Recursive Filter Demo

This project demonstrates the application of a recursive filter, specifically a Simple Moving Average, on a noisy signal. The Jupyter Notebook includes Python code for generating noisy data, applying the recursive filter, and visualizing the results.

Introduction

Recursive filters are widely used in signal processing to smooth or enhance signals by considering previous observations. In this demo, we focus on a Simple Moving Average as a recursive filter to illustrate its impact on a noisy signal.

```python

Function to generate noisy data

def generate_noisy_data(size=100):

```
    # Implementation...
```

Recursive filter function (Simple Moving Average)

def recursive_filter(data, alpha=0.1):

```
    # Implementation...
```

In [1]:

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

# Function to generate noisy data
def generate_noisy_data(size=100):
    time = np.arange(size)
    true_signal = np.sin(0.1 * time)  # True underlying signal
    noise = np.random.normal(0, 0.5, size)  # Gaussian noise
    noisy_data = true_signal + noise
    return time, noisy_data

# Recursive filter function (Simple Moving Average)
def recursive_filter(data, alpha=0.1):
    filtered_data = [data[0]]  # Initial value
    for i in range(1, len(data)):
        filtered_data.append((1 - alpha) * filtered_data[-1] + alpha * data[i])
    return np.array(filtered_data)

# Generate noisy data
time, noisy_data = generate_noisy_data()

# Apply recursive filter (Simple Moving Average)
alpha = 0.1  # Smoothing factor
filtered_data = recursive_filter(noisy_data, alpha=alpha)

# Plotting
plt.figure(figsize=(12, 6))

plt.plot(time, noisy_data, label='Noisy Data', linestyle='--', color='blue', alpha=0.7)
plt.plot(time, filtered_data, label=f'Recursive Filter (alpha={alpha})', color='orange')

plt.title('Recursive Filter Demo')
plt.xlabel('Time')
```

```
plt.ylabel('Signal Value')
plt.legend()
plt.show()
```