My Introduction to Digital Signal Processing (DSP)

Conor Sheehan

July 19, 2024

Contents

1 What is Digital Signal Processing (DSP)?

Digital Signal Processing (DSP) is the mathematical manipulation of an information signal to modify or improve it in some way. It involves applying algorithms to digital signals to achieve various objectives such as filtering, compression, and feature extraction. The primary goal of DSP is to measure, filter, and/or compress continuous real-world analog signals.

1.1 Applications of DSP

DSP is used in a wide range of applications across various fields. Some notable applications include:

- **Audio and Speech Processing:** Noise reduction, echo cancellation, and audio compression.
- **Image and Video Processing:** Image enhancement, compression, and object recognition.
- **Communications:** Modulation, demodulation, error detection, and correction.
- **Biomedical Engineering:** ECG signal processing, medical imaging, and hearing aids.
- **Air Traffic Control:** Signal processing for radar and communication systems, leveraging sparse matrices for efficient data handling and processing.

1.1.1 DSP in Air Traffic Control

In air traffic control, DSP is crucial for processing radar signals and communication systems. Radar systems rely on DSP techniques to detect and track aircraft by processing the reflected radar signals. Sparse matrices are particularly useful in this context as they efficiently handle the large, sparse datasets typical in radar signal processing, reducing computational load and improving real-time performance.

1.2 Overview of DSP Filters and Concepts

Below is a brief introduction to some fundamental DSP filters and concepts:

- **Low Pass Filter:** Allows signals with frequencies lower than the cutoff frequency to pass through and attenuates frequencies higher than the cutoff frequency.
- **High Pass Filter:** Allows signals with frequencies higher than the cutoff frequency to pass through and attenuates frequencies lower than the cutoff frequency.

- **Band Pass Filter:** Allows signals within a specific frequency range to pass through and attenuates frequencies outside this range.
- **Band Stop (Notch) Filter:** Attenuates signals within a specific frequency range and allows frequencies outside this range to pass through.
- **Sampling and Nyquist Theorem:** Ensures a signal is sampled at a rate that captures all necessary information without aliasing.
- **Fast Fourier Transform (FFT):** Efficiently computes the discrete Fourier transform (DFT) of a signal, revealing its frequency components.
- **Windowing:** Applies a window function to a signal to reduce spectral leakage before performing an FFT.
- **Convolution:** Combines two signals to form a third signal, representing the amount of overlap between the signals as one is shifted over the other.
- **Quantization:** Converts a continuous range of values into a finite range of discrete values, simulating analog-to-digital conversion.

1.3 Impulse Response

What is Impulse Response?

In digital signal processing, the impulse response of a system is the output when the input is a discrete-time unit impulse signal, usually denoted as $\delta[n]$. The impulse response characterizes the system completely and can be used to determine the output for any input signal through the process of convolution.

Mathematical Representation

For a discrete LTI (Linear Time-Invariant) system, if $h[n]$ is the impulse response and $x[n]$ is the input signal, the output $y[n]$ is given by:

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

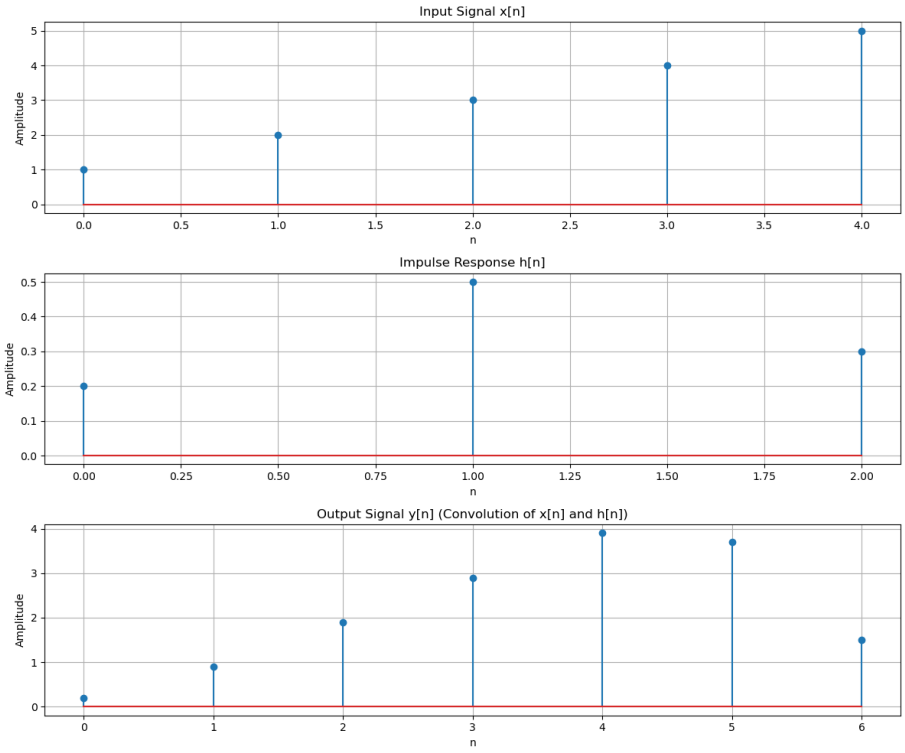
Example in Python

Let's consider a simple Finite Impulse Response filter with an impulse response $h[n] = [0.2, 0.5, 0.3]$. We'll show how an input signal can be represented as a linear combination of shifted impulses and then compute the output.

Explanation

- **Input Signal** $x[n]$: The discrete input signal we are working with.
- **Impulse Response** $h[n]$: The filter coefficients or impulse response of the system.

Figure 1: Impulse Response Example



- **Convolution**: The output signal $y[n]$ is computed by convolving the input signal with the impulse response, effectively applying the filter to the input. This uses the equation mentioned above for convolution. As the Impulse Response here is only 3 items, only 3 terms in the input impulse contribute to the final output.

Plot

1. The first subplot shows the input signal $x[n]$.
2. The second subplot shows the impulse response $h[n]$.
3. The third subplot shows the output signal $y[n]$, which is the result of convolving $x[n]$ with $h[n]$.

Linear Combination of Inputs

Any discrete input signal can be represented as a linear combination of shifted impulses. For example, the input signal $x[n] = [1, 2, 3, 4, 5]$ can be written as:

$$x[n] = 1 \cdot \delta[n] + 2 \cdot \delta[n-1] + 3 \cdot \delta[n-2] + 4 \cdot \delta[n-3] + 5 \cdot \delta[n-4]$$

Each shifted impulse $\delta[n-k]$ is scaled by the corresponding value of the input signal.

Output Calculation

By convolving the input signal $x[n]$ with the impulse response $h[n]$, we get the output $y[n]$. This operation sums up the scaled and shifted versions of the impulse response, giving us the complete output signal.

This simple example and code demonstrate the fundamental concept of impulse response and its role in determining the output of a discrete-time LTI system.

1.4 Frequency Response in DSP

Introduction

In digital signal processing (DSP), the frequency response of a system is a crucial concept that characterizes how the system responds to different frequency components of an input signal. It is a measure of the system's output spectrum in response to an input signal and is typically represented by two functions: the magnitude response and the phase response.

Magnitude Response

The magnitude response indicates how the amplitude of each frequency component is modified by the system. It is essential for understanding how different frequencies are attenuated or amplified by the system.

Phase Response

The phase response shows the phase shift introduced by the system for each frequency component. This is important for understanding the time delay and phase distortion characteristics of the system.

Mathematical Representation

Mathematically, the frequency response is the Fourier transform of the system's impulse response. Given an impulse response $h[n]$, the frequency response $H(\omega)$ can be computed using the Fast Fourier Transform (FFT).

Python Example

In the Python code, we use the frequency response of a simple FIR filter with an impulse response $h[n] = [0.2, 0.5, 0.3]$.

Visualization

The frequency response of the FIR filter is visualized in the following plots, which show the magnitude and phase responses. These plots help in understanding how the filter affects different frequency components of the input signal.

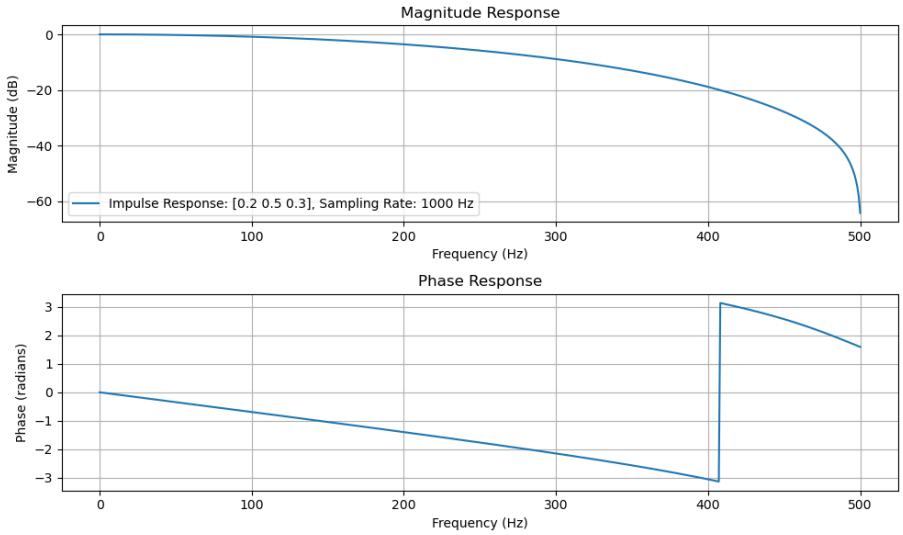


Figure 2: Frequency Response of the FIR Filter

2 Low Pass Filters

- **Definition:** A low pass filter allows signals with a frequency lower than a certain cutoff frequency to pass through and attenuates (reduces the amplitude of) signals with frequencies higher than the cutoff frequency.
- **Applications:** Removing high-frequency noise from a signal, smoothing signals, and anti-aliasing in digital audio.
- **Types:**
  - **Butterworth Filter:** Maximally flat frequency response in the passband.
  - **Chebyshev Filter:** Has a ripple in the passband but a sharper cutoff.
  - **Bessel Filter:** Linear phase response, maintaining the shape of filtered signals.

- **Finite Impulse Response (FIR) Filters:** Non-recursive and always stable with a linear phase response.
- **Infinite Impulse Response (IIR) Filters:** Recursive and can achieve sharper cutoffs with fewer coefficients but may have phase distortion.

Low Pass Filter Example

Introduction

In this example, we design and apply a low pass filter to a noisy signal. The original signal consists of a 5 Hz sine wave with added 50 Hz noise. We use a Butterworth filter with a cutoff frequency of 10 Hz to remove the high-frequency noise.

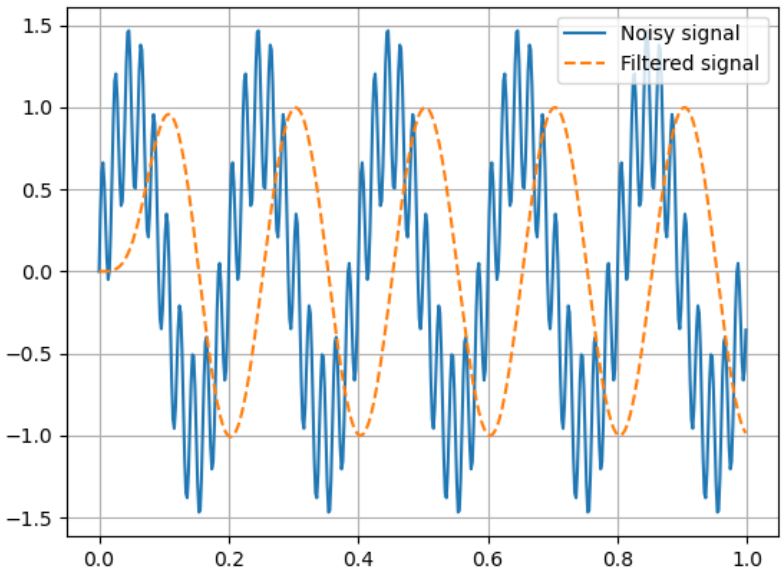


Figure 3: Comparison of Noisy Signal and Filtered Signal

Explanation

- **Signal Generation**: The code generates a sample signal consisting of a 5 Hz sine wave with added 50 Hz noise.

- **Filter Design**: A Butterworth low pass filter with a cutoff frequency of 10 Hz is designed using the `butter` function from the `scipy.signal` module. The filter is of order 5.
- **Filtering the Signal**: The filter is applied to the noisy signal using the `lfilter` function, which outputs the filtered signal.
- **Plotting**: The original noisy signal and the filtered signal are plotted for comparison. The noisy signal is plotted with a solid line, while the filtered signal is plotted with a dashed line.

Graphical Output

The plot shows two signals:

- **Noisy Signal**: This signal (solid line) contains the original 5 Hz sine wave with additional 50 Hz noise.
- **Filtered Signal**: This signal (dashed line) is the result of applying the low pass filter to the noisy signal. The high-frequency noise is significantly reduced, leaving a cleaner 5 Hz sine wave.

3 High Pass Filters

- **Definition:** A high pass filter allows signals with a frequency higher than a certain cutoff frequency to pass through and attenuates signals with frequencies lower than the cutoff frequency.
- **Applications:** Removing low-frequency noise such as DC offsets, rumble noise in audio recordings, and in image processing for edge detection.
- **Types:**
  - **Butterworth High Pass:** Smooth response with no ripple.
  - **Chebyshev High Pass:** Allows a sharper cutoff at the expense of ripple in the passband.
  - **FIR High Pass:** Non-recursive and stable with linear phase.
  - **IIR High Pass:** Recursive with sharper cutoffs but possible phase distortion.

High-Pass Filter Example

The high-pass filter allows frequencies above the cutoff frequency to pass through while attenuating frequencies below the cutoff. In the following plot, the original signal contains a low-frequency 5 Hz component and a high-frequency 50 Hz component. After applying the high-pass filter with a cutoff frequency of 30 Hz, the low-frequency component is attenuated, and the high-frequency component is preserved.

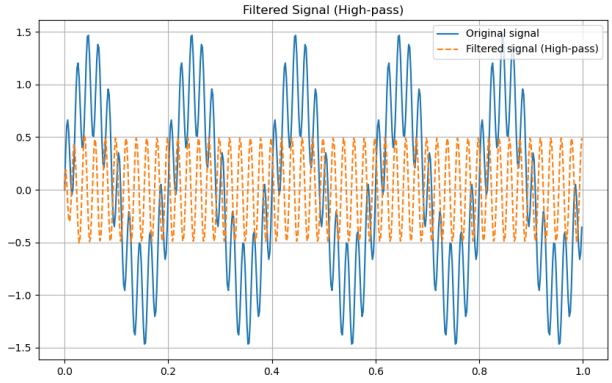


Figure 4: High-Pass Filter Result: The original signal (solid line) and the filtered signal (dashed line).

4 Band Pass Filters

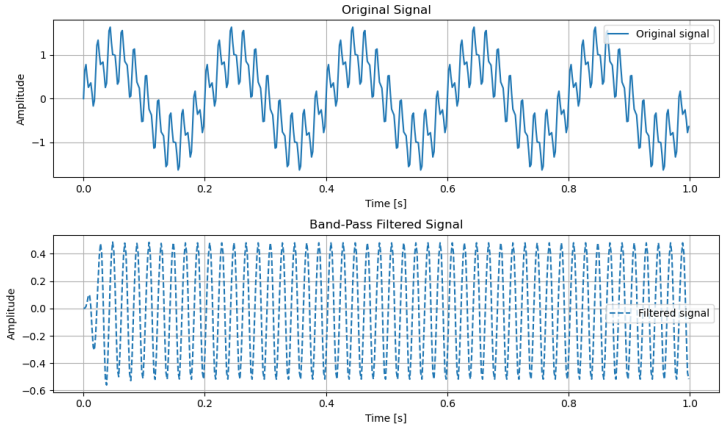


Figure 5: Band-Pass Filter Result: The original signal (solid line) and the filtered signal (dashed line).

- **Definition:** Allows frequencies within a certain range (band) to pass and attenuates frequencies outside this range.
- **Applications:** Communication systems to isolate specific frequency bands, audio equalization, and biomedical signal processing.

Band-Pass Filter Example

The band-pass filter allows frequencies within a specified range to pass through while attenuating frequencies outside this range. In the above plot, the original signal contains components at 5 Hz, 50 Hz, and 100 Hz. The band-pass filter with cutoff frequencies of 20 Hz and 80 Hz allows the 50 Hz component to pass while attenuating the 5 Hz and 100 Hz components.

5 Band Stop (Notch) Filters

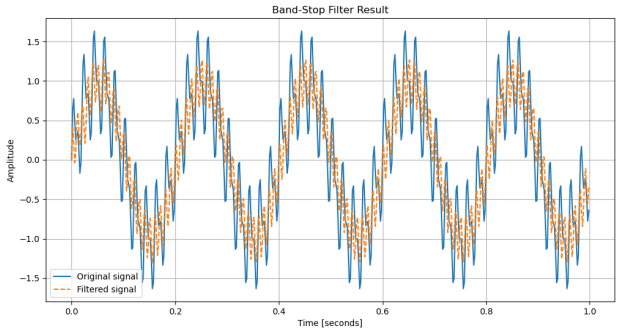


Figure 6: Band-Stop Filter Result: The original signal (blue) and the filtered signal (orange) are shown. The 50 Hz component is significantly attenuated while the 5 Hz and 100 Hz components remain largely unaffected.

- **Definition:** Attenuates frequencies within a certain range and allows frequencies outside this range to pass.
- **Applications:** Removing specific frequency components such as power line noise (e.g., 50 Hz or 60 Hz).

In the figure, the original signal and the filtered signal are compared. The band-stop filter effectively attenuates the frequencies within the range of 40 Hz to 60 Hz, demonstrating its ability to reduce unwanted frequency components from the signal.

6 Sampling and Nyquist Theorem

- **Sampling:** Converting a continuous signal into a discrete signal by taking samples at regular intervals.
- **Nyquist Theorem:** To accurately reconstruct a signal, it must be sampled at least twice its highest frequency component (Nyquist rate).

Example of Sampling

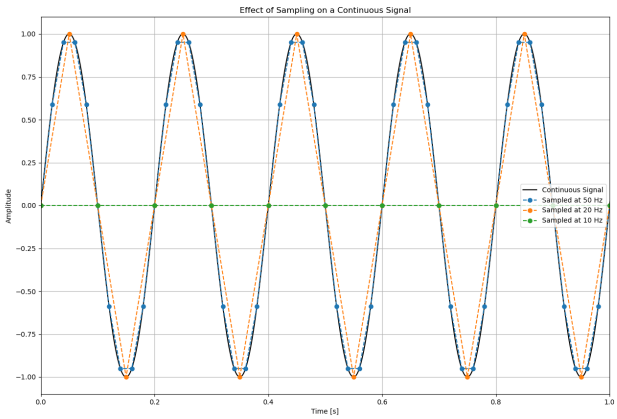


Figure 7: Continuous and Sampled Signals

1. **Continuous Signal**: The black solid line in Figure 7 represents the continuous sine wave with a frequency of 5 Hz sampled at 1000 Hz. This high sampling rate captures the signal's true shape accurately.

2. **Sampled Signal at 50 Hz**: The blue dashed line with circular markers in Figure 7 shows the sine wave sampled at 50 Hz. This rate is sufficient to accurately capture the sine wave, as it is well above the Nyquist rate (10 Hz for a 5 Hz signal).

3. **Sampled Signal at 20 Hz**: The green dashed line with circular markers in Figure 7 illustrates the sine wave sampled at 20 Hz. This is still above the Nyquist rate, but the representation starts to show slight deviations from the original continuous signal.

4. **Sampled Signal at 10 Hz**: The red dashed line with circular markers in Figure 7 represents the sine wave sampled at 10 Hz, which is exactly at the Nyquist rate. This results in a less smooth representation, but it still captures the general shape of the sine wave.

Overall, the plot demonstrates how sampling at rates above the Nyquist rate preserves the original signal, while sampling at the Nyquist rate results in a rougher approximation.

7 Fast Fourier Transform (FFT)

The Fast Fourier Transform (FFT) is a crucial algorithm in digital signal processing (DSP) that efficiently computes the Discrete Fourier Transform (DFT)

of a sequence. The DFT transforms a time-domain signal into its frequency-domain representation, revealing the different frequency components present in the signal.

The FFT reduces the computational complexity of the DFT from $O(N^2)$ to $O(N \log N)$, making it feasible to analyze large datasets. This efficiency is particularly important in applications like spectral analysis, filtering, image processing, communication systems, and more.

Example of FFT

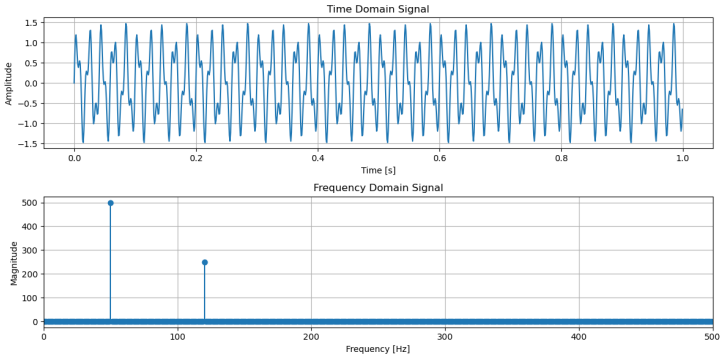


Figure 8: Time Domain and Frequency Domain Representation of the Signal

The graph produced by the code illustrates the significance of the FFT in analyzing a signal. It includes:

1. **Time Domain Signal**: The top subplot in Figure 8 shows the original time-domain signal, which is a combination of two sine waves with frequencies of 50 Hz and 120 Hz. This representation depicts how the signal varies over time.

2. **Frequency Domain Signal**: The bottom subplot in Figure 8 displays the magnitude of the FFT result. The x-axis represents the frequency in Hz, while the y-axis represents the magnitude of each frequency component. The peaks at 50 Hz and 120 Hz clearly indicate the presence of these frequencies in the original signal.

This graph demonstrates how the FFT can decompose a complex signal into its constituent frequencies, providing insights that are not immediately apparent in the time domain representation.

8 Windowing

In digital signal processing, windowing is a technique used to mitigate spectral leakage when performing a Fourier Transform on a signal. Spectral leakage

occurs because the Fourier Transform assumes the signal is periodic, which can cause discontinuities at the boundaries of the sampled signal.

A window function, such as the Hamming window, is applied to the signal to taper its edges, reducing these discontinuities. This process helps in obtaining a more accurate frequency spectrum.

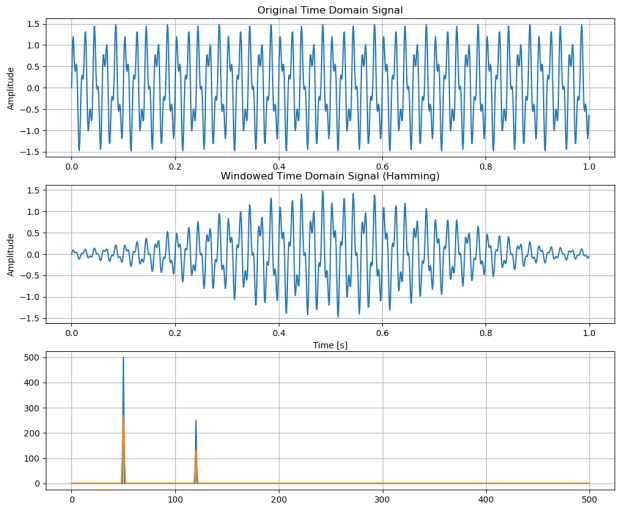


Figure 9: Time Domain and Frequency Domain Representation of the Original and Windowed Signals

Description of the Code Results

The graphs produced by the code illustrate the effect of windowing on a signal. Figure 9 includes:

1. **Original Time Domain Signal**: The top subplot shows the original time-domain signal, which is a combination of two sine waves with frequencies of 50 Hz and 120 Hz.

2. **Windowed Time Domain Signal**: The middle subplot shows the signal after applying a Hamming window. The windowed signal has tapered edges, reducing discontinuities.

3. **Frequency Domain Signal**: The bottom subplot shows the frequency domain representation of both the original and windowed signals. The application of the window reduces spectral leakage, resulting in a clearer frequency spectrum.

9 Convolution and Correlation

In digital signal processing, convolution and correlation are fundamental operations used for signal analysis, filtering, and system characterization.

Convolution is an operation that expresses how the shape of one signal is modified by another signal. It is widely used in filtering, where one signal (the filter) modifies another signal to enhance or suppress certain features.

Correlation, on the other hand, measures the similarity between two signals as a function of the displacement of one relative to the other. It is often used in signal detection and pattern recognition.

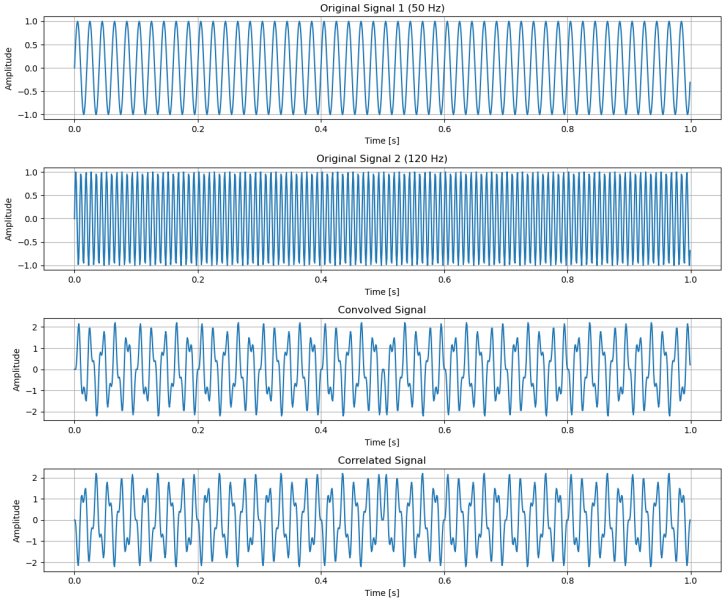


Figure 10: Time Domain Representation of Original, Convolved, and Correlated Signals

Description of the Code Results

The graphs produced by the code illustrate the effects of convolution and correlation on two signals. Figure 10 includes:

1. **Original Signal 1 (50 Hz)**: The first subplot shows the original time-domain signal, which is a 50 Hz sine wave.

2. **Original Signal 2 (120 Hz)**: The second subplot shows the original time-domain signal, which is a 120 Hz sine wave.

3. **Convolved Signal**: The third subplot shows the result of convolving the two signals. This plot demonstrates how one signal modifies the shape of the other.

4. **Correlated Signal**: The fourth subplot shows the result of correlating the two signals. This plot indicates the similarity between the two signals as a function of time shift.

10 Quantization

Quantization is the process of converting a continuous range of values into a finite set of discrete values. This is a critical step in the analog-to-digital conversion process, where a continuous signal is sampled and then mapped to a limited number of levels. Quantization introduces some error, known as quantization noise, which is the difference between the actual analog value and the quantized digital value.

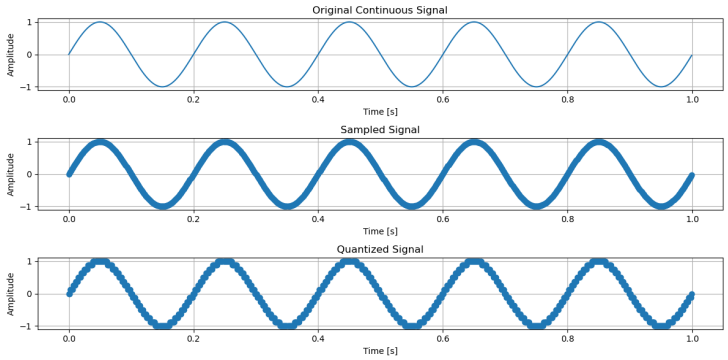


Figure 11: Time Domain Representation of Original, Sampled, and Quantized Signals

Description of the Code Results

The Python code generates a continuous-time signal, samples it, and quantizes the sampled signal into discrete levels. The results are displayed in a series of plots:

1. **Original Continuous Signal**: The continuous-time signal before any processing. 2. **Sampled Signal**: The signal measured at discrete time intervals. 3. **Quantized Signal**: The sampled signal rounded to the nearest discrete level.

The graph demonstrates the process of quantization, showing how the continuous signal is approximated by discrete levels and illustrating the introduction of quantization noise.