Design and Implementation of FIR and IIR Filters to Clean Noisy Signals Using Python

PYTHON CRT PROJECT

EEE-B/SECTION

Team members

1. 23701A0248
2. 23701A0254
3. 23701A0261
4. 23701A0276
5. 23701A0278
6. 23701A0288
7. 24705A0215
8. 24705A0217
9. 24705A0218
10. 24705A0219
11. 24705A0222
12. 24705A0224
13. 24705A0226
14. 24705A0229
15. 24705A0231
16. 24705A0235

Table of Contents

1.Introduction

Background:-

In the field of digital signal processing (DSP), filters are used extensively to manipulate signals to enhance useful information and reduce noise or unwanted components. Many real-world signals—whether audio, biomedical, or sensor outputs—contain noise which can obscure important characteristics. Filtering is essential to improving signal quality for analysis or further processing.

Objective:-

This project focuses on two primary digital filter types: FIR

(Finite Impulse Response) and IIR (Infinite Impulse Response). Using Python, SciPy, and Matplotlib, the goal is to design, implement, and compare these filters to clean noisy signals.

Structure:-

- Introduce theory behind FIR and IIR filters.
- Design filters using Python libraries.
- Generate noisy test signals.
- Apply filters and visualize results.
- Compare filter performance.

2.Fundamentals of Digital Filters

Discrete-Time Signals:-

Signals processed in digital systems are discrete in time, sampled at a sampling frequency f_s. The Nyquist frequency $f_N=f_s/2$defines the maximum frequency representable without aliasing.

Digital Filters Overview:-

Filters process discrete samples to modify frequency components:

- Lowpass filters allow frequencies below cutoff f_c.
- Highpass filters allow frequencies above f_c.
- Bandpass filters allow frequencies between two cutoffs.
- Bandstop filters attenuate frequencies between two cutoffs.

Impulse Response and System Function:-

A filter's impulse response defines how it responds to a single impulse. The system function H(z) characterizes filter behavior in the z-domain.

3. FIR Filters: Theory

FIR Filter Definition

An FIR filter has a finite number of coefficients

$$y[n] = \sum_{i=0}^{N} b_i x[n-i]$$

b_k:

where N is the order of the filter.

Characteristics:-

- Non-recursive: output depends only on

input. Always stable.

Can have exactly linear phase.

Typically higher order needed for sharp transitions.

Design Using Window Method:-

The ideal lowpass filter impulse response is truncated using a window function (e.g., Hamming window) to limit length and reduce sidelobes.

4: IIR Filters: Theory

IIR Filter Definition:-

An IIR filter uses both inputs and previous

outputs: $$y[n] = \sum_{l=1}^{N} a_l y[n-l] + \sum_{k=0}^{M} b_k x[n-k]$$

Characteristics:-

Recursive filter; output feedback.

Can achieve sharper cutoff with fewer coefficients. Stability depends on pole locations.

Phase response is typically nonlinear.

Butterworth Filter:-

A Butterworth filter is maximally flat in the passband and has monotonic magnitude response, commonly used for general purpose lowpass filtering.

5: Problem Statement and Objectives

Problem Statement

Noisy signals complicate analysis and reduce system performance. The goal is to remove noise effectively without distorting signal features.

Objectives

 Generate noisy test signals with known frequencies. Design FIR and IIR lowpass filters using Python. Apply filters to noisy signals.
 Plot signals and filter frequency responses.
 Compare filter performance.

6: Software and Libraries

Python 3.x:-

The project is implemented in Python for its extensive scientific ecosystem.

Libraries:-

- NumPy: Efficient numerical computations and array handling.
- SciPy.signal: Signal processing functions including filter design and application.
- Matplotlib: Visualization of time series and frequency responses.

Installation:-

```
pip install numpy scipy matplotlib
```

7: Signal Generation and Noise Model

Clean Signal:-

We create a clean sine wave:

$x(t) = \sin(2\pi f t)$

with frequency $f = 5_{Hz}$.

Noise Model:-

Additive Gaussian noise and sinusoidal interference at 50 Hz are added

where noise is white Gaussian noise.

8: FIR Filter Design in Python

Design Parameters:-

- Filter order
- Cutoff frequency $f_c = 10$ Hz.

- Sampling frequency fs=500 Hz.

Design Code:-

```
from scipy.signal import firwin

nyq = 0.5 * fs
normalized_cutoff = cutoff / nyq

fir_coeff = firwin(numtaps=order+1,
cutoff=normalized_cutoff,
pass_zero='lowpass')
```

Explanation:-

- numtaps is filter length = order + 1.
- pass_zero='lowpass' designs a lowpass filter.

9: IIR Filter Design in Python

Butterworth Design:-

```
from scipy.signal import butter

b, a = butter(N=order,
Wn=normalized_cutoff, btype='low')
```

- N is filter order.
- Wn is normalized cutoff.
- btype='low' for lowpass.

10: Applying Filters

FIR Filtering:-

from scipy.signal import lfilter

filtered_fir = lfilter(fir_coeff, 1.0, noisy_signal)

IIR Filtering:-

Use zero-phase filtering to avoid phase

distortion: from scipy.signal import filtfilt

filtered_iir = filtfilt(b, a, noisy_signal)

11: Frequency Response

FIR Filter:-

from scipy.signal import freqz

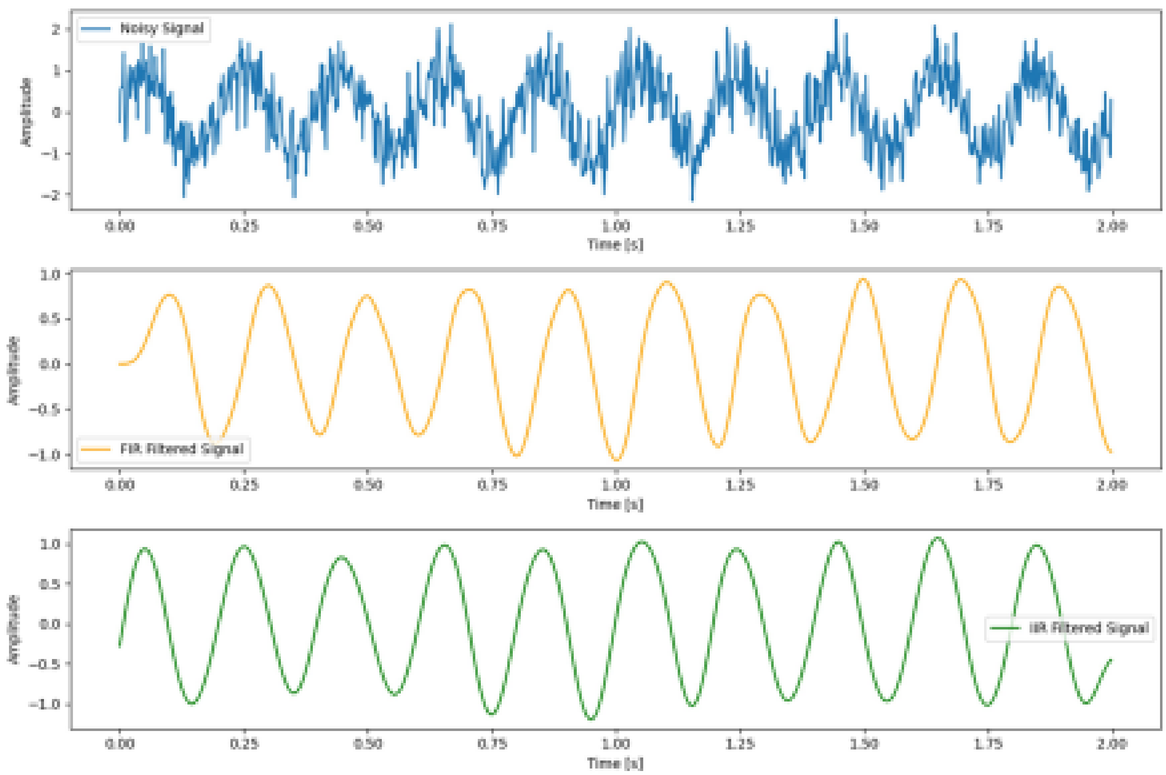
w, h = freqz(fir_coeff)
plt.plot(w * fs / (2*np.pi), 20 *
np.log10(abs(h)))

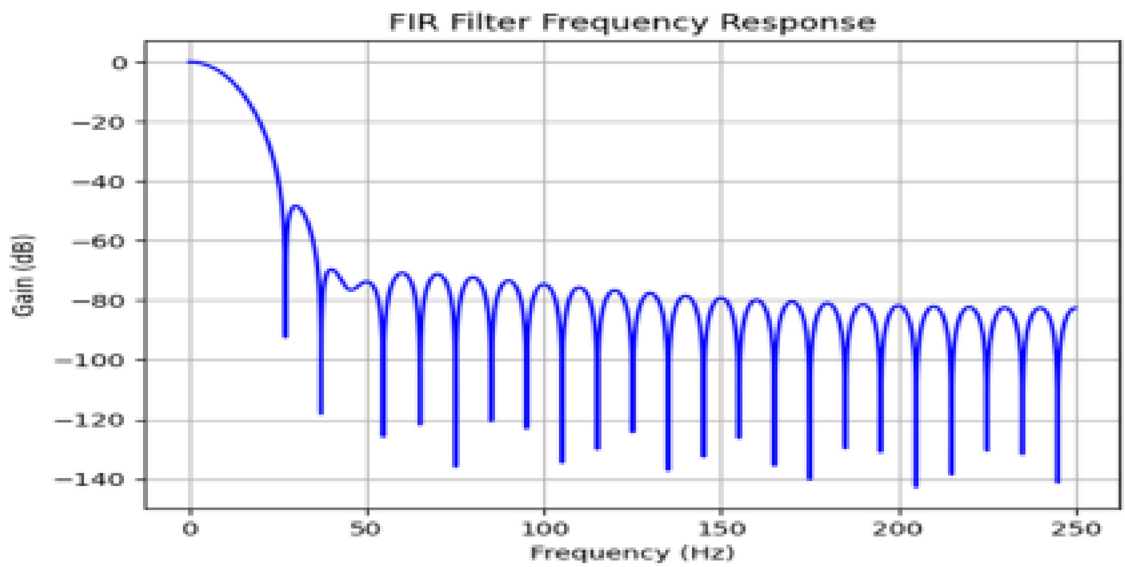
IIR Filter:-

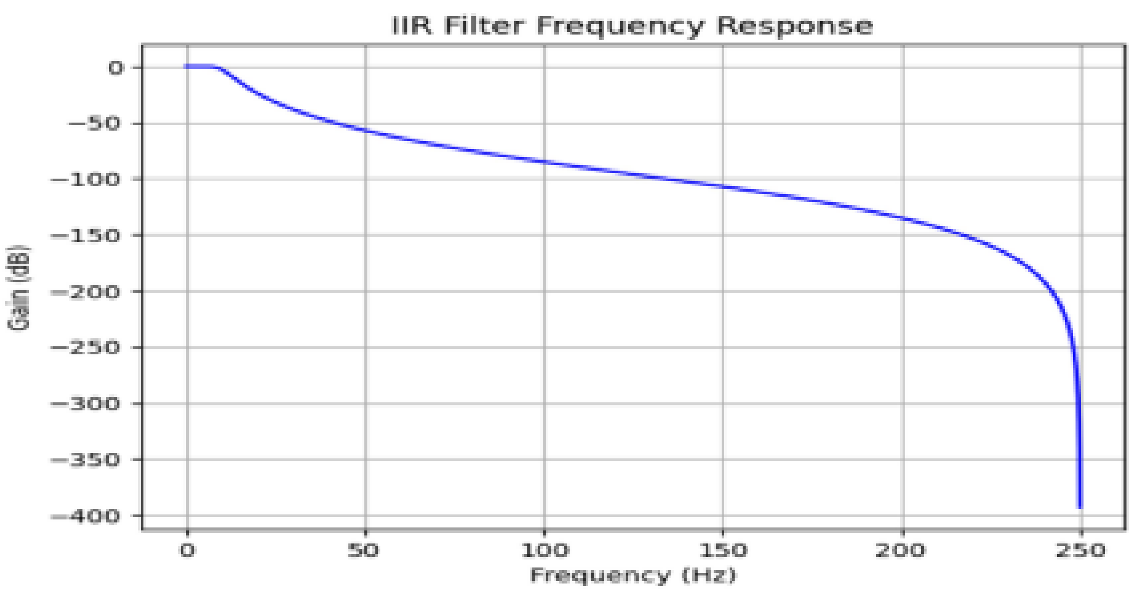
Same method with numerator and denominator coefficients.

12: Visualization of Signals

noisy signal,FIR filtered and IIR filtered signals in time domain to visually assess noise reduction.







13: Performance Comparison Goal:-

Effectively suppress undesired components such as high frequency noise (e.g., 50 Hz interference) while preserving the desired signal (5 Hz sine wave).

FIR Filter:

- Strengths: Offers strong attenuation of frequencies above the cutoff, especially with higher filter orders.
- Observed Performance: The FIR filter successfully removed high-frequency noise, showing a smooth waveform in the filtered signal.
- Limitation: Requires a higher order to achieve a sharp cutoff, especially for narrow transition bands.

IIR Filter:

- Strengths: Achieves sharp roll-off near the cutoff frequency with relatively lower order.
- Observed Performance: The IIR filter effectively eliminated the 50 Hz interference with fewer coefficients.
- Trade-off: More susceptible to ripple or gain irregularities near the cutoff band.

Verdict: Both filters attenuate noise well; IIR does it more efficiently (lower order), but FIR does so more predictably.

14: Applications

Filering is vital in:-

- Audio denoising.
- Biomedical signal cleanup (ECG, EEG).
- Communications.
- Sensor data processing.

15: Extending the Project Ideas

include:

- Designing bandpass and highpass filters.
- Adaptive filtering techniques.
- Real-time filtering.
- Multirate filtering.

16: Challenges

- Choosing filter parameters.
- Managing phase distortion.
- Balancing order vs complexity.
- Noise modeling accuracy.

17: Full Python Code Listing

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import firwin, lfilter, freqz, butter, filtfilt


def design_fir_filter(order, cutoff, fs, filter_type='low'):

 nyq = 0.5 * fs

 normalized_cutoff = np.array(cutoff) / nyq
 if filter_type == 'low':

 pass_zero = True

 elif filter_type == 'high':

 pass_zero = False

 elif filter_type in ['bandpass', 'bandstop']:

 pass_zero = filter_type

 else:

 raise ValueError("Invalid filter_type. Choose from 'low', 'high', 'bandpass', 'bandstop'.")


 fir_coeff = firwin(order + 1, normalized_cutoff,

pass_zero=pass_zero)   return fir_coeff


def design_iir_filter(order, cutoff, fs, filter_type='low'):

 nyq = 0.5 * fs

 normalized_cutoff = np.array(cutoff) / nyq

 b, a = butter(order, normalized_cutoff, btype=filter_type,
```

```
analog=False)   return b, a


def apply_filter(b, a, signal, use_filtfilt=False):
 if use_filtfilt:
 filtered_signal = filtfilt(b, a, signal) # zero-phase filtering
 else:
 filtered_signal = lfilter(b, a, signal)
 return filtered_signal


def plot_frequency_response(b, a=1, fs=1.0, title='Frequency
Response'):   w, h = freqz(b, a, worN=8000)
 plt.plot((fs * 0.5 / np.pi) * w, 20 * np.log10(abs(h)),
'b')   plt.title(title)
 plt.xlabel('Frequency (Hz)')
 plt.ylabel('Gain (dB)')
 plt.grid(True)
 plt.show()
if __name__ == "__main__":
 fs = 500.0 # Sampling frequency (Hz)
 t = np.arange(0, 2.0, 1/fs) # Time vector for 2 seconds


 # Create a noisy signal: 5 Hz sine + white noise + 50 Hz
interference   freq_signal = 5.0
 noisy_signal = (
 np.sin(2 * np.pi * freq_signal * t) +
 0.5 * np.random.randn(len(t)) +
 0.3 * np.sin(2 * np.pi * 50 * t)
```

```
)

# Filter parameters
fir_order = 50
iir_order = 4
cutoff = 10 # cutoff frequency (Hz)
filter_type = 'low' # lowpass filter

# Design FIR filter
fir_b = design_fir_filter(fir_order, cutoff, fs,

filter_type)   # Design IIR filter

iir_b, iir_a = design_iir_filter(iir_order, cutoff, fs, filter_type)

# Apply filters
fir_filtered = lfilter(fir_b, [1.0], noisy_signal)
iir_filtered = apply_filter(iir_b, iir_a, noisy_signal, use_filtfilt=True)

# Plot time domain signals
plt.figure(figsize=(12, 8))

plt.subplot(3, 1, 1)
plt.plot(t, noisy_signal, label='Noisy Signal')
plt.legend()
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
```

```
plt.subplot(3, 1, 2)

plt.plot(t, fir_filtered, label='FIR Filtered Signal',
color='orange')   plt.legend()

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')


plt.subplot(3, 1, 3)

plt.plot(t, iir_filtered, label='IIR Filtered Signal',
color='green')   plt.legend()

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')


plt.tight_layout()
plt.show()


# Plot frequency responses

plot_frequency_response(fir_b, fs=fs, title='FIR Filter Frequency
Response')   plot_frequency_response(iir_b, iir_a, fs=fs, title='IIR Filter
Frequency Response'
```

18: Experimental Results

- Filter effectiveness.
- Frequency response plots.

- Signal waveforms.

19: Conclusions

Summary of findings: FIR filters provide stable linear phase response; IIR filters are efficient but phase nonlinear. Both can effectively clean noisy signals when properly designed.