (each  $k$  corresponds to a different frequency)
- ⑤  $e^{-j2\pi kn/N}$  the complex exponential that represents sinusoidal waves at different frequencies.

The DFT produces complex numbers that represent both the amplitude and phase of different frequency components in the signal.

# FFT Results

```
my_fft(signal, fs); my_ifft(mag)
```



**Figure: FFT Results**

# FFT Denoising

**FFT denoising** employs the Fast Fourier Transform (FFT) to analyze and filter out unwanted frequency components from a signal. By transforming the signal into the frequency domain, one can selectively remove or attenuate specific frequency bands associated with noise. Mathematically, denoising is achieved by zeroing or attenuating certain frequency components in the Fourier-transformed signal, followed by an inverse FFT to obtain the denoised signal.

# FFT: How it Works

- ① **Transform the Signal to the Frequency Domain:** Use the Fast Fourier Transform (FFT) to convert the time-domain signal into the frequency domain. The FFT decomposes the signal into its frequency components.
- ② **Identify the Noise:** In the frequency domain, noise is typically present in the higher-frequency bins, especially if the noise is high-frequency (e.g., white noise). Depending on the signal, this can vary.
- ③ **Filter the Frequencies:** To remove the noise, we apply a low-pass filter (or sometimes a band-pass or band-stop filter, depending on the situation) that reduces or eliminates the high-frequency components that are considered noise while retaining the low-frequency components that are considered the signal.
- ④ **Inverse FFT:** After filtering out the noise, we apply the Inverse Fast Fourier Transform (IFFT) to convert the filtered signal back to the time domain. This results in a denoised version of the original signal.

# FFT Steps

1. **Apply FFT** to the noisy signal to obtain its frequency components.
2. **Thresholding:** Identify and remove (or reduce) the frequency components that represent noise. This can be done by:
  - ① Setting a threshold frequency (cutoff), below which we retain the components and above which we reduce or remove the components.
  - ② Alternatively, use soft thresholding or hard thresholding techniques to selectively suppress small components.
3. **Apply IFFT** to get the denoised signal back in the time domain.

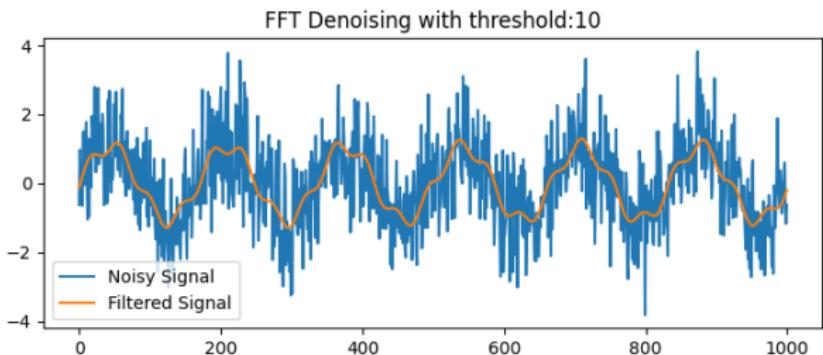
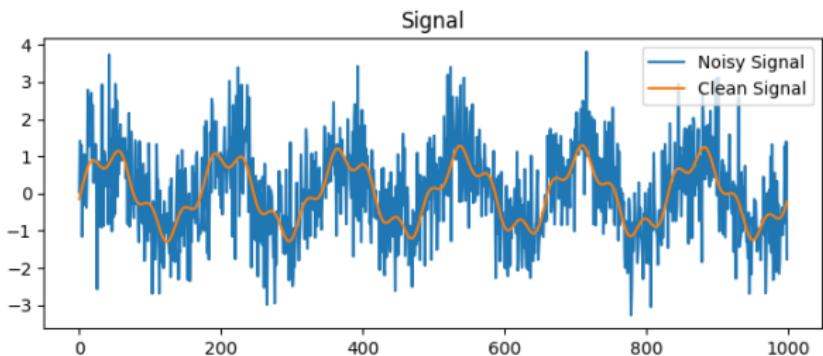


Figure: FFT Denoising

# Power Spectral Density (PSD)

**Power Spectral Density (PSD)** is a measure that describes how the power of a signal or time series is distributed over various frequency components. It is a fundamental concept in signal processing, used to analyze the frequency characteristics of a signal, particularly in fields like telecommunications, audio processing, and vibration analysis.

In simple terms, PSD tells us how the power (or energy) of a signal is spread across different frequencies.

# PSD Mathematics

**PSD** can be computed by taking the squared magnitude of the Fourier transform of the signal. Specifically, for a time-domain signal  $x(t)$ , the PSD  $S_x(f)$  at frequency  $f$  is calculated as:

$$S_x(f) = \lim_{T \rightarrow \infty} \frac{1}{T} \left| \int_{-T/2}^{T/2} x(t) e^{-j2\pi ft} dt \right|^2$$

**Where:**

- ①  $S_x(f)$  is the PSD of  $x(t)$
- ②  $f$  is the frequency in Hz
- ③  $T$  is the period
- ④ The integral represents the **Fourier Transform** of  $x(t)$
- ⑤ The magnitude of the Fourier Transform squared gives the power at that frequency.

# PSD Results

`psd(signal, fs)`

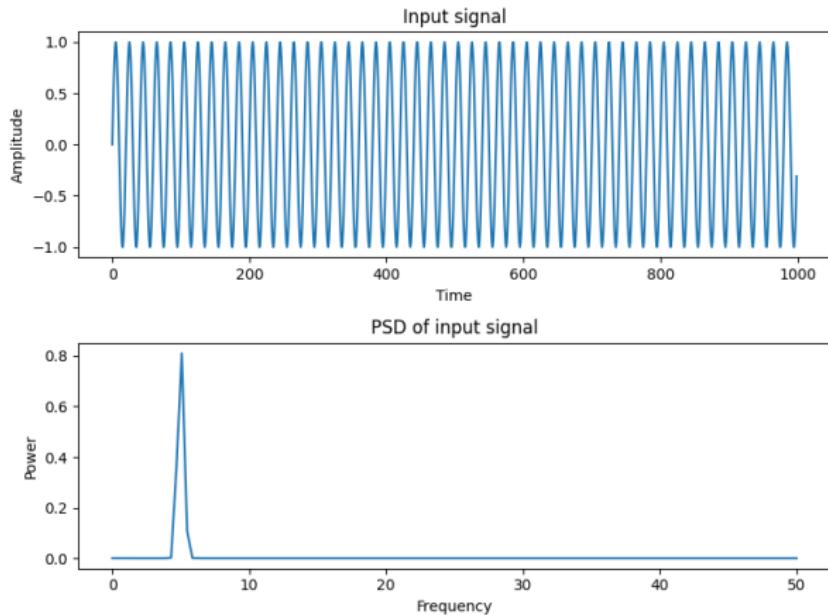


Figure: Input & PSD of Input

# Short Time Fourier Transform (STFT)

The **Short-Time Fourier Transform (STFT)** is a signal processing technique used to analyze the frequency content of a signal as it changes over time. Unlike the traditional **Fourier Transform (FT)**, which provides the frequency content of a signal over the entire time period, the STFT allows us to observe how the frequency content evolves over time, making it particularly useful for non-stationary signals (signals whose frequency content changes over time).

In simple terms, the STFT gives a **time-frequency representation** of a signal, allowing you to see how the frequencies of a signal change as it progresses through time.

The STFT is calculated by taking the FT of a windowed segment of the signal. In other words, the signal is divided into small overlapping or non-overlapping windows, and the FFT is computed for each window. The time and frequency information of the signal are captured together, producing a spectrogram.

# STFT Mathematics

Mathematically, the STFT of a signal  $x(t)$  is given by:

$$X(t, f) = \int_{-\infty}^{+\infty} x(\tau)w(t - \tau)e^{-j2\pi f\tau}d\tau$$

Where:

- ①  $X(t, f)$  is the STFT of the signal  $x(t)$  at time  $t$  and frequency  $f$
- ②  $w(t - \tau)$  is a **window function** (like a Gaussian or Hamming window) that limits the signal to a small time segment around time  $t$
- ③  $e^{-j2\pi f\tau}$  is the complex exponential (the Fourier basis function)
- ④  $\tau$  is the variable of integration (a dummy time variable),
- ⑤ The result  $X(t, f)$  is a complex-valued function representing both the **amplitude** and **phase** of the frequency components at time  $t$  and frequency  $f$

# STFT Steps

1. **Windowing:** The signal is multiplied by a window function  $w(t - \tau)$ . This window limits the signal to a small portion of time, allowing us to focus on local sections of the signal.
2. **Fourier Transform:** The Fourier Transform is then computed on this windowed signal to get the frequency content of that section.
3. **Sliding Window:** This process is repeated by sliding the window across the signal (either by overlapping or non-overlapping windows), producing a time-frequency representation of the signal.

The result is a spectrogram, which is a 2D plot showing the signal's frequency content as a function of time.

# SST Parameters

**Window Function:** The choice of window function affects the time and frequency resolution of the STFT. Common window functions include:

- ① **Rectangular Window:** Simple, but poor frequency resolution.
- ② **Hamming/Hanning Window:** Smoother windows that provide a good balance between time and frequency resolution.
- ③ **Gaussian Window:** Provides excellent localization in both time and frequency.

**Window Length:** The size of the window determines how much of the signal is captured in each analysis. A larger window gives better frequency resolution but poorer time resolution, and vice versa.

**Overlap:** The window can overlap with the previous window by some amount. Overlapping windows provide better time resolution but increase computational cost.

# SFST Pros and Cons

## Pros:

1. **Time-Frequency Analysis:** Provides both time and frequency information, which is crucial for analyzing non-stationary signals.
2. **Localized Frequency Content:** Helps understand how the frequency components evolve over time.
3. **Widely Used:** STFT is widely used in applications such as audio processing, speech recognition, and biomedical signal analysis.

## Cons:

1. **Resolution Trade-off:** The choice of window size introduces a trade-off between time and frequency resolution:
  - ① A large window gives better frequency resolution but poorer time resolution.
  - ② A small window gives better time resolution but poorer frequency resolution.
2. **Fixed Window:** The window size is fixed, which means it may not be optimal for signals with varying frequency content across different time scales.

# STFT Results

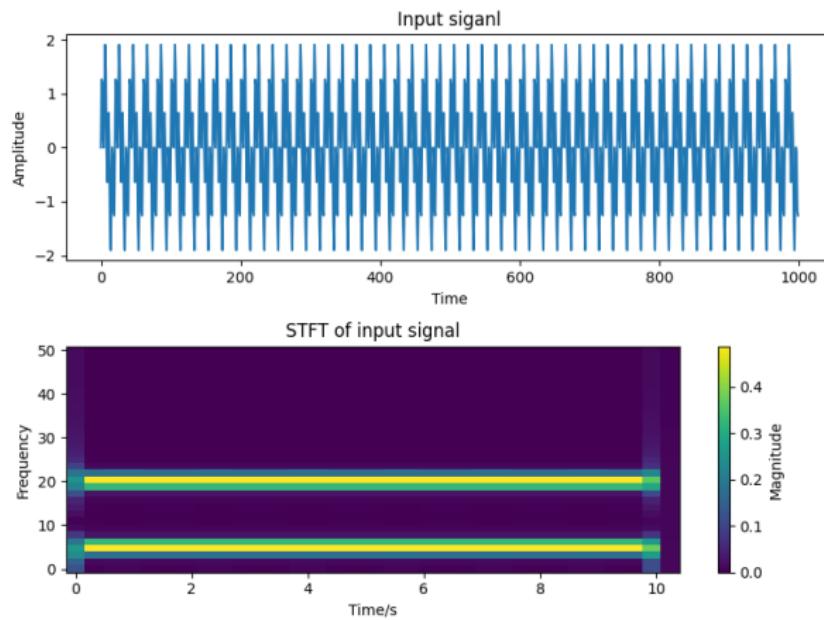


Figure: STFT

# Wavelet Analysis

STFT has a serious problem, that is we can't get a good time and frequency resolution. The bigger the nperseg, the better the frequency resolution, but the worse time resolution, vice versa. The above problem is caused by the fact that the nperseg is fixed. So we introduce the wavelet analysis which is a MRA method. MRA (multiresolution analysis) is designed to give good time resolution and poor frequency resolution at high frequencies and good frequency resolution and poor time resolution at low frequencies. Generally, the basic function of wavelet transform is orthogonal and normalized (normalized makes the transformed signal have the same energy at every scale).

# Mexican Hat Wavelet

Mexican Hat Wavelet is the second derivative of the Gaussian function.

Gaussian function:

$$w(t) = \frac{1}{\sqrt{2\pi} \cdot \sigma} e^{\frac{-t^2}{2\sigma^2}}$$

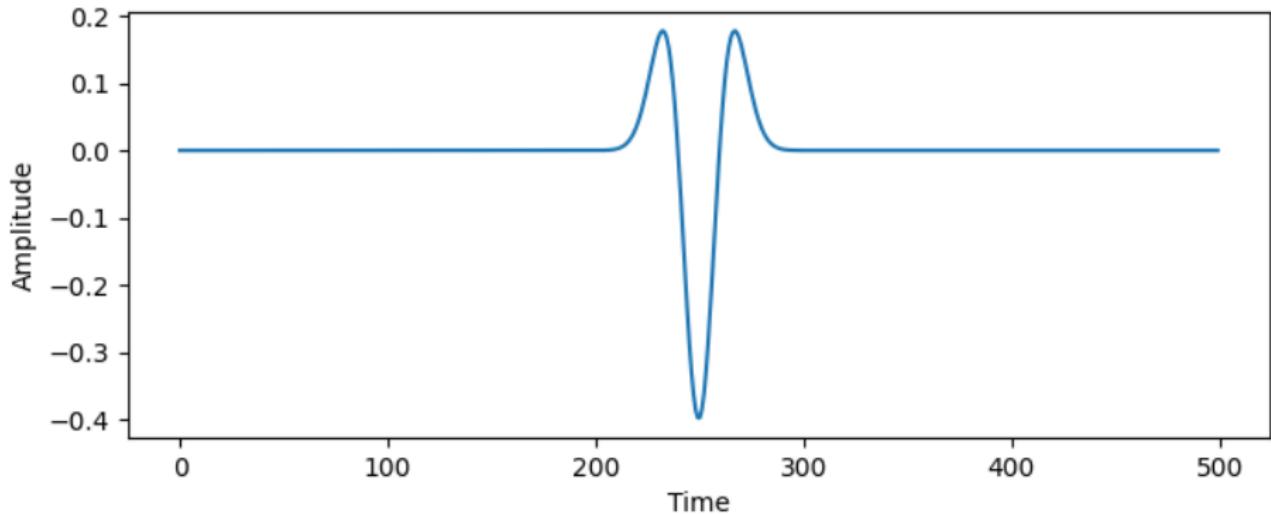
where  $\sigma$  is the standard deviation and  $t$  is time.

Second derivative of the Gaussian function:

$$\psi(t) = \frac{1}{\sqrt{2\pi} \cdot \sigma^3} \left( e^{\frac{-t^2}{2\sigma^2}} \cdot \left( \frac{t^2}{\sigma^2} - 1 \right) \right)$$

# Mexican Hat Continued

`mexican_hat_wavelet(sigma, length)`



**Figure:** Mexican Hat Wavelet

# Morlet Wavelet

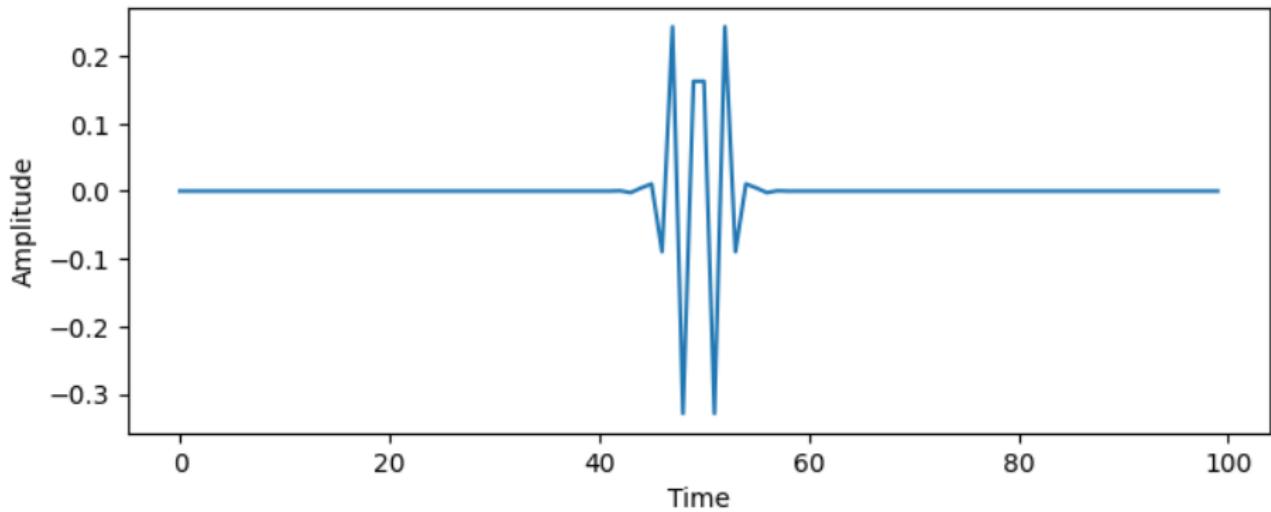
Morlet Wavelet is defined as follows:

$$w(t) = e^{iat} \cdot e^{-\frac{t^2}{2\sigma}}$$

where  $t$  is the time variable,  $a$  controls the frequency of the sinusoidal oscillation, and  $\sigma$  controls the width of the Gaussian envelope.

# Morlet Wavelet Continued

`morlet_wavelet(length, sigma, a)`



**Figure:** Morlet Wavelet

# Continuous Wavelet Transform (CWT)

Continuous Wavelet Transform (CWT): the signal can be transformed by:

$$CWT_x^\psi(\tau, s) = \Psi_x^\psi(\tau, s) = \frac{1}{\sqrt{|s|}} \int x(t)\psi^* \left( \frac{t - \tau}{s} \right) dt$$

where the  $\psi$  denotes the mother wavelet,  $\tau$  denotes the translation of the wavelet, and  $s$  denotes the scale of the wavelet.  $\tau$  and  $s$  are all incremented continuously. However, if this transform needs to be computed by a computer, then both parameters are increased by a sufficiently small step size. This corresponds to sampling the time-scale plane.

# Inverse CWT

Inverse CWT:

$$x(t) = \frac{1}{c_\psi^2} \int_s \int_\tau \Psi_x^\psi(\tau, s) \frac{1}{s^2} \psi\left(\frac{t-\tau}{s}\right) d\tau ds$$

where  $c_\psi$  is a constant that depends on the wavelet used and the other symbols are described in above cell.  $c_\psi$  can be calculated as follows:

$$c_\psi = \left\{ 2\pi \int_{-\infty}^{\infty} \frac{|\hat{\psi}(\xi)|^2}{|\xi|} d\xi \right\}^{1/2}$$

where  $\hat{\psi}(\xi)$  is the FT of  $\psi(t)$

# CWT Results

coefficients, frequencies = my\_cwt(signal, scales, wavelet, fs)

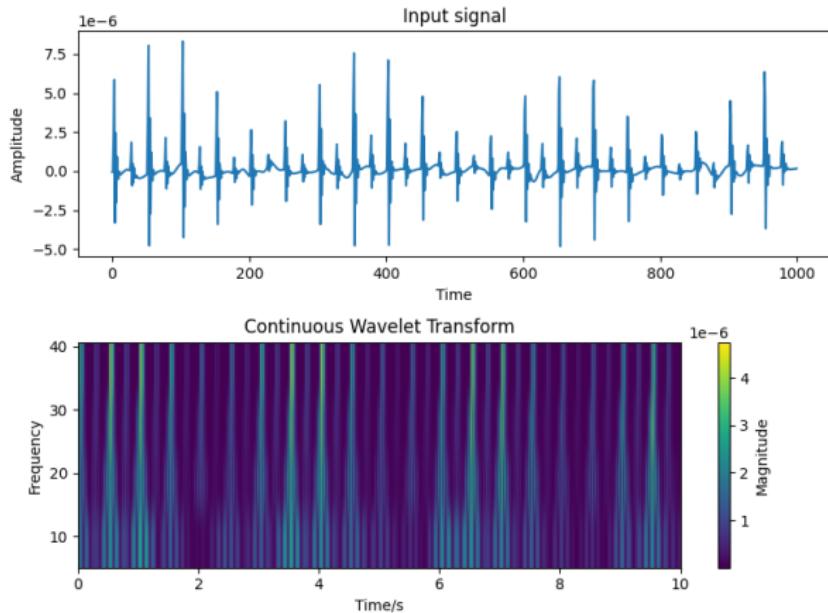


Figure: CWT with a Morlet Wavelet

# Wavelet Denoising Transform (DWT)

Wavelet denoising is a technique that utilizes wavelet transforms to remove noise from signals or images. It decomposes the signal into different frequency components, thresholds the coefficients, and reconstructs the signal, effectively reducing noise.

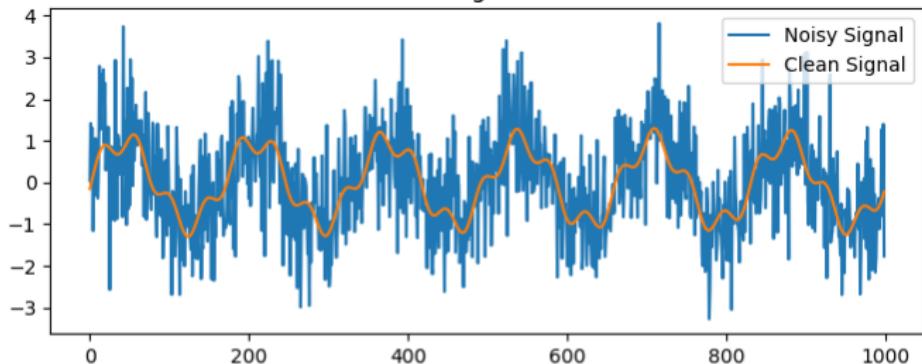
Mathematically, the denoised signal  $y(t)$  is obtained by thresholding wavelet coefficients  $W_j$  at a certain level:  $y(t) = \sum_j \text{Threshold}(W_j)$ .

The denoising steps are the following:

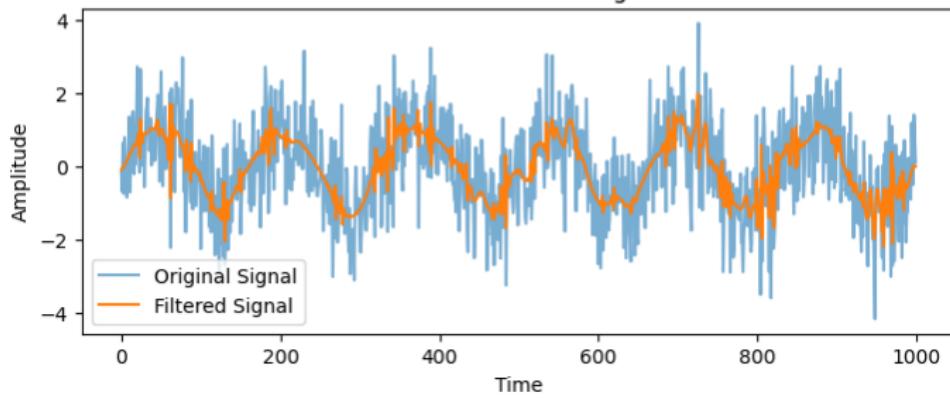
- ① Apply the DWT to the signal
- ② Compute the threshold corresponding to the chosen level
- ③ Only keep coefficients with a value higher than the threshold
- ④ Apply the inverse DWT to retrieve the signal

```
wavelet_denoise(noisy_signal, wav, 0.5, show=True)
```

## Signal



## Wavelet Denoising



# Polynomial Chirplet Transform (PCT)

Polynomial Chirplet Transform (PCT) is a signal processing technique and a variant of the Chirplet Transform. The Chirplet Transform is a method used to analyze non-stationary modulations in signals, and PCT further introduces polynomial functions to enhance its analytical capabilities.

A "Chirp" refers to a signal with a frequency that varies over time, and a "Chirplet" is a wavelet associated with a Chirp. PCT introduces polynomial modulation functions, allowing the frequency, phase, and amplitude of Chirplets to vary according to the shape of the polynomial.

The core idea of PCT is to use polynomials to describe the nonlinear modulation characteristics of a signal, thereby better capturing the complex structures and non-stationary nature of the signal. This method is effective for analyzing signals where frequency, phase, and amplitude change over time.

# Chirplet

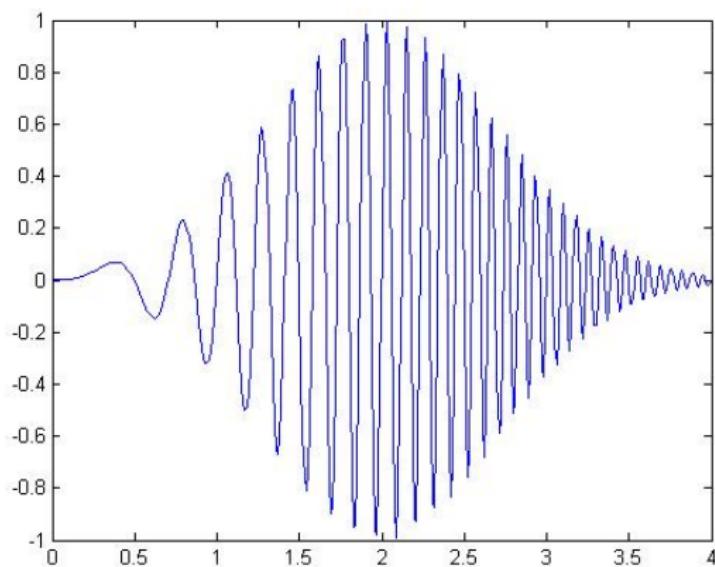


Figure: Sample Chirplet

Image from: [Wikimedia Commons: Chirplet](#)

# Chirplet Transform

The Chirplet Transform of a signal  $x(t)$  is often represented as:

$$C(a, b, \omega, \tau) = \int_{-\infty}^{\infty} x(t) \psi_{a,b,\omega,\tau}^*(t) dt$$

where  $*$  denotes the complex conjugate and  $\psi_{a,b,\omega,\tau}(t)$  is the Chirplet defined as:

$$\psi_{a,b,\omega,\tau}(t) = e^{j(\omega t + \frac{a}{2}t^2 + bt + \tau)}$$

# Polynomial Chirplet Transform

The Polynomial Chirplet Transform extends the Chirplet Transform by introducing a polynomial modulation. The general form is:

$$C(a, b, \omega, \tau, P(t)) = \int_{-\infty}^{\infty} x(t) \psi_{a,b,\omega,\tau,P(t)}^*(t) dt$$

Here,  $P(t)$  is a polynomial function that modulates the Chirplet parameters.

$$\psi_{a,b,\omega,\tau,P(t)}(t) = e^{j(\omega t + \frac{a}{2}t^2 + bt + \tau + P(t))}$$

# Chirplet Transform Results

chirplet\_transform(signal)

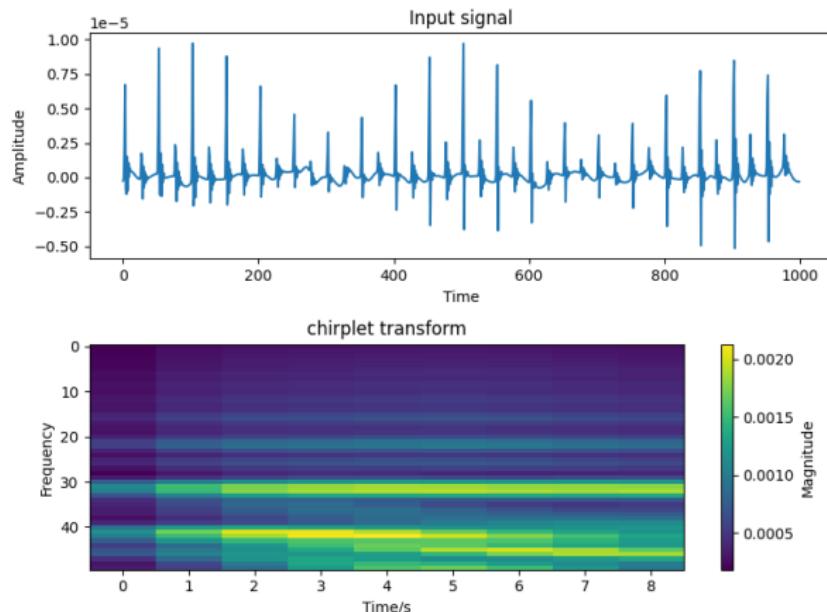


Figure: Chirplet Transform

# Table of Contents

1 Signals and Noise

2 Filters

3 Decomposition

4 Time and Frequency Domain

5 Contact Information

# Thank you!

Stephen Coshatt, [stephen.coshatt@uga.edu](mailto:stephen.coshatt@uga.edu)  
Prof. WenZhan Song, [wsong@uga.edu](mailto:wsong@uga.edu)