Digital Signal Processing Lecture Notes - 7

E Yazici - SRH Uni Heidelberg

16.6.2023

Linear filters

In signal processing, filters play an integral role in extracting useful information from signals. A filter is a system or device that allows specific components of a signal to pass through while suppressing others. In this context, linear filters and adaptive processing form a key part of many systems, ranging from audio processing to communications and control systems.

Convolution and Filtering

In the context of signal processing, convolution is a mathematical operation that is fundamental to many common signal processing operators. In simple terms, convolution provides a way of 'mixing' two signals, to form a third signal. It is used, among other things, to apply a filter to an input signal.

A filter is a system or device that allows specific components of a signal to pass through while suppressing others. In this context, linear filters and adaptive processing form a key part of many systems, ranging from audio processing to communications and control systems.

The term "filter" in signal processing refers to a mathematical function that modifies an input signal to achieve a specific output. This could be as simple as reducing certain frequencies of the signal, amplifying certain frequencies, or even more complex operations such as introducing an echo or reverberation effect. A digital filter is simply a filter that operates on digital signals, such as those processed by a computer.

Filtering a signal is done by convolving the signal with a function known as the impulse response, usually denoted $h[n]$. This function characterizes the filter's behavior: given an impulse as an input, the impulse response is the output of the filter. Therefore, it describes the reaction of the system to an impulse.

In the context of convolution, the input signal is usually denoted $x[n]$ and the output signal (the filtered signal) is denoted $y[n]$. In the time domain, the convolution of the input signal $x[n]$ with the impulse response $h[n]$ is computed as:

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

Here, the asterisk denotes convolution. This equation expresses that at each time instant n, the output $y[n]$ is a weighted sum of the input signal's values at every time instant m, where the weights are the values of the impulse response $h[n-m]$ at these instants.

In essence, convolution allows us to calculate the output of a linear time-invariant system given the input and the system's impulse response.

Problem 1: Convolution

Given an input signal $x[n] = \{1, 2, 3, 4, 5\}$ and impulse response $h[n] = \{2, 1\}$, calculate the convolved output $y[n]$.

Solution

The output $y[n]$ can be calculated by using the formula:

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

This means for each value of n in the output, we calculate the sum of products of the input signal $x[m]$ and the impulse response $h[n-m]$ for all m.
Let's calculate the output $y[n]$ for the given $x[n]$ and $h[n]$.

First, extend $x[n]$ and $h[n]$ for all n. That gives $x[n] = 1, 2, 3, 4, 5, 0, 0, 0, ...$ and $h[n] = 2, 1, 0, 0, 0, ...$

Using the formula, the convolution $y[n]$ is calculated as follows:

$$\begin{aligned}
y[0] &= x[0]h[0] + x[-1]h[1] = 1 \cdot 2 + 0 \cdot 1 = 2, \\
y[1] &= x[1]h[0] + x[0]h[1] = 2 \cdot 2 + 1 \cdot 1 = 5, \\
y[2] &= x[2]h[0] + x[1]h[1] = 3 \cdot 2 + 2 \cdot 1 = 8, \\
y[3] &= x[3]h[0] + x[2]h[1] = 4 \cdot 2 + 3 \cdot 1 = 11, \\
y[4] &= x[4]h[0] + x[3]h[1] = 5 \cdot 2 + 4 \cdot 1 = 14, \\
y[5] &= x[5]h[0] + x[4]h[1] = 0 \cdot 2 + 5 \cdot 1 = 5, \\
y[6] &= x[6]h[0] + x[5]h[1] = 0 \cdot 2 + 0 \cdot 1 = 0,
\end{aligned}$$

So, the convolved output is $y[n] = 2, 5, 8, 11, 14, 5, 0$.

Low-Pass Filter: In the field of signal processing, a low-pass filter is a filter that allows signals with a frequency lower than a certain cutoff frequency to pass through and attenuates frequencies higher than the cutoff frequency. The amount of attenuation for each frequency depends on the filter design. A low-pass filter is the opposite of a high-pass filter, which allows signals with frequencies higher than the cutoff frequency.

Let's think about listening to music. When you adjust the bass control on your stereo, you're manipulating a type of low-pass filter. Turning up the bass allows more of the low-frequency (bass) sounds to pass through to the speakers, while reducing the high-frequency (treble) sounds. In signal processing terms, a low-pass filter does something similar: it "filters out" the high-frequency components of a signal and allows the low-frequency components to "pass through."

Cutoff Frequency: The cutoff frequency, often denoted by ω_c in the digital signal processing literature, is the frequency boundary in a system's frequency response at which energy flowing through the system begins to be reduced (attenuated or reflected) rather than passing through. In the context of a low-pass filter, frequencies below the cutoff frequency are passed relatively unattenuated while frequencies above the cutoff frequency are significantly attenuated. In this problem, the cutoff frequency is $\pi/4$.

Continuing with the music example, the cutoff frequency is like the point on your stereo's control dial where you've set the bass level. Frequencies below this point (i.e., the bass sounds) get through to the speakers relatively unchanged. Frequencies above this point (the treble sounds) are reduced, or attenuated. In the low-pass filter, the cutoff frequency is the "boundary" between the frequencies that are allowed to pass through and those that are not.

Impulse Response: This concept can be a bit more abstract, but let's try a physical example. Imagine dropping a stone into a calm pond. The ripples that spread out from the point where the stone entered the water could be thought of as the pond's "impulse response." It's the pattern of waves caused by a single, sudden "impulse." In signal processing, the impulse response is the output of a system (like a filter) when you input an impulse. In this case, the "impulse" is a signal that's zero everywhere except at one point.

The impulse response of a system, denoted by $h[n]$, is its output when the input is an impulse. An impulse is a signal that is zero everywhere except at $n = 0$, where it is equal to 1. The impulse response completely characterizes a linear time-invariant system. If we know the impulse response of a system, we can predict its output for any given input using the convolution operation.

By looking at the impulse response, you can tell how a system will react to different inputs. In the case of the low-pass filter, the impulse response helps us understand how different frequency components will be filtered out.

Problem 2: Filter Impulse Response

Given a simple moving average filter of length 3, calculate its impulse response $h[n]$.

Solution

A moving average filter of length N is a low-pass filter whose impulse response is a rectangular function of length N:

$$h[n] = \begin{cases} 1/N & \text{if } 0 \leq n < N, \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, for a moving average filter of length 3, the impulse response is:

$$h[n] = \begin{cases} 1/3 & \text{if } 0 \leq n < 3, \\ 0 & \text{otherwise,} \end{cases}$$

which gives $h[n] = \{1/3, 1/3, 1/3, 0, 0, \ldots\}$.

* * *

The term ***linear*** refers to a system's property where the output is directly proportional to the input. In other words, if the input signal is scaled or summed, the output will change accordingly without modifying the system's other properties. This is a desirable property as it simplifies analysis and implementation.

Adaptive processing, on the other hand, refers to systems that adjust their properties based on the input or some other parameter. This adaptability makes these systems powerful tools for dealing with dynamic or unpredictable environments.

In the following sections, we will delve into the details of digital filters, their design techniques, and how to analyze and process random signals using these filters. We will also introduce the concept of adaptive signal processing, discussing various algorithms and their applications.

Linear Filter Problems and Solutions

Problem 1

Consider a simple low-pass filter with a cutoff frequency $\omega_c = \pi/4$. The filter has an impulse response $h[n] = \frac{1}{\pi}\frac{\sin(\omega_c n)}{n}$.

This formula is derived from the sinc function, which is the ideal impulse response of a low-pass filter in the frequency domain. However, in real-world applications, the ideal low-pass filter is not physically realizable, and we often use approximations to it.

Problem 1a

Determine the response of this filter to the input signal $x[n] = \cos(\pi n/2)$.

Solution:

The output $y[n]$ of a linear filter is the convolution of the input signal $x[n]$ and the impulse response $h[n]$. In this case, it can be represented as:

$$y[n] = x[n] * h[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m] \tag{1}$$

Given that $x[n] = \cos(\pi n/2)$ and $h[n] = \frac{1}{\pi}\frac{\sin(\omega_c n)}{n}$, we can compute $y[n]$ by substituting these values into the convolution equation and solving it.

```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt

# Define the input signal x[n]
n = np.arange(-100, 101)
x = np.cos(np.pi * n / 2)

# Define the impulse response h[n]
omega_c = np.pi / 4
h = 1/np.pi * np.sinc(omega_c * n / np.pi)

# Compute the convolution y[n] = x[n] * h[n]
y = signal.convolve(x, h, mode='same')

# Plot the input, impulse response and output signals
plt.figure(figsize=(10, 6))
plt.subplot(3, 1, 1)
plt.stem(n, x, use_line_collection=True)
plt.title('Input signal x[n] = cos(\pi n / 2)')
plt.subplot(3, 1, 2)
plt.stem(n, h, use_line_collection=True)
plt.title('Impulse response h[n]')
plt.subplot(3, 1, 3)
plt.stem(n, y, use_line_collection=True)
plt.title('Output signal y[n]')
plt.tight_layout()
plt.show()
```

This code creates an input signal x[n], an impulse response h[n], and then computes the convolution y[n] to produce the output signal. It then plots these three signals. The output signal y[n] represents the result of passing the input signal through the low-pass filter defined by h[n].

Problem 1b

Determine the frequency response $H(\omega)$ of this filter:

Solution:

The frequency response $H(\omega)$ of a linear, time-invariant filter is the Fourier transform of its impulse response $h[n]$. Hence, we have:

$$H(\omega) = \mathcal{F}\{h[n]\} = \sum_{n=-\infty}^{\infty} h[n]e^{-j\omega n} \tag{2}$$

We can calculate $H(\omega)$ by substituting the given $h[n]$ into this equation and solving it.

If you are familiar with the Fourier Transform properties and tables, you would recognize the impulse response $h[n]$ as a sinc function, which has a well-known Fourier Transform - a rectangular function. However, the Fourier Transform of a sinc function is usually taken over continuous time, while here we are dealing with a discrete-time signal.

In practice, to compute the frequency response of a filter given its impulse response, you would use the Discrete Fourier Transform (DFT) or the Fast Fourier Transform (FFT) algorithm, which is a computationally efficient way of implementing the DFT.

Here is a Python code that shows how to compute and plot the frequency response:

```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt

# Define the impulse response h[n]
n = np.arange(-100, 101)
omega_c = np.pi / 4
h = 1/np.pi * np.sinc(omega_c * n / np.pi)

# Compute the frequency response H(omega)
H = np.fft.fftshift(np.fft.fft(h))

# Define the frequency axis
omega = np.linspace(-np.pi, np.pi, len(H))

# Plot the impulse response and frequency response
plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
plt.stem(n, h, use_line_collection=True)
plt.title('Impulse response h[n]')
plt.subplot(2, 1, 2)
plt.plot(omega, np.abs(H))
plt.title('Frequency response |H(\omega)|')
plt.tight_layout()
plt.show()
```

Digital Filters

Definition and Function

A digital filter can be thought of as a filter in the time domain. For example, a coffee filter allows water to pass through, but it blocks the coffee grounds. In the same way, a digital filter is designed to allow certain parts of the signal (that you want) to pass through and block out the parts of the signal (that you don't want).

The signal that we use in the digital filter is discrete, which means it only exists at certain times (as opposed to a continuous signal which exists at all times). In other words, the signal is sampled, like taking a snapshot at regular intervals. The formula given:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k] \tag{3}$$

can be read as "the output at time n (y[n]) is the sum of the past M inputs

($x[n-k]$) scaled by the corresponding filter coefficients (b_k), minus the sum of the past N outputs (y[n-k]) scaled by their corresponding coefficients (a_k)."

The scaling and adding of samples is often referred to as a linear combination, which makes this type of filter a linear filter.

Let's break it down a bit further:

$y[n]$: This represents the output of the filter at time n.

$\sum_{k=0}^{M} b_k x[n-k]$: This term is the sum of the most recent M input values $x[n]$, each scaled by a corresponding coefficient b_k. This is the part of the filter's operation that depends on the input signal. The coefficients b_k determine how much each input value contributes to the output. This is sometimes referred to as the "feedforward" part of the filter.

$\sum_{k=1}^{N} a_k y[n-k]$: This term is the sum of the previous N output values $y[n]$, each scaled by a corresponding coefficient a_k. This is the part of the filter's operation that depends on its own previous outputs. The coefficients a_k determine how much each past output value feeds back into the output. This is sometimes referred to as the "feedback" part of the filter.

The values M and N determine the order of the filter. A high order filter has more terms in the sum, which means it uses more past input or output values to calculate the current output. Higher order filters can achieve more complex frequency responses, but they also require more computation.

In simpler terms, this mathematical expression is like a recipe for how to calculate the filter's output based on its inputs and its past outputs. Each input and past output is multiplied by a specific weight (the coefficients b_k and a_k), and then these weighted values are all added together to produce the output.

Filter Types: FIR and IIR

Filters are categorized into two types: Finite Impulse Response (FIR) filters and Infinite Impulse Response (IIR) filters.

FIR Filters

FIR filters have "Finite Impulse Response" because if you were to input an impulse (a signal that is 0 everywhere except at one point), the filter would output a signal that eventually becomes zero and stays zero. You could think of it like dropping a stone into a calm pond; the ripples (the response) would eventually die out.

Mathematically, an FIR filter is characterized by the equation:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] \tag{4}$$

This means that the output y[n] at any time n is a weighted sum of the current and past input values $x[n-k]$. Here, b_k are the weights, and M is the order of the filter, which refers to the number of previous inputs the filter uses.

Let's illustrate with an example:

Consider a simple moving average filter (which is a type of FIR filter) of order 3. It averages the current and two preceding values of the input signal. Its operation can be defined by:

$$y[n] = \frac{1}{3}x[n] + \frac{1}{3}x[n-1] + \frac{1}{3}x[n-2] \tag{5}$$

In this case, all of the coefficients b_k are equal to $\frac{1}{3}$, and $M = 2$ (the filter order is typically defined as the maximum of M and N). This filter does not use any previous output values in its calculation, so $N = 0$.

As an example, suppose we have a sequence of input values $x = [3, 4, 5, 6, 7, 8]$. The output of the filter for the first few values would be:

$$\begin{aligned}
y[2] &= \frac{1}{3}x[2] + \frac{1}{3}x[1] + \frac{1}{3}x[0] = \frac{1}{3} * 5 + \frac{1}{3} * 4 + \frac{1}{3} * 3 = 4 \\
y[3] &= \frac{1}{3}x[3] + \frac{1}{3}x[2] + \frac{1}{3}x[1] = \frac{1}{3} * 6 + \frac{1}{3} * 5 + \frac{1}{3} * 4 = 5 \\
y[4] &= \frac{1}{3}x[4] + \frac{1}{3}x[3] + \frac{1}{3}x[2] = \frac{1}{3} * 7 + \frac{1}{3} * 6 + \frac{1}{3} * 5 = 6
\end{aligned}$$

As you can see, each output value is simply the average of the current and two preceding input values.

As a Python code:

```
import numpy as np

# input sequence
x = np.array([3, 4, 5, 6, 7, 8])
# filter coefficients
b = np.array([1/3, 1/3, 1/3])
# output sequence
y = np.convolve(x, b, mode='valid')  # [4. 5. 6. 7.]
```

Note: In this Python code, we use the 'valid' mode in the convolution operation which means the convolution product is only given for points where the sequences overlap completely.

FIR filters have the property of being always stable, which means their output will not blow up, given bounded input. However, they might require more computational power compared to IIR filters of similar performance.

IIR Filters

The name "Infinite Impulse Response" comes from the idea that if you were to input an impulse, the filter would output a signal that theoretically continues forever. It's like ringing a bell; the sound (the response) takes a long time to fade away, and theoretically, it never completely disappears.

IIR filters use past output samples as well as past input samples to compute the current output:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k] \tag{6}$$

This means the output $y[n]$ at any time n is a weighted sum of the current and past input values x[n-k], minus a weighted sum of the past output values y[n-k]. The weights b_k and a_k are the filter coefficients, and M and N are the orders of the filter.

IIR filters can achieve a given filtering characteristic using fewer coefficients and calculations than a similar FIR filter, but they may become unstable, causing the output to blow up.

FIR Filter Example

A very simple example of an FIR filter is a moving average filter, where each output sample is the average of the last M input samples. This is used for smoothing out fluctuations in the data.

For example, consider a 3-point moving average filter, which is an FIR filter with M=3 and the coefficients $b_k = 1/3$. For this filter, the equation becomes:

$$y[n] = \frac{1}{3}x[n] + \frac{1}{3}x[n-1] + \frac{1}{3}x[n-2] \tag{7}$$

This means each output sample is the average of the current and previous two samples.

IIR Filter Example

A simple example of an IIR filter is the exponential averaging filter. In this filter, each output sample is a weighted combination of the current input sample and the previous output sample.

For instance, consider an exponential averaging filter with the weighting factor 0.5. This can be expressed as an IIR filter with M=1, N=1, $b_0 = 0.5$, and $a_1 = 0.5$. For this filter, the equation becomes:

$$y[n] = 0.5x[n] + 0.5y[n-1] \tag{8}$$

This means each output sample is the average of the current input sample and the previous output sample. This type of IIR filter can be used to implement a "leaky integrator" or a simple low-pass filter.

Impulse Response and Frequency Response

The impulse response of a digital filter is a sequence that shows how the filter responds to an impulse, which is a signal that is zero everywhere except at one point where it is 1. If we feed an impulse into a filter, the output we get is the impulse response. This response is a complete characterization of the filter – if you know the impulse response, you can predict how the filter will react to any signal.

The frequency response of a digital filter provides information about how the filter will react to signals of different frequencies. It is essentially a plot that shows how the filter attenuates or amplifies each frequency. If the filter is designed to eliminate signals of a certain frequency, the frequency response will be low (ideally zero) at that frequency. If the filter is designed to allow signals of a certain frequency to pass through without attenuation, the frequency response will be high (ideally one) at that frequency.

Convolution and Transfer Function

Convolution is a mathematical operation that's very important in digital signal processing. It is used to compute the output of a filter given the filter's input and impulse response. The convolution operation takes two sequences of numbers and combines them into a third sequence of numbers.

The transfer function is a mathematical description of the relationship between the input and output of a filter, given in the frequency domain. In other words, it describes how the filter responds to different frequencies of input. The transfer function is calculated by taking the Z-transform of the impulse response.

Stability and Causality

Stability in the context of digital filters relates to whether the output of the filter will remain bounded (i.e., not become infinitely large) for a bounded input signal. This is an important property for practical filters, because an unstable filter can rapidly produce very large output values in response to relatively small input signals, which is generally undesirable.

Causality refers to a system where the output at any given time depends only on the current and past input samples, but not on future inputs. In practical terms, a causal filter is one that doesn't anticipate future input values - it only responds to the current and past values. This is a necessary condition for real-time filtering applications, because in real time, future input values are unknown.

Example: Impulse Response

Let's consider a simple FIR filter with coefficients $h[n] = 0.2, 0.5, 0.3$. This means that the output at each time n is 0.2 times the current input, 0.5 times the previous input, and 0.3 times the input two time steps ago. Therefore, if we input an impulse signal $x[n] = 1, 0, 0, ...$, the output will be $y[n] = 0.2, 0.5, 0.3, 0, 0, ...$ which is exactly the sequence of coefficients $h[n]$. Hence, the impulse response of this filter is $h[n] = 0.2, 0.5, 0.3$.

Example: Frequency Response

Consider an FIR filter with impulse response $h[n] = 1, -1$. The frequency response of this filter is given by the discrete Fourier transform of the impulse response, which is $H(\omega) = \sum_{n=0}^{N-1} h[n]e^{-j\omega n} = 1 - e^{-j\omega}$. The magnitude of the frequency response can be obtained by taking the absolute value of $H(\omega)$, which is $|H(\omega)| = \sqrt{(1 - \cos(\omega))^2 + (\sin(\omega))^2} = \sqrt{2 - 2\cos(\omega)}$. The plot of this function will show how the filter amplifies or attenuates the different frequency components.

Example: Convolution

Suppose we have an input sequence $x[n] = 1, 2, 3$ and a filter with impulse response $h[n] = 2, 1$. The output of the filter can be computed by convolving the input sequence with the impulse response, which yields $y[n] = x[n] * h[n] = 2, 5, 8, 3$.

Example: Transfer Function

Consider an FIR filter with impulse response $h[n] = 1, -1$. The transfer function is given by the z-transform of the impulse response, which is $H(z) = \sum_{n=0}^{N-1} h[n]z^{-n} = 1 - z^{-1}$.

Example: Stability and Causality

An IIR filter with impulse response $h[n] = 1, 0.5, 0.25, 0.125, ...$ is both stable and causal. It is stable because the impulse response is absolutely summable (the sum of the absolute values of the impulse response coefficients is finite). It is causal because the output at any time depends only on the current and past inputs. An FIR filter with any set of coefficients is always stable and causal.

Problems and Solutions

Problem 1

Given the following FIR filter equation:

$$y[n] = 0.5x[n] + 0.3x[n-1] - 0.8x[n-2] \tag{9}$$

Determine the output sequence $y[n]$ for the input $x[n] = [3, 4, 2, 1, 0]$.

Solution 1

To find the output sequence $y[n]$ for the given input $x[n]$, we need to substitute the values of $x[n]$, $x[n-1]$, and $x[n-2]$ into the filter equation for each n.

Firstly, we should extend $x[n]$ for all n as follows: $x[n] = 3, 4, 2, 1, 0, 0, 0, ...$ Then we compute $y[n]$:

$$\begin{aligned}
y[0] &= 0.5x[0] + 0.3x[-1] - 0.8x[-2] = 0.5 \cdot 3 = 1.5 \\
y[1] &= 0.5x[1] + 0.3x[0] - 0.8x[-1] = 0.5 \cdot 4 + 0.3 \cdot 3 = 2.9 \\
y[2] &= 0.5x[2] + 0.3x[1] - 0.8x[0] = 0.5 \cdot 2 + 0.3 \cdot 4 - 0.8 \cdot 3 = 0.2 \\
y[3] &= 0.5x[3] + 0.3x[2] - 0.8x[1] = 0.5 \cdot 1 + 0.3 \cdot 2 - 0.8 \cdot 4 = -2.1 \\
y[4] &= 0.5x[4] + 0.3x[3] - 0.8x[2] = 0.3 \cdot 1 - 0.8 \cdot 2 = -1.3
\end{aligned}$$

So, the output sequence $y[n]$ is $[1.5, 2.9, 0.2, -2.1, -1.3]$.

Problem 2

A causal, linear, and time-invariant system has the impulse response $h[n] = (0.5)^n u[n]$, where $u[n]$ is the unit step function. Determine if the system is stable.

Solution 2

The system is stable if it is BIBO (Bounded Input, Bounded Output) stable, which means that for every bounded input, the output is also bounded. This is equivalent to the condition that the impulse response be absolutely summable:

$$\sum_{n=0}^{\infty} |h[n]| < \infty \tag{10}$$

For the given impulse response, this becomes:

$$\sum_{n=0}^{\infty} |(0.5)^n| = \sum_{n=0}^{\infty} (0.5)^n = \frac{1}{1-0.5} = 2 < \infty \tag{11}$$

So, the system is stable.

More Examples

Example 1: FIR Filter Response

Let's consider a simple FIR filter with impulse response $h[n] = 0.5, 0.5$. The input to the filter is a sequence $x[n] = 1, 2, 3, 4$.

Problem: Calculate the output $y[n]$ of the filter.

Solution:
The output of a FIR filter can be calculated by convolving the input sequence $x[n]$ with the impulse response $h[n]$. Here, we have $x[n] = 1, 2, 3, 4$ and $h[n] = 0.5, 0.5$.

In terms of formulas, for discrete sequences as we have here, convolution is defined as a finite sum:

$$(y * h)[n] = \sum_{k=0}^{N} y[k] \cdot h[n-k] \tag{12}$$

where N is the length of the sequences, which in this case is 4.

The convolution can be calculated as follows:

For $n = 0$, we get $y[0] = x[0]h[0] + x[-1]h[1] = 1 \times 0.5 + 0 \times 0.5 = 0.5$.
For $n = 1$, we get $y[1] = x[1]h[0] + x[0]h[1] = 2 \times 0.5 + 1 \times 0.5 = 1.5$.
For $n = 2$, we get $y[2] = x[2]h[0] + x[1]h[1] = 3 \times 0.5 + 2 \times 0.5 = 2.5$.
For $n = 3$, we get $y[3] = x[3]h[0] + x[2]h[1] = 4 \times 0.5 + 3 \times 0.5 = 3.5$.

After $n = 3$, the input sequence $x[n]$ is zero. So, $y[4] = x[4]h[0] + x[3]h[1] = 0 \times 0.5 + 4 \times 0.5 = 2$.

Therefore, the output sequence is $y[n] = 0.5, 1.5, 2.5, 3.5, 2$.

Example 2: Frequency Response of a System

Consider a system with the impulse response $h[n] = 1, 1, 1$.

Problem: Compute the frequency response of the system.

Solution:
The frequency response $H(\omega)$ of a system is the Fourier transform of its impulse response $h[n]$. Hence, we can calculate $H(\omega)$ using the definition of the discrete-time Fourier transform (DTFT):

$$H(\omega) = \sum_{n=0}^{2} h[n]e^{-j\omega n} = 1 + e^{-j\omega} + e^{-2j\omega} \tag{13}$$

So, the frequency response of the system is $H(\omega) = 1 + e^{-j\omega} + e^{-2j\omega}$. This function tells us how the system will amplify or attenuate signals of different frequencies. For instance, when $\omega = 0$, all frequency components are passed through without attenuation ($|H(0)| = 3$). But when $\omega = \pm\frac{2\pi}{3}$, the system will completely remove those frequency components ($|H(\pm\frac{2\pi}{3})| = 0$).

Example 3: Stability of an IIR Filter

Consider an IIR filter defined by the difference equation: $y[n] = x[n]+0.5y[n-1]$.

Problem: Determine if the filter is stable.

Solution:
The filter is an IIR filter because the output depends on the previous outputs. The stability of an IIR filter can be determined by looking at the roots of its characteristic equation. If all the roots lie inside the unit circle in the complex plane, then the filter is stable.

What is a characteristic Equation?

The characteristic equation of a filter is obtained by setting the input of the filter to zero and solving the resulting homogeneous difference equation. For a filter defined by a difference equation, the characteristic equation can tell us important properties about the filter, including its stability.

In the context of filters, stability is an important concept. A filter is said to be "stable" if, given a bounded input (i.e., an input that doesn't go to infinity), the output is also bounded (it doesn't go to infinity either).

For the given IIR filter, the difference equation is:

$$y[n] = x[n] + 0.5y[n-1] \tag{14}$$

Setting the input to zero, the equation becomes:

$$y[n] = 0.5y[n-1] \tag{15}$$

Assuming a solution of the form $y[n] = Az^n$, where z is a complex number and A is the amplitude, we can substitute this into the homogeneous equation:

$$Az^n = 0.5Az^{n-1} \tag{16}$$

Solving this equation for z, we get:

$$z = 0.5 \tag{17}$$

For a discrete-time system such as this one, stability is ensured if all the solutions of the characteristic equation lie inside the unit circle in the complex plane, that is, their absolute value is less than 1.

Since $z = 0.5$ and $|z| < 1$, we can conclude that the filter is stable.

- - -

The concept of BIBO (Bounded Input, Bounded Output) stability is an important one in the study of systems and signals. A system is said to be BIBO stable if every bounded input leads to a bounded output.

In simpler terms, this means that if we input a signal into the system that doesn't go to infinity (it is "bounded"), then the system will not produce an output that goes to infinity either - the output will also be "bounded".

So, when we say that a filter is BIBO stable, we mean that for any input sequence that is bounded, the output sequence will also be bounded, regardless of the length of the input sequence.

In the case of the IIR filter defined by the difference equation $y[n] = x[n] + 0.5y[n-1]$, the characteristic equation is indeed $1 - 0.5z^{-1} = 0$, and this has a root at $z = 0.5$.

In the z-domain, the unit circle corresponds to the set of complex numbers with an absolute value (magnitude) of 1. Any roots of the characteristic equation that lie inside the unit circle (have a magnitude less than 1) correspond to components of the system's response that decay over time.

Since the magnitude of 0.5 is less than 1 and thus lies inside the unit circle, this means that the corresponding component of the system's response will decay over time, and hence the system is BIBO stable.

Example 4: FIR Filter Response

In this example, we will implement the FIR filter and calculate the output $y[n]$.

```
import numpy as np

# define the input and impulse response
x = np.array([1, 2, 3, 4])
h = np.array([0.5, 0.5])

# compute the output y[n]
y = np.convolve(x, h)

print("The output sequence is:", y)
```

Example 5: Frequency Response of a System

In this example, we will compute and plot the frequency response of a system using the scipy library.

This script will compute and plot the magnitude of the frequency response of the system:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft

# define the impulse response
h = np.array([1, 1, 1])

# compute the frequency response
H = fft(h)

# plot the frequency response
freq = np.linspace(-np.pi, np.pi, len(H))
plt.plot(freq, np.abs(H))
plt.title('Frequency Response')
plt.xlabel('Frequency (rad/sample)')
plt.ylabel('Magnitude')
plt.grid(True)
plt.show()
```

Example 6: Stability of an IIR Filter

In this example, we will determine the stability of an IIR filter by looking at the roots of its characteristic equation.

```
import numpy as np

# define the coefficients of the characteristic equation
coeff = np.array([1, -0.5])

# compute the roots of the characteristic equation
roots = np.roots(coeff)

# check if the filter is stable
stable = np.all(np.abs(roots) < 1)

print("The filter is stable:", stable)
```

Filter Design

Designing a digital filter involves a series of steps that aim to meet a given set of specifications. These might include the desired gain in the passband, the allowable ripple in the passband, the required attenuation in the stopband, and the transition band's width.

Filter Specifications

The typical specifications for a filter are:

- Passband edge frequency, f_p
- Stopband edge frequency, f_s
- Maximum passband ripple, A_p
- Minimum stopband attenuation, A_s

FIR Filter Design Methods

There are several standard methods to design FIR filters. The most common ones include:

Window Method

This method involves creating an ideal impulse response and then multiplying it by a window function. The type of window function used affects the filter's characteristics. For example, a rectangular window yields a filter with poor stopband attenuation, while a Hamming window provides better stopband attenuation but a wider main lobe.

The main steps in the window method are:

1. Determine the ideal lowpass filter impulse response, $h_d[n]$.
2. Choose a window function, $w[n]$.
3. The designed filter response, $h[n] = h_d[n]w[n]$.

Frequency Sampling Method

This method involves specifying the desired frequency response, $H_d(\omega)$, at a set of evenly spaced frequency points and then computing the impulse response using the inverse discrete Fourier transform.

Optimal Methods

These methods aim to minimize the error between the designed filter and the ideal one according to some criterion. The most common optimal method is the Parks-McClellan method, which uses the Remez exchange algorithm and Chebyshev approximation theory.

IIR Filter Design Methods

Designing IIR filters usually involves designing an analog filter and then transforming it to the digital domain. The most popular methods are based on analog prototype filters:

Butterworth Filters

Butterworth filters have a maximally flat frequency response in the passband. They provide a good compromise between passband flatness and stopband attenuation.

Chebyshev Filters

There are two types of Chebyshev filters, Type I and Type II. Type I filters have an equiripple passband and a monotonically decreasing stopband. Type II filters have a monotonically decreasing passband and an equiripple stopband.

Elliptic Filters

Elliptic filters (or Cauer filters) have an equiripple behavior in both the passband and the stopband. They provide the steepest transition band for a given filter order.

Analysis of Filter Properties

After a filter has been designed, it is important to verify its performance. The characteristics of a filter can be analyzed using its impulse response, frequency response, phase response, and pole-zero plot.

Practical Considerations in Filter Design

The design of practical digital filters also involves considerations about the implementation of the filter. These include the computational complexity of the filter, the effects of coefficient quantization, the stability of the filter, and the trade-off between filter order and performance.

Example: FIR Filter Design using Window Method

Suppose we want to design a low-pass FIR filter with a cut-off frequency of 0.3π rad/sample using a Hamming window. The ideal impulse response of this filter is given by:

$$h_d[n] = \frac{\sin(0.3\pi(n - M/2))}{\pi(n - M/2)}, \quad 0 \leq n \leq M \tag{18}$$

where M is the length of the filter.

The Hamming window is given by:

$$w[n] = 0.54 - 0.46\cos\left(\frac{2\pi n}{M}\right), \quad 0 \leq n \leq M \tag{19}$$

The actual impulse response of the filter is the product of the ideal impulse response and the window function:

$$h[n] = h_d[n] \cdot w[n] \tag{20}$$

Here, we want to create a FIR (Finite Impulse Response) filter using a method known as the window method. In this method, we start with the ideal filter response (the one we want) and then we "window" it, meaning we reduce the influence of the points towards the edges of the response, which helps to reduce ripples in the actual filter response. This is done using a "window function". In this example, we're using the Hamming window.

The ideal impulse response $h_d[n]$ of a low-pass filter is a sinc function, where the main lobe is centered at the cutoff frequency of the filter.

The Hamming window $w[n]$ is a function that starts and ends at zero and has a peak of one in the middle.

By multiplying the ideal impulse response with the window function (i.e., taking their pointwise product), we get the actual impulse response $h[n]$ of the filter. This process is called windowing.

However, actually calculating the values of $h[n]$ for a given M (filter length) and cutoff frequency can get a bit tricky because it involves the sinc function and the cosine function.

Example: IIR Filter Design using Butterworth Method

Suppose we want to design a low-pass IIR filter with a cut-off frequency of 0.3π rad/sample and a filter order of 2. The analog prototype of a Butterworth filter of order 2 is given by:

$$H_a(s) = \frac{1}{(s^2 + \sqrt{2}s + 1)} \tag{21}$$

We can transform this analog filter to a digital filter using the bilinear transformation, $s = \frac{2}{T}(\frac{1-z^{-1}}{1+z^{-1}})$, where T is the sampling period. After some algebraic manipulation, we get the transfer function of the digital filter.

Designing an IIR (Infinite Impulse Response) filter is a bit more complex. Here, we're using the Butterworth method. Butterworth filters are known for their flat frequency response in the passband (i.e., the range of frequencies that are allowed to pass through the filter).

The design process starts with an analog prototype, which is a continuous-time filter. For a Butterworth filter of order 2, the analog prototype $H_a(s)$ is a rational function with a quadratic polynomial in the denominator.

The tricky part is converting this analog filter to a digital filter. This involves replacing the complex frequency s in the analog filter with a function of the digital frequency z. The bilinear transformation is a common method for doing this, and it ensures that the frequency response of the digital filter matches that of the analog filter.

After this transformation, we get the transfer function of the digital filter, which can be used to compute the filter coefficients.

Again, in practice, you'd usually use software tools to perform these calculations and design filters.

1

* * *

Note that the above examples are highly simplified. In practice, you would use a software tool such as MATLAB or Python's SciPy library to design and analyze digital filters, as doing it manually can be quite tedious and error-prone. The main aim of these examples is to illustrate the basic concepts and steps involved in filter design.

Example: Design of a High-Pass FIR Filter

Suppose we want to design a high-pass FIR filter with a cutoff frequency of 3000 Hz for a signal that is sampled at 8000 Hz. We can use the window method for this.

First, we'd design a low-pass filter with the same cutoff frequency. The impulse response of this filter, $h_{LP}[n]$, would be given by:

$$h_{LP}[n] = \frac{\sin\left(2\pi \frac{3000}{8000}(n - \frac{M}{2})\right)}{\pi(n - \frac{M}{2})} w[n], \tag{22}$$

where $w[n]$ is the Hamming window function and M is the filter length.

Then, we'd convert this to a high-pass filter by spectral inversion, which involves subtracting the low-pass impulse response from a delta impulse:

$$h_{HP}[n] = \delta[n - \frac{M}{2}] - h_{LP}[n]. \tag{23}$$

Python Code:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import firwin, freqz, hamming

# Filter specifications.
sample_rate = 8000
cutoff_freq = 3000
filter_length = 101  # Should be odd.

# Design the low-pass filter.
lowpass_coef = firwin(filter_length, cutoff_freq, window='hamming', pass_zero='lowpass', fs=

# Design the high-pass filter by spectral inversion.
highpass_coef = -lowpass_coef
highpass_coef[(filter_length - 1) // 2] += 1

# Plot the impulse responses.
time = np.arange(-(filter_length - 1) // 2, (filter_length - 1) // 2 + 1) / sample_rate
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(time, lowpass_coef)
plt.title('Impulse Response of Low-Pass Filter')
plt.xlabel('Time [s]')
plt.grid()
plt.subplot(2, 1, 2)
plt.plot(time, highpass_coef)
plt.title('Impulse Response of High-Pass Filter')
plt.xlabel('Time [s]')
plt.grid()
plt.tight_layout()
plt.show()

# Plot the frequency responses.
freq, response = freqz(lowpass_coef, fs=sample_rate)
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(freq, np.abs(response))
plt.title('Frequency Response of Low-Pass Filter')
plt.xlabel('Frequency [Hz]')
plt.grid()
freq, response = freqz(highpass_coef, fs=sample_rate)
plt.subplot(2, 1, 2)
plt.plot(freq, np.abs(response))
plt.title('Frequency Response of High-Pass Filter')
plt.xlabel('Frequency [Hz]')
plt.grid()
plt.tight_layout()
plt.show()
```

Design of a Band-Pass IIR Filter

Suppose we want to design a band-pass IIR filter with passband frequencies of 2000 Hz and 3000 Hz for a signal that is sampled at 8000 Hz. We can use the Butterworth method for this.

The design process would start with an analog prototype that passes the normalized frequency range of 1 to 1.4142 (the square root of 2). This prototype would have a transfer function of the form:

$$H_a(s) = \frac{1}{(s+1)(s+1)}, \tag{24}$$

which corresponds to a 2nd-order Butterworth filter.

Then, we'd perform a frequency transformation to shift and stretch the passband of the analog prototype to the desired frequencies. This would involve replacing s in the analog filter with a function of the form:

$$s = \frac{1}{\tan\left(\frac{\pi(B-W)}{2f_s}\right)}\left(\frac{1-z^{-1}}{1+z^{-1}}\right), \tag{25}$$

where B is the center frequency of the passband, W is the width of the passband, and f_s is the sample rate.

After this transformation, we'd get the transfer function of the digital filter. The filter coefficients can then be obtained from this transfer function.

Python Code:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import butter, lfilter, freqz

# Sample rate and desired cutoff frequencies (in Hz).
fs = 8000.0
lowcut = 2000.0
highcut = 3000.0

# Butterworth filter design
from scipy.signal import butter, sosfreqz
order = 6
sos = butter(order, [lowcut, highcut], btype='band', output='sos', fs=fs)

# Frequency response
w, h = sosfreqz(sos, worN=2000, fs=fs)

# Plot the frequency response.
plt.figure(figsize=(10, 6))
plt.plot(w, abs(h), 'b')
plt.plot(lowcut, 0.5*np.sqrt(2), 'ko')
plt.plot(highcut, 0.5*np.sqrt(2), 'ko')
plt.xlim(0, 0.5*fs)
plt.title("Bandpass Filter Frequency Response")
plt.xlabel("Frequency [Hz]")
plt.ylabel("Gain")
plt.grid(True)
plt.show()
```

Analyzing and Processing Random Signals and Designing Filters

Random signals, also known as stochastic signals, are signals that cannot be described accurately by an exact mathematical function or rule. These signals are unpredictable and do not follow a regular pattern. Examples of random signals include noise signals (such as the static you might hear when tuning a radio), signals from communication channels, signals in biology, and many more.

Random Signals

When dealing with random signals, we often have to rely on statistical properties to describe the signals. The mean, variance, and autocorrelation function are some of these properties:

Mean or Expected Value

The mean, or expected value, is the average value that the signal is expected to take on. In other words, if we were to observe the signal an infinite number of times and average all the values, the result would be the expected value.

For a discrete-time random signal $x[n]$, the mean or expected value $E\{x[n]\}$ is given by:

$$E\{x[n]\} = \sum_{-\infty}^{\infty} x[n]p(x[n]) \tag{26}$$

where $p(x[n])$ is the probability density function of $x[n]$.

Variance

The variance measures how spread out the signal values are around the mean. If the variance is small, it means the signal values are close to the mean, whereas a large variance means the signal values are spread out over a large range.

The variance $\text{var}\{x[n]\}$ is given by:

$$\text{var}\{x[n]\} = E\{(x[n] - E\{x[n]\})^2\} \tag{27}$$

Autocorrelation Function

The autocorrelation function measures the similarity between the signal's values at different points in time. For instance, if the autocorrelation function is high at a particular time difference, it means the signal is likely to be similar at those two times.

The autocorrelation function $R_{xx}[m]$ of a random signal $x[n]$ is defined as:

$$R_{xx}[m] = E\{x[n]x[n+m]\} \tag{28}$$

Processing Random Signals

Processing random signals often involves extracting useful information or enhancing certain features of the signal. This usually involves designing and applying filters to the random signals.

Designing Filters for Random Signals

When we design a filter for a random signal, we typically have a certain set of criteria that we want our filtered signal to satisfy. This could be a specific frequency response, time-domain response, or some other characteristic.

One common method to design filters for random signals is the Wiener filter. A Wiener filter tries to estimate a random signal $x[n]$ based on a noisy observation $y[n] = x[n] + v[n]$, where $v[n]$ is an additive noise signal.

The Wiener filter that minimizes the mean square error (the average of the squares of the differences) between the estimated signal and the actual signal is given by:

$$h_{opt}[n] = R_{vv}^{-1}[n] * R_{xv}[n] \tag{29}$$

where $R_{vv}[n]$ is the autocorrelation of the noise signal, $R_{xv}[n]$ is the cross-correlation between the signal and the noise, and $*$ denotes convolution. This formula shows how we can use the statistical properties of the signal and the noise to design a filter that minimizes the error.

For example, consider a case where we are receiving a signal that has been corrupted by additive white Gaussian noise (AWGN). If we know the statistical properties of the signal and the noise, we can design a Wiener filter that will minimize the mean square error between the estimated and actual signal. This will result in a cleaner, less noisy signal.

EXAMPLE:
Let's consider a simple random signal $x[n]$ which takes the values $-1, 0, 1$ with probabilities $0.2, 0.6, 0.2$ respectively.

1. Expected Value (Mean)

The expected value $E\{x[n]\}$ is given by the sum of each value multiplied by its probability:

$$E\{x[n]\} = -1 * 0.2 + 0 * 0.6 + 1 * 0.2 \tag{30}$$

$$= -0.2 + 0 + 0.2 \tag{31}$$

$$= 0 \tag{32}$$

So, the expected value of $x[n]$ is 0.

2. Variance

The variance is the expected value of the square of the deviation of $x[n]$ from its mean. Here, the mean is 0, so we get:

$$\text{var}\{x[n]\} = E\{(x[n] - 0)^2\} \tag{33}$$

$$= (-1 - 0)^2 * 0.2 + (0 - 0)^2 * 0.6 + (1 - 0)^2 * 0.2 \tag{34}$$

$$= 0.2 + 0 + 0.2 \tag{35}$$

$$= 0.4 \tag{36}$$

So, the variance of $x[n]$ is 0.4.

3. **Autocorrelation Function**

For a discrete random signal, the autocorrelation at lag m is the expected value of the product of $x[n]$ and $x[n + m]$. However, without additional information about the structure of our signal, we can't compute the autocorrelation.

Python Code:

```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt

# Generate a random signal
np.random.seed(0)  # for reproducibility
n_samples = 1000
x = np.random.normal(size=n_samples)

# Add noise to the signal
noise = np.random.normal(scale=0.5, size=n_samples)
y = x + noise

# Design a Butterworth filter
b, a = signal.butter(4, 0.2)  # 4th order Butterworth filter with cutoff frequency 0.2
y_filtered = signal.lfilter(b, a, y)

# Plot the original, noisy, and filtered signals
plt.figure(figsize=(12, 8))
plt.plot(x, label='Original signal')
plt.plot(y, label='Noisy signal')
plt.plot(y_filtered, label='Filtered signal')
plt.legend()
plt.show()
```

Problem Set - 7

1. Given a linear system with the impulse response $h[n] = (0.5)^n u[n]$, and an input signal $x[n] = \delta[n] - 2\delta[n-1]$. Compute the output signal $y[n]$.

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

2. Consider a FIR filter with the following coefficients: $b = [1, -1, 2, -2]$. If the input to the filter is $x[n] = [1, 2, 3, 4, 5]$, what is the output $y[n]$?

$$y[n] = \sum_{k=0}^{M} b_k x[n-k]$$

3. For an IIR filter with the coefficients $b = [1, 2]$ and $a = [1, -0.5]$, if the input to the filter is $x[n] = [1, 0, 0, 0, 0]$, what is the output $y[n]$?

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k]$$

4. Given a random signal $x[n]$ where $x[n] = -1$ with probability 0.2, $x[n] = 0$ with probability 0.6, and $x[n] = 1$ with probability 0.2. Compute the expected value and variance of $x[n]$.

$$E\{x[n]\} = \sum_{\forall n} x[n] \cdot p(x[n])$$

$$\text{var}\{x[n]\} = E\{(x[n] - E\{x[n]\})^2\}$$