Digital Signal Processing Tutorial

Stephen Coshatt[1] and Dr. WenZhan Song[1]

The University of Georgia[1]

Sensor Data Science and AI
Spring 2025

UNIVERSITY OF
GEORGIA

Outline

Table of Contents

Introduction

- Point 1
- Point 2
- Point 3

Table of Contents

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

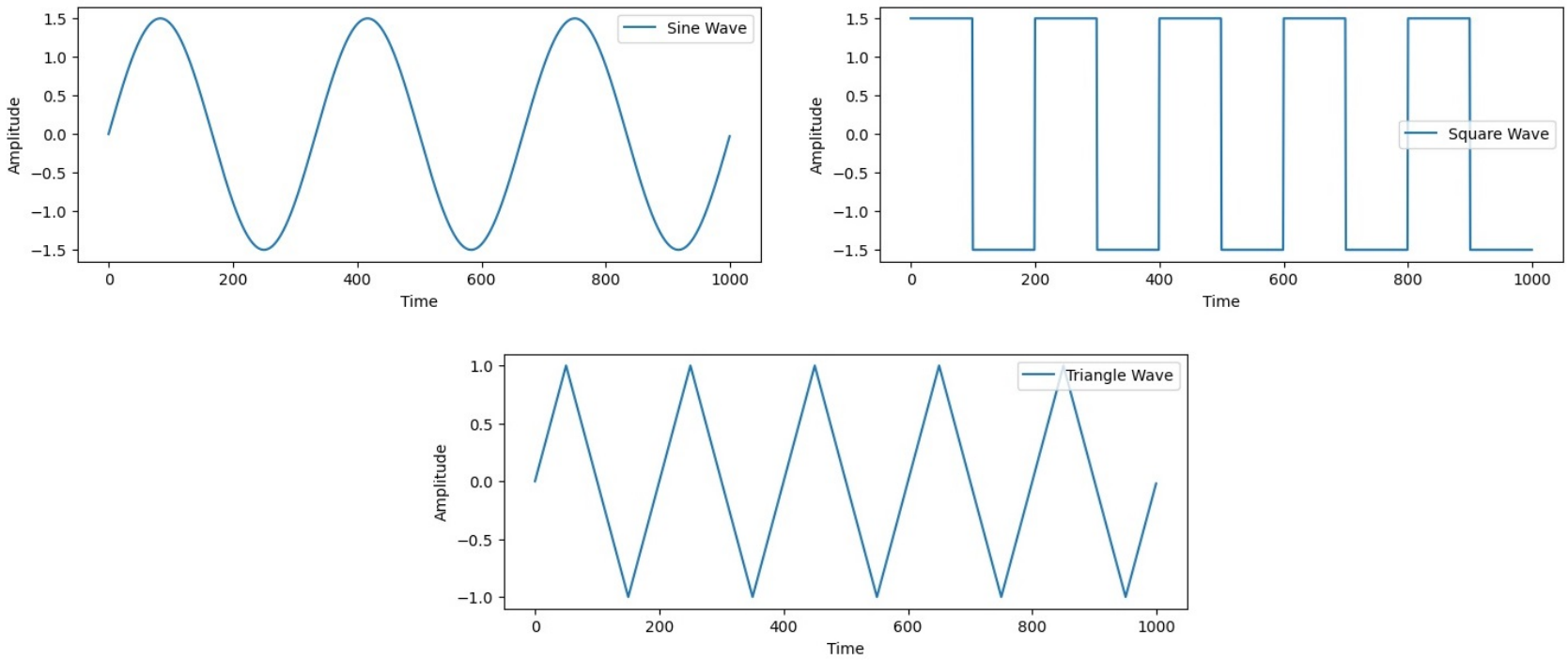


Figure: Sine, Square, and Triangle Waves

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

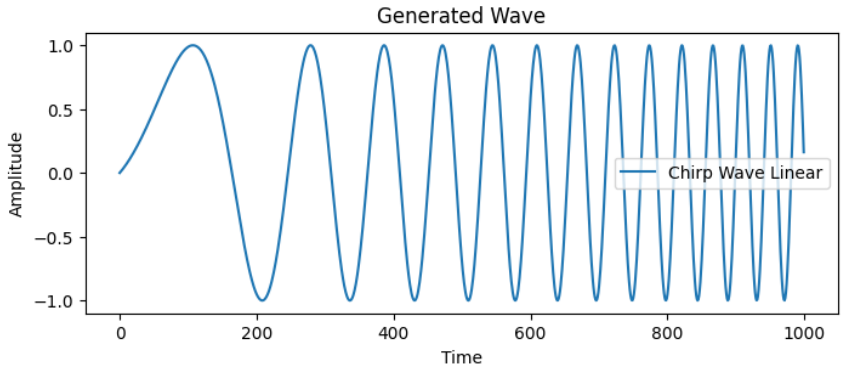


Figure: Linear Chirp

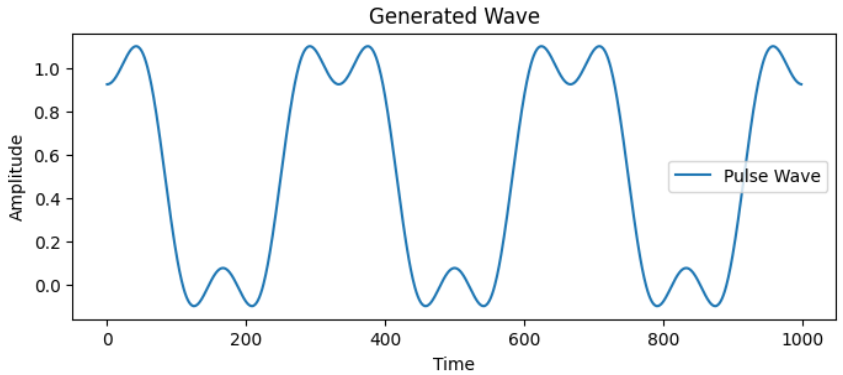


Figure: Pulse

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

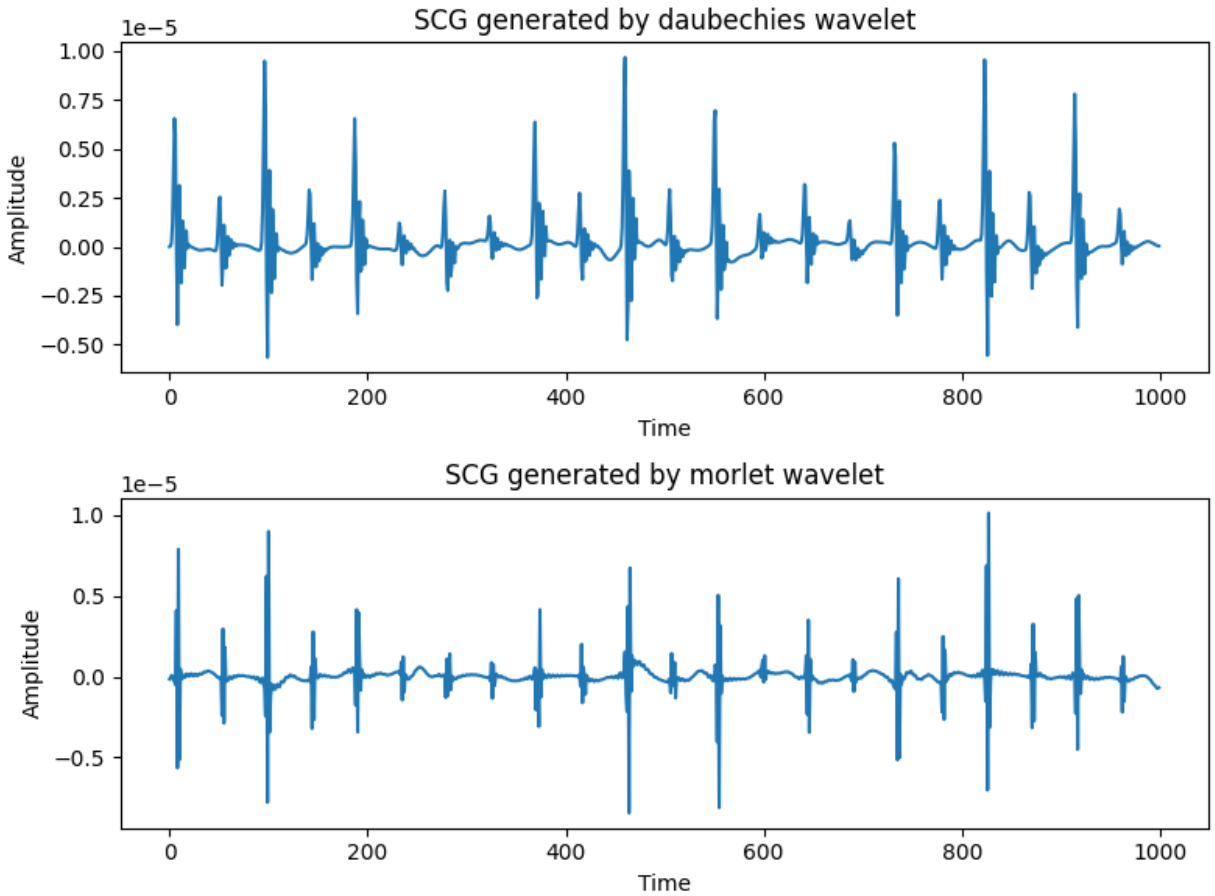


Figure: Two types of SCG generation

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

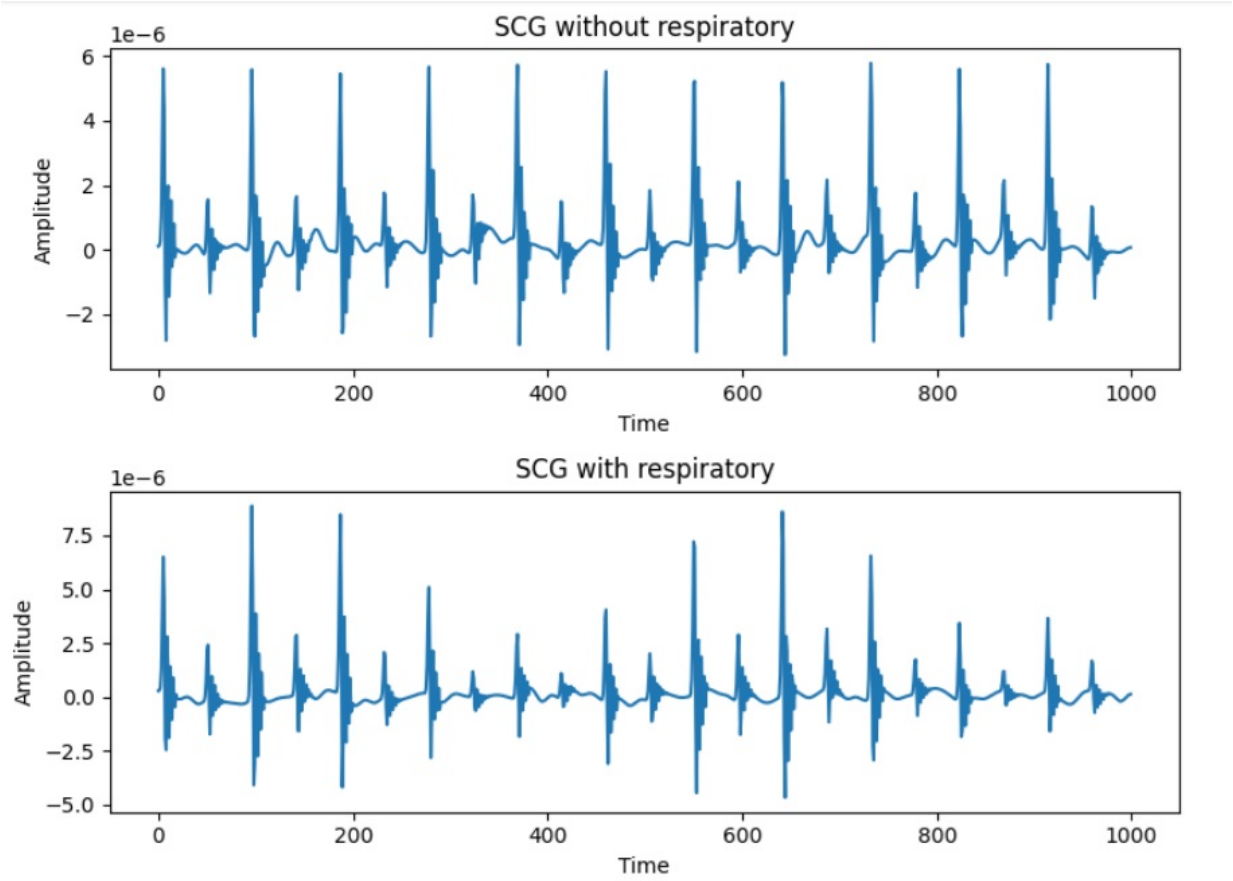


Figure: Adding Respiratory to SCG

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

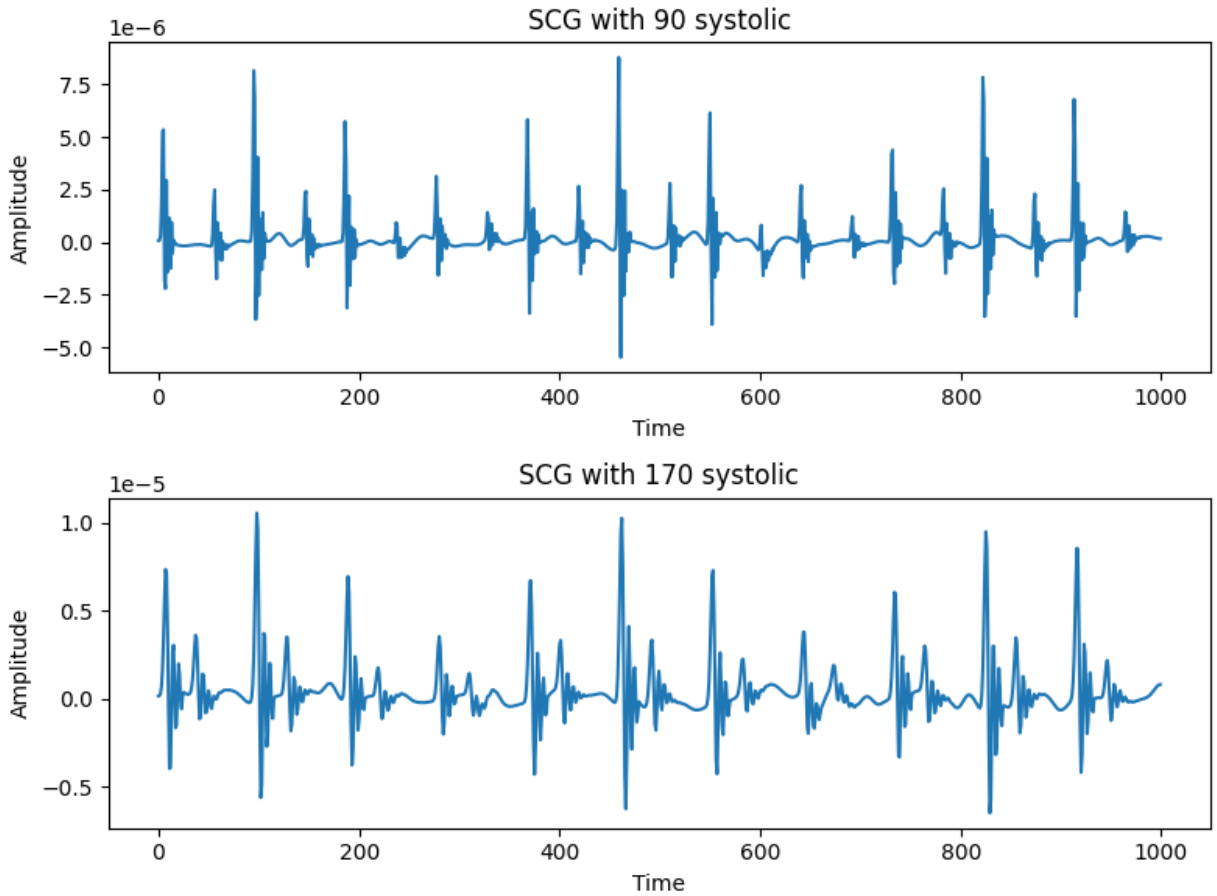


Figure: SCG with Systolic pressures of 90 and 170

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)

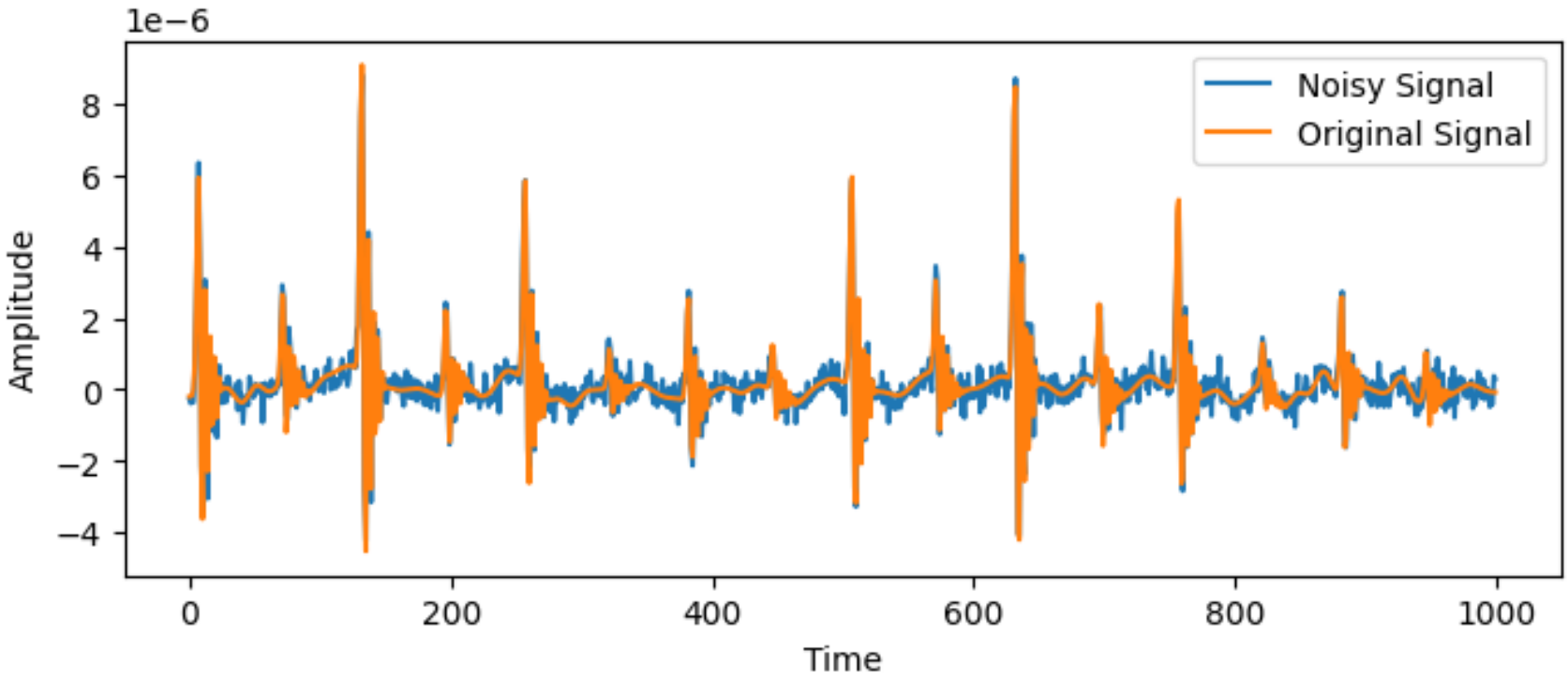


Figure: Signal with Gaussian White Noise

Table of Contents

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

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

where s is the complex frequency variable, ω_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

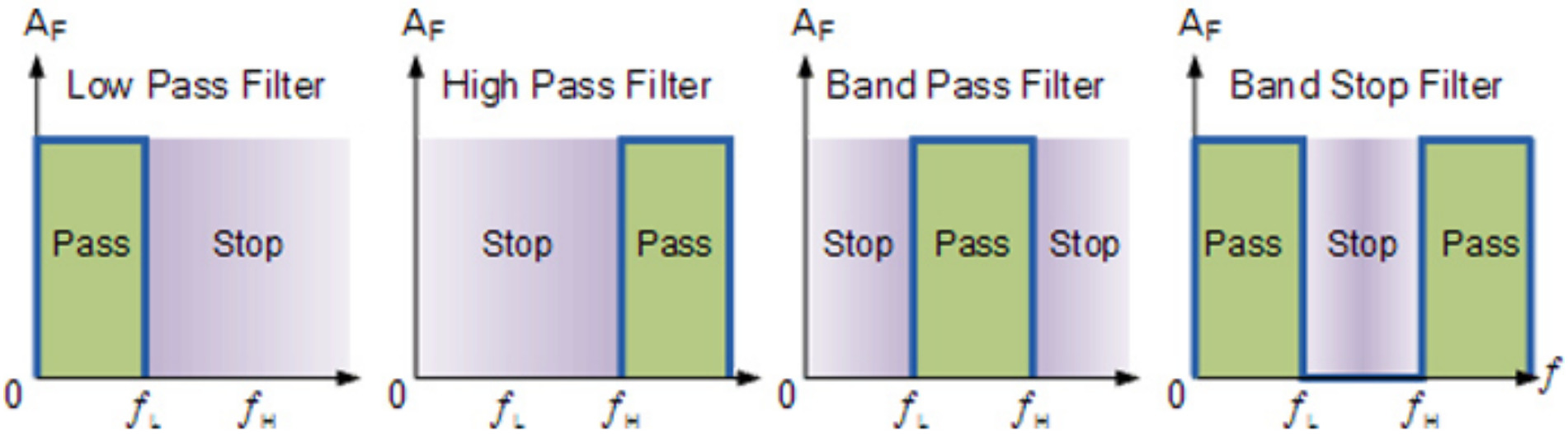


Figure: Linear Filters

Image from Frequency Filter

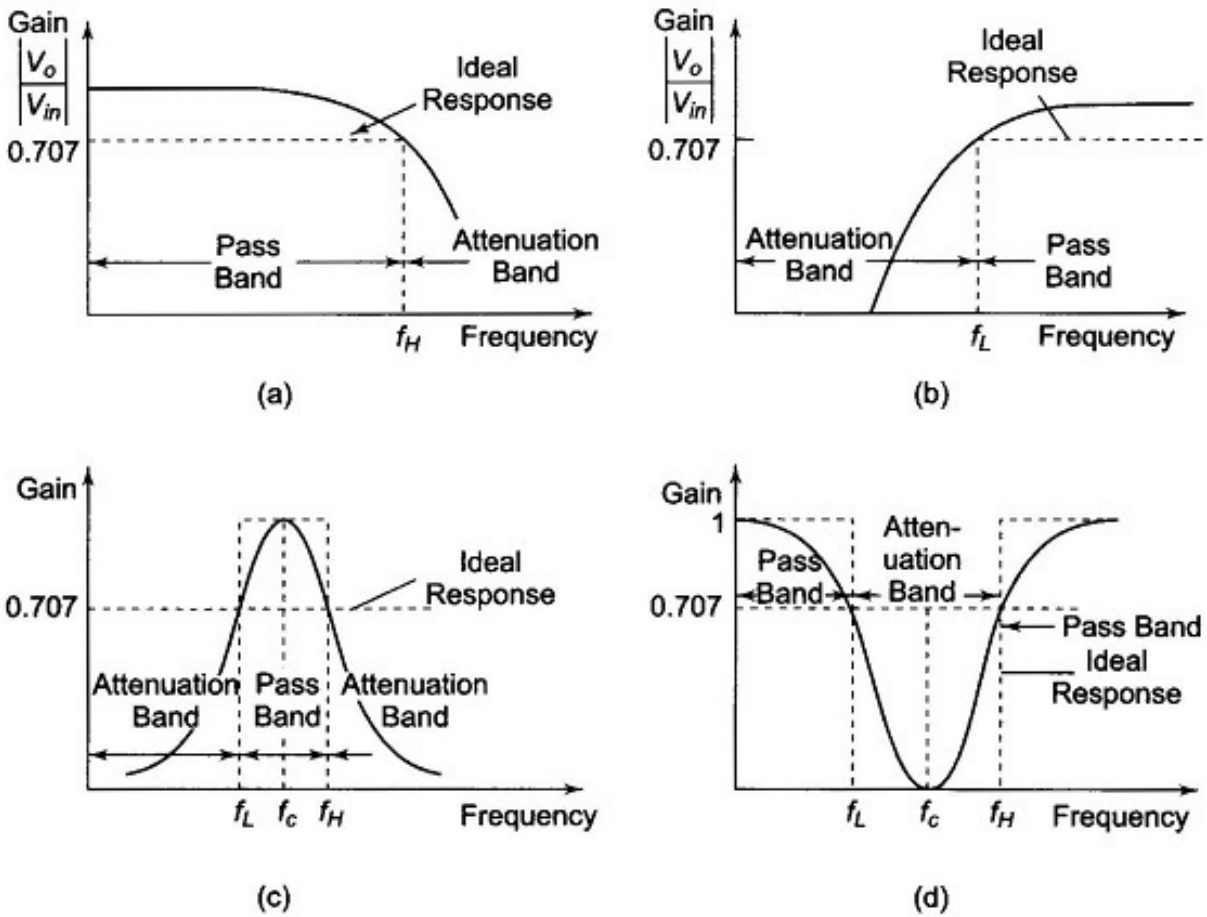


Figure: Butterworth Filter Responses - (a) Low Pass,(b) High Pass, (c) Band Pass, (d) Band Stop

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)

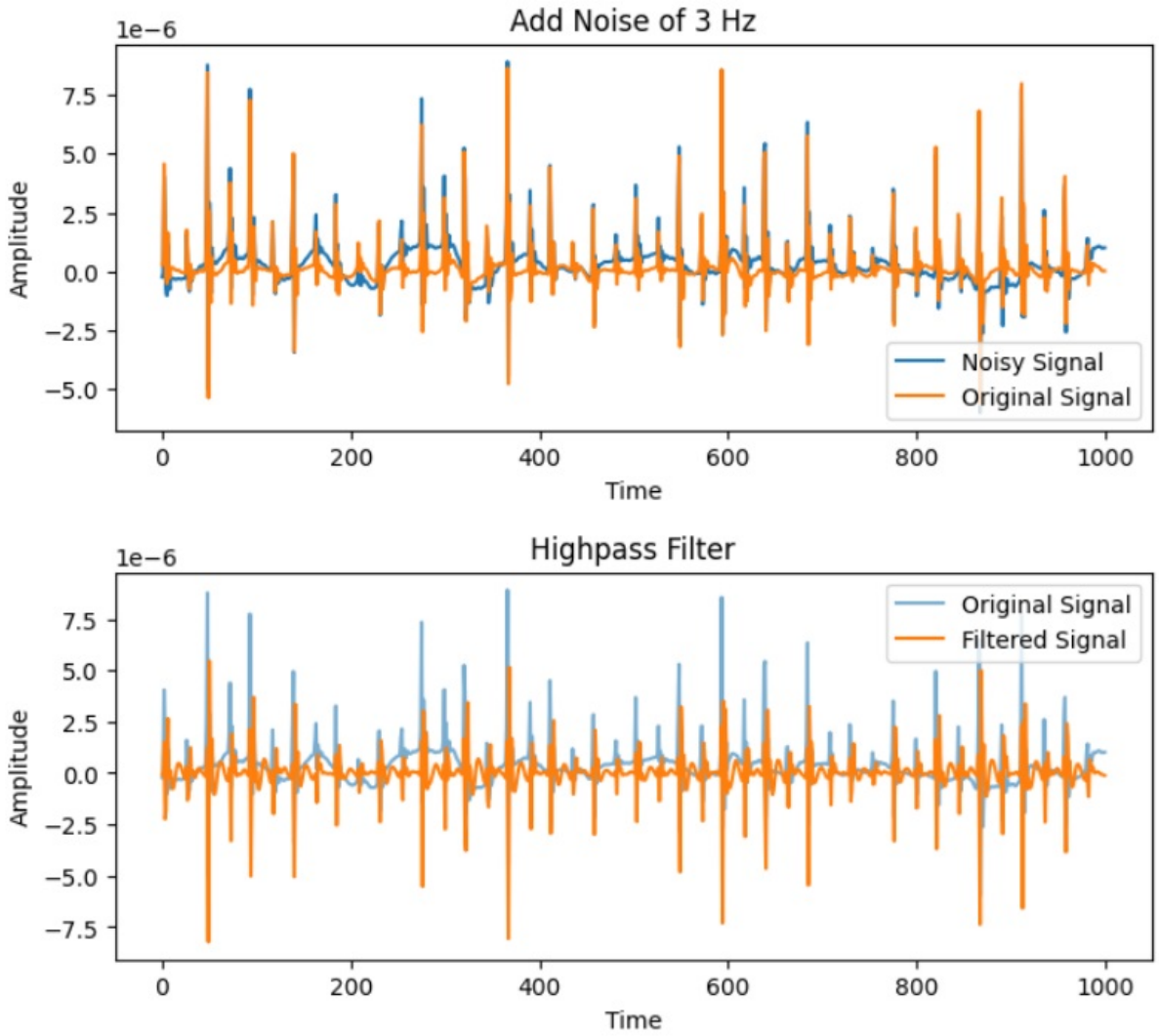


Figure: Filter 3 Hz Noise with Butterworth

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 α 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)

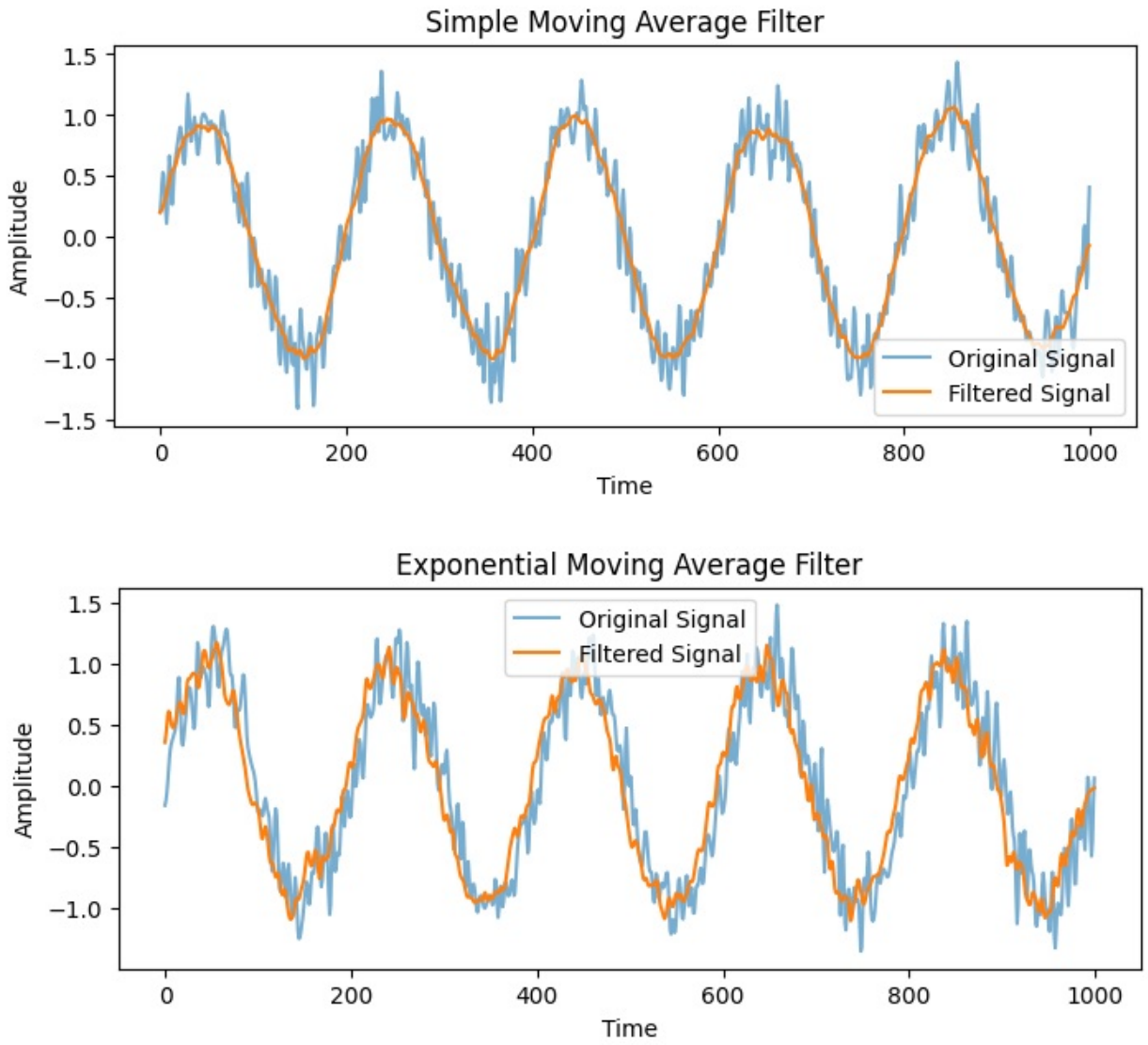


Figure: Moving Average Filters

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.

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

Example:

If polynomial is $y = a_0 + a_1 x + a_2 x^2$

then $Minimize \ \sum_{i=-2}^{2}(y_{k+i} - (a_0 + a_1 x_{k+i} + a_2 x_{k+i}^2))^2$

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)

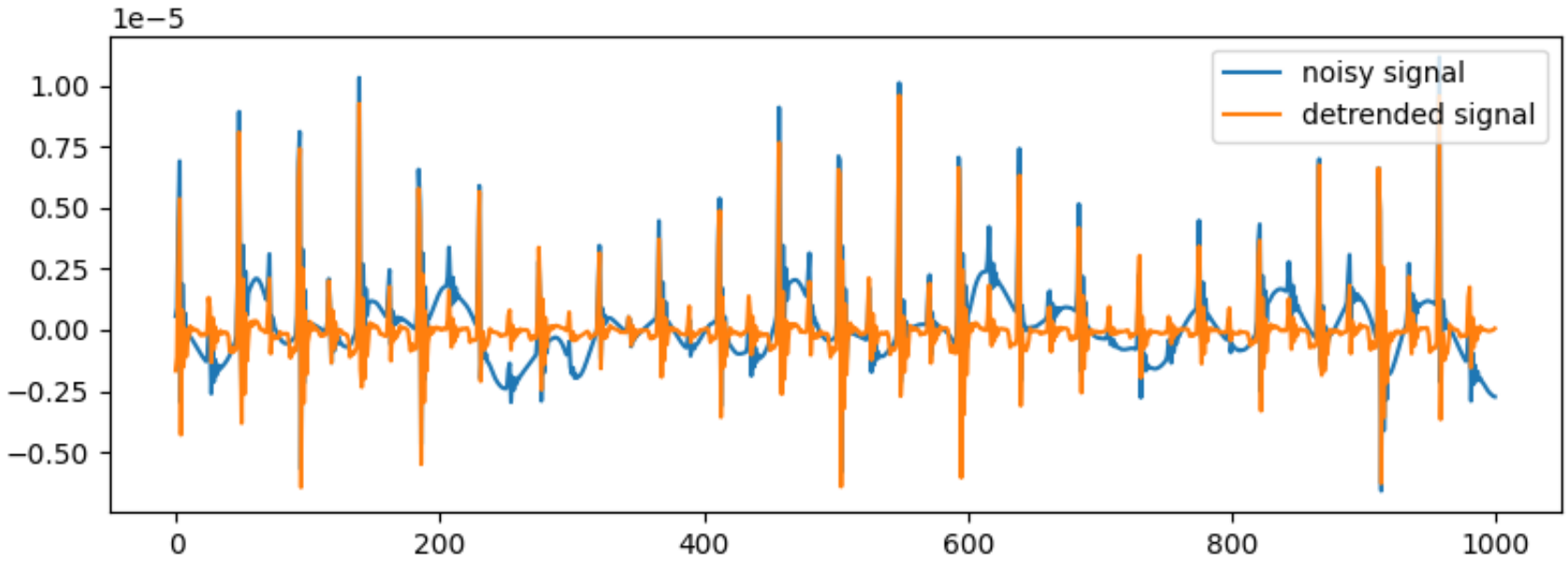


Figure: Savgol Filter

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 $Sx(f)$ is the signal power spectrum, $Sn(f)$ 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)

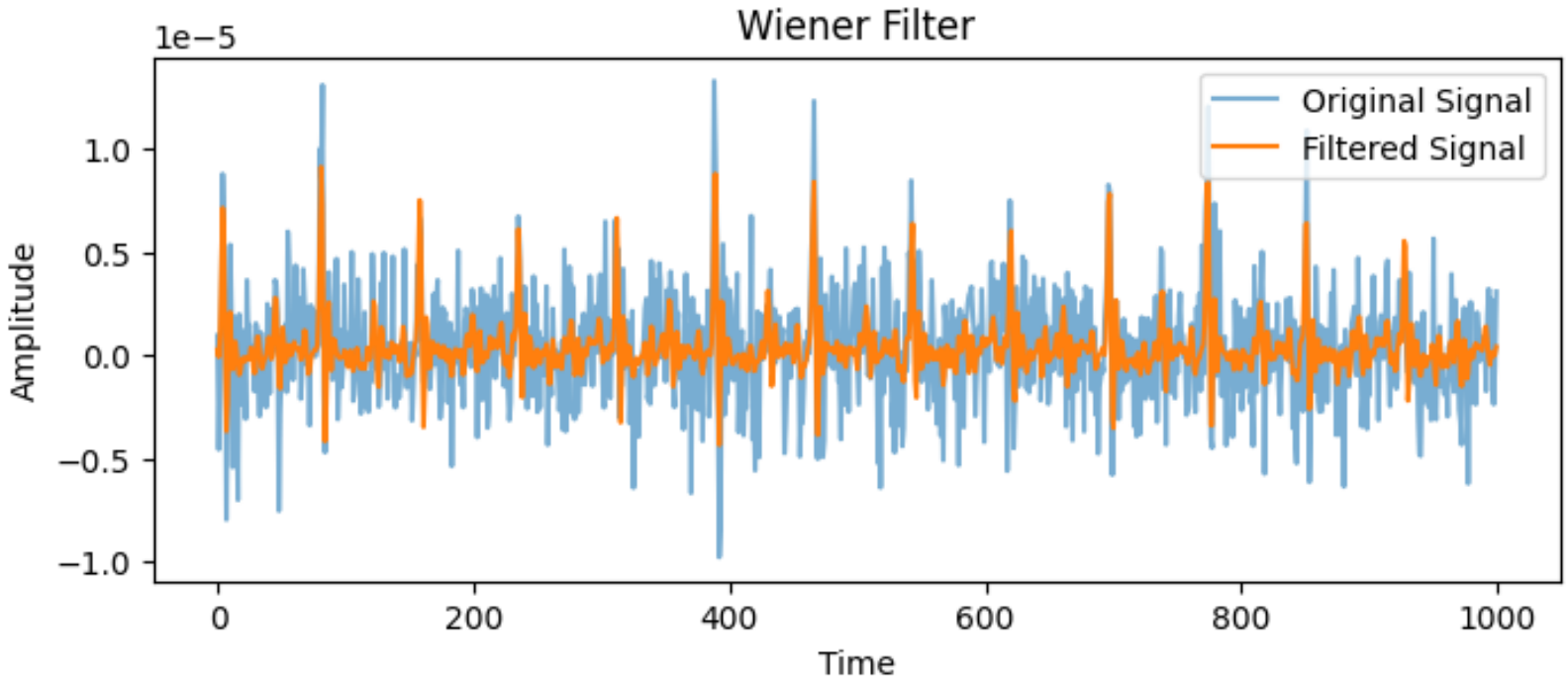


Figure: Wiener Filter

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)

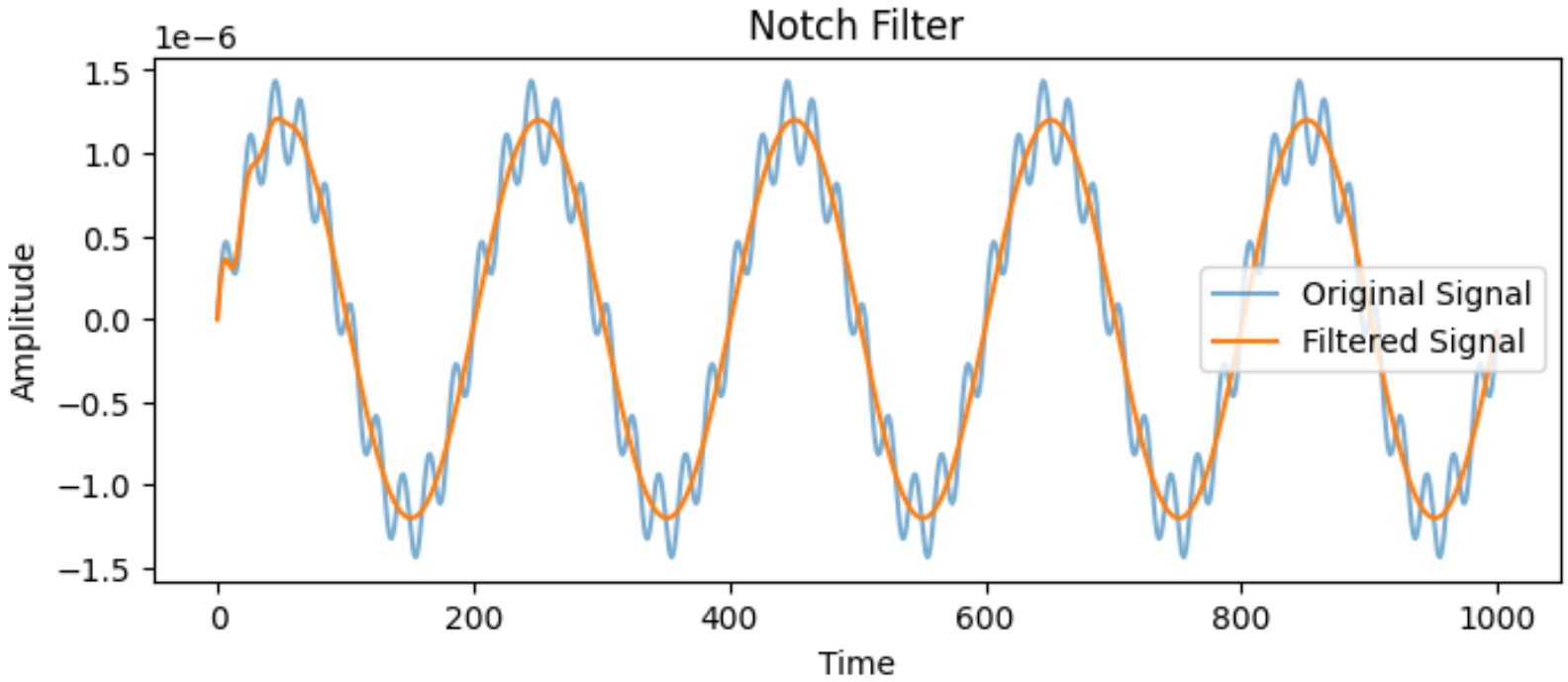


Figure: Notch Filter

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

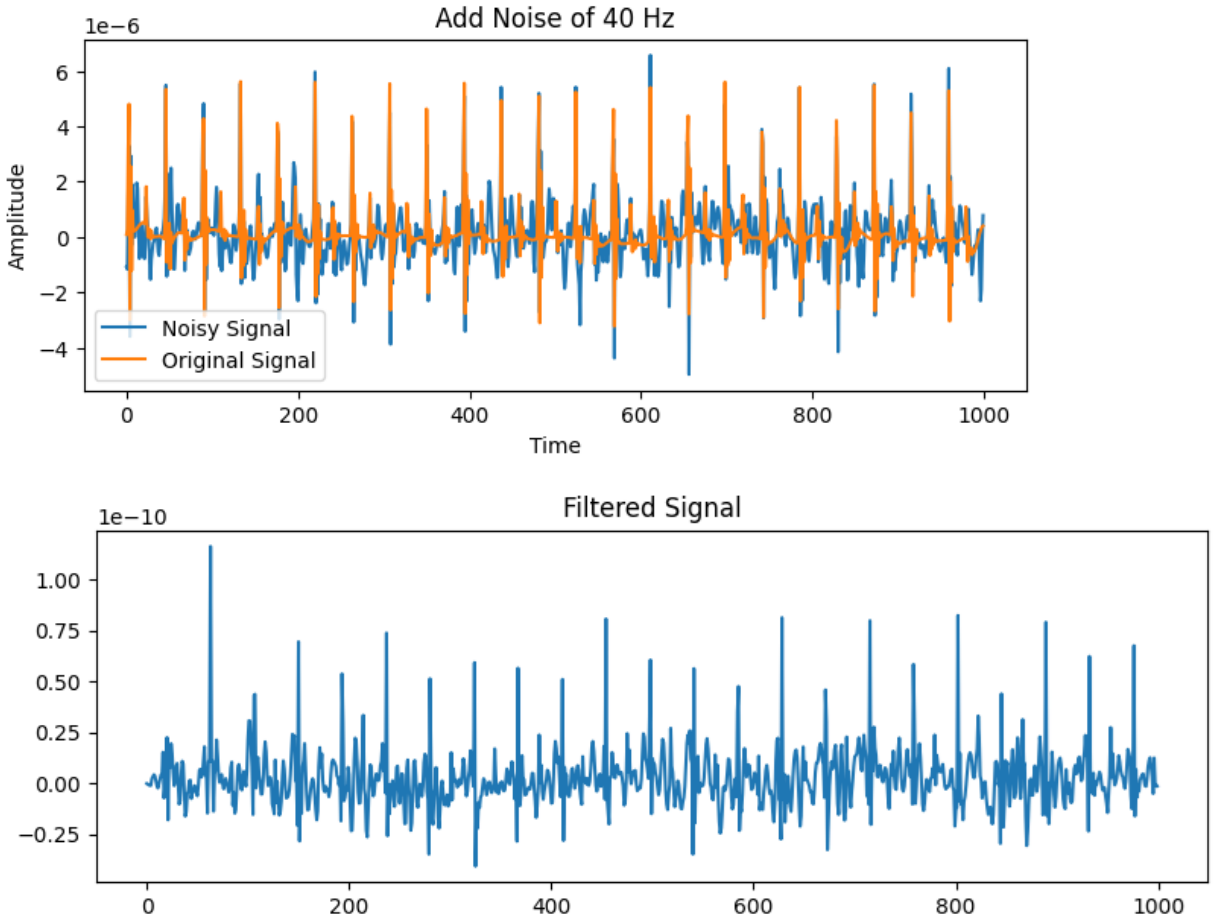


Figure: Matched Filter

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 Eqations

$$K_k = AP_kC^T(CP_KC^T + S_z)^{-1}$$

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

$$P_{k+1} = AP_kA^T + S_w - AP_kC^TS_z^{-1}CP_kA^T$$

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)

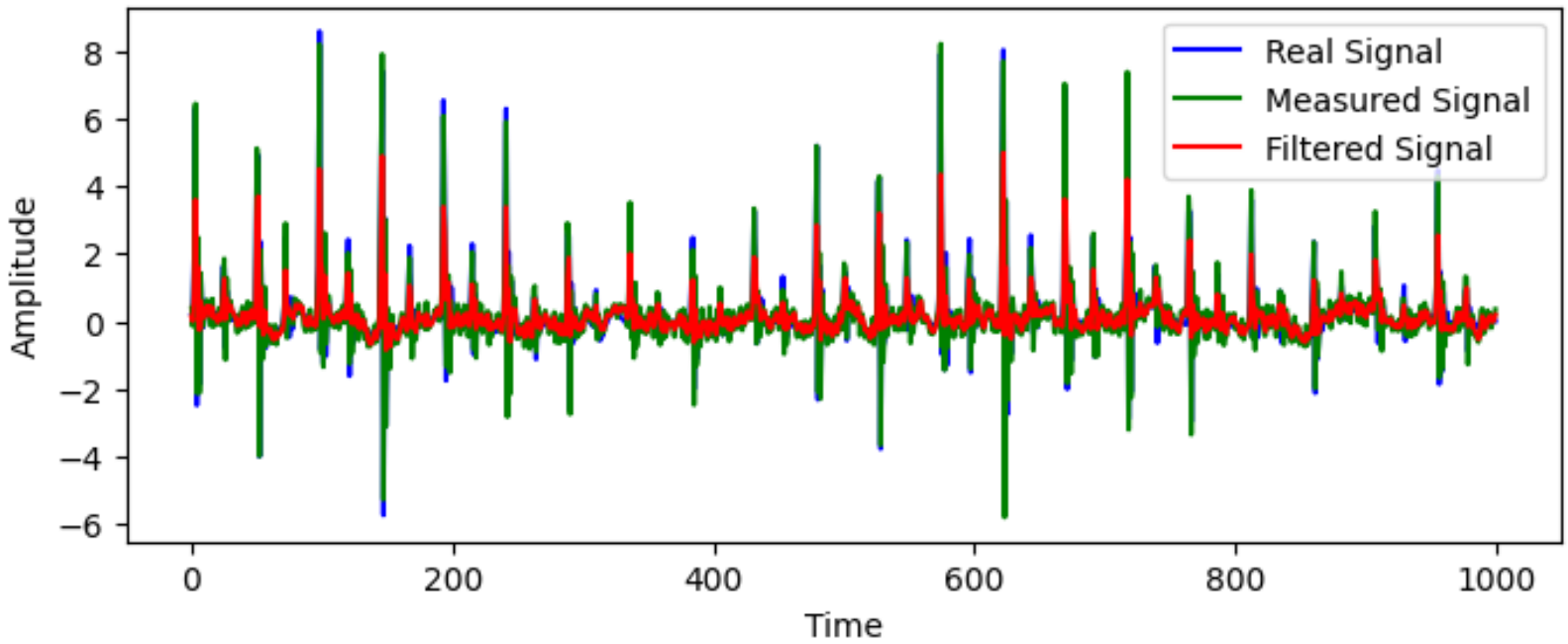


Figure: Kalman Filter

Adaptive Filters

An adaptive filter is a filter with non-constant coefficients. The filter coefficients are adjusted based on an 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 tutorail, we will limit ourselves to adaptive FIR filters.

The filter accepts an input signal **x** and produces an output signal **y**. The FIR coefficients are adjustable, meaning that at every new sample of **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 **e**. The error signal **e** is typically computed as the difference between the actual output signal **y** and a desired output signal **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 $w(n) = [w_0(n), w_1(n), ..., w_{N-1}(n)]^T$. we try to minimize the expected square error:

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

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

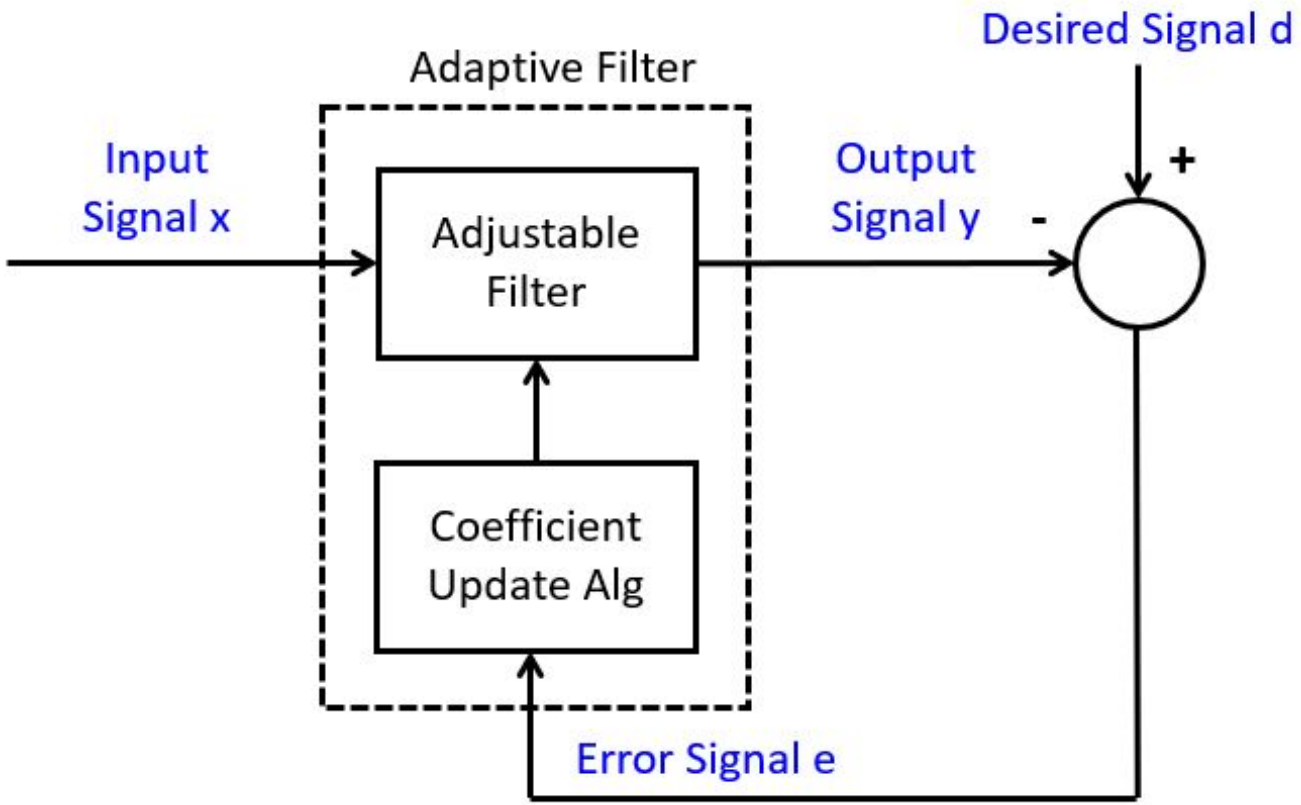


Figure: Adaptive Filter

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

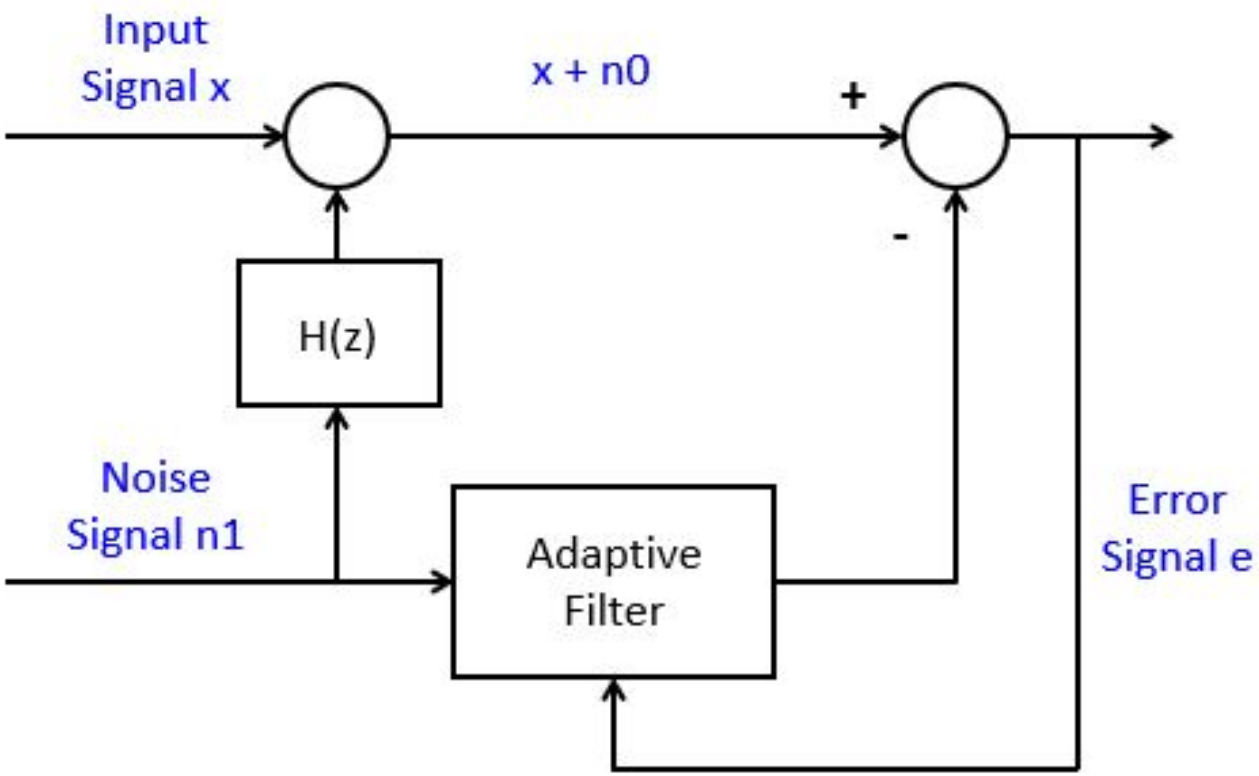


Figure: Adaptive Filter for Noise Cancellation

Image from Adaptive Filters

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

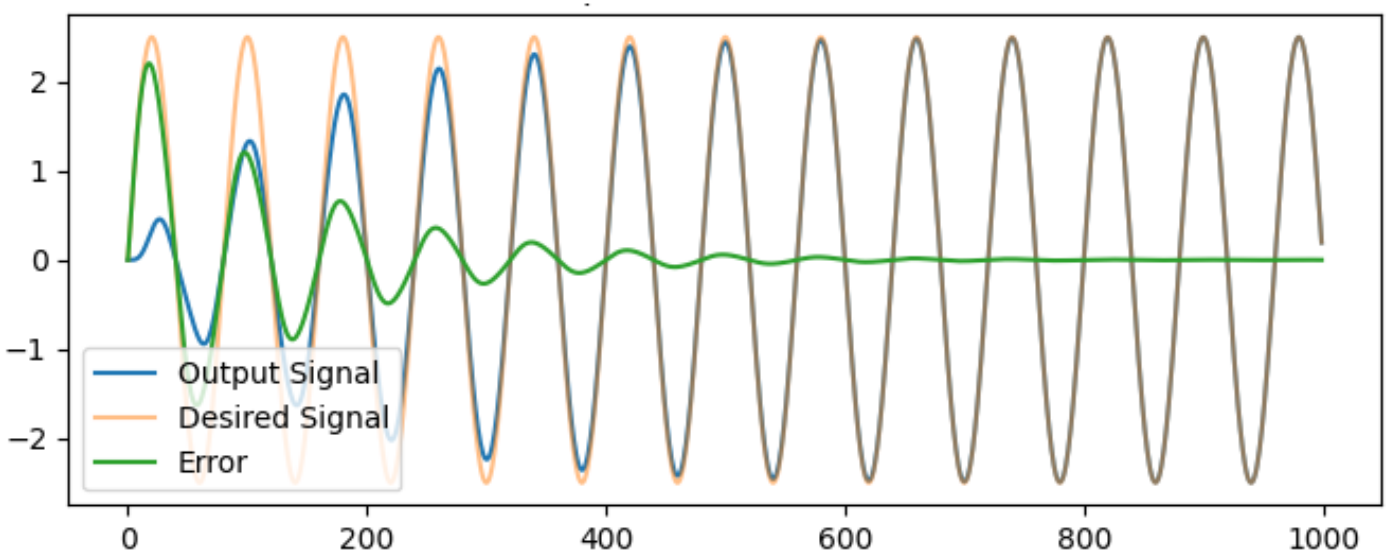


Figure: FIR Filter

Fast-Fourier Transform Denoising

The Fast Fourier Transform (FFT) can be used 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.

The denoising steps are the following

1. Apply the fft to the signal
2. Keep only the coefficients which have a low enough frequency (in absolute)
3. Compute the inverse fft

fft_denoise(noisy_signal, threshold=threshold)

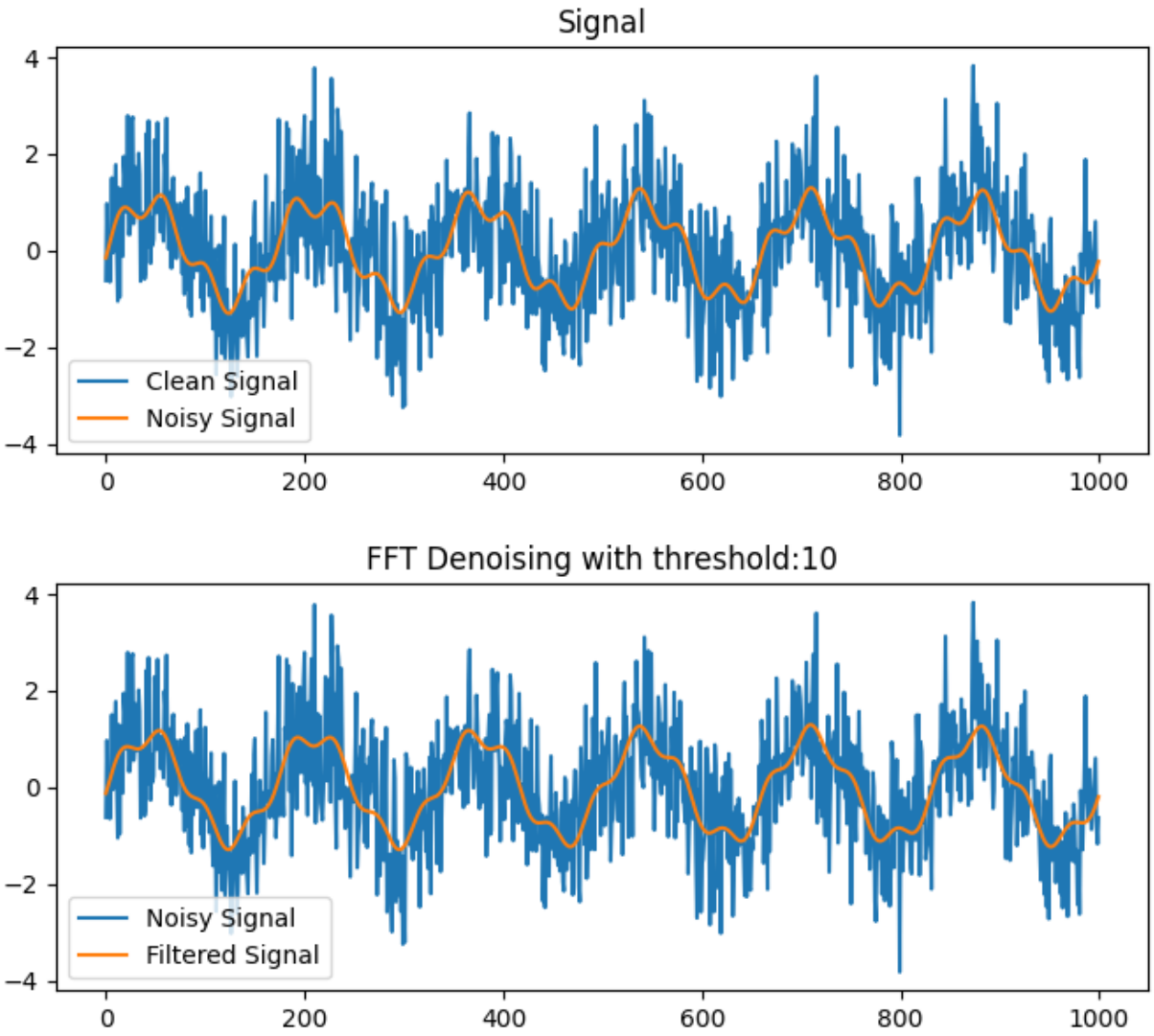


Figure: FFT Denoising

Wavelet Denoising Transform (DWT)

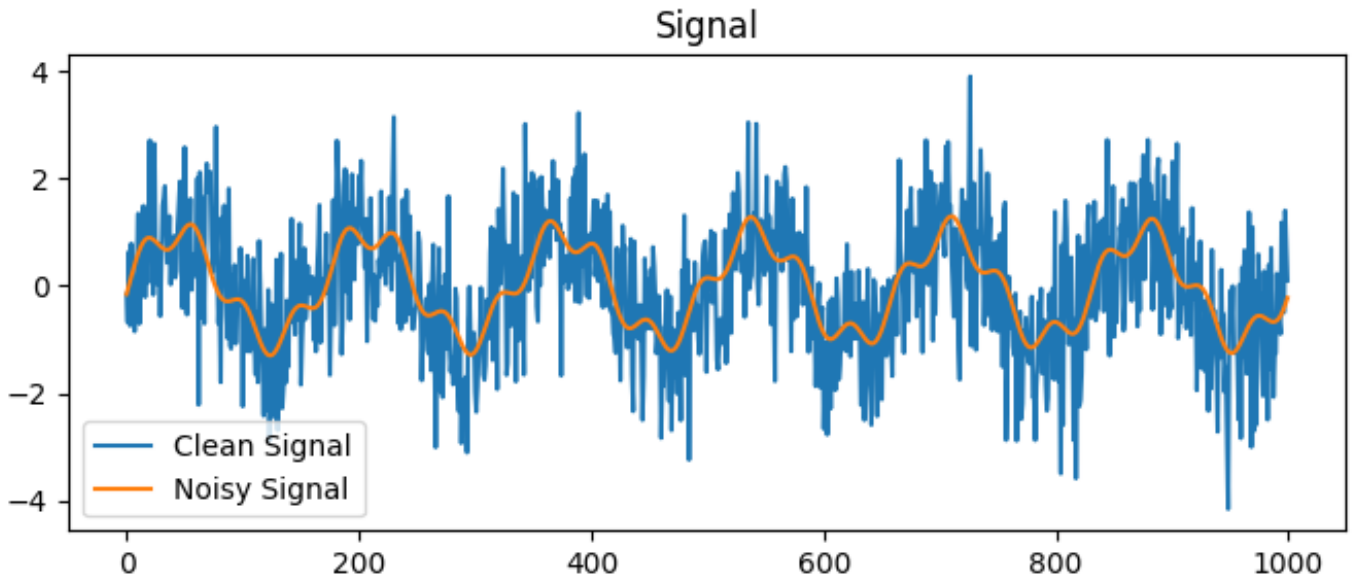
Wavelet denoising is a technique that utilizes wavelet transforms to remove noise from signals or images. It decomposes the signal into different frequency components, thresholds the coefficients, and reconstructs the signal, effectively reducing noise.

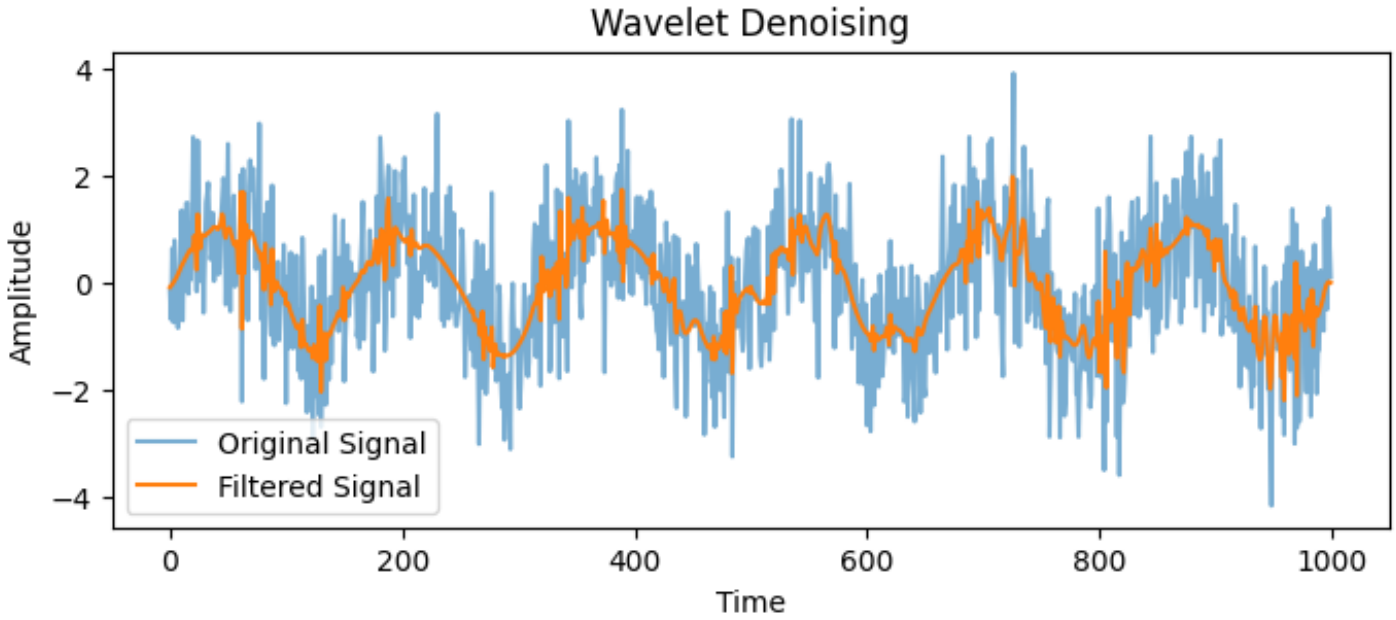
Mathematically, the denoised signal $y(t)$ is obtained by thresholding wavelet coefficients W_j at a certain level: $y(t) = \Sigma_j Threshold(W_j)$.

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
Clean Signal
Noisy Signal
4
2
0
−2
−4
0
200
400
600
800
1000


Wavelet Denoising
Original Signal
Filtered Signal
Amplitude
Time
4
2
0
−2
−4
0
200
400
600
800
1000

Signal Averaging

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

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 $A = [a_1, a_2, ..., a_n]$ and $B = [b_1, b_2, ..., b_m]$, 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.

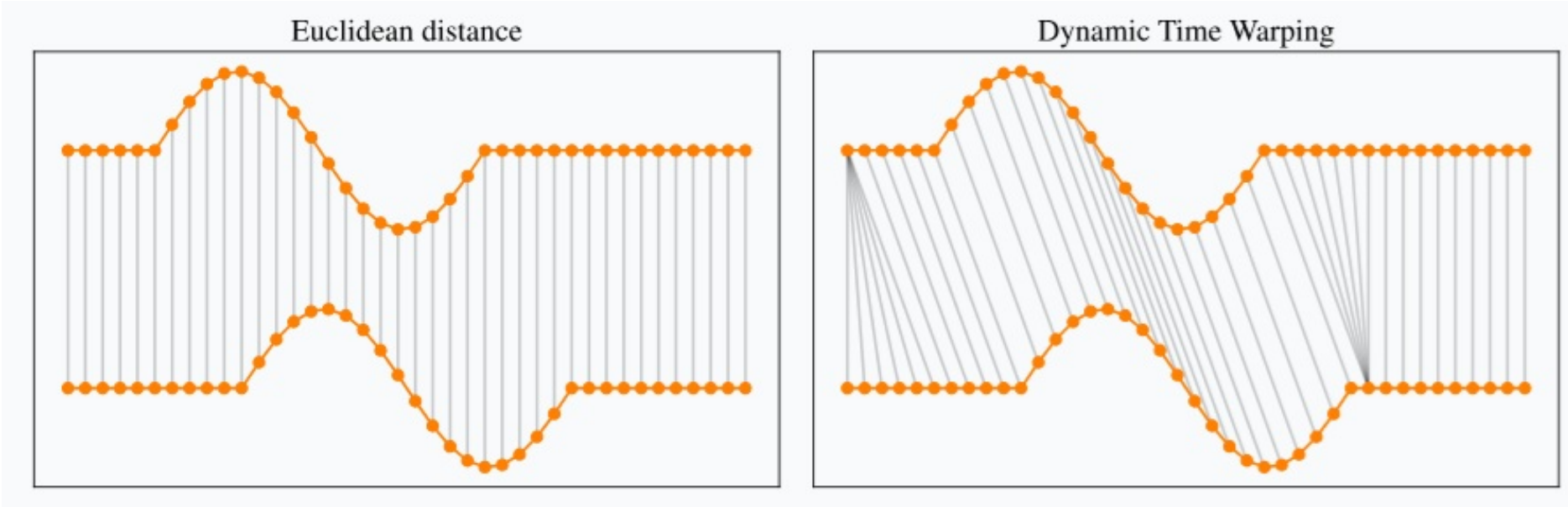


Figure: DTW and Euclidean Distance Comparison.

Image from An Introduction to Dynamic Time Warping by Romain Tavenard

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. performNLAAF1(list(series), show=True)
2. performNLAAF2(list(series), show=True)
3. performICDTW(list(series), show=True)
4. performDBA(series, show=True)

Averaging

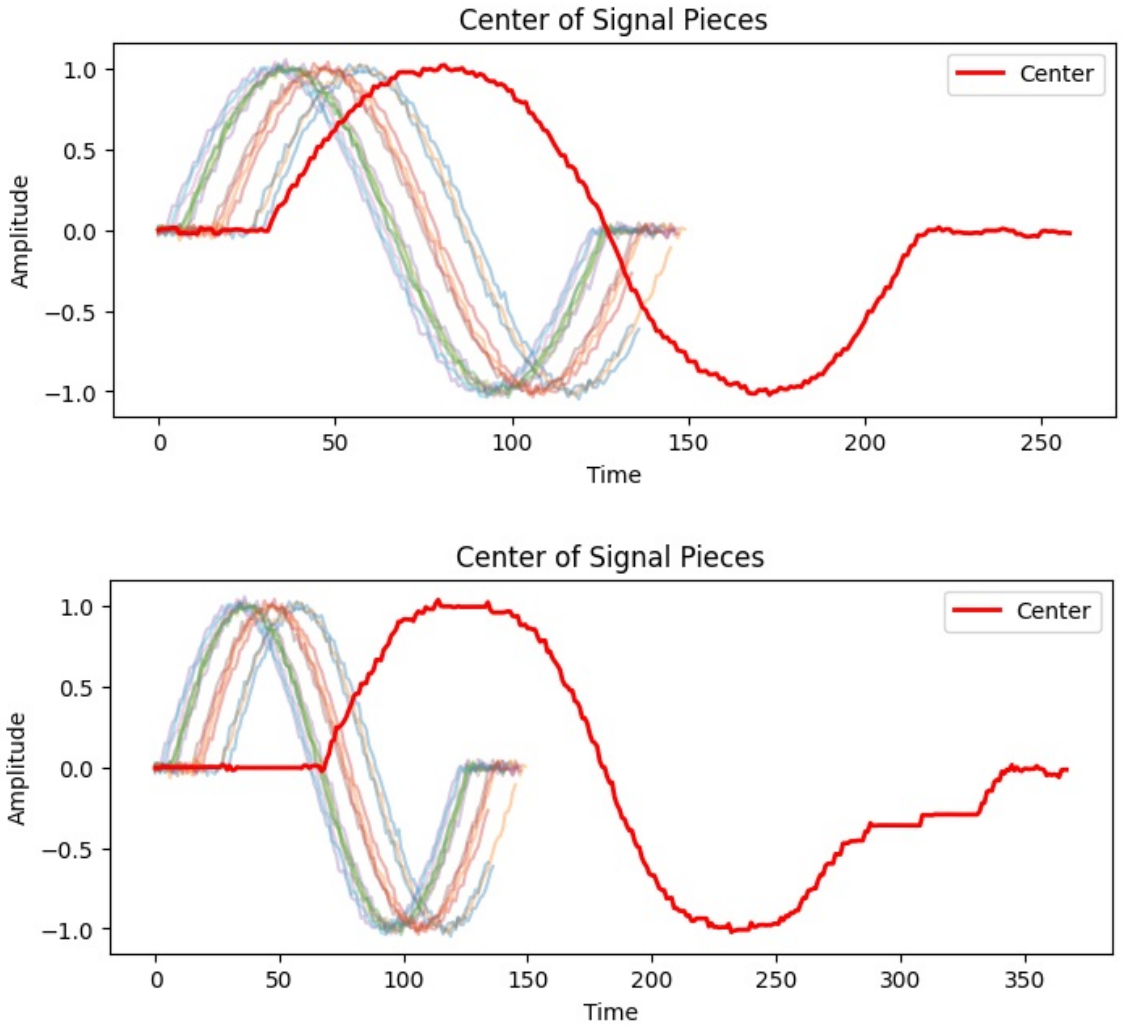


Figure: NLAAF1 and NLAAF2

Averaging

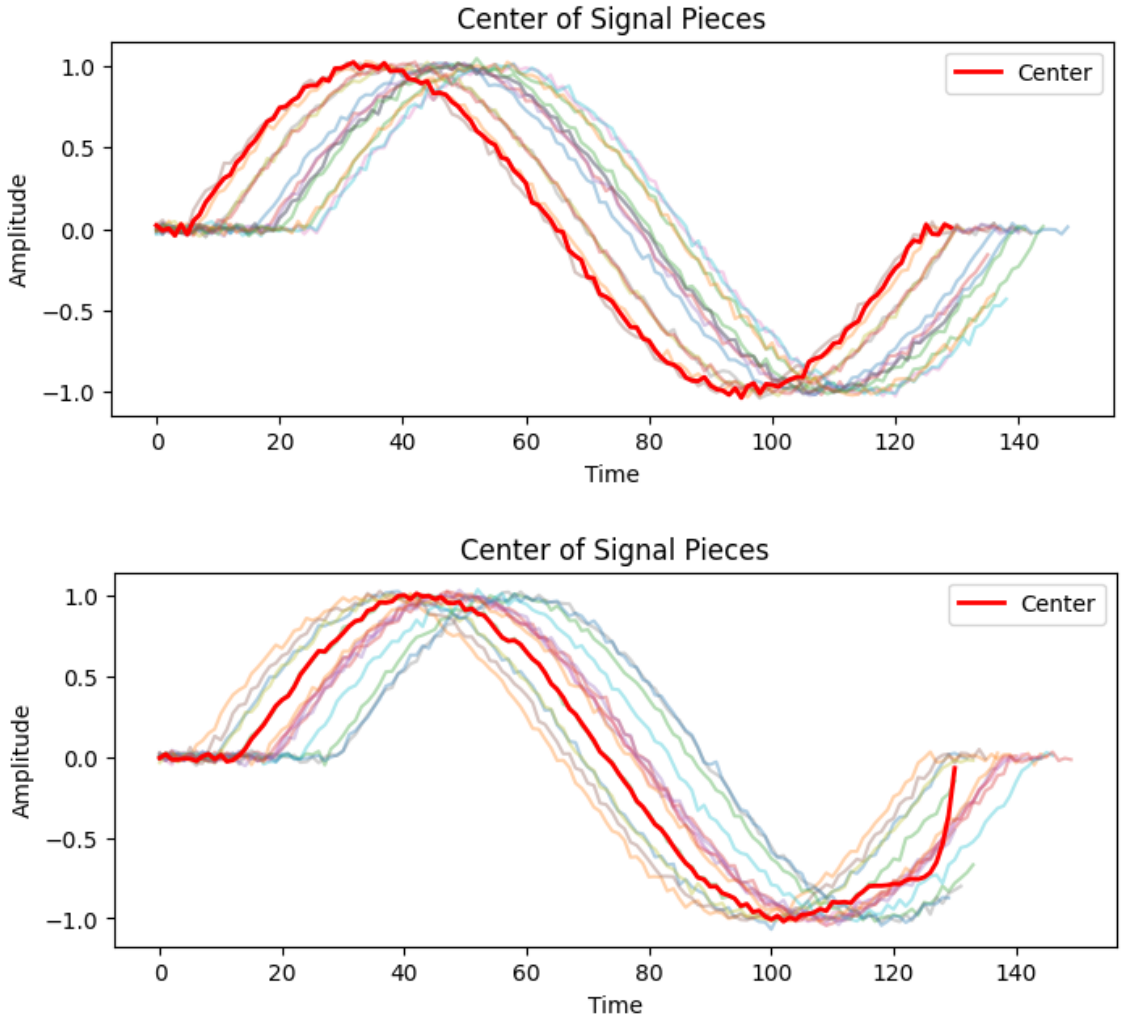


Figure: ICDTW and DBA

Table of Contents

Decomposition

1. Decompposition in signal processing is the breakdown of a complex signal into simpler components.

Seasonal Decomposition

Seasonal decomposition is a time series analysis method that separates a time series into its trend, seasonal, and residual components. 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.

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

$$Seasonal = sine_wave(duration = 10, sampling_rate = 100, frequency = 0.5)$$

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

$$signal = Seasonal + Trend + Residual$$

$$\_ = seasonal\_decomposition(signal, period = 200, show = True)$$

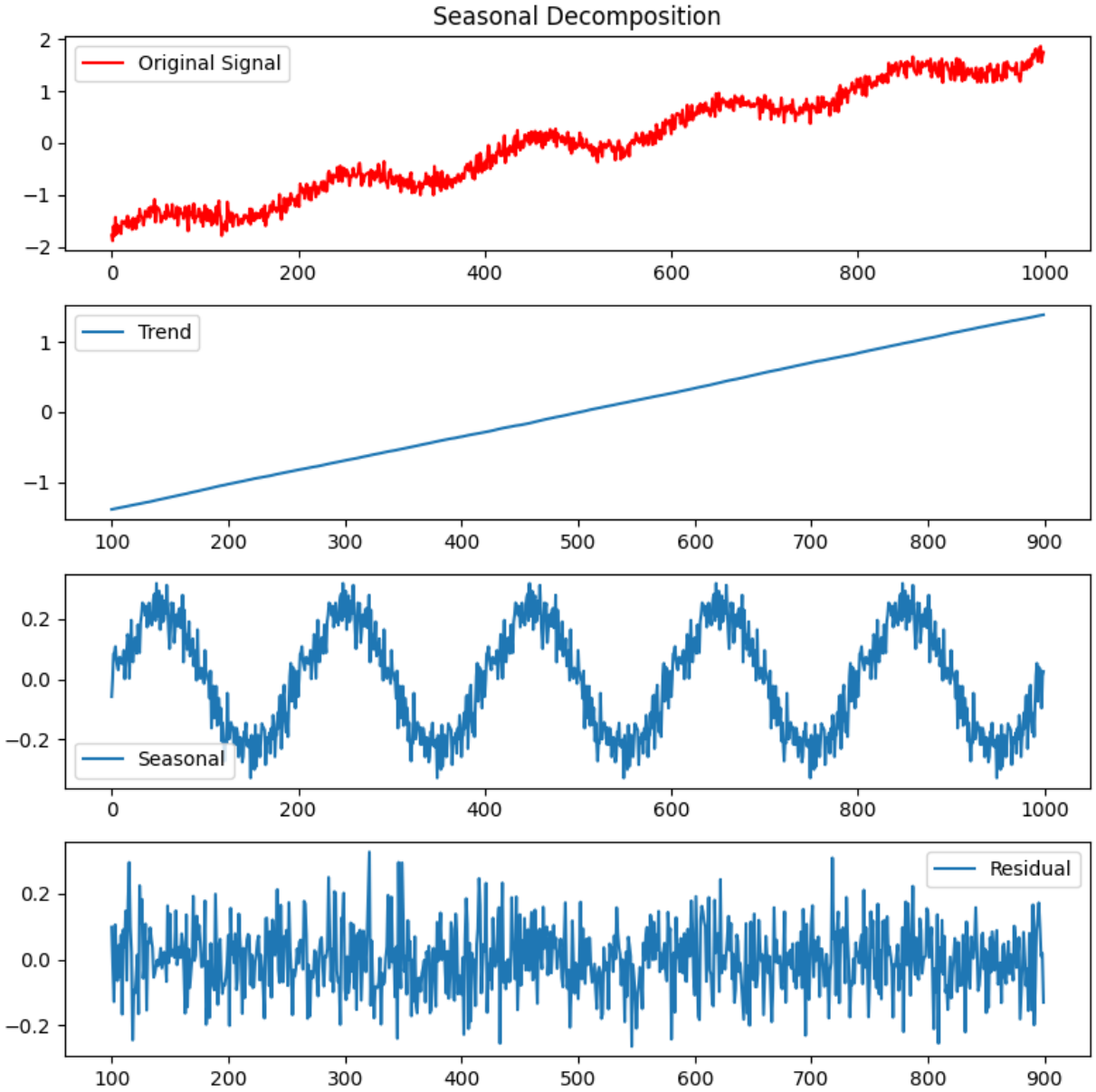


Figure: Seasonal Decomposition: Original, Trend, Seasonal, and Residual

Empirical Mode Decomposition (EMD)

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_decomposition(signal, show=True)

EMD Continued

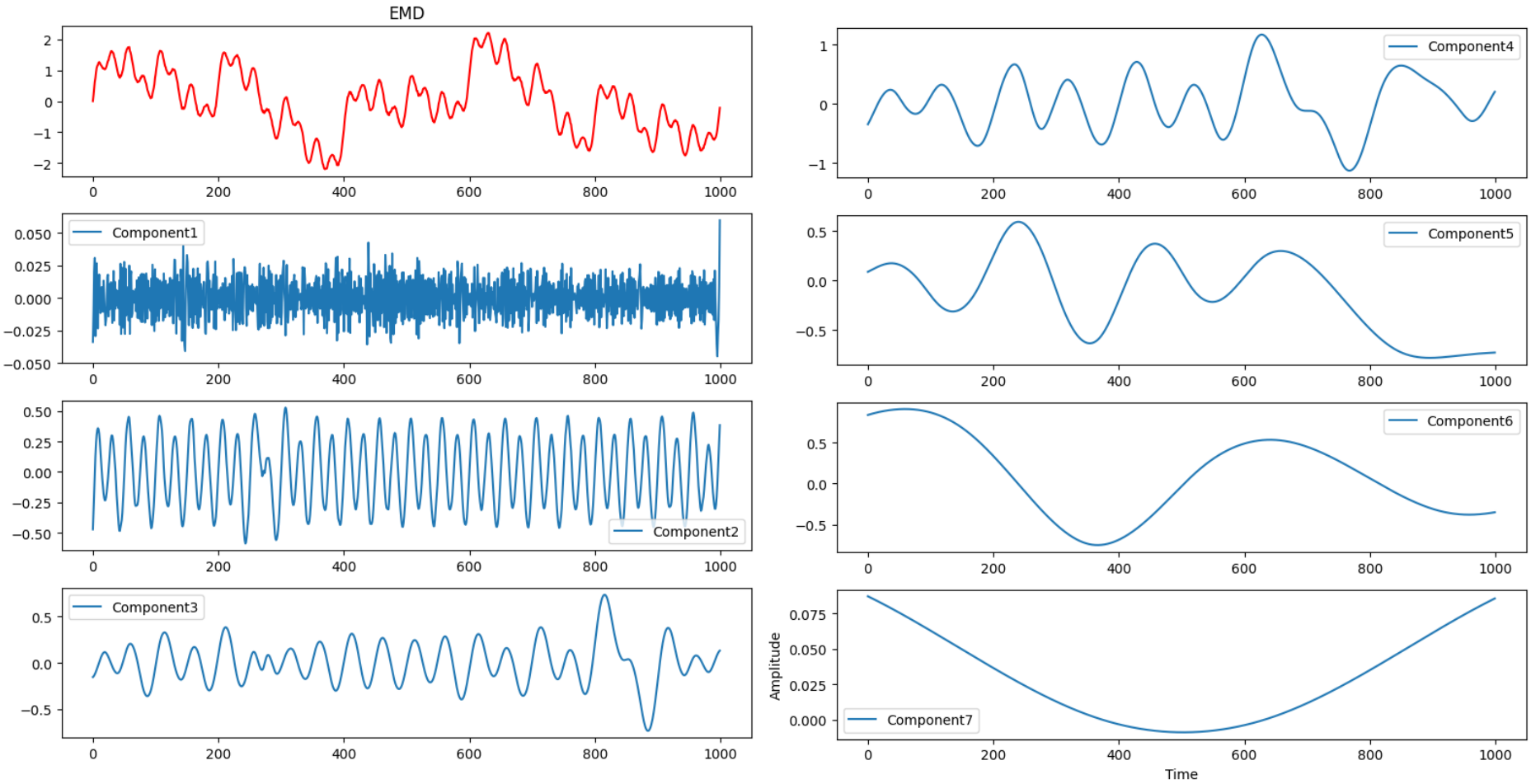


Figure: EMD

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.

eemd_decomposition(signal, show=True)

EEMD Continued

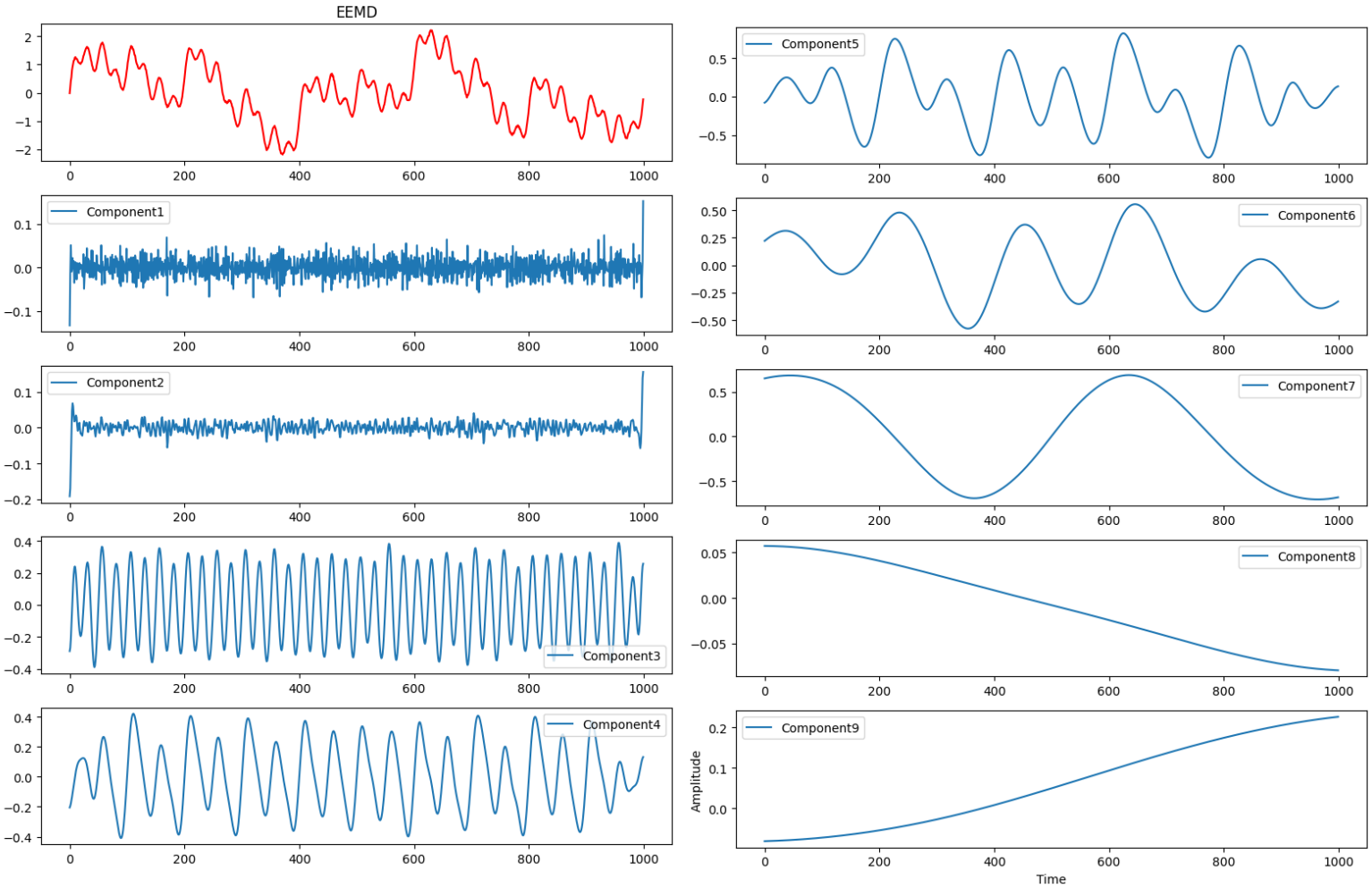


Figure: EEMD

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.

ceemd_decomposition(signal, show=True)

CEEMD Continued

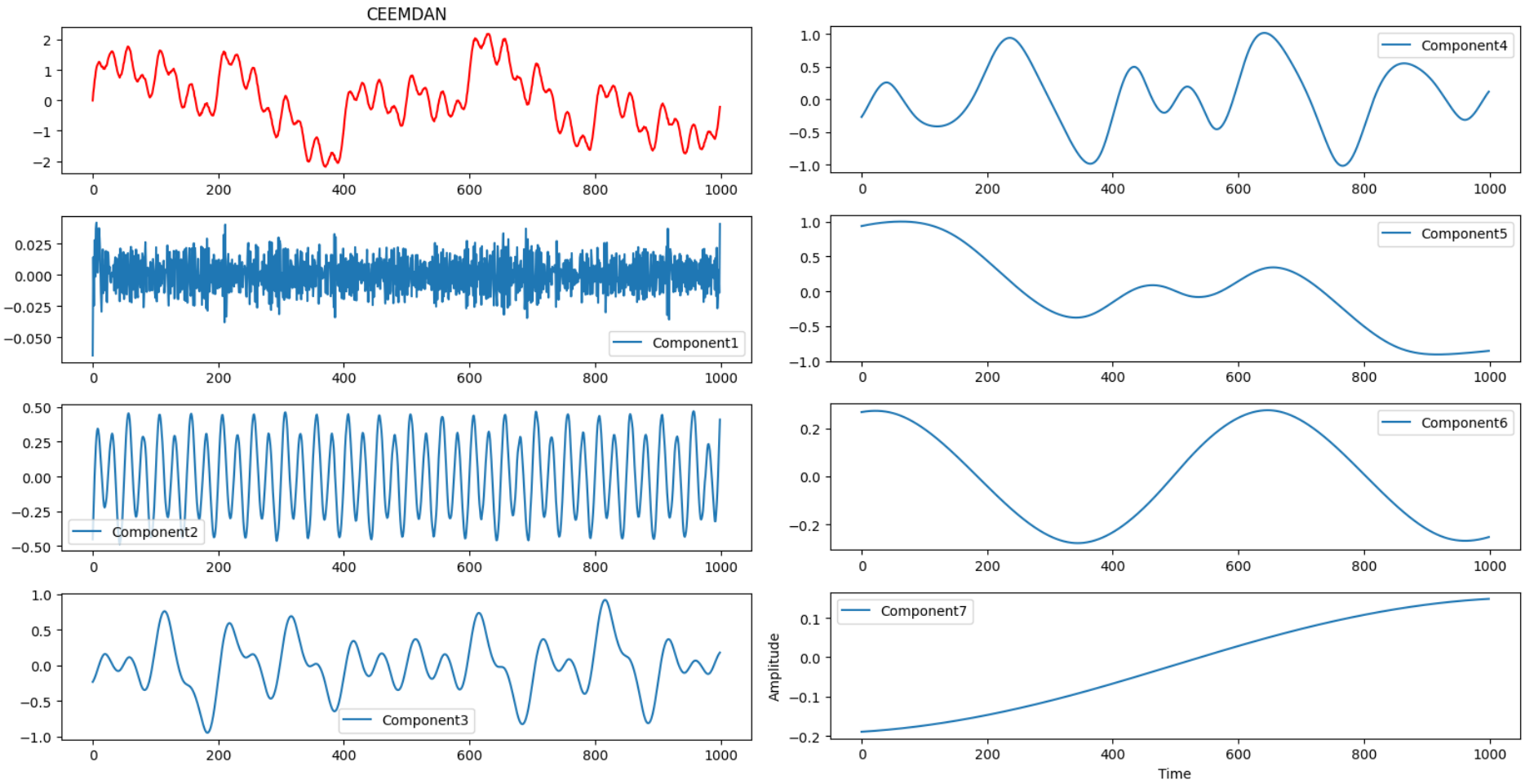


Figure: CEEMD

Variational Mode Decomposition (VMD)

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.

vmd_decomposition(signal, show=True)

VMD Continued

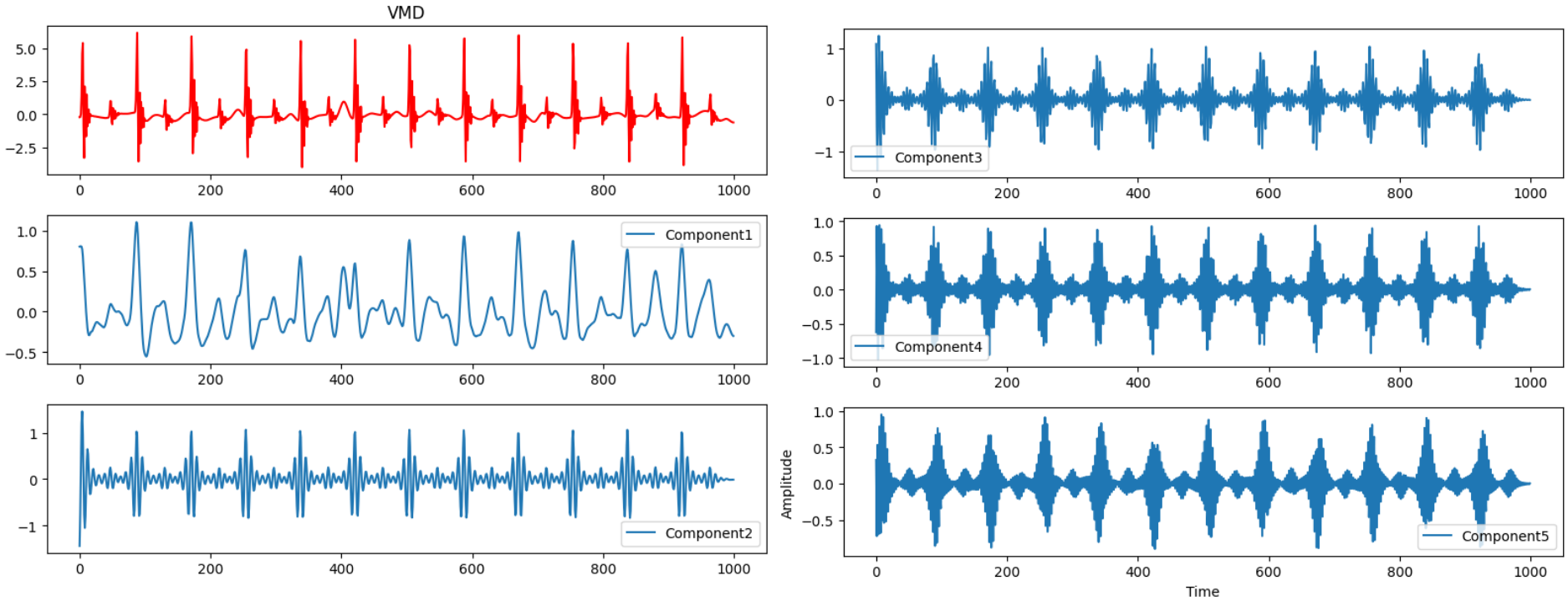


Figure: VMD

Singular-Spectrum Analysis

Singular-Spectrum Analysis (SSA) is a data analysis method used primarily for time series data. It is a non-parametric technique that transforms the data into a form that is easier to analyze and interpret, particularly for detecting patterns such as trends, cycles, and noise. SSA decomposes a time series into a set of components that can be analyzed individually to better understand underlying structures in the data.

Steps

1. **Windowing** Begins by creating a trajectory matrix from the time series data. It embeds the series in a higher-dimensional space using a sliding window (or lag) of a fixed length. It maps the time series into a matrix, where each row represents a shifted version of the original series.

SSA Continued

Steps Coninued

1. **Singular Value Decomposition (SVD)** The trajectory matrix is then subjected to Singular Value Decomposition, which decomposes it into singular vectors and singular values. SVD is a technique that helps identify the most important patterns in the data, effectively isolating dominant components from noise.

2. **Reconstruction** After SVD, the original time series is reconstructed using the most significant components (singular vectors). By reconstructing the time series with fewer components, SSA filters out noise and highlights key trends and periodic behaviors.

3. **Component Analysis** The resulting components can be analyzed to detect trends, oscillations, and anomalies, and can be used for forecasting, denoising, or feature extraction.

SSA Continued

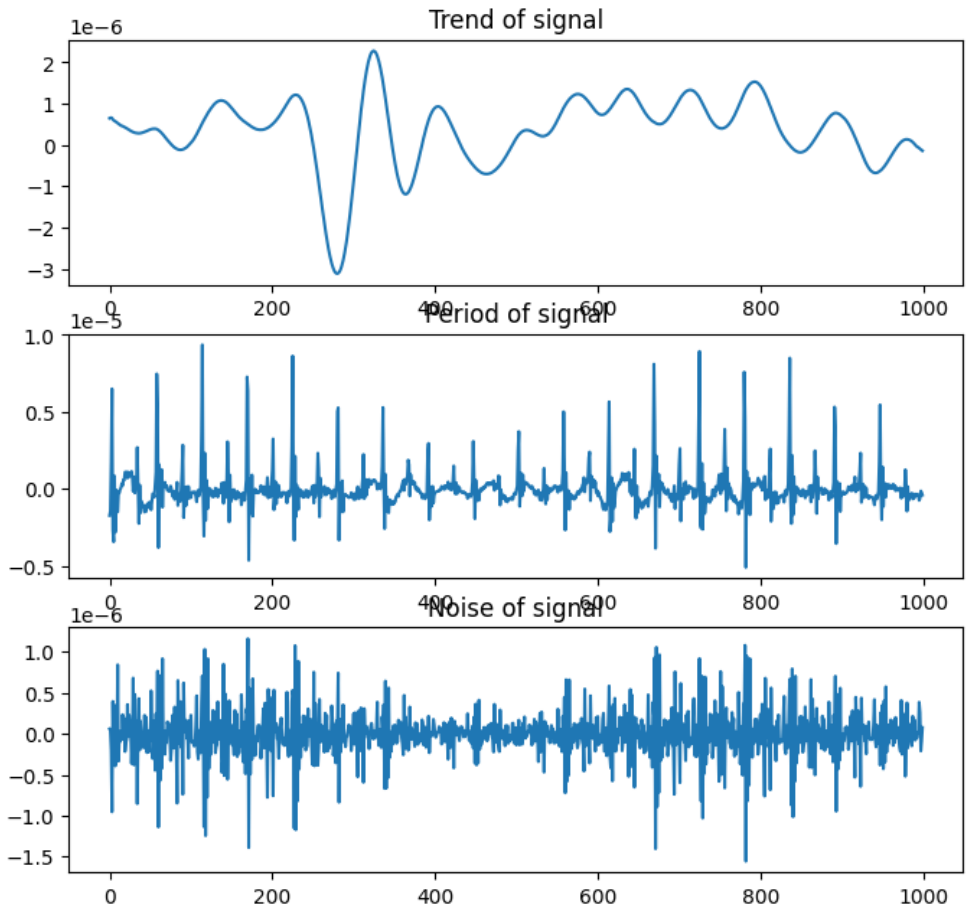


Figure: SSA: Trend, Period, Noise

Blind Source Seperation (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.

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

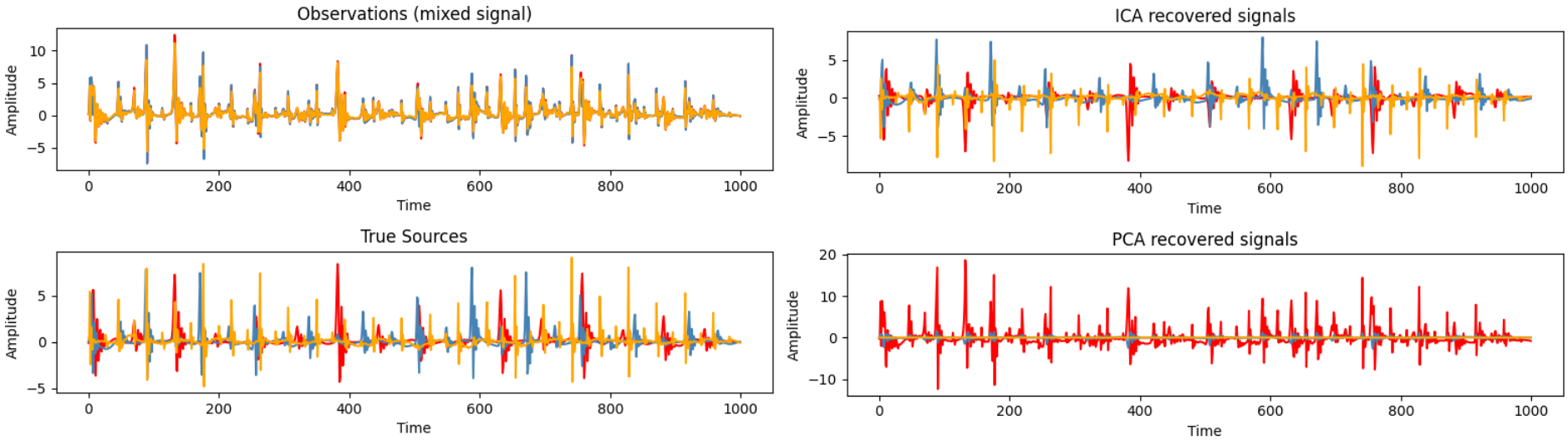


Figure: BSS: Observations, True Sources, ICA Recovery, PCA Recovery

Table of Contents

Time and Frequency Domain

1. 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.

Peaks & Envelopes

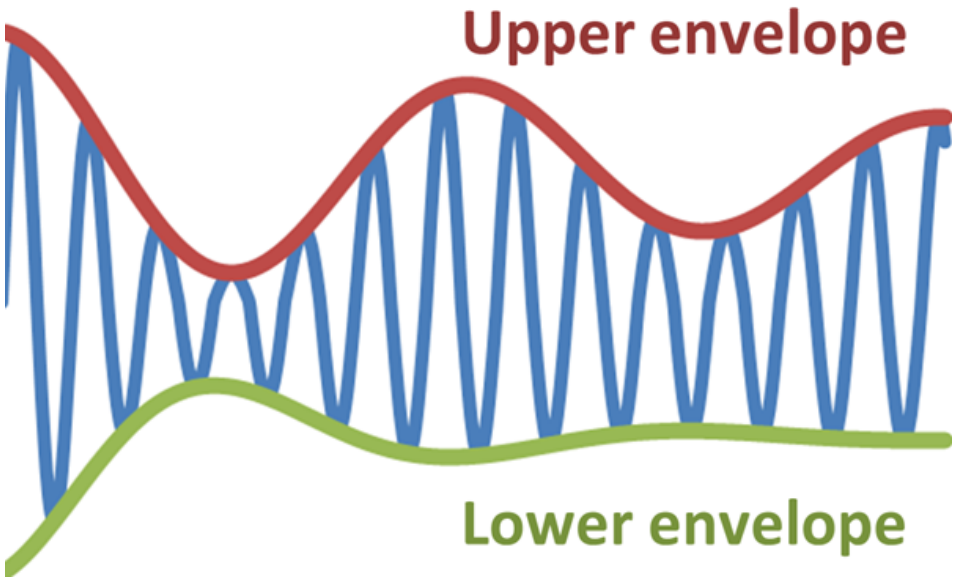


Figure: Peaks & Envelopes

Image from Wikipedia

Peaks

get_peaks(signal)

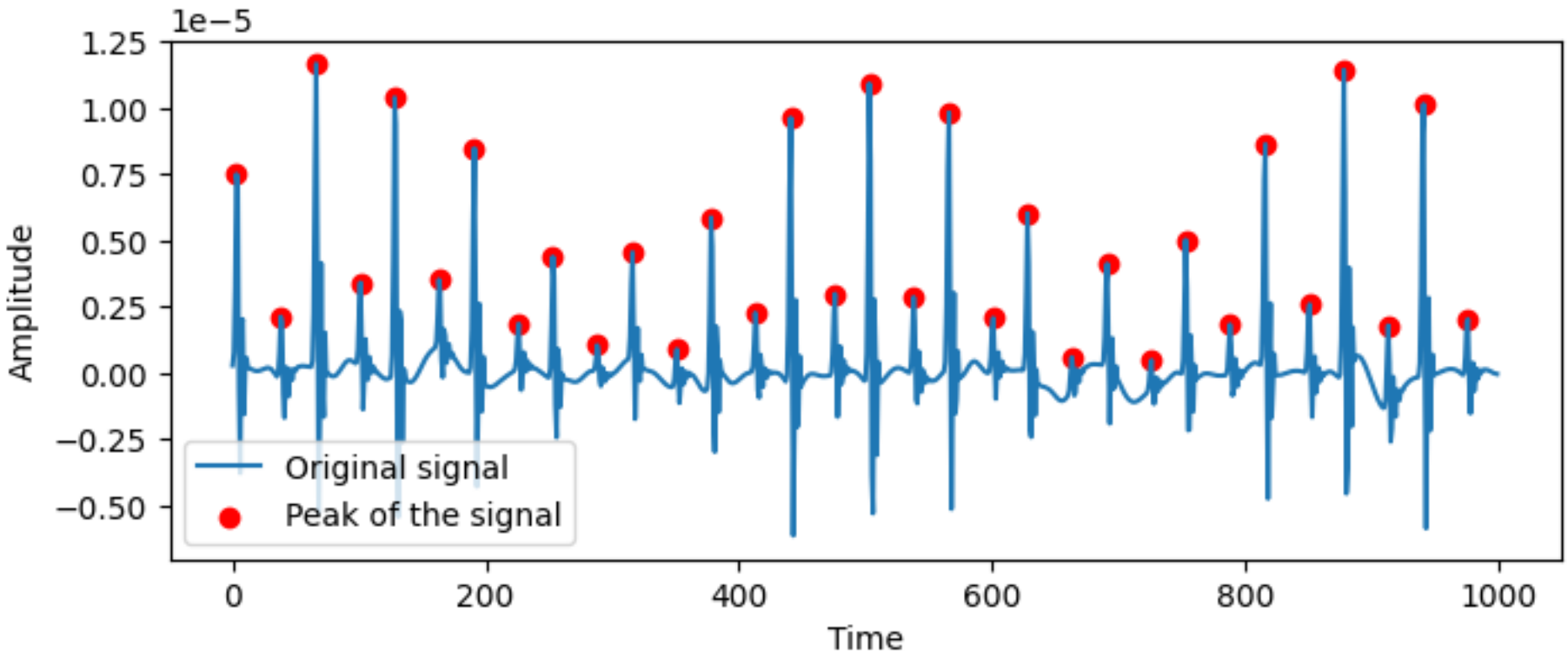


Figure: Get Peaks

Envelope

envelope_from_peaks(signal)

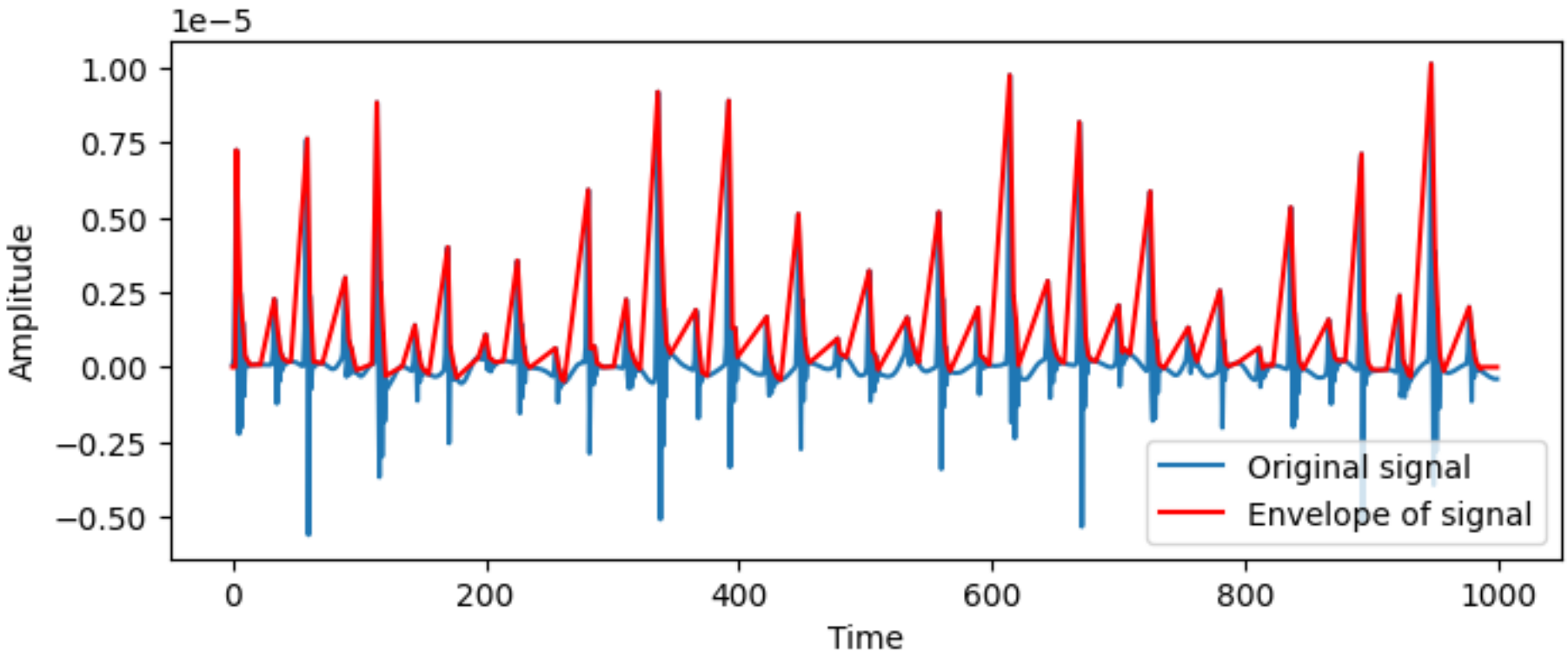


Figure: Get Envelope from Peaks

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.

$$envelope[i] = signal[i - \frac{window\_length}{2}] + ... + signal[i] + ... + signal[i + \frac{window\_length}{2}]$$

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)

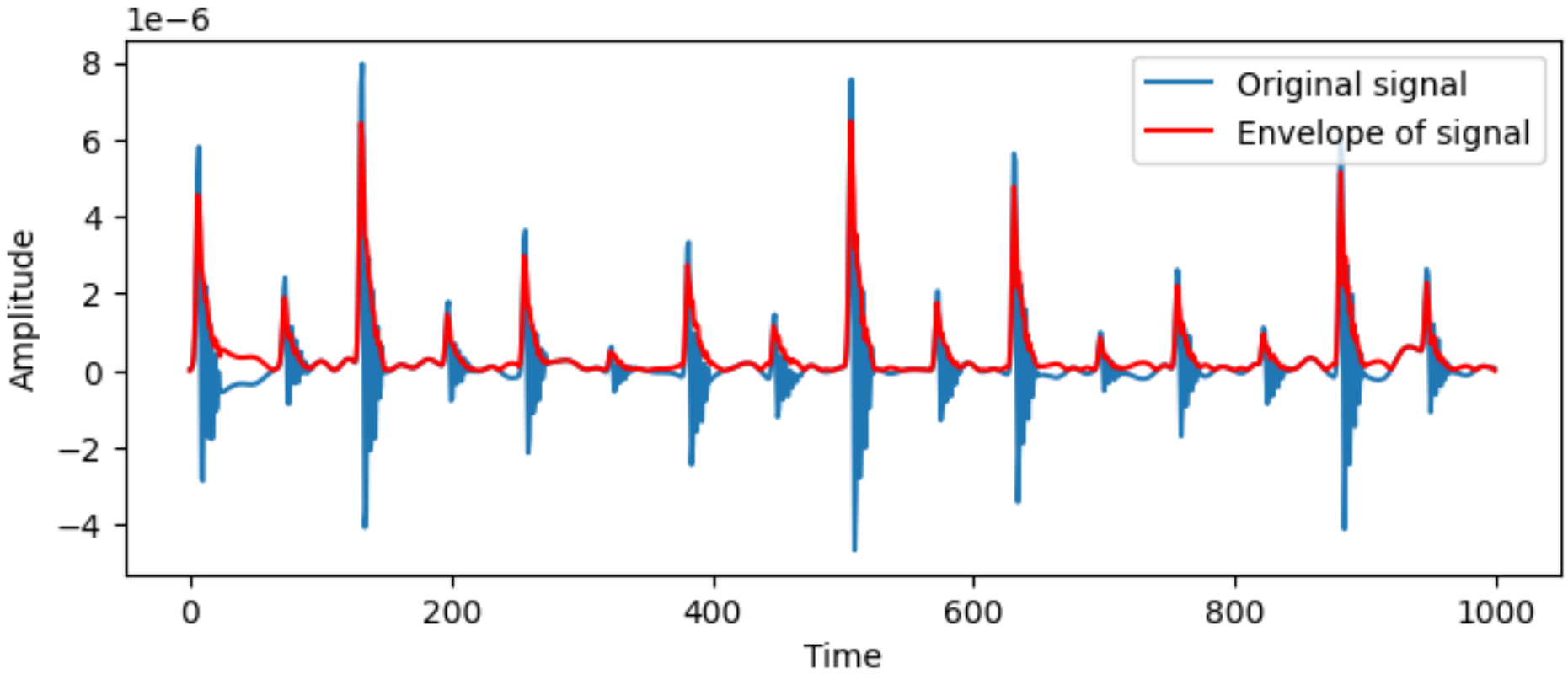


Figure: Get Average Envelope

Envelope & Phase Extraction with Hilbert Transformation

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.

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.

Hilbert Transformation Continued

inst_amplitude(signal)

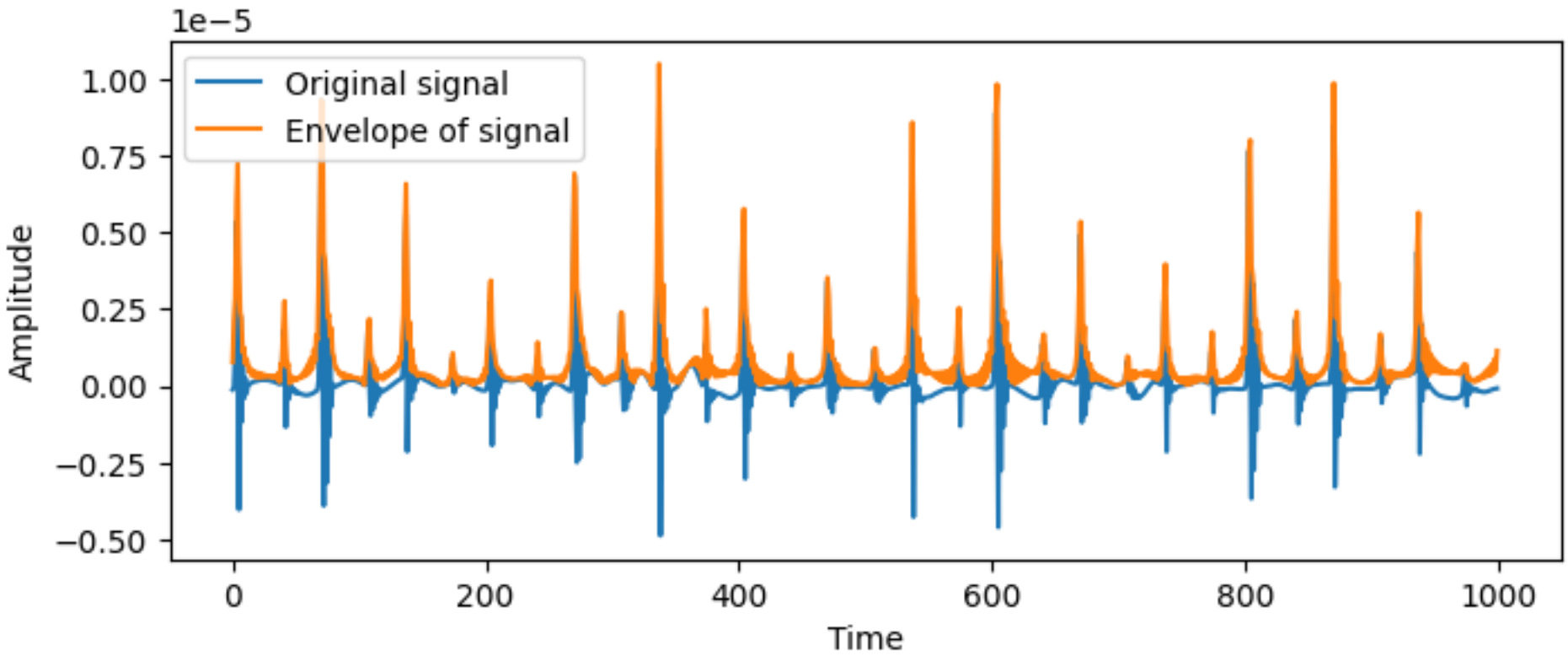


Figure: Envelope from Hilbert Transform

Singlular Spectrum Transform (SST)

Singular Spectrum Transform (SST) is a signal processing technique based on singular spectrum analysis, used for extracting trends and periodic components in a signal. SST is primarily employed for decomposing signals and reconstructing their components to better understand the structure and features of the signal. This technique is usually used for the change point detection. In the example, we can find that where the signal changes a lot has a relative high score.

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, Σ, and V^T. Here, U contains the left singular vectors, Σ 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.

SST Results

sst(signal, win_length=50)

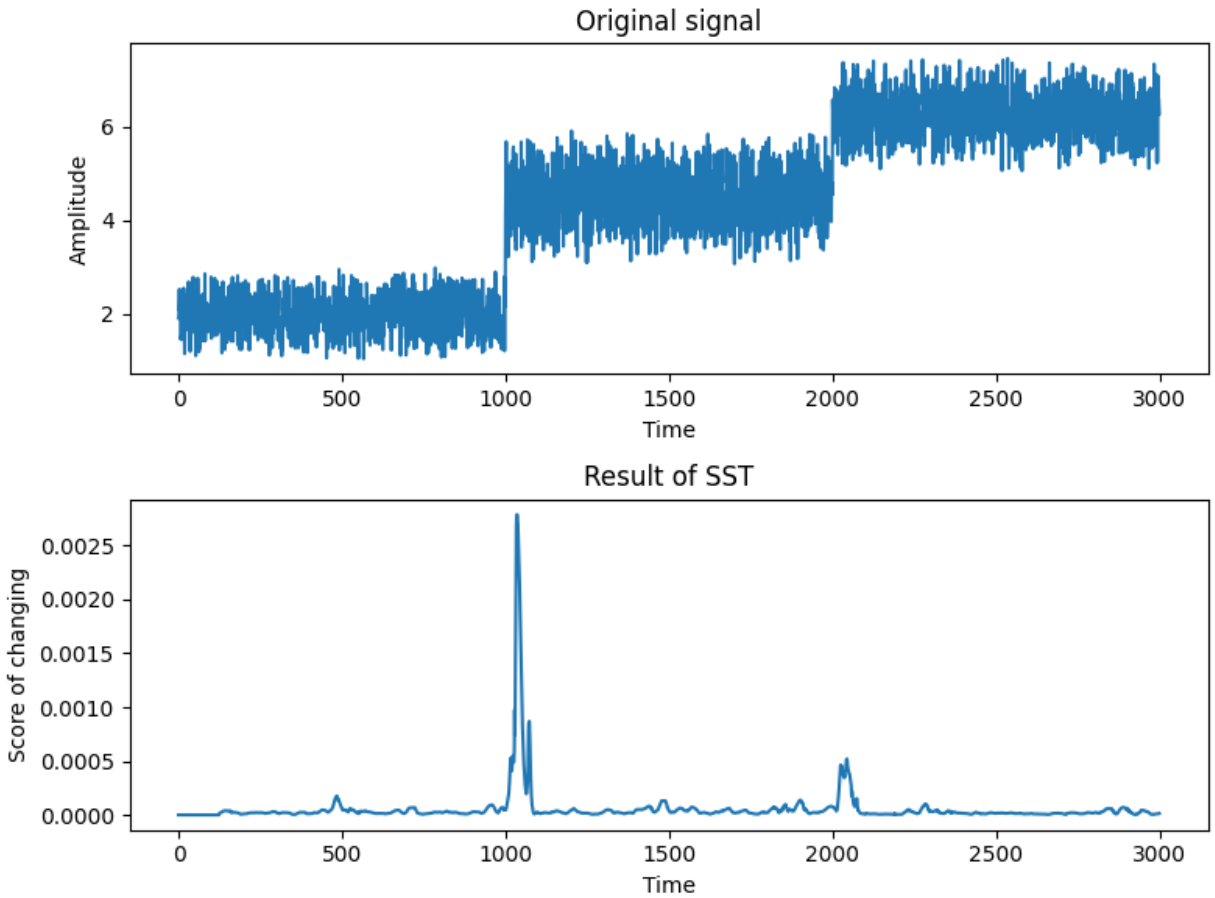


Figure: SST Results

Petrosian Fractal Dimension (PFD)

Petrosian Fractal Dimension (PFD) is a method used to measure the complexity of time-series signals, particularly applicable in biomedical signals such as electrocardiograms (ECG). It calculates the fractal dimension of a signal to describe its complexity and irregularity. A higher Petrosian Fractal Dimension value indicates a more complex signal.

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

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

The example shows that the signal with noise is more complex than the original signal, so that the noisy signal has a bigger PFD than the orignal signal.

PFD Continued

pfd(signal1)); pfd(signal2))

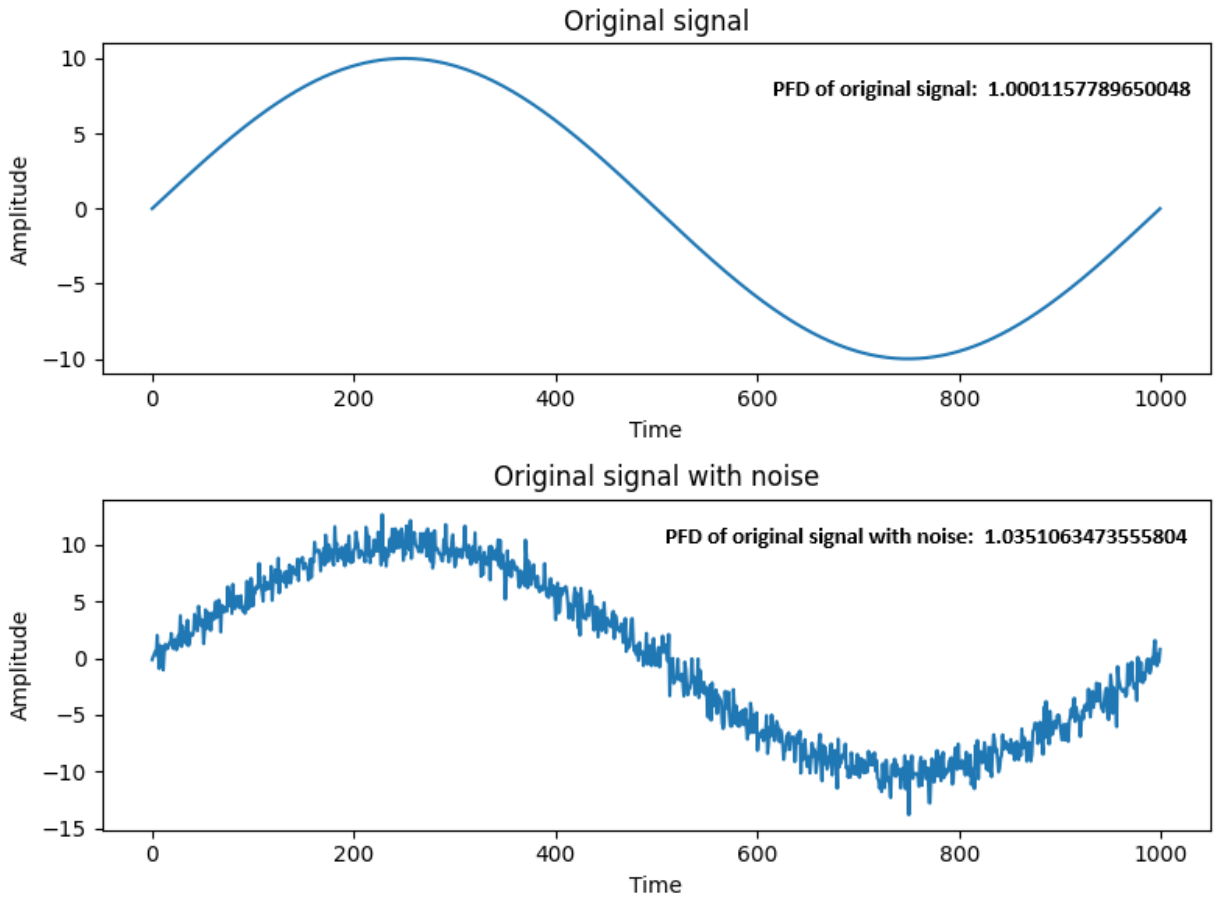


Figure: PFD Scores

Skewness

Skewness is a statistical measure that describes the degree of asymmetry of a distribution. It quantifies the extent to which a distribution deviates from symmetry around its mean.

$$Skewness = E[(X - \mu)^3]/\sigma^3$$

where E is the expected value operator,X is the random variable, μ is the mean of the distribution, σ is the standard deviation of the distribution. A skewness $value > 0$ means there is more weight in the right tail of the distribution. A skewness $value < 0$ means there is more weight in the left tail of the distribution.

Skewness Continued

skew(array1); skew(array2)

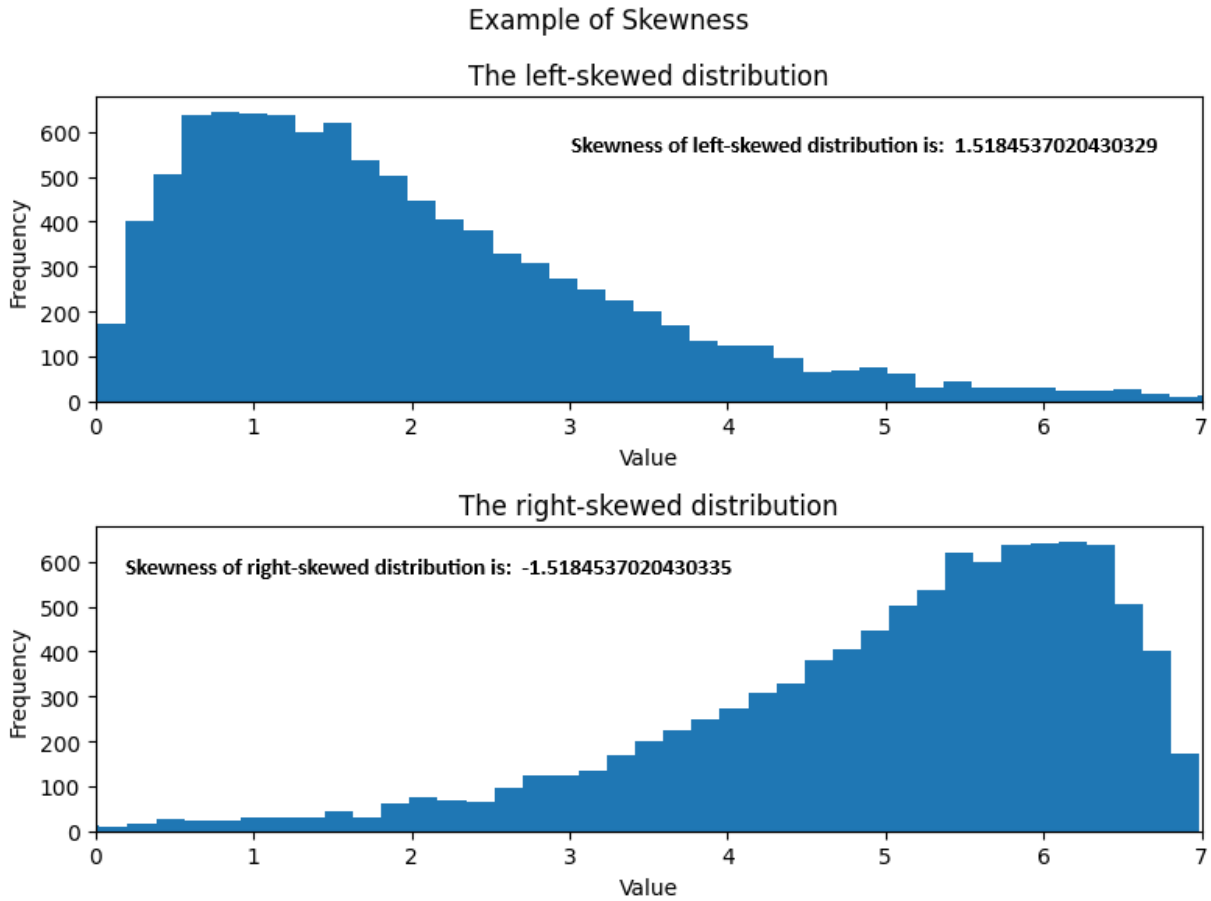


Figure: Skewness: Left Skewed & Right Skewed

Kurtosis

Kurtosis is a statistical measure that describes the "tailedness" of a distribution. It measures the thickness of the tails of a distribution relative to the rest of the data.

$$Kurtosis = E[(X - \mu)^4]/\sigma^4 - 3$$

where E is the expected value operator, X is the random variable, μ is the mean of the distribution, σ is the standard deviation of the distribution. The subtraction of 3 in Fisher's definition adjusts the kurtosis so that the normal distribution has a kurtosis of zero. Positive kurtosis indicates a "leptokurtic" distribution with heavier tails and a sharper peak. Negative kurtosis indicates a "platykurtic" distribution with lighter tails and a flatter peak.

Kurtosis Continued

kurtosis(array3); kurtosis(array4)

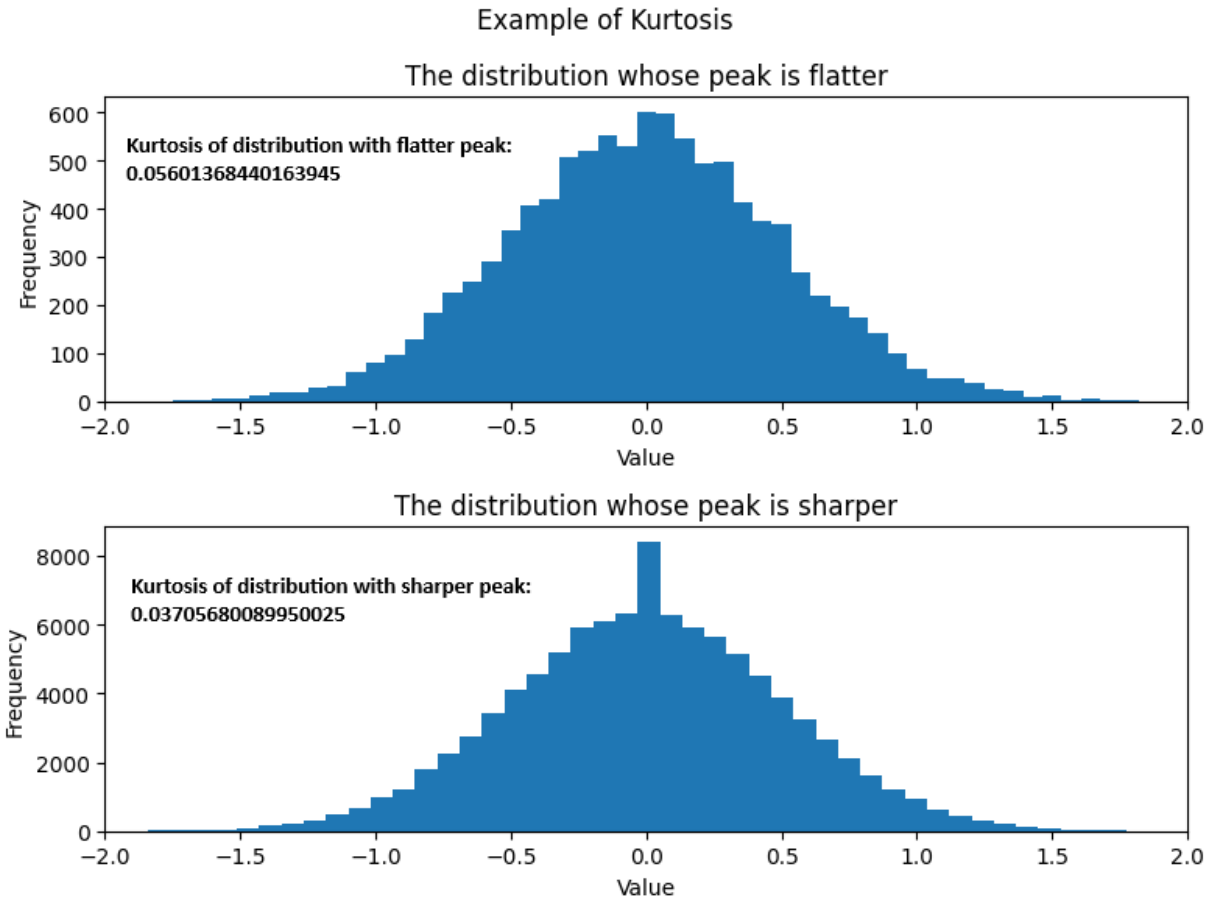


Figure: Kurtosis: Flat & Sharp

Fast Fourier Transform (FFT)

Fast Fourier Transform (FFT) is a fast algorithm to calculate the Discrete Fourier Transform (DFT)

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

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

where X denotes the result of FFT which is the spectrum, x denotes the time-domain signal and j is the imaginary unit.

my_fft function gets the original signal and the sampling rate as input and outputs the frequency and corresponding magnitude (a complex number). The length of the frequency is the half of the fs.

my_ifft function gets the magnitude of the **my_fft** function as input and outputs the original signal.

FFT Results

my_fft(signal, fs); my_ifft(mag)

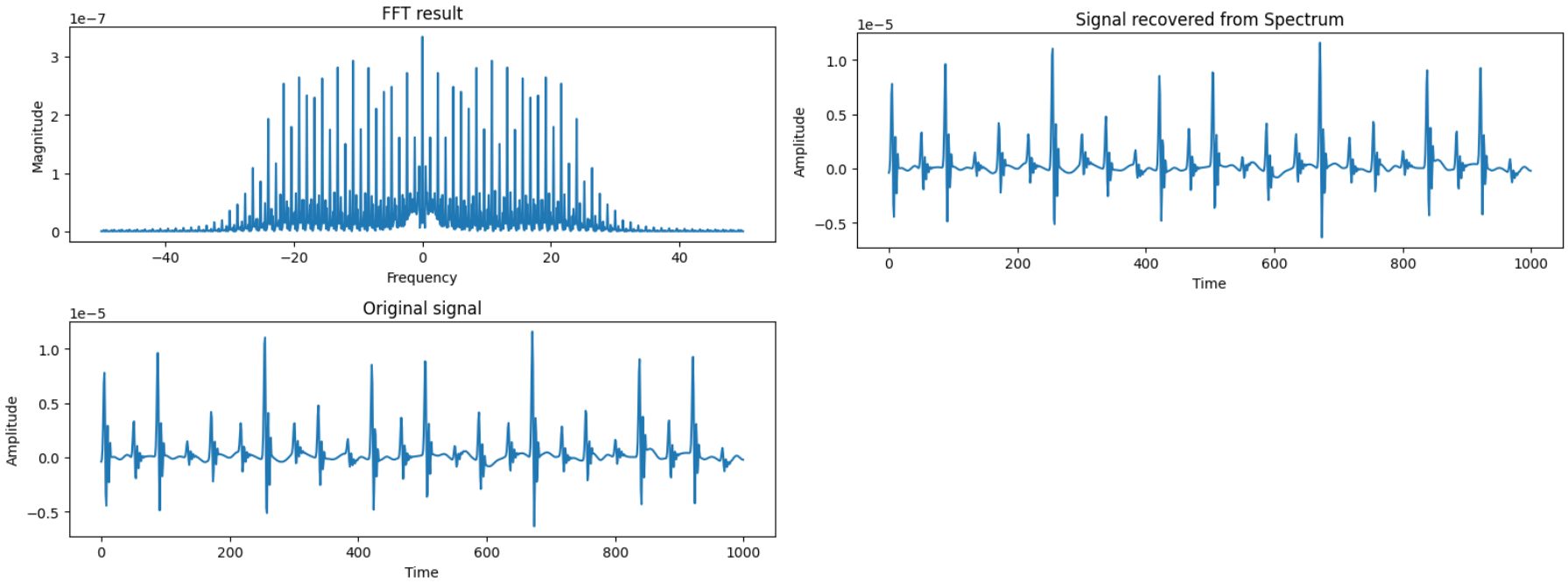


Figure: FFT Results

Power Spectral Density (PSD)

Power Spectral Density (PSD) is a tool used for frequency spectrum analysis to describe how the power of a signal varies with frequency.

The PSD represents the distribution of power across different frequencies in a signal, typically measured in power per Hertz (Hz). It can be computed by taking the squared magnitude of the Fourier transform of the signal. Specifically, for a time-domain signal x(t), the Power Spectral Density $S_{xx}(f)$ at frequency f is calculated as:

$$S_{xx}(f) = \lim_{T \to \infty} \frac{1}{T} \left| \int_{-T/2}^{T/2} x(t) e^{-j2\pi f t} \, dt \right|^2$$

where x denotes the time-domain signal and j denotes the imaginary unit.

PSD Results

psd(signal, fs)

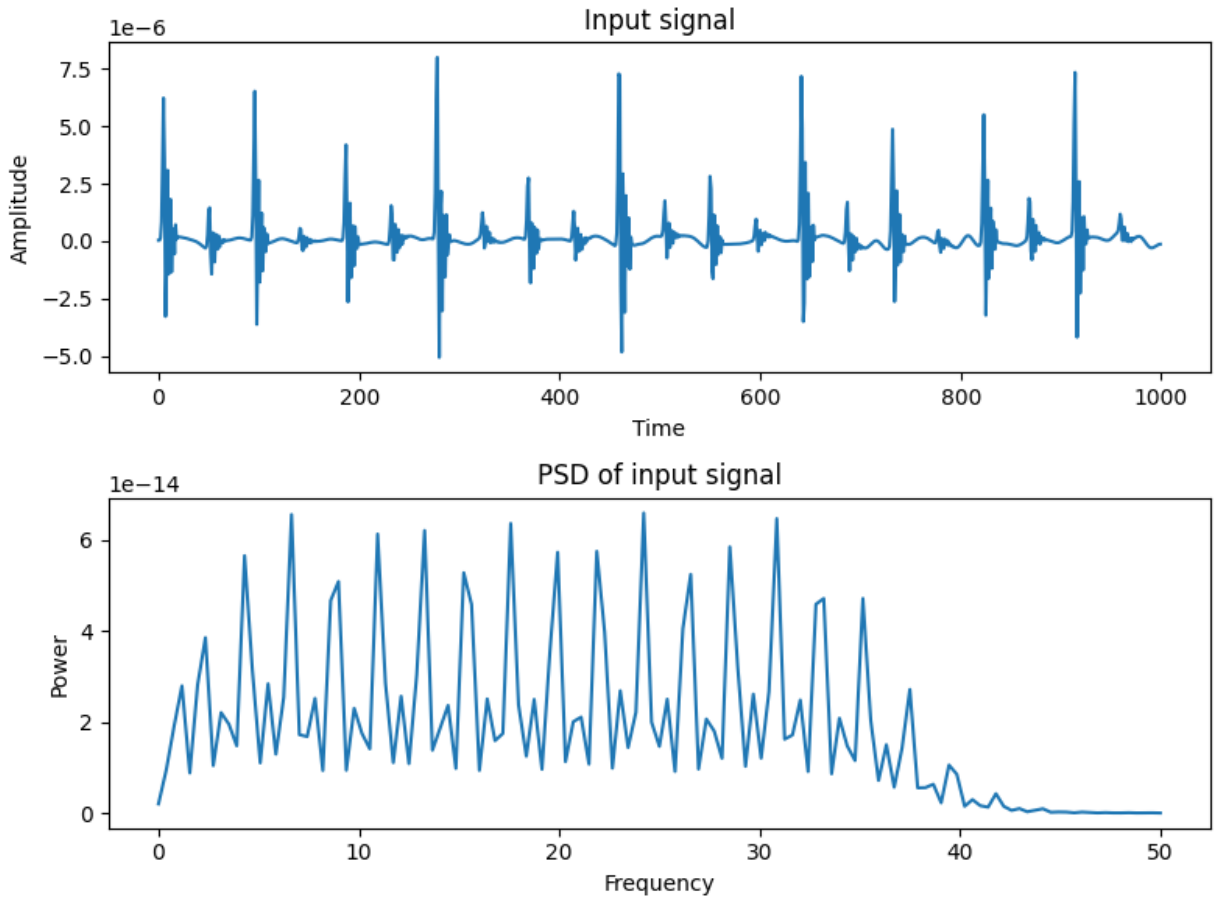


Figure: Input & PSD of Input

Short Time Fourier Transform (STFT)

Short Time Fourier Transform (STFT) - To extract signal information, we segment the signal and view the segment as stationary. However, when the signal does not complete an integer number of cycles, an issue known as Frequency Leakage may occur which can pose challenges in reconstructing the original signal based on the spectrum. To minimize the impact of Frequency Leakage, the original signal can be multiplied by a window function. After framing, we get many framed segment. Then we do the DFT to get frequency information of each frame. The scale of y axis is frequency. Then we concatenate the spectrum horizontally to get the spectrogram. The spectrogram has both the frequency and time frequency of a signal.

STFT Continued

The longer the frame, the frequency information is more accurate, the time information is less accurate. The above is what we call Heisenberg uncertainty principle.

The output of the STFT is y-axis (frequency), x-axis (time) and the value of each point (magnitude). The max number of y-axis (frequency) is the half of the fs which is based on the Nyquist sampling princple, but the number of frequency in y-axis is the half of the nperseg. The number of the x-axis depends on the nperseg and the noverlap and the max number of the x-axis is (length of signal / fs). The output Z is complex-valued.

STFT Results

f, t, Z = my_stft(signal, fs=fs, window=window, nperseg=nperseg)

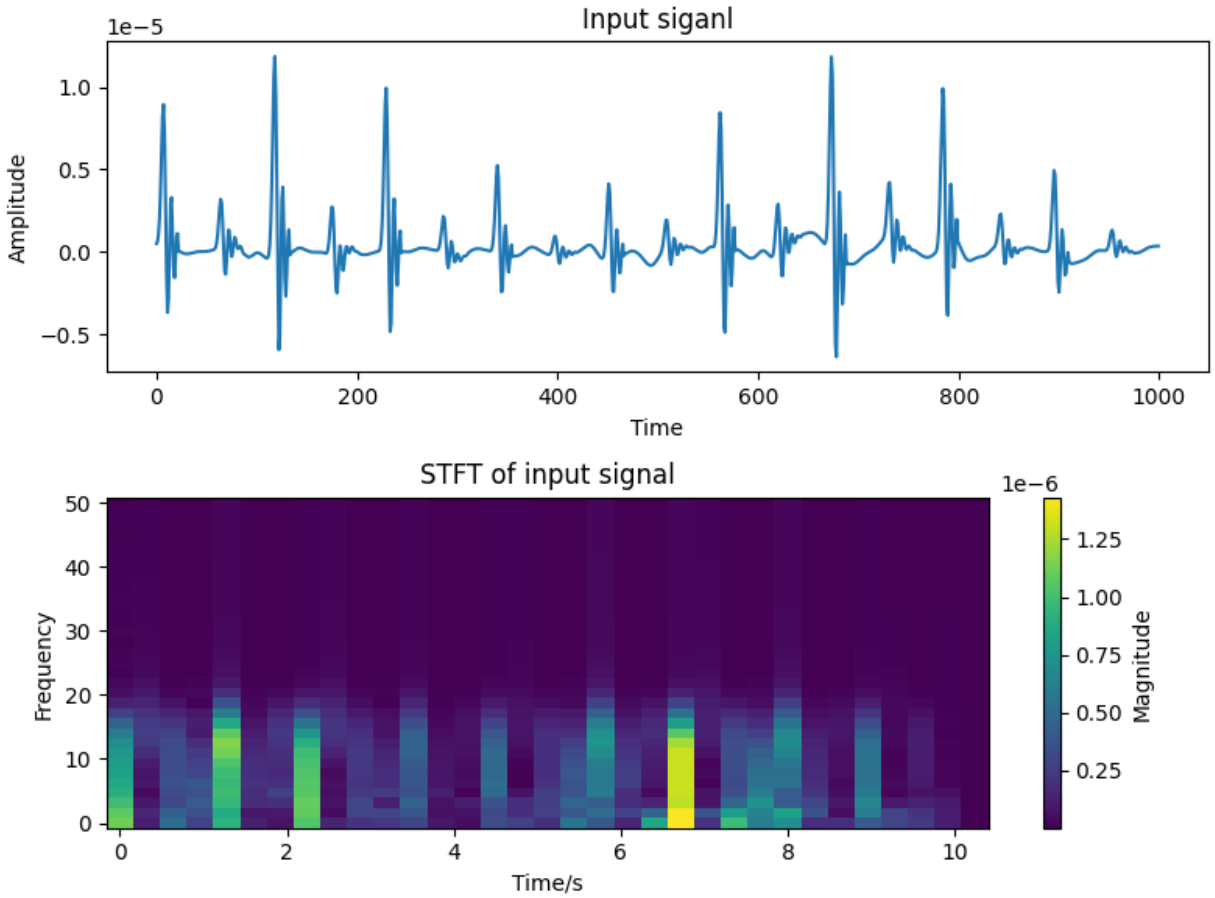


Figure: STFT

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 σ 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)

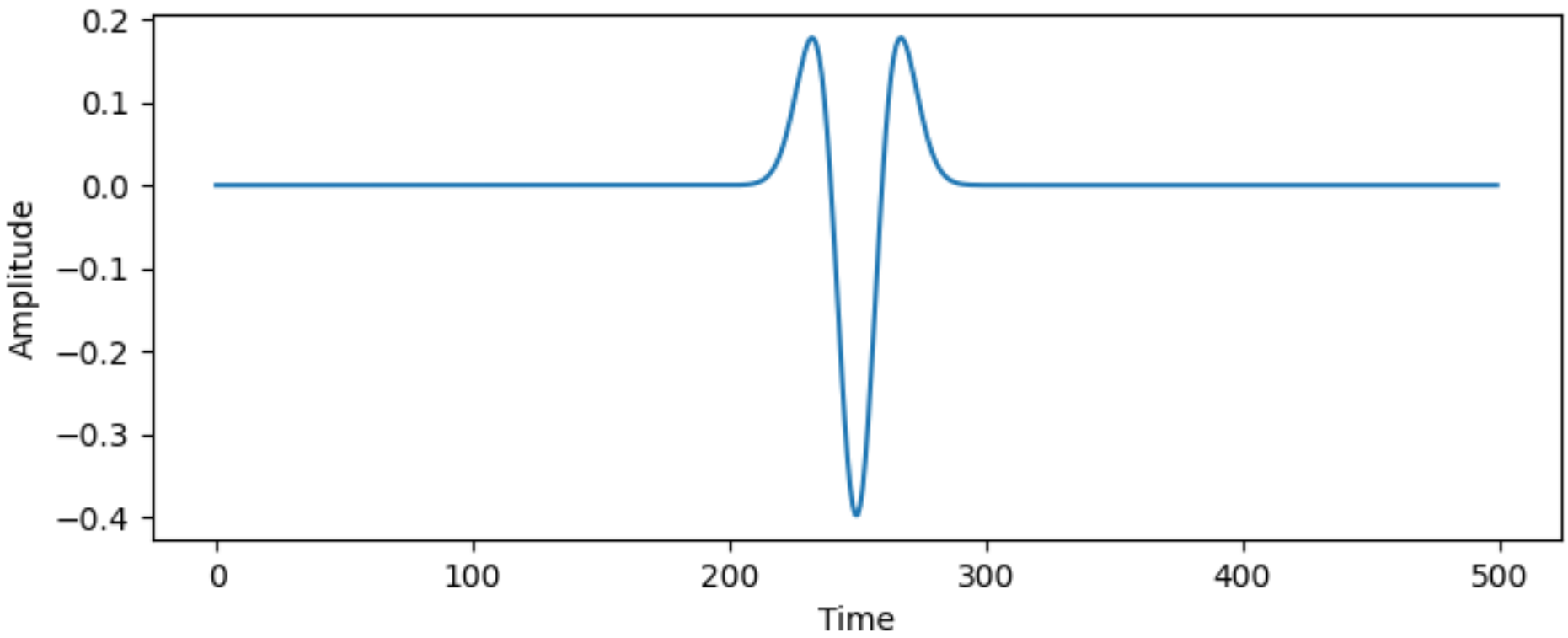


Figure: Mexican Hat Wavelet

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 σ controls the width of the Gaussian envelope.

Mortlet Wavelet Continued

morlet_wavelet(length, sigma, a)

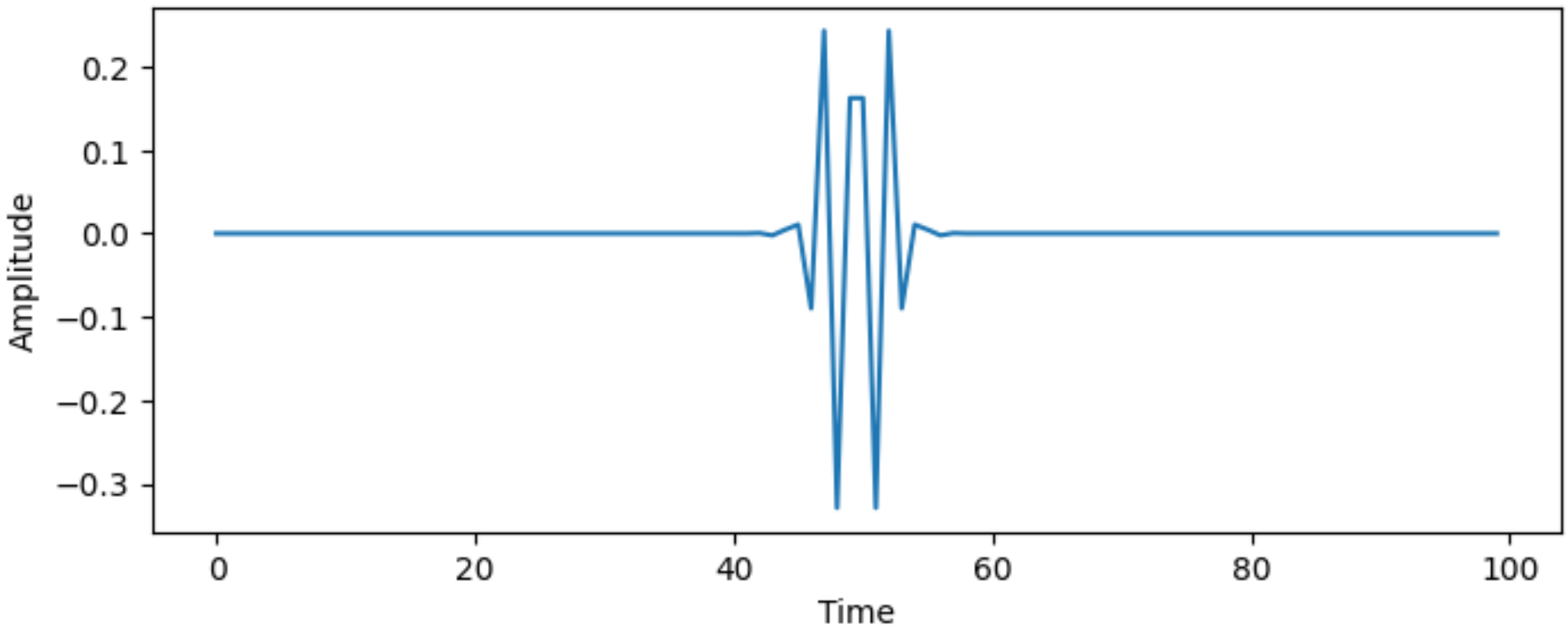


Figure: Morlet Wavelet

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 ψ denotes the mother wavelet, τ denotes the translation of the wavelet, and s denotes the sacle of the wavelet. τ 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_ψ is a constant that depends on the wavelet used and the other symbols are described in above cell. c_ψ 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)

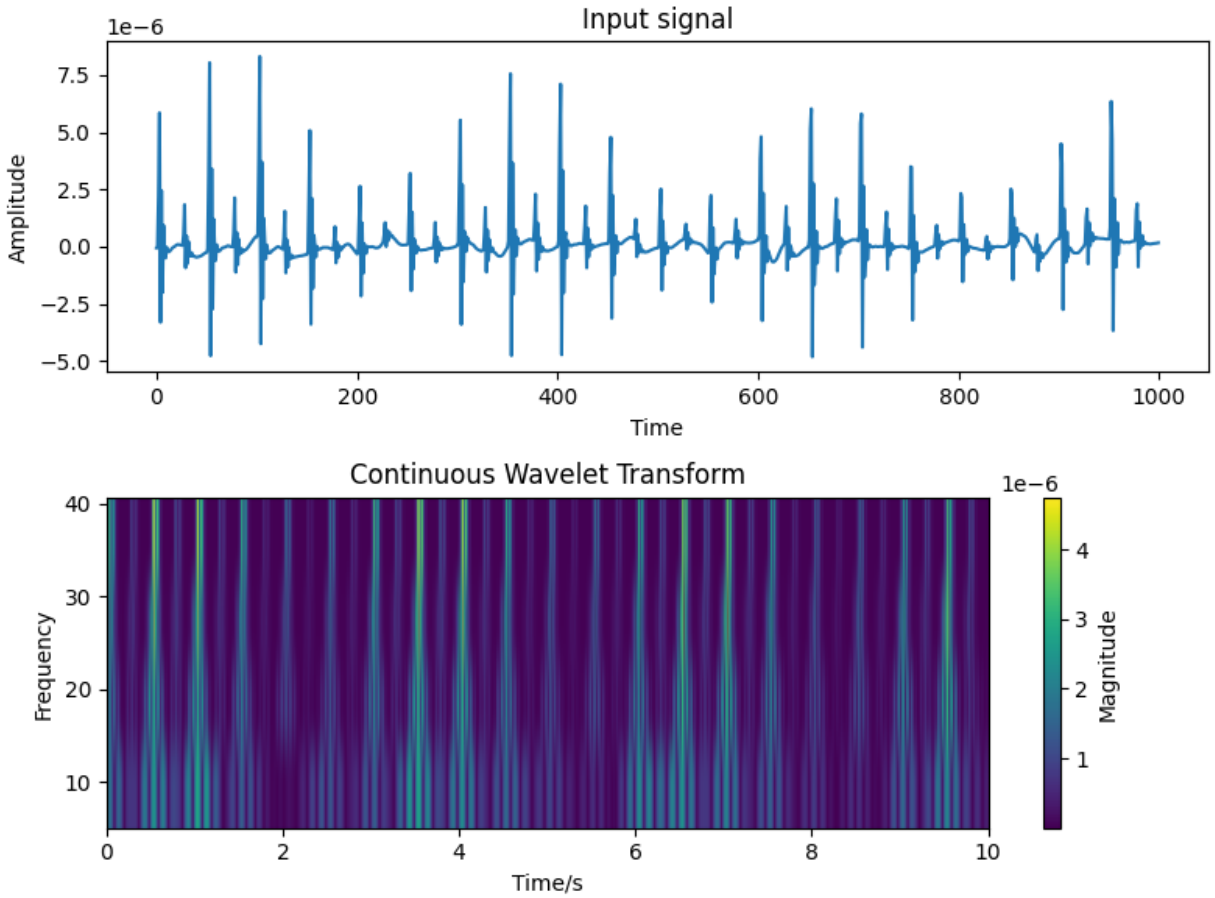


Figure: CWT with a Morlet Wavelet

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 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)

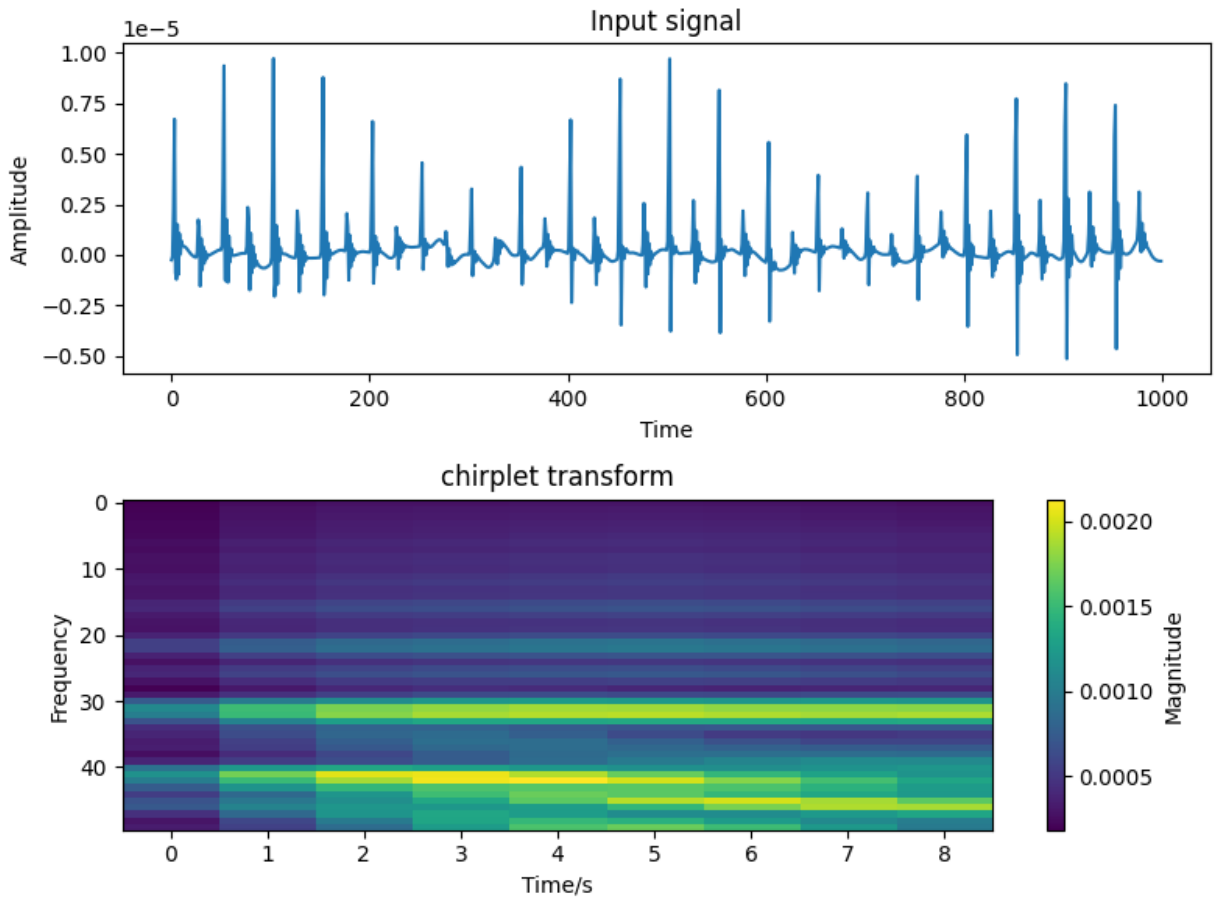


Figure: Chirplet Transform

Table of Contents

Thank you!

E-mail: stephen.coshatt@uga.edu