

Hands-on 2

by Arsyadana Estu Aziz (121140068)

Task 1

Analisis Pengaruh Noise pada Sinyal Sinusoid

Tujuan:

- Memahami pengaruh noise terhadap spektrum frekuensi
- Membandingkan performa DFT dan FFT
- Mempelajari visualisasi sinyal dalam domain waktu dan frekuensi

```
In [1]: ## Install Library
import numpy as np
import matplotlib.pyplot as plt
from scipy import fft, signal
import time
```

```
In [2]: ## Implementation of scipy_fft
def calculate_scipy_fft(signal, fs):
    """
    Calculate FFT using SciPy's implementation.

    Parameters:
    signal (array): Input time domain signal
    fs (float): Sampling frequency in Hz

    Returns:
    tuple: (frequencies, fft_result)
    """
    N = len(signal)

    # Calculate FFT
    fft_result = fft.fft(signal)

    # Calculate frequency axis
    frequencies = fft.freq(N, d=1/fs)

    return frequencies, fft_result
```

```
In [3]: ## Implementation of DFT
def calculate_dft(signal):
    """
    Calculate Discrete Fourier Transform (DFT) manually.

    Parameters:
    signal (array): Input time domain signal

    Returns:
    array: Frequency domain representation
    """
    N = len(signal)
```

```

dft = np.zeros(N, dtype=complex)

for k in range(N): # Frequency
    for n in range(N): # Time
        dft[k] += signal[n] * np.exp(-2j * np.pi * k * n / N)

return dft

```

Task 1

- Jelaskan pengaruh level noise terhadap:
 - Bentuk sinyal dalam domain waktu
 - Spektrum frekuensi yang dihasilkan
- Pada level noise berapa sinyal asli (10 Hz) masih dapat diidentifikasi dengan jelas?
- Bandingkan waktu komputasi antara DFT dan FFT untuk sinyal ini

```

In [4]: # 1. Buat sinyal sinusoid sederhana
fs = 1000 # sampling frequency
t = np.linspace(0, 2, 2*fs) # 2 detik sinyal
f1 = 10 # frekuensi 10 Hz
signal = np.sin(2 * np.pi * f1 * t)

# 2. Tambahkan noise dengan berbagai Level
noise_levels = [0.1, 0.5, 1.0] # Level noise yang berbeda
plt.figure(figsize=(15, 10))

for i, noise_level in enumerate(noise_levels, 1):
    # Tambahkan noise
    noisy_signal = signal + noise_level * np.random.normal(0, 1, len(t))

    # Hitung FFT and Print the time operation
    start_time = time.time()
    freq, fft_result = calculate_scipy_fft(noisy_signal, fs)
    scipy_fft_time = time.time() - start_time
    print(f"Scipy FFT time with noise level {noise_level}: {scipy_fft_time:.4f}")

    # Hitung DFT and Print the time operation
    start_time = time.time()
    dft = calculate_dft(noisy_signal)
    dft_time = time.time() - start_time
    print(f"DFT time with noise level {noise_level}: {dft_time:.4f} s")

    # Plot
    plt.subplot(3, 2, 2*i-1)
    plt.plot(t[:100], noisy_signal[:100]) # Plot 100 sampel pertama
    plt.title(f'Sinyal dengan Noise Level {noise_level}')
    plt.xlabel('Waktu (s)')
    plt.ylabel('Amplitudo')

    plt.subplot(3, 2, 2*i)
    freq_mask = (freq >= 0) & (freq <= 30) # Plot 0-30 Hz
    plt.plot(freq[freq_mask], np.abs(fft_result)[freq_mask])
    plt.title(f'Spektrum Frekuensi')
    plt.xlabel('Frekuensi (Hz)')
    plt.ylabel('Magnitude')

```

```

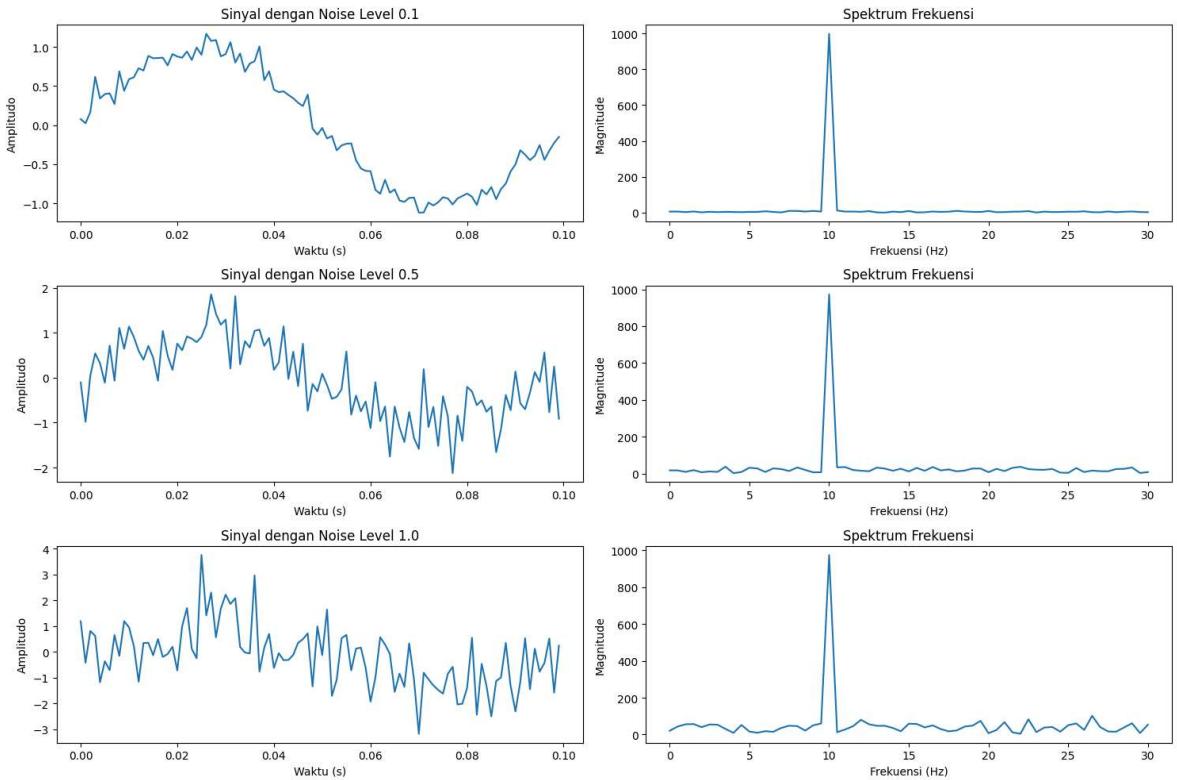
plt.tight_layout()
plt.show()

```

```

Scipy FFT time with noise level 0.1: 0.0021 s
DFT time with noise level 0.1: 25.4464 s
Scipy FFT time with noise level 0.5: 0.0000 s
DFT time with noise level 0.5: 20.5106 s
Scipy FFT time with noise level 1.0: 0.0000 s
DFT time with noise level 1.0: 24.6489 s

```



Explanation

Berdasarkan hasil dari grafik tersebut, pengaruh noise jelas berpengaruh dengan sinyal dalam domain waktu dan domain frekuensi. Definisi dari Fourier Series yang menjelaskan bahwa sinyal merupakan kombinasi dari sinyal-sinyal dengan frekuensi yang berbeda. Sehingga penambahan noise sebagai sebuah frekuensi akan berdampak sebagai berikut:

1. Pada time-domain: Semakin tinggi sebuah noise-level, maka sinyal tersebut akan semakin terdistorsi dan semakin menyimpang dari sinyal awal.
2. Pada frekuensi-domain: Akan muncul sebuah mini-puncak (peak) pada grafik frekuensi domain, yang tentu berpengaruh pada sinyal pada time-domain.

Analisis on Signal

Sinyal asli (10 Hz) masih dapat diidentifikasi dengan jelas ketika nilai dari noise cukup kecil (noise_level = 0.1). Sinyal mulai terdistorsi ketika noise sudah mencapai nilai 0.5.

Time Complexity

DFT sendiri memiliki waktu time complexity sebesar $O(N^2)$, dimana metode ini akan melakukan perluangan sebanyak 2 kali dan sangat tidak efisien dengan data sinyal yang

besar. Hasil dari operasi menunjukan bahwa FFT menang mutlak dalam proses time computation.

Disisi lain, FFT yang hanya memiliki time complexity sebesar $O(N \log N)$, dimana proses ini jauh lebih cepat.

```
yml
Scipy FFT time with noise level 0.1: 0.0000 s
DFT time with noise level 0.1: 39.2945 s
Scipy FFT time with noise level 0.5: 0.0010 s
DFT time with noise level 0.5: 32.7847 s
Scipy FFT time with noise level 1.0: 0.0000 s
DFT time with noise level 1.0: 31.9600 s
```

Task 2

- Jelaskan pengaruh panjang window terhadap:
 - Frequency resolution (kemampuan membedakan frekuensi yang berdekatan)
 - Magnitude spectrum yang dihasilkan
- Window size berapa yang paling optimal untuk mendeteksi ketiga frekuensi?
- Hitung frequency resolution ($\Delta f = fs/N$) untuk setiap window size
- Jelaskan trade-off antara frequency resolution dan time resolution

```
In [5]: # 1. Buat sinyal dengan tiga frekuensi yang berdekatan
fs = 1000 # sampling frequency
t = np.linspace(0, 5, 5*fs) # 5 detik sinyal

# Buat tiga sinyal dengan frekuensi berdekatan
f1, f2, f3 = 10, 12, 15 # frekuensi dalam Hz
signal = (np.sin(2 * np.pi * f1 * t) +
          0.8 * np.sin(2 * np.pi * f2 * t) +
          0.5 * np.sin(2 * np.pi * f3 * t))

# 2. Analisis dengan panjang window berbeda
window_sizes = [fs//2, fs, fs*2] # 0.5s, 1s, 2s windows

plt.figure(figsize=(15, 10))
for i, window_size in enumerate(window_sizes, 1):
    # Ambil sebagian sinyal sesuai window
    windowed_signal = signal[:window_size]
    t_window = t[:window_size]

    # Hitung FFT
    freq, fft_result = calculate_scipy_fft(windowed_signal, fs)

    # Plot
    plt.subplot(3, 2, 2*i-1)
    plt.plot(t_window, windowed_signal)
    plt.title(f'Sinyal dengan Window {window_size/fs}s')
    plt.xlabel('Waktu (s)')
    plt.ylabel('Amplitudo')

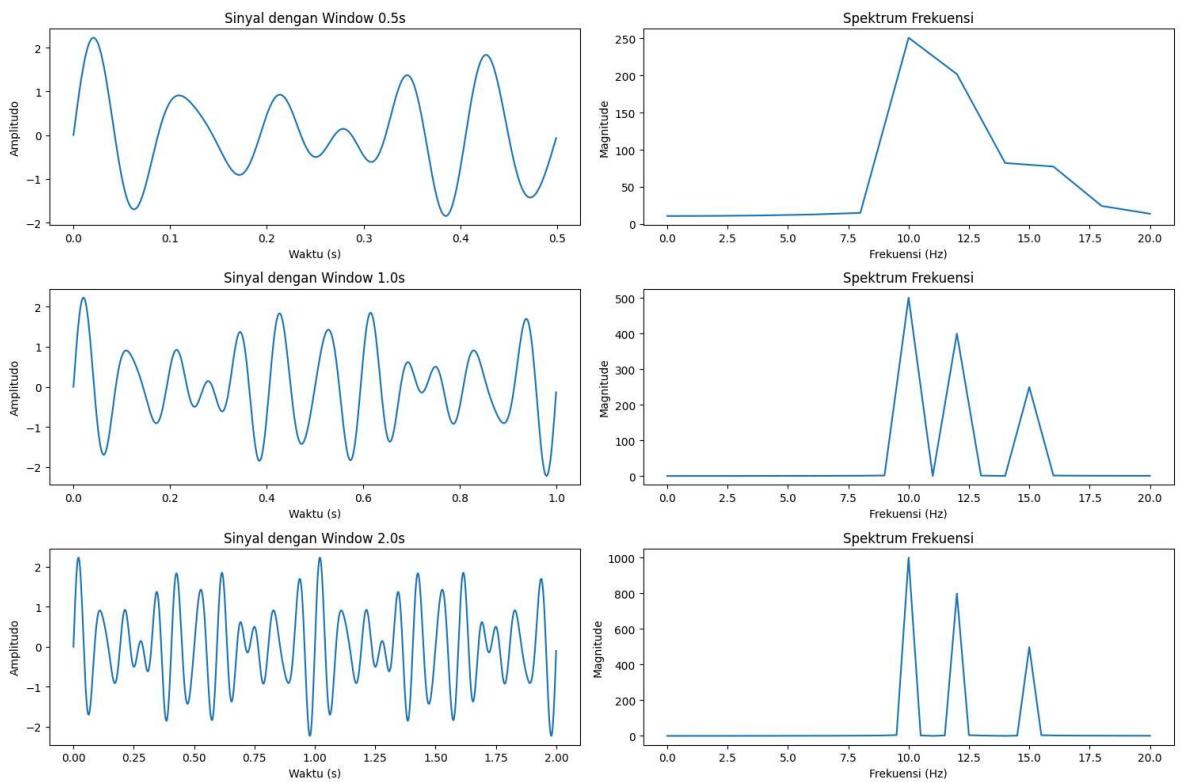
    plt.subplot(3, 2, 2*i)
    freq_mask = (freq >= 0) & (freq <= 20)
    plt.plot(freq[freq_mask], np.abs(fft_result)[freq_mask])
```

```

plt.title('Spektrum Frekuensi')
plt.xlabel('Frekuensi (Hz)')
plt.ylabel('Magnitude')

plt.tight_layout()
plt.show()

```



Explanation

Sebelum masuk ke penjelasan, saya akan menjelaskan tentang apa itu Frequencies Resolution. Frequencies Resolution adalah seberapa jelas kita dapat memisahkan sebuah komponen frekuensi dari sebuah sinyal, bisa di petakan dalam kurun waktu window length (detik)

- Semakin lama waktu window length, kita dapat melihat seperasi dari setiap frekuensi komponen, memungkinkan untuk melihat bagian frekuensi dengan jelas.
- Hal ini bisa dilihat dari puncak di bagian frequencies - domain, dimana pada window 0.5 detik, sangat sulit untuk memisahkan frekuensi komponen dari sinyal.
- Secara optimal, dari grafik ini, semakin lama sebuah window length, maka akan semakin baik juga visualisasi frekuensi yang dihasilkan, pada kasus ini, window length 1 detik menjadi nilai minimum untuk melihat frekuensi resolution yang efektif.

Frequencies Resolution

Frequencies Resolution dapat dihitung dengan formula berikut: $\Delta f = \frac{f_s}{N}$
Dimana f_s adalah jumlah sampling dikalikan dengan window length, dan N adalah

Jumlah sampling secara keseluruhan. Maka frequency resolution untuk masing masing window adalah (sampling rates = 1000).

- 0.5 detik window = $\frac{1000}{500}$, 500 sample dengan 1000 Hz sampling rate = 2 Hz
- 1 detik window = $\frac{1000}{1000}$ = 1 Hz
- 2 detik window = $\frac{1000}{2000}$ = 0.5 Hz

Semakin tinggi sebuah nilai frequency resolution, maka akan semakin sulit untuk membedakan frequencies yang berdekatan, misalkan untuk window 0.5 detik dengan f_s 2 Hz, maka akan sulit untuk membedakan frekuensi 10 and 12 Hz.

Trade off frequencies resolution and time resolution.

Kedua hal ini menggunakan window length sebagai media perhitungan, semakin lama sebuah length window, maka akan semakin baik pula frequencies resolution yang dihasilkan, namun performa dari time resolution akan berkurang, dan sebaliknya.

Task 3

- Mengapa nyquist_rate dihitung sebagai `sampling_rate / 2`? Dan mengapa frekuensi cutoff (low dan high) dibagi dengan nyquist_rate?

`Nyquist theorem` adalah sebuah pernyataan bahwa untuk mensample sebuah sinyal analog tanpa adanya aliasing, maka jenis frekuensi sampling yang digunakan adalah 2x dari maximum frekuensi yang ada di sinyal tersebut. Disisi lain `nyquist rate` merepresentasikan sinyal yang dapat di representasikan tanpa aliasing (biasanya di sekitar `sampling rate / 2`).

Dengan membagi frekuensi `cutoff` dengan nyquist rate, maka hasil dari frekuensi cutoff yang dihasilkan akan berada pada rentang [0,1] untuk mempermudah proses kalkulasi. Metode filter seperti `scipy.butter` atau `scipy.firwin` dapat berjalan dengan memasukan input berupa target `low / high cutoff` beserta sampling rate nya, jadi tidak perlu repot untuk melakukan perhitungan `nyquist rate`

Task 4

- Jelaskan apa itu IIR
- Jelaskan bagaimana merancang filter IIR secara manual (tanpa menggunakan `scipy.signal`)
- Lakukan eksperimen dengan membandingkan filter IIR yang dibuat secara manual Vs. menggunakan `scipy.signal`

IIR (Infinite Impulse Response) adalah metode untuk membuat filter dengan menggunakan konsep feedback loops, artinya menggunakan nilai dari input saat ini dan sebelumnya untuk menghitung nilai dari output saat ini. Berbeda dengan `FIR` (Finite Impulse Response) yang menggunakan konvolusi dalam proses perhitungan, `IIR` menggunakan konsep rekursif sehingga proses yang dilakukan menjadi jauh lebih

efektif dibandingkan dengan FIR . IIR Filter sendiri dapat dijelaskan dengan formula berikut $\$ y[n] = \sum_{k=0}^M b_k x[n - k] - \sum_{j=1}^N a_j y[n - j]$ $\$$ Dimana:

- $x[n]$ adalah input sinyal
- $y[n]$ adalah output sinyal
- b adalah Feedforwards constant (menggabungkan input saat ini dan sebelumnya)
- a adalah Feedback constant (menggabungkan output sebelumnya.)

Creating a IIR Low Filter

Untuk menghitung IIR sendiri, ada beberapa langkah yang perlu dilakukan, pertama menentukan koefisien a dan b lalu menghitung dengan formula IIR.

```
In [6]: import numpy as np

def butterworth_lowpass_manual(x, cutoff, fs):
    # Normalize the frequency (cutoff in rad/s)
    Wn = 2 * np.pi * cutoff # Analog cutoff frequency
    T = 1 / fs # Sampling period

    # Calculate b0, b1, and a1
    b0 = Wn / (2 / T + Wn)
    b1 = b0
    a1 = (2 / T - Wn) / (2 / T + Wn)

    # Normalize the coefficients (by ensuring the gain is 1 at DC)
    b = np.array([b0, b1]) # b coefficients (numerator)
    a = np.array([1, -a1]) # a coefficients (denominator)

    y = np.zeros_like(x)
    for i in range(1, len(x)):
        y[i] = b[0] * x[i] + b[1] * x[i-1] - a[1] * y[i-1]

    return y
```

```
In [7]: from scipy.signal import filtfilt, butter

## Scipy IIR Filter
def scipy_iir_filter(x, cutoff, fs):
    b, a = butter(1, cutoff, btype='low', fs=fs)
    print(b, a)
    return filtfilt(b, a, x)
```

```
In [8]: # Example signal (sine wave + noise)
fs = 1000 # Sampling frequency (Hz)
t = np.linspace(0, 1, fs, endpoint=False) # Time array
x = np.sin(2 * np.pi * 50 * t) + 0.5 * np.random.normal(0, 1, fs) # Signal with

# Apply the Lowpass IIR filter
cutoff = 30 # Cutoff frequency (Hz)
filtered_signal = butterworth_lowpass_manual(x, cutoff, fs)

## Filter the signal using scipy
filtered_signal_scipy = scipy_iir_filter(x, cutoff, fs)
```

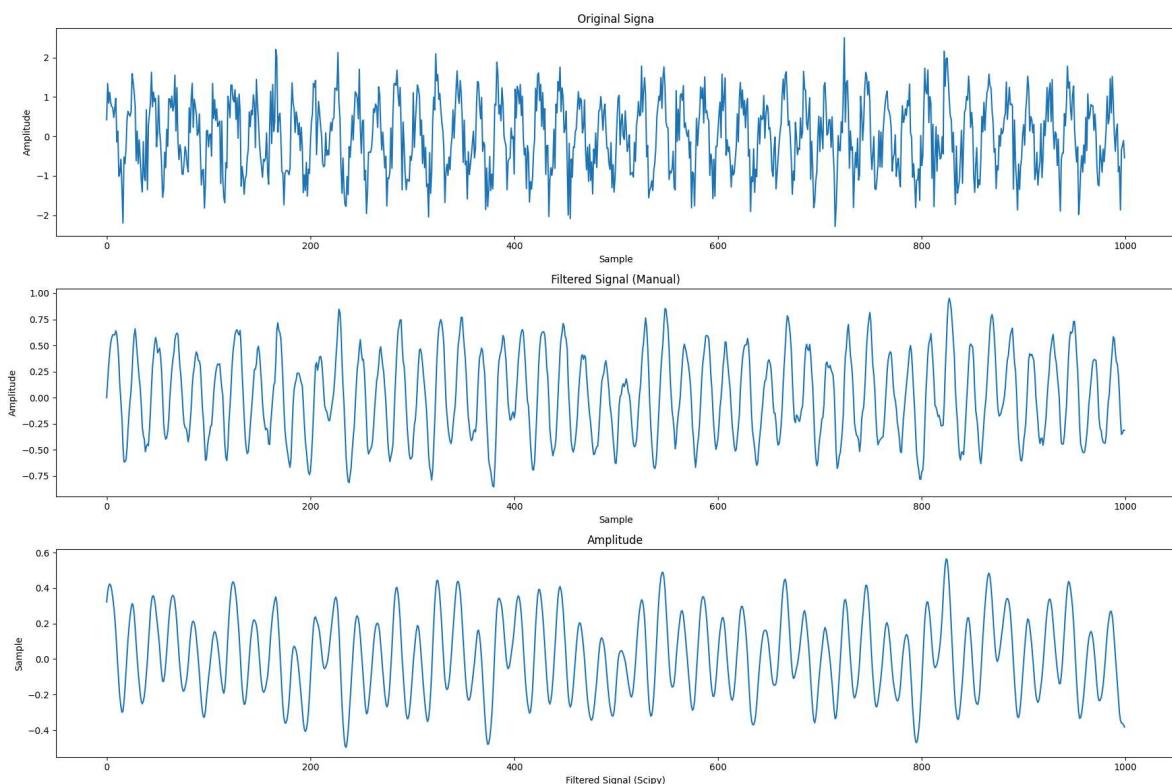
```
[0.08636403 0.08636403] [ 1. -0.82727195]
```

```
In [9]: ## Visualize the signal and the filtered signal
fig, ax = plt.subplots(3, 1, figsize=(18, 12))
ax[0].plot(x)
ax[0].set_title("Original Signal")
ax[0].set_xlabel("Sample")
ax[0].set_ylabel("Amplitude")

ax[1].plot(filtered_signal)
ax[1].set_title("Filtered Signal (Manual)")
ax[1].set_xlabel("Sample")
ax[1].set_ylabel("Amplitude")

ax[2].plot(filtered_signal_scipy)
ax[2].set_xlabel('Filtered Signal (Scipy)')
ax[2].set_ylabel('Sample')
ax[2].set_title('Amplitude')

plt.tight_layout()
plt.show()
```



Explanation

Pada kasus ini, terdapat penggunaan 2 metode yakni dengan IIR Filter secara manual dengan dengan `scipy.signal`. Untuk metode manual sendiri ada 2 langkah penting yang perlu dilakukan.

1. Menentukan koefisien a dan b. Disini akan ditentukan koefisien a dan b, di set sebuah variable `Wn` sebagai analog cutoff untuk menentukan frekuensi untuk menghilangkan frekuensi tinggi pada low-pass filter. Analog cutoff sendiri ini di definisikan sebagai $W_n = 2 \pi f_{cutoff}$

- Sampling period T adalah kebalikan dari sampling rate fs dimana ini merupakan waktu antara setiap sampel yang di definisikan sebagai $T = \frac{1}{f_s}$
- Untuk koefisien a_1, b_0, b_1 merupakan nilai past and current values yang digunakan untuk perhitungan IIR filter nanti.

Task 5

- Jelaskan karakteristik dari filter FIR dan IIR. Apa kelebihan dan kekurangan dari masing-masing filter? FIR Meskipun kedua metode ini menghasilkan filter yang bisa digunakan, namun FIR dan IIR memiliki beberapa kelebihan dan kekurangan.

FIR

- Respon Impulse Terbatas, FIR tidak memiliki feedback layaknya IIR sehingga FIR cenderung stabil di banding IIR
- Waktu proses yang lama, karena proses koefisien FIR yang bergantung pada jumlah sampel, maka waktu komputasi yang dilakukan akan semakin lama karena banyak operasi yang akan dilakukan.
- Desain yang sederhana, desain filter FIR cukup sederhana ketimbang IIR karena melibatkan koefisien filter, mudah untuk di implementasikan

IIR

- Respons Impuls Tak Terbatas, IIR memiliki respon impuls tak terbatas karena mengandung feedback sebagai variable untuk nilai berikutnya.
- Desain yang lebih kompleks, IIR memiliki desain filter yang lebih kompleks karena mengandung banyak koefisien (koefisien sekarang dan feedback)
- Efisiensi komputasi, Filter IIR lebih efisien dalam hal jumlah koefisien untuk mencapai performa yang sama dibandingkan dengan FIR.
- Stabilitas: IIR bisa menjadi tidak stabil jika tidak dirancang dengan hati-hati, karena adanya umpan balik.

Task 6

- Buatlah Respiration Signal dengan library `neurokit2`
- Aturlah `noise = 0.2` untuk ECG atau `noise = 0.05` untuk Respiration Signal
- Gunakan NIM (121140068) anda sebagai random seed
- Sampling rate dan durasi signal bebas
- Lakukan komparasi antara filter FIR dan IIR pada sinyal yang telah dibuat. Gunakan `scipy.signal` untuk filter FIR dan IIR agar lebih mudah.
- Lakukan cut off yang menurut anda paling baik untuk kedua filter tersebut dan jelaskan alasannya. (Pernah dibahas di kelas)

```
In [10]: ## Setup the respiration signal
```

```
import neurokit2 as nk
import matplotlib.pyplot as plt
```

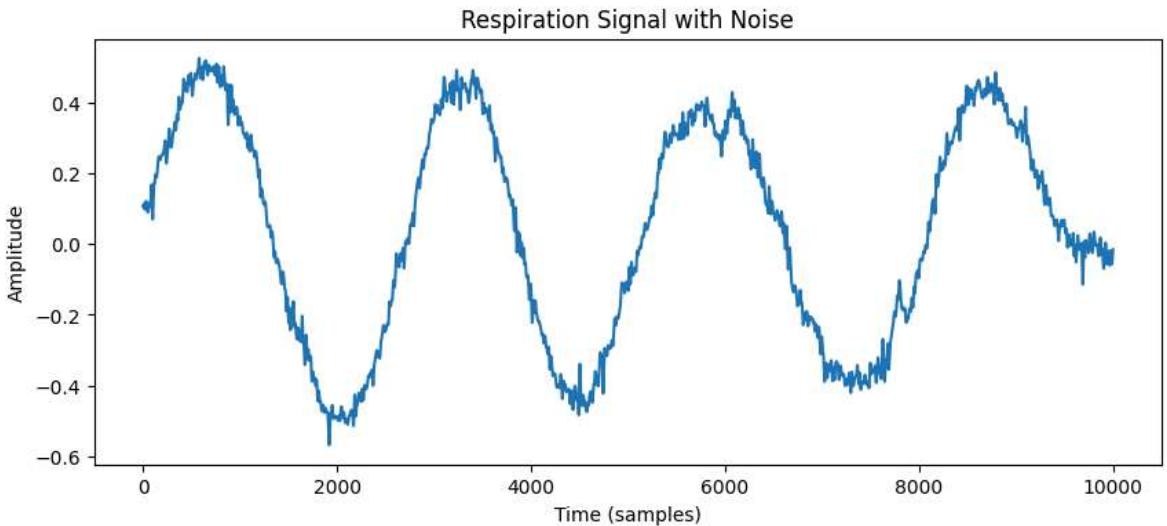
```

# Generate respiration signal with noise
duration = 10 # in seconds
sampling_rate = 1000 # in Hz

respiration_signal = nk.rsp_simulate(duration=duration, sampling_rate=sampling_r

# Plot the respiration signal
plt.figure(figsize=(10, 4))
plt.plot(respiration_signal)
plt.title("Respiration Signal with Noise")
plt.xlabel("Time (samples)")
plt.ylabel("Amplitude")
plt.show()

```



```

In [13]: ## Visualisasi Frekuensi

# Langkah 1: Hitung Panjang Sinyal
N = len(respiration_signal)

# Langkah 2: Hitung FFT Menggunakan Fungsi FFT SciPy
fft_results = fft.fft(respiration_signal)
fft_magnitude = np.abs(fft_results) # Get the magnitude of the FFT
fft_freqs = np.fft.fftfreq(len(respiration_signal), 1 / sampling_rate) # Freque

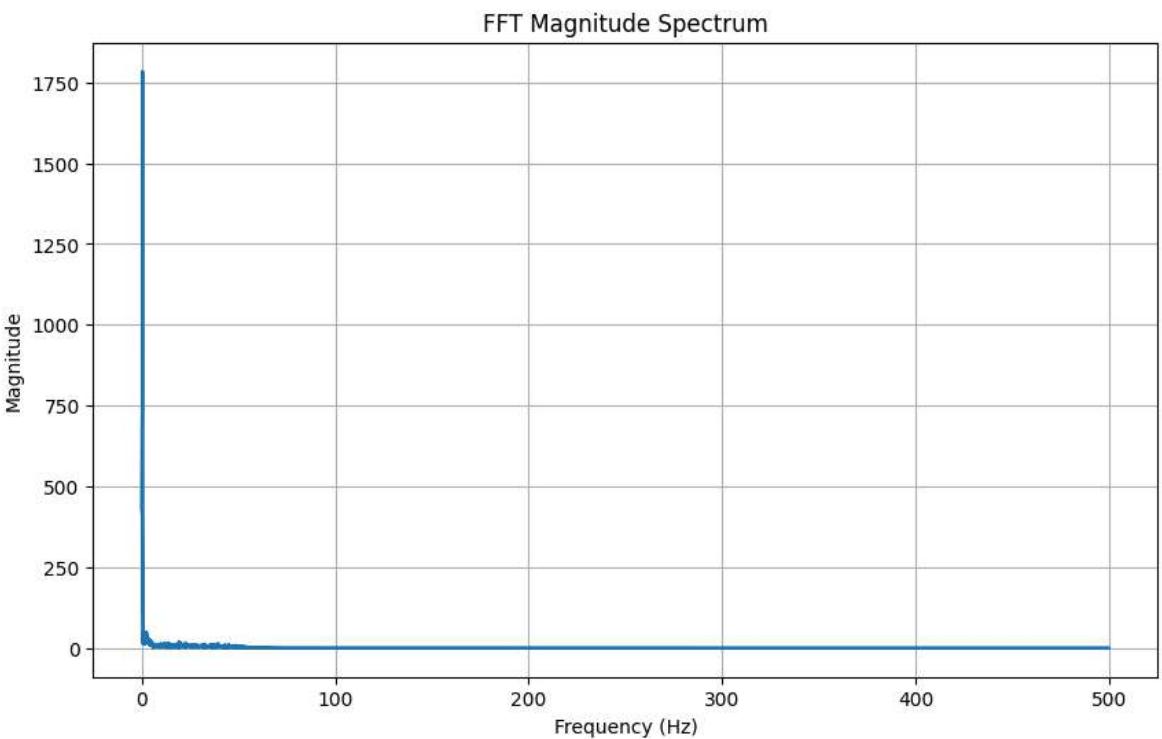
# Langkah 3: Hitung x-axisnya
freq = fft.fftfreq(N, 1/sampling_rate)

# Menemukan frekuensi dominan
dominant_freq = fft_freqs[np.argmax(fft_magnitude)]
print(f"The most dominant frequency is {dominant_freq:.2f} Hz")

# Plot the FFT magnitude spectrum
plt.figure(figsize=(10, 6))
plt.plot(abs(fft_freqs), fft_magnitude)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude')
plt.title('FFT Magnitude Spectrum')
plt.grid(True)
plt.show()

```

The most dominant frequency is 0.40 Hz



```
In [ ]: ## Filtering

from scipy import signal

# Define cutoff frequency
cutoff = 0.5

# Design FIR filter (Low-pass)
fir_order = 101 # Filter order
fir_coeff = signal.firwin(fir_order, cutoff, fs=sampling_rate)

# Apply FIR filter to the respiration signal
fir_filtered = signal.filtfilt(fir_coeff, 1.0, respiration_signal)

# Design IIR filter (Low-pass)
iir_b, iir_a = signal.butter(4, cutoff / (0.5 * sampling_rate), btype='low')

# Apply IIR filter to the respiration signal
iir_filtered = signal.filtfilt(iir_b, iir_a, respiration_signal)

# Plot the original and filtered signals
plt.figure(figsize=(12, 6))

# Original signal
plt.subplot(3, 1, 1)
plt.plot(respiration_signal)
plt.title("Original Respiration Signal")
plt.xlabel("Time (samples)")
plt.ylabel("Amplitude")

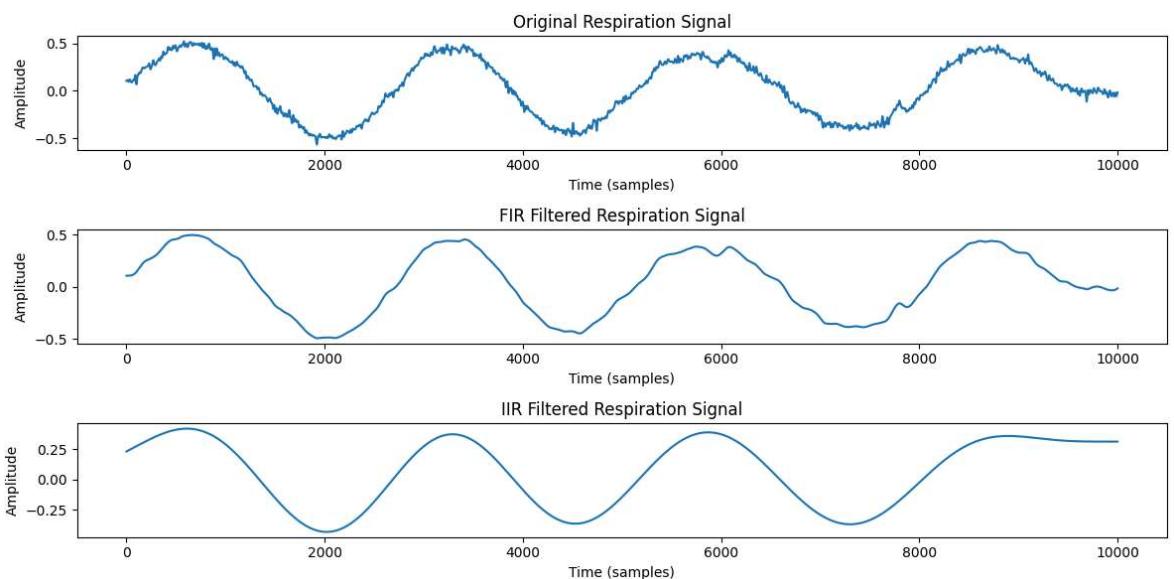
# FIR filtered signal
plt.subplot(3, 1, 2)
plt.plot(fir_filtered)
plt.title("FIR Filtered Respiration Signal")
plt.xlabel("Time (samples)")
plt.ylabel("Amplitude")
```

```

# IIR filtered signal
plt.subplot(3, 1, 3)
plt.plot(iir_filtered)
plt.title("IIR Filtered Respiration Signal")
plt.xlabel("Time (samples)")
plt.ylabel("Amplitude")

plt.tight_layout()
plt.show()

```



Explanation

Alasan memilih frekuensi cutoff pada 0,5 Hz adalah frekuensi maksimum berapa pada frekuensi 0.4, kita akan melakukan filter low-pass untuk mengizinkan frekuensi dibawah 0.5 untuk lewat (Filter noise di frekuensi yang lebih tinggi.)

Referensi:

- [3b1b Fourier Transform Video](#)
- [FFT Tourism SO](#)
- [Intuition behind sampling rates](#)