5. Digital Filter Design and Analysis: Implementing FIR and IIR filters in Python.
6. Adaptive Filtering: Applying adaptive filtering algorithms to noise reduction.

Contents

1 Introduction

This manual provides an overview of digital filter design and implementation methods, including Finite Impulse Response (FIR), Infinite Impulse Response (IIR), and Adaptive Filtering techniques for noise reduction. Mathematical foundations and complete Python implementations are provided.

Digital filtering plays a crucial role in signal processing to remove unwanted noise, enhance desired signals, and extract information. This manual explores:

- **FIR Filters**: Finite Impulse Response filters.
- **IIR Filters**: Infinite Impulse Response filters.
- **Adaptive Filters**: Dynamically adjusting filters to reduce noise based on learning algorithms.

2 Finite Impulse Response (FIR) Filters

An FIR filter has the transfer function:

$$H(z) = \sum_{m=0}^{M} b_m z^{-m}$$

The FIR filter computes its output using the equation:

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

where:

- M: Filter order.
- b_k: Filter coefficients.
- $x[n-k]$: Delayed input samples.

Key Characteristics:

- **Linear Phase**: Achieved if coefficients are symmetric.
- **Stability**: Always stable as the impulse response is finite.
- **No Feedback**: Only input coefficients are used.

3 Infinite Impulse Response (IIR) Filters

An IIR filter has the transfer function:

$$H(z) = \frac{\sum_{m=0}^{M} b_m z^{-m}}{1 + \sum_{n=1}^{N} a_n z^{-n}}$$

The IIR filter computes its output using:

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

where:

- b_k: Numerator coefficients.
- a_k: Denominator coefficients.
- M: Order of numerator polynomial.
- N: Order of denominator polynomial.

Key Characteristics:

- **Infinite Impulse Response**: Feedback introduces recursion.
- **Stability**: Poles must lie within the unit circle in the z-domain.
- **Efficient**: Requires fewer coefficients compared to FIR for similar performance.

4 LMS Adaptive Filtering Algorithm

The Least Mean Squares (LMS) algorithm is an adaptive filter that iteratively adjusts its coefficients to minimize the error between the desired signal and the output of the filter. It is widely used in signal processing for applications like noise cancellation and system identification.

The LMS adaptive filter aims to minimize the cost function defined as the mean squared error (MSE):

$$J(\mathbf{w}) = \mathbb{E}\left[\left|d[n] - \mathbf{w}^T \mathbf{x}[n]\right|^2\right]$$

where:

- $d[n]$ is the desired signal at time step n.
- $\mathbf{w}$ is the filter's weight vector, which is updated at each iteration.
- $\mathbf{x}[n]$ is the input vector (typically a vector of past inputs).
- The superscript T denotes the transpose of the vector.

4.1 Filter Length M

- **Filter length** is the number of coefficients w_n that are adjusted during this adaptation process. A longer filter length means more coefficients to adapt, which can increase the filter's ability to capture more complex patterns in the input signal but can also lead to more computational complexity and slower adaptation.

- For example, if you set $M = 4$, the filter will have 4 coefficients to adjust, and the input vector at each time step would consist of the current and previous 3 samples of the input signal (a 4-tap filter).

Example: If we specify a filter length of $M = 4$, the adaptive filter will use 4 taps (coefficients), and at each time step, it will adjust the 4 coefficients based on the input signal and the error signal. This determines the filter's response to the input signal and how it adapts to reduce the noise.

Summary:

- **Filter length** M specifies how many past input samples the filter considers to compute the output at any time. A higher M gives the filter more memory (it looks at more past samples) and can lead to better noise reduction but requires more computations and memory.

4.2 LMS Algorithm Steps

The algorithm updates the filter weights in the direction that reduces the MSE. The update rule for the weight vector $\mathbf{w}[n]$ is given by:

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \mu \cdot e[n] \cdot \mathbf{x}[n]$$

where:

- μ is the step size or learning rate, which controls the convergence speed.
- $e[n]$ is the error signal, defined as:

$$e[n] = d[n] - \mathbf{w}^T[n]\mathbf{x}[n]$$

- $\mathbf{x}[n]$ is the input vector at time step n.
- $\mathbf{w}[n]$ is the filter coefficient vector at time step n.

Thus, the LMS algorithm updates the filter coefficients based on the error signal and the input vector. The weight vector $\mathbf{w}[n]$ is adjusted to reduce the error at each iteration.

4.3 Convergence of the LMS Algorithm

The convergence of the LMS algorithm is influenced by the choice of the step size μ. If μ is too large, the algorithm may diverge, whereas if μ is too small, the convergence will be slow. A commonly used condition for stability is:

$$0 < \mu < \frac{2}{\lambda_{\max}}$$

where $\lambda_{\max}$ is the maximum eigenvalue of the input correlation matrix $\mathbb{R}_{\curvearrowleft} = \mathbb{E}[\mathbf{x}[n]\mathbf{x}[n]^T]$.

4.4 Adaptive Filters

The Least Mean Squares (LMS) adaptive filter updates its weights iteratively:

$$w_k[n+1] = w_k[n] + \mu e[n] x_k[n],$$

where:

- μ: Step size.
- $e[n]$: Error signal $e[n] = d[n] - y[n]$.
- $w_k[n]$: Filter weights.
- $x_k[n]$: Input data sample.

5 Python Implementation

5.1 FIR Filter Implementation

The FIR filter implementation in Python is as follows:

```python
import numpy as np
import matplotlib.pyplot as plt

def fir_filter(x, b):
    """
    FIR filter implementation.

    Parameters:
    x : ndarray
        Input signal.
    b : ndarray
        Filter coefficients.

    Returns:
    y : ndarray
        Filtered output signal.
    """
    M = len(b)
    y = np.zeros(len(x))
    for n in range(M, len(x)):
        y[n] = np.dot(b, x[n-M+1:n+1][::-1])
    return y

# Example usage and plotting
fs = 1000  # Sampling frequency
t = np.linspace(0, 1, fs)
x = np.sin(2 * np.pi * 5 * t) + 0.5 * np.random.randn(len(t))  #
    Signal with noise
b = [0.2, 0.2, 0.2, 0.2, 0.2]  # FIR coefficients

y = fir_filter(x, b)
```

```python

plt.figure(figsize=(10, 6))
plt.plot(t, x, label="Noisy Signal")
plt.plot(t, y, label="Filtered Signal", linewidth=2)
plt.legend()
plt.title("FIR Filter")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```

5.2 IIR Filter Implementation

The IIR filter implementation in Python is as follows:

```python
def iir_filter(x, b, a):
    """
    IIR filter implementation.

    Parameters:
    x : ndarray
        Input signal.
    b : ndarray
        Numerator coefficients.
    a : ndarray
        Denominator coefficients.

    Returns:
    y : ndarray
        Filtered output signal.
    """
    M = len(b)
    N = len(a)
    y = np.zeros(len(x))
    for n in range(len(x)):
        y[n] = np.dot(b, x[max(0, n-M+1):n+1][::-1])
        if n > 0:
            y[n] -= np.dot(a[1:], y[max(0, n-N+1):n][::-1])
    return y

# Example usage and plotting
a = [1, -0.5, 0.1]  # IIR coefficients
b = [0.1, 0.2, 0.3]

y = iir_filter(x, b, a)

plt.figure(figsize=(10, 6))
plt.plot(t, x, label="Noisy Signal")
plt.plot(t, y, label="Filtered Signal", linewidth=2)
plt.legend()
plt.title("IIR Filter")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```

5.3 Adaptive LMS Filter Implementation

The LMS adaptive filter implementation in Python is as follows:

```python
def lms_filter(x, d, mu, num_taps):
    """
    LMS adaptive filter implementation.

    Parameters:
    x : ndarray
        Input signal (noisy).
    d : ndarray
        Desired signal.
    mu : float
        Step size.
    num_taps : int
        Number of filter taps.

    Returns:
    y : ndarray
        Filtered output signal.
    e : ndarray
        Error signal.
    w : ndarray
        Final filter weights.
    """
    n = len(x)
    w = np.zeros(num_taps)
    y = np.zeros(n)
    e = np.zeros(n)

    for i in range(num_taps, n):
        x_segment = x[i-num_taps:i][::-1]
        y[i] = np.dot(w, x_segment)
        e[i] = d[i] - y[i]
        w += mu * e[i] * x_segment

    return y, e, w

# Example usage and plotting
d = np.sin(2 * np.pi * 5 * t)  # Desired signal
mu = 0.01  # Step size
num_taps = 5

y, e, w = lms_filter(x, d, mu, num_taps)

```

```python
plt.figure(figsize=(10, 6))
plt.plot(t, d, label="Desired Signal")
plt.plot(t, y, label="Filtered Signal", linewidth=2)
plt.legend()
plt.title("LMS Adaptive Filter")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```

6 Conclusion

This manual provides the mathematical foundation and Python implementations for FIR, IIR, and LMS adaptive filters. Additionally, visualization has been included to demonstrate the effectiveness of the filters.

7 Tasks

Each task requires you to implement all three types of filters: FIR, IIR, and Adaptive LMS, using different parameters and observe the performance for noise reduction.

Variant 1: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{0, 0, 1\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{1, -1.5, 0.7\}, \quad a = \{1, 0.5, 0.1\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.01$ and filter length $M = 4$ to reduce noise in the same noisy sinusoidal signal.

Variant 2: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{1, 1, 2\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{0.5, 0.2\}, \quad a = \{1, -0.8\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.05$ and filter length $M = 4$ to reduce noise in the same noisy sinusoidal signal.

Variant 3: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{0, 0, 3\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{0.6, 0.3, 0.1\}, \quad a = \{1, 0.5, 0.2\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.01$ and filter length $M = 6$ to reduce noise in the same noisy sinusoidal signal.

Variant 4: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{0.2, 0.3, 0.2\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{1, 0.5, 0.3\}, \quad a = \{1, -0.6\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.1$ and filter length $M = 4$ to reduce noise in the same noisy sinusoidal signal.

Variant 5: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{0.5, 0.3, 0.1\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{0.3, 0.4, 0.3\}, \quad a = \{1, -0.4, 0.2\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.05$ and filter length $M = 4$ to reduce noise in the same noisy sinusoidal signal.

Variant 6: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{0.25, 0.25, 0.25, 0.25\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{1, 1, 1\}, \quad a = \{1, -0.5\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.1$ and filter length $M = 6$ to reduce noise in the same noisy sinusoidal signal.

Variant 7: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{1, -1, 0.5\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{0.3, 0.4\}, \quad a = \{1, -0.5, 0.2\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.05$ and filter length $M = 4$ to reduce noise in the same noisy sinusoidal signal.

Variant 8: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{1, 0, 1\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{0.5, 0.5\}, \quad a = \{1, -0.3\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.05$ and filter length $M = 5$ to reduce noise in the same noisy sinusoidal signal.

Variant 9: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{0, 1, 0\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{0.2, 0.8\}, \quad a = \{1, 0.5\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.05$ and filter length $M = 3$ to reduce noise in the same noisy sinusoidal signal.

Variant 10: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{0.4, 0.4, 0.4\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{1, -0.6\}, \quad a = \{1, 0.5\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.1$ and filter length $M = 5$ to reduce noise in the same noisy sinusoidal signal.

Variant 11: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{1, -0.2, 0.5\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{0.6, 0.3\}, \quad a = \{1, 0.4\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.05$ and filter length $M = 6$ to reduce noise in the same noisy sinusoidal signal.

Variant 12: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{1, 0, -1\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{0.4, 0.6\}, \quad a = \{1, -0.3\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.02$ and filter length $M = 5$ to reduce noise in the same noisy sinusoidal signal.

Variant 13: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{1, 0.5, 0.5\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{1, -0.5, 0.2\}, \quad a = \{1, -0.3\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.1$ and filter length $M = 4$ to reduce noise in the same noisy sinusoidal signal.

Variant 14: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{0.5, 0.5\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{0.3, 0.4\}, \quad a = \{1, -0.4\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.1$ and filter length $M = 3$ to reduce noise in the same noisy sinusoidal signal.

Variant 15: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{0.2, 0.3, 0.5\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{1, 0.5\}, \quad a = \{1, -0.7\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.05$ and filter length $M = 6$ to reduce noise in the same noisy sinusoidal signal.

Variant 16: - Design an FIR filter with the following coefficients and implement it in Python to reduce noise in a noisy sinusoidal signal.

$$\text{FIR Filter Coefficients: } b = \{0.5, 0.5\}$$

- Design an IIR filter with the following coefficients and implement it in Python to reduce noise in the same noisy sinusoidal signal.

$$\text{IIR Filter Coefficients: } b = \{0.4, 0.3\}, \quad a = \{1, -0.5\}$$

- Implement an adaptive LMS filter in Python with a step size $\mu = 0.05$ and filter length $M = 5$ to reduce noise in the same noisy sinusoidal signal.