Assignement 5 Digital Filters

May 16, 2021

By Dimitriy Georgiev

Assignment Description

The assignment requires frequency domain analysis of a sampled signal and implementation of a digital filter based on predefined amplitude response.

The data for this assignment represents IMU measurements (simulated), sampled at 100 Hz by a microcontroller.

Background

A digital filter performs mathematical operations on a sampled, discrete-time signal in order to reduce or enhance certain features of that signal. There are two main types of discrete-time digital filters, namely infinite impulse response (*IIR*) and finite impulse response (*FIR*). *IIR* filters are suitable for application where the phase response of the system is not of much concern. For numerical computation, *IIR* filters can achieve specific filtering characteristics with less memory and number of arithmetic operations, compared to *FIR*. On the other hand, they are less numerically stable and require detailed analysis during design due to quantisation effects and non-linear phase characteristics. The structure of a *FIR* filter ensures linear phase and stability which can be preferred in many applications. *FIR* filters have no analog equivalent, contrary to *IIR* filters which can map to analog filters via the Bilinear (Tustin) transform.

Mathematical Definitions

In practice, an IIR filter can be defined with a linear constant coefficient difference equation given by:

$$y(n) = \frac{1}{a_0}\left(\sum_{i=0}^{P} b_i x[n-i] - \sum_{j=1}^{Q} a_i y[n-j]\right)$$

where, a_i and b_i are the filter's denominator and numerator polynomial coefficients, who's roots are equal to the filter's poles and zeros respectively. It is essential that the poles' magnitudes remain in the unit circle for the filter's stability.

If the feedback coefficients (a_i) are set to zero, the resulting formula defines the equation of a FIR filter which is based solely on past and present input values:

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

There are several practical ways to implement these filters on a digital system. The Direct Form I and Direct Form II (Direct Form Transposed) are the most commmonly used. The Direct Form I is tailored towards fixed-point arithmetic, which in turn is suitable for implementation on microcontrollers without FPU, while Direct Form II is optimized for floating-point math delivering higher numerical accuracy.

Implementation in Python

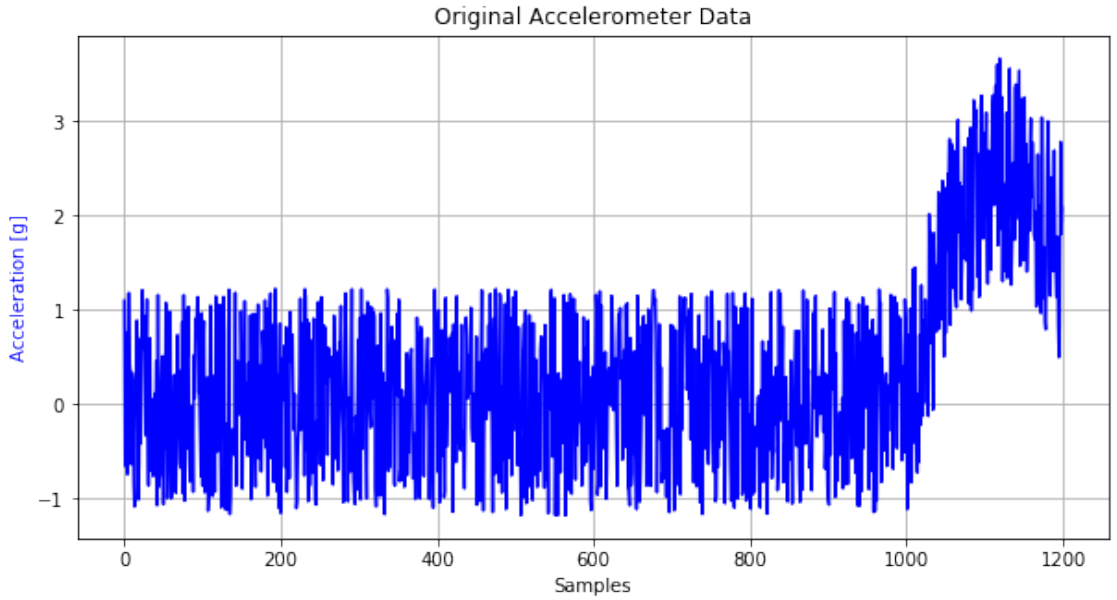
To streamline the design of the filter, the *scipy.signal* package can be utilized to calculate the filter coefficients based on given amplitude response requirements. The amplitude response requirements can feature passband critical frequencies, stopband critical frequencies, passband ripple, and stopband attenuation.

The signal is first analyzed via a discrete fourier transform and the underlying spectrum is plotted. Considering that the data represents accelerometer measurements, the aim is to implement low-pass filter which attenuates noise. The accelerometer can be used during the semester project to track the movement of the HyDrone. As an inspection vessel, the HyDrone moves slowly and steadily. Therefore, it is reasonably to assume that the real movements can be captured under **10 Hz**. (Perhaps even lower).

[1]:
```
import pandas as pd
import numpy as np
from scipy import signal, fft
from matplotlib import pyplot as plt
```

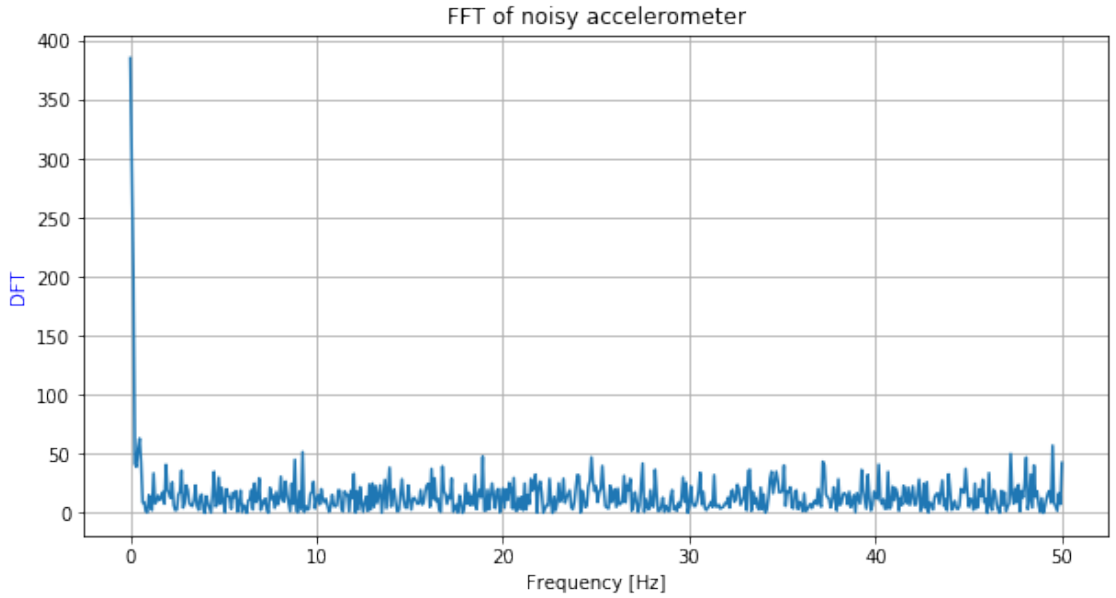
[2]:
```
sampling_rate = 100 # [Hz]

# The signal is imported as 2D array
df = pd.read_csv("acc_data.csv")[6000:7200]

# Convert the data to a single dimensional array
data = df.to_numpy()[:,0] / 2
num_samples = len(data)
sample_spacing = 1.0 / sampling_rate

# Plot
plt.figure(figsize=(10, 5))
plt.title("Original Accelerometer Data")
plt.ylabel("Acceleration [g]", color='b')
plt.xlabel("Samples")
plt.grid(True)
plt.plot(data, 'b')
```

```
plt.show()
```



Apply Discrete Fourier Transform

[3]:
```
# Compute the one-dimensional n-point discrete Fourier Transform (DFT) of a␣
 ↪real-valued array by means of the Fast Fourier Transform.
# Apply a windowing function to the data before calculating the FFT, to reduce␣
 ↪spectral leakage.
dft_y = fft.rfft(data, axis=0)
magnitude = abs(dft_y.real)

# Get the frequency bin centers in cycles per unit of the sample spacing (with␣
 ↪zero at the start).
dft_frequencies = fft.rfftfreq(num_samples, d=sample_spacing)

# Plot
plt.figure(figsize=(10, 5))
plt.title("FFT of %s" % "noisy accelerometer")
plt.ylabel("DFT", color='b')
plt.xlabel("Frequency [Hz]")
plt.grid(True)
plt.plot(dft_frequencies, magnitude)
plt.show()
```



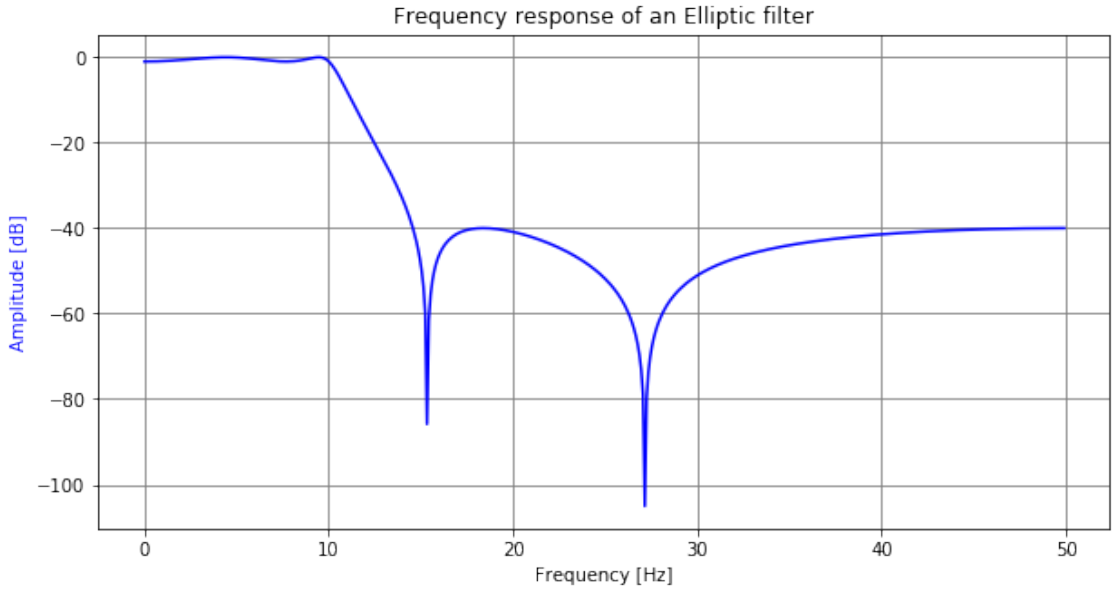
Filter Design

Considering IIR filters, there are several types one can choose from. The most commonly used ones are the Butterworth, Chebyshev I, Chebyshev II and Elliptic. The Butterworth type has the slowest roll-off while the Elliptic has the fastest. On the other hand, Elliptic filters suffer from equiripple in both passband and stopband frequencies. Since equiripple is not big of a concern in this particular application, Elliptic filter can be used.

[4]:
```
wp = 0.2 # Cutoff frequency, normalized from 0 to 1. 1 -> Nyquist frequency.␣
 ↪Hence,  0.2 -> 10 Hz

ws = 0.3 # Stopband edge frequency, 0.3 -> 15 Hz

gpass = 1 # Passband gain (dB). 1 -> Unity Gain

gstop = 40 # Stopband attenuation 40 dB.

# Construct a digital IIR filter of minimum order for given specificaitons
iir_filter = signal.iirdesign(wp, ws, gpass, gstop, ftype='ellip', output='ba')
b, a = iir_filter
print("[INFO] Filter coefficients:")
print("b: ", b[:])
print("a: ", a[:])
print("Order of the filter: ", len(a))

# Calculate frequency response of the filter
```

```
w, h = signal.freqz(b, a)
freq = w * sampling_rate * 1.0 / (2 * np.pi)

# Plot
plt.figure(figsize=(10,5))
plt.ylabel("Amplitude [dB]", color='b')
plt.xlabel("Frequency [Hz]")
plt.title("Frequency response of an Elliptic filter")
plt.grid(which="both", linestyle='-', color="grey")
plt.plot(freq, 20 * np.log10(abs(h)), 'b')
plt.show()
```

```
[INFO] Filter coefficients:
b:  [ 0.01967691 -0.01714282  0.03329653 -0.01714282  0.01967691]
a:  [ 1.         -3.03302405  3.81183153 -2.29112937  0.5553678 ]
Order of the filter:  5
```



Difference Equation

The expanded form of the difference equation of an IIR filter is given by:

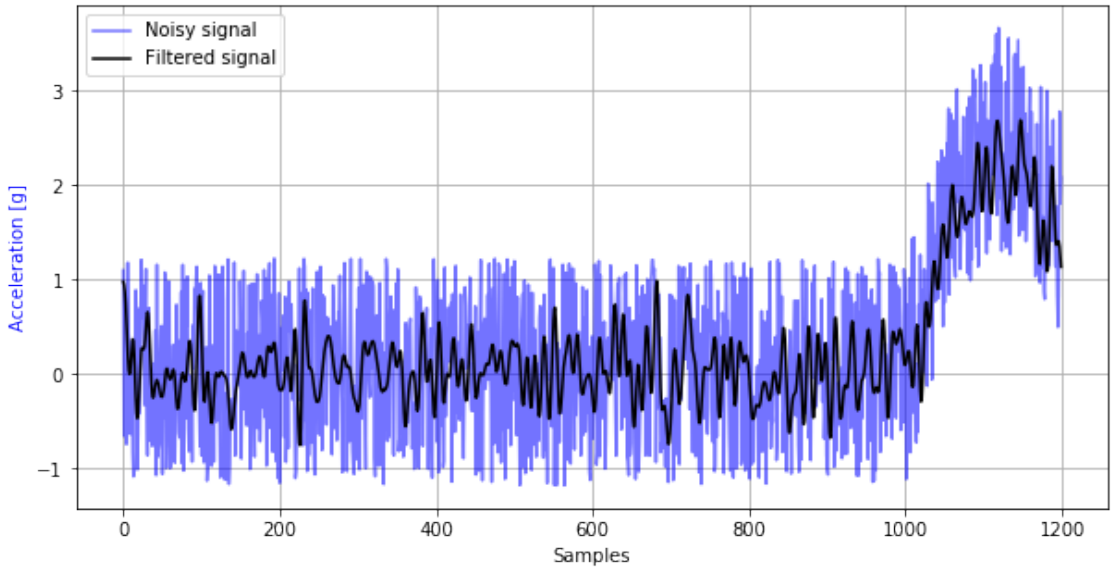
$$y[n] = \frac{1}{a_0}(b_0 x[n] + b_1 x[n-1] + \cdots + b_P x[n-P] - a_1 y[n-1] - a_2 y[n-2] - \cdots - a_Q y[n-Q])$$

Hence,

$$y[n] = (0.02x[n] - 0.017x[n-1] + 0.033x[n-2] - 0.017x[n-3] + 0.02x[n-4] - 3.033y[n-1] - 3.812y[n-2] - 2.291y[n-2] - 0.555y[n-2])$$

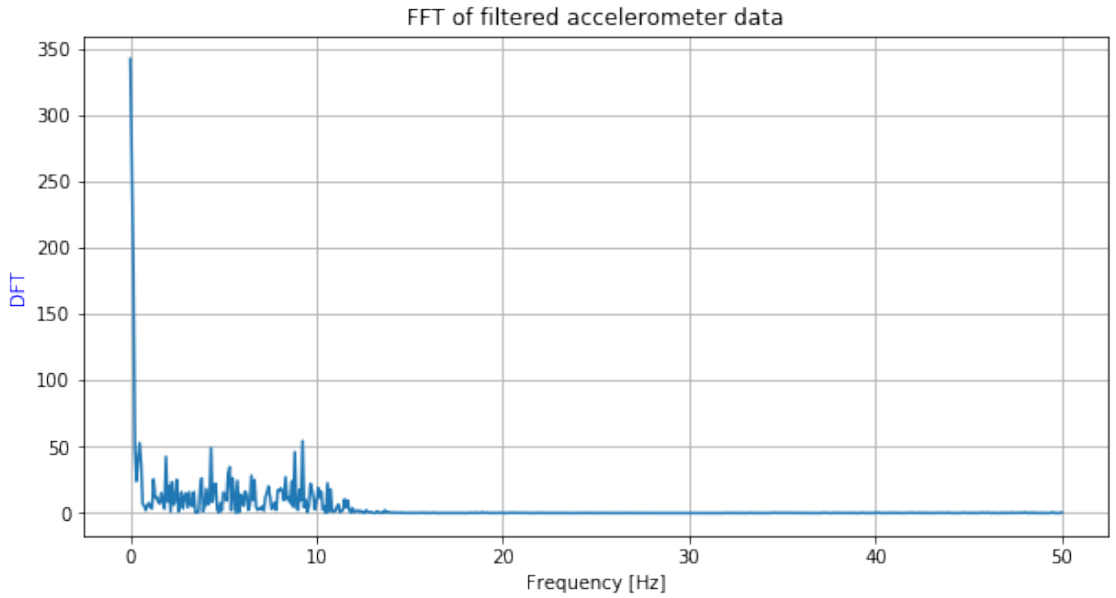
Filter Results

[5]:
```
# Compute the initial state of the filter so that the output of the filter␣
↪starts at the same value as the first element of the signal
zi = signal.lfilter_zi(b, a)

# Filter data with given filter coefficients
# Perform convolution of the input data with the impulse response of the filter.
filtered_data, delay_values = signal.lfilter(b, a, data, zi=zi*data[0])

# Plot
plt.figure(figsize=(10,5))
plt.ylabel("Acceleration [g]", color='b')
plt.xlabel("Samples")
plt.grid(True)
plt.plot(data, 'b', alpha=0.55)
plt.plot(filtered_data, 'k')
plt.legend(('Noisy signal', 'Filtered signal'), loc='best')
plt.show()
```



The results can be improved if the filter is applied twice, once forward and once backwards. The resulting filter will have zero phase and better performance due to increasing the order of the original one by a factor of 2. Increasing the order of the filter, however, will increase computation time and memory usage.

[6]:
```
# Compute the one-dimensional n-point discrete Fourier Transform (DFT) of a␣
 ↪real-valued array by means of the Fast Fourier Transform.
# Apply a windowing function to the data before calculating the FFT, to reduce␣
 ↪spectral leakage.
dft_y = fft.rfft(filtered_data, axis=0)
magnitude = abs(dft_y.real)

# Get the frequency bin centers in cycles per unit of the sample spacing (with␣
 ↪zero at the start).
dft_frequencies = fft.rfftfreq(num_samples, d=sample_spacing)

# Plot
plt.figure(figsize=(10, 5))
plt.title("FFT of %s" % "filtered accelerometer data")
plt.ylabel("DFT", color='b')
plt.xlabel("Frequency [Hz]")
plt.grid(True)
plt.plot(dft_frequencies, magnitude)
plt.show()
```



References

[1] *"Digital filter"* Available: https://en.wikipedia.org/wiki/Digital_filter

[2] *"Difference between IIR and FIR filters: a practical design guide"* Available: https://www.advsolned.com/difference-between-iir-and-fir-filters-a-practical-design-guide/

[3] *"Signal processing (scipy.signal)"* Available: https://docs.scipy.org/doc/scipy/reference/signal.html#filtering