1. Sinyal PPG

Sinyal PPG merupakan sinyal aliran darah dari jantung ke tangan. Pada soal 1 ini, terlebih dahulu saya akan membuat sinyalnya. Saya akan membuat 2 sinyal, sinyal PPG yang bersih (ppg_clean) dan sinyal noise, yang nantinya akan digabung menjadi 1 sinyal. Pertama, import library-library yang akan digunakan.

In [89]:
```
import neurokit2 as nk
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import decimate
from scipy.signal import resample
```

Kemudian, kita definisikan parameter-parameter yang ada (berdasarkan soal). Setelah didefinisikan, masukkan parameter ke fungsi nk.ppg_simulate().

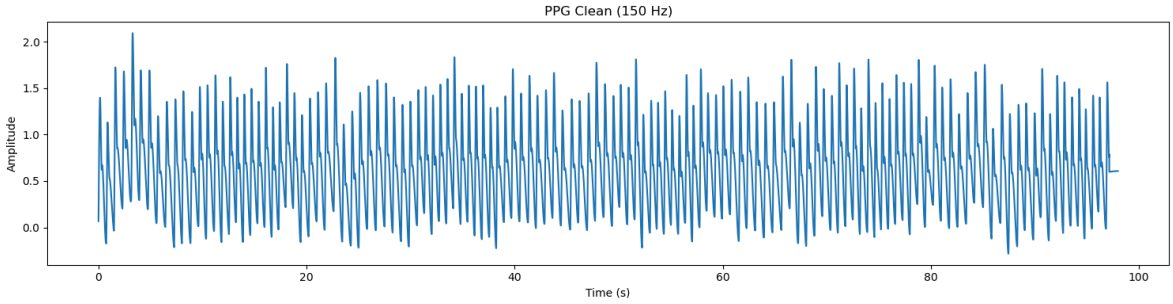
In [90]:
```
# Parameter utama
duration       = 98         # detik
fs             = 150        # Hz
heart_rate     = 80        # BPM
random_state   = 40311      # seed
noise_level    = 0.98       # sigma Gaussian

# Simulasi PPG "bersih"
ppg_clean = nk.ppg_simulate(
    duration=duration,
    sampling_rate=fs,
    heart_rate=heart_rate,
    random_state=random_state
)
```

Lalu kita lakukan pengaturan untuk sumbu x (waktu -> t). 0 merupakan titik waktu mulai, variabel 'duration' menjadi titik selesai, dan 1/fs merupakan interval waktu antar satu sampel sinyal dengan sampel sinyal lain.

In [91]:
```
# Waktu untuk setiap sampel
t = np.arange(0, duration, 1/fs)
```

Lalu, kita lakukan plot untuk sinyal PPG bersih. Berikut adalah tampilan plotnya:

In [92]:
```
# Plot sinyal PPG clean
plt.figure(figsize=(15,4))
plt.plot(t, ppg_clean, label="Clean")
plt.title("PPG Clean (150 Hz)")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.tight_layout()
plt.show()
```

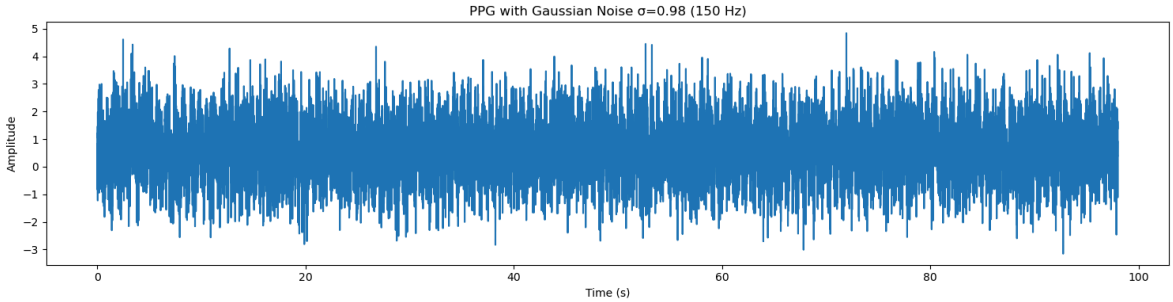


Berikutnya adalah membuat sinyal noise. Di sini, alasan kenapa saya membuat 2 sinyal terpisah yang kemudian digabungkan adalah karena fungsi ppg_simulate dari library neurokit2 tidak memiliki parameter yang membuat variabel noise_level. Saya menggunakan Gaussian noise dengan σ = noise_level. nilai random_state akan digunakan sebagai nilai dari np.random.seed yang berfungsi sebagai nilai awal dari generator bilangan acak. Kemudian noise akan didefinisikan dengan fungsi np.random.normal yang akan meng-generate bilangan acak menurut distribusi normal. Parameter dari noise ada 3 yaitu mean = 0 (artinya rata-rata noise di sekitar nol), std = noise_level = 0.98 (artinya penyebaran noise akan berada di sekitar ini. semakin besar nilainya, berarti amplitudo dari noise akan semakin besar), dan panjang noise yang = panjang dari sinyal ppg_clean. Setelahnya sinyal ppg_clean akan digabungkan dengan sinyal noise dan membentuk sinyal baru yaitu sinyal ppg_noisy.

In [93]:
```
# Tambah Gaussian noise dengan σ = noise_level
np.random.seed(random_state)
noise = np.random.normal(0, noise_level, size=len(ppg_clean))
ppg_noisy = ppg_clean + noise
```

Berikut adalah hasil plot dari sinyal ppg_noisy. Dapat kita lihat bahwa sinyal ppg_noisy ini cukup jauh berbeda dengan sinyal aslinya karena penambahan noise tadi.

In [94]:
```
# Plot sinyal PPG dengan noise
plt.figure(figsize=(15,4))
plt.plot(t, ppg_noisy, label="Noisy")
plt.title(f"PPG with Gaussian Noise σ={noise_level} (150 Hz)")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.tight_layout()
plt.show()
```



a. Downsampling, fs rendah, Aliasing

Downsampling merupakan proses mengurangi laju sampling (sampling rate) sebuah sinyal. Pada sub bagian ini, saya akan membuktikan bahwa downsampling dapat

menghilangkan informasi dari sinyal asli. Di sini, terlebih dahulu saya mendefinisikan ulang variabel yang akan digunakan.

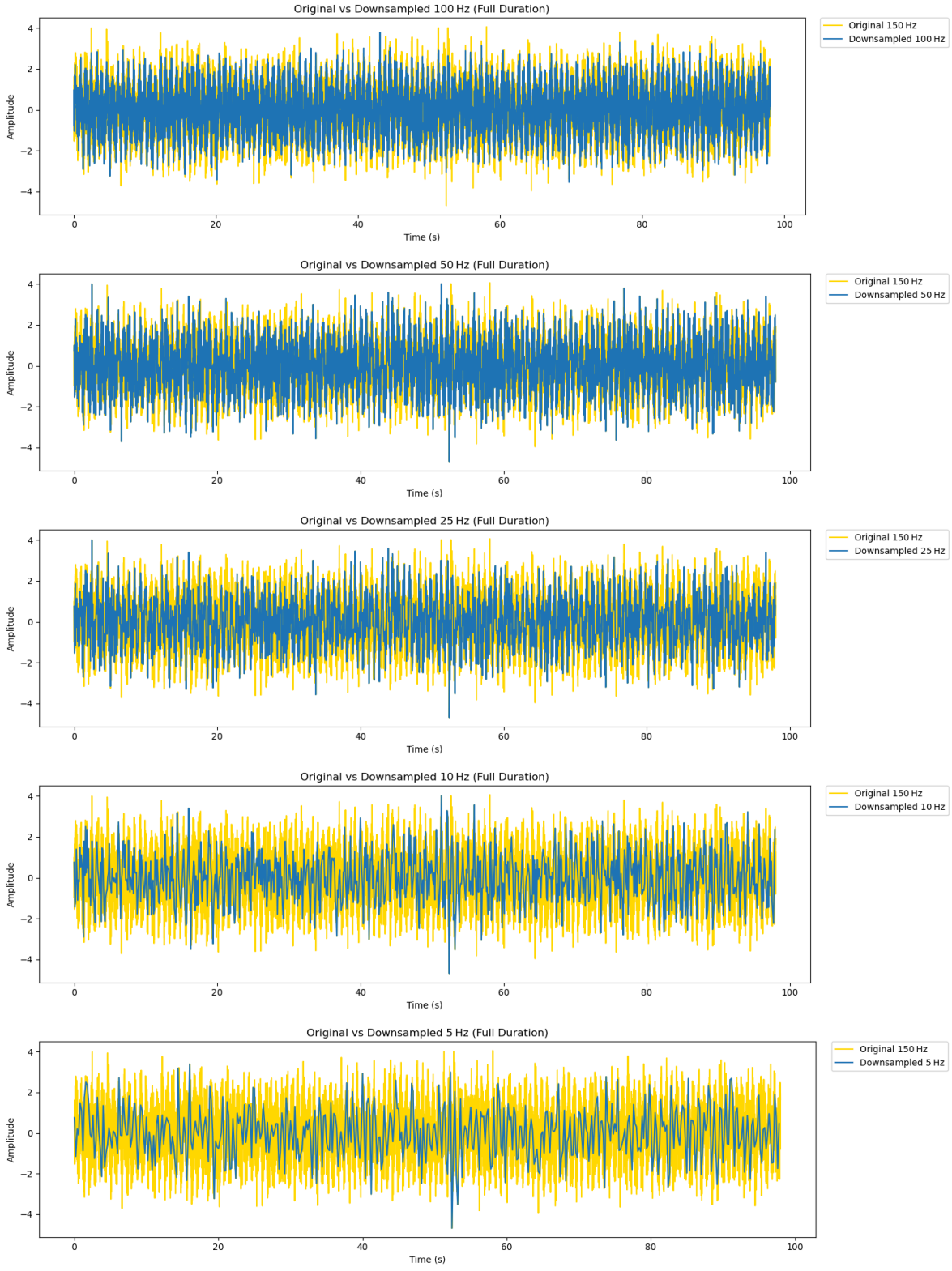
In [95]:
```
# Simulate PPG noisy signal
duration = 98
fs = 150
t = np.arange(0, duration, 1/fs)
np.random.seed(40311)
noise = np.random.normal(0, 0.98, size=len(t))
ppg_clean = np.sin(2 * np.pi * 1.33 * t)
ppg_noisy = ppg_clean + noise
```

Downsampling akan dilakukan dari 150 Hz ke 100, 50, 25, 10, dan 5 Hz. Untuk faktor-faktor non integer misal dari 150 Hz ke 100 Hz akan melalui proses FFT based (resampling), karena apabila menggunakan decitimation akan terjadi error karena parameter tidak bulat. Sementara untuk faktor-fator integer misal dari 150 Hz ke 50 Hz akan menggunakan decimation.

In [96]:
```
# Downsampling target rates and signals
fs_targets = [100, 50, 25, 10, 5]
downsamples = {}

# Resample for non-integer factor (150 -> 100 Hz)
downsamples[100] = (resample(ppg_noisy, int(len(ppg_noisy) * 100 / fs)),
                    np.linspace(0, duration, int(len(ppg_noisy) * 100 / fs), end

# Integer factor downsampling
for fs_new in [50, 25, 10, 5]:
    factor = int(fs / fs_new)
    downsamples[fs_new] = (ppg_noisy[::factor], t[::factor])
```

Berikutnya, kita plot hasil downsampling untuk setiap frekuensi. Didapatkan hasil seperti di bawah. Dapat dilihat, bahwa terjadi perbedaan antara sinyal asli (kuning) dengan sinyal downsampling (biru), apalagi ketika nilai frekuensi nya semakin kecil. Misal pada frekuensi 50 Hz, sinyal mengikuti lembah-puncak besar dari sinyal kuning, tetapi fluktasi kecil-kecil (noise frekuensi tinggi) tampak mulai hilang. Perubahan mendadak di sinyal asli juga kadang terlewat (titik biru seperti 'melompat' melewati beberapa puncak noise kecil).

In [97]:
```
# Plot full duration comparisons with solid lines
orig_color = 'gold'
ds_color = 'tab:blue'

for fs_new, (sig_ds, t_ds) in downsamples.items():
    plt.figure(figsize=(15, 4))
    plt.plot(t, ppg_noisy, label="Original 150 Hz", color=orig_color)
    plt.plot(t_ds, sig_ds, label=f"Downsampled {fs_new} Hz", color=ds_color)
    plt.title(f"Original vs Downsampled {fs_new} Hz (Full Duration)")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.legend(loc='upper left', bbox_to_anchor=(1.02, 1), borderaxespad=0.)
    plt.tight_layout()
    plt.show()
```



Untuk lebih jelasnya, kita bisa menggunakan frekuensi spektrum. Frekuensi spektrun merupakan representasi sinyal dalam lingkup frekuensi. Terlebih dahulu saya menyiapkan variabel yang dibutuhkan.

In [98]:
```
# Simulate PPG noisy signal
duration = 98
fs = 150
t = np.arange(0, duration, 1/fs)
np.random.seed(40311)
noise = np.random.normal(0, 0.98, size=len(t))
```

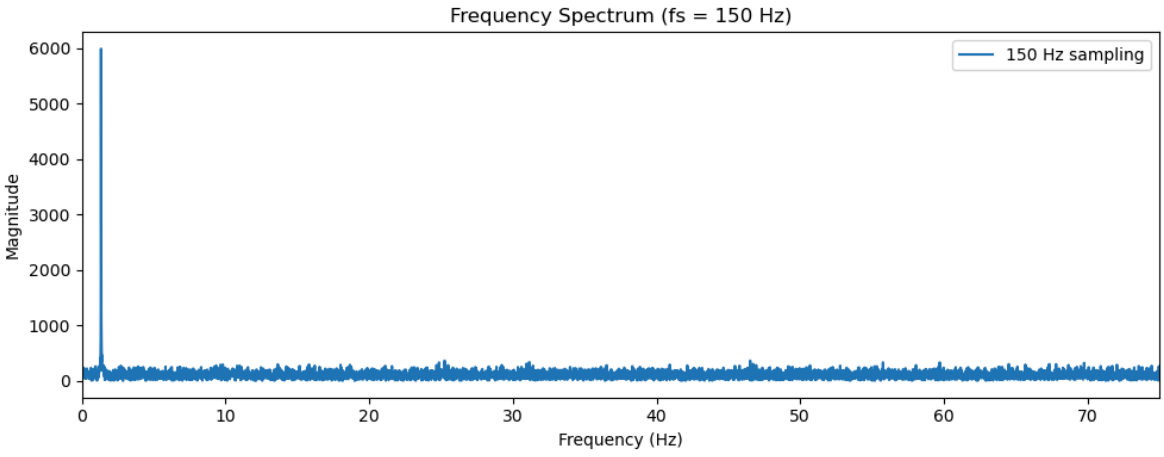
```
ppg_clean = np.sin(2 * np.pi * 1.33 * t)
ppg_noisy = ppg_clean + noise
```

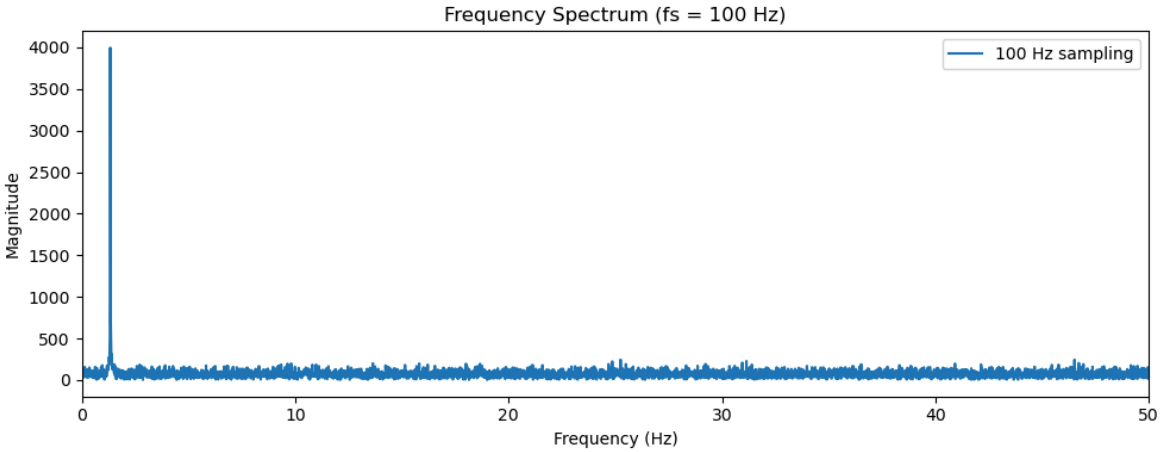
Lalu, membuat sekumpulan versi sinyal PPG yang sama pada berbagai laju sampling—mulai dari sinyal asli di 150 Hz, lalu didownsample ke 100 Hz (menggunakan fungsi resample untuk faktor non-integer) serta ke 50, 25, 10, dan 5 Hz (dengan mengambil tiap nth sampel untuk faktor integer)—lalu menyimpannya dalam dictionary signals sebagai pasangan (sinyal, laju_sampling) agar nanti bisa dibandingkan pengaruh decimation pada tiap laju sampling.

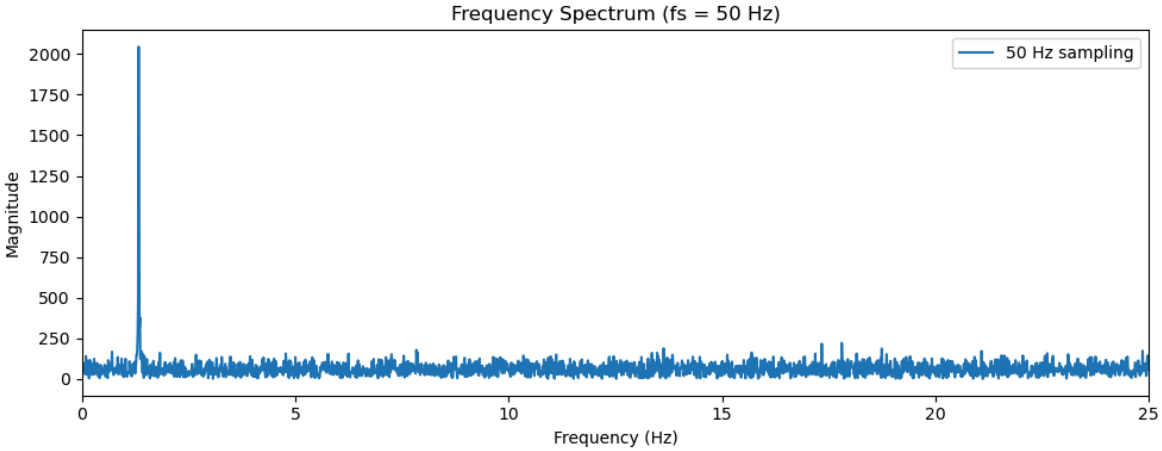
In [99]:
```
# Downsampling target rates and signals
fs_targets = [100, 50, 25, 10, 5]
signals = {}
signals[150] = (ppg_noisy, fs)  # original

# create downsampled signals
sig_100 = resample(ppg_noisy, int(len(ppg_noisy) * 100 / fs))
signals[100] = (sig_100, 100)
for fs_new in [50, 25, 10, 5]:
    factor = int(fs / fs_new)
    signals[fs_new] = (ppg_noisy[::factor], fs_new)
```

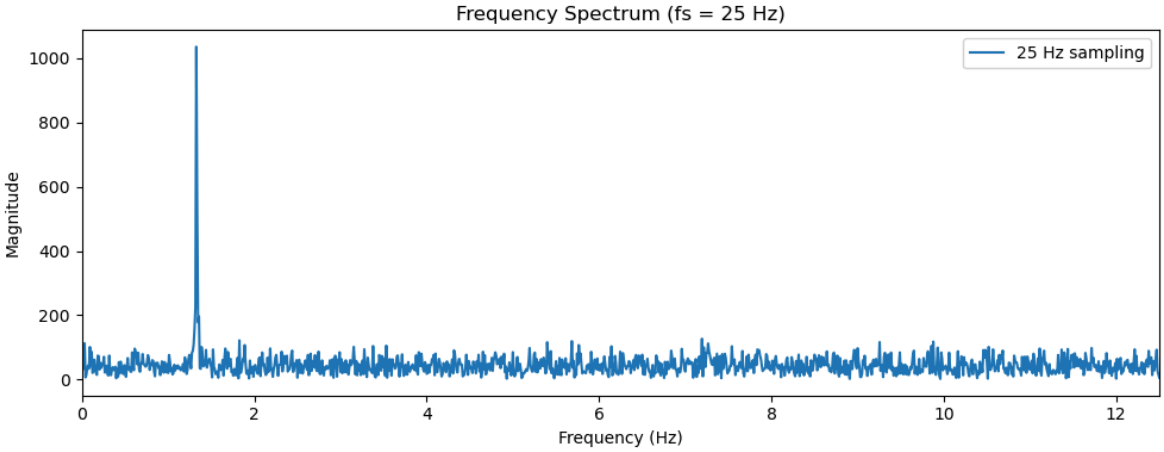
Kemudian, lakukan plot.

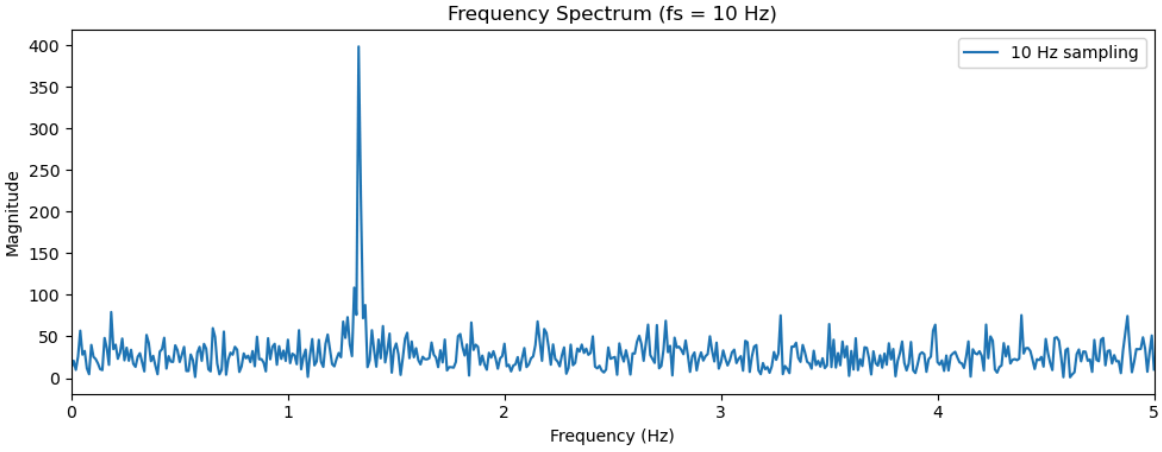
In [100…
```
# Plot frequency spectrum for each
for fs_new, (sig, fsig) in signals.items():
    N = len(sig)
    # FFT and frequency axis
    fft_vals = np.abs(np.fft.rfft(sig))
    freqs = np.fft.rfftfreq(N, d=1/fsig)

    plt.figure(figsize=(10, 4))
    plt.plot(freqs, fft_vals, label=f"{fsig} Hz sampling")
    plt.title(f"Frequency Spectrum (fs = {fsig} Hz)")
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Magnitude")
    plt.xlim(0, fsig/2)  # up to Nyquist
    plt.legend()
    plt.tight_layout()
    plt.show()
```

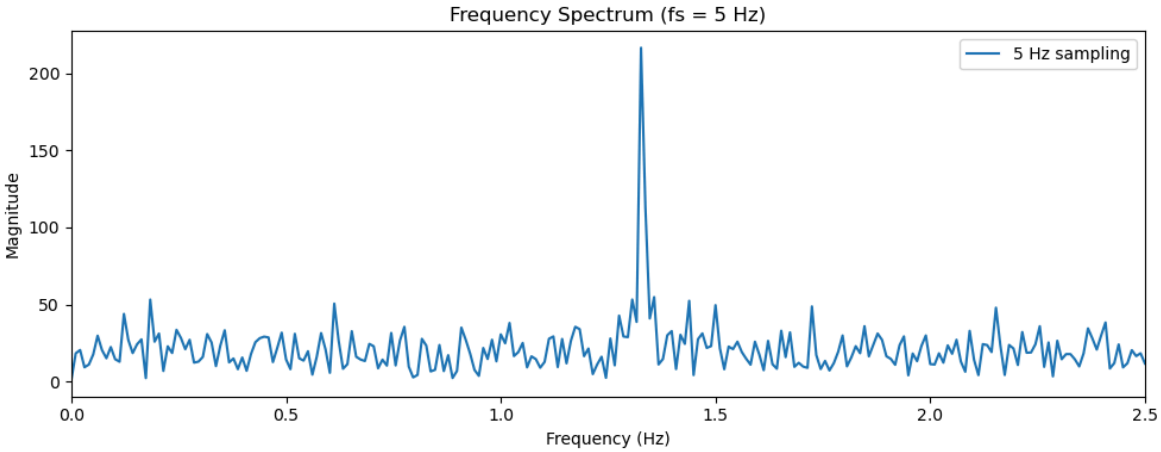



Frequency Spectrum (fs = 100 Hz)
100 Hz sampling
Magnitude
Frequency (Hz)
0
500
1000
1500
2000
2500
3000
3500
4000
0
10
20
30
40
50


Frequency Spectrum (fs = 50 Hz)
50 Hz sampling
Magnitude
Frequency (Hz)
0
250
500
750
1000
1250
1500
1750
2000
0
5
10
15
20
25


Frequency Spectrum (fs = 25 Hz)
25 Hz sampling
Magnitude
Frequency (Hz)
0
200
400
600
800
1000
0
2
4
6
8
10
12


Frequency Spectrum (fs = 10 Hz)
10 Hz sampling
Magnitude
Frequency (Hz)
0
50
100
150
200
250
300
350
400
0
1
2
3
4
5



Berdasarkan grafik dapat dilihat bahwa **terbukti** proses downsampling memberi perubahan informasi pada sinyal asli. Saat fs semakin turun, semua frekuensi di atas fs/2 terpotong. Puncak sinyal juga menjadi semakin lebar dan kurang tajam. fs yang semakin rendah membuat bentuk sinyal semakin terdistorsi dan gagal menangkap ritme jantung secara akurat. Pada fs yang sangat rendah (misal di fs 10 Hz dan 5 Hz), terjadi aliasing (fenomena sinyal terrekonstruksi dari sampel aslinya sehingga menampilkan komponen frekuensi rendah yang sebenarnya tidak ada di sijyal asli). Aliasing terjadi karena fs terlalu rendah, sehingga tidak dapat menangkap detail sinyal secara sempurna. Terjadinya aliasing dapat menandakan bahwa sinyal tidak lagi mewakili sinyal asli.

b. Order

Order adalah parameter yang menunjukkan seberapa “tajam” atau “curam” kemampuan filter memisahkan frekuensi yang diinginkan dari yang tidak diinginkan. Di sini saya akan menunjukkan pengaruh nilai order terhadap sinyal. Prosesnya akan menggunakan filter lowpass. Pertama, definisikan library yang akan digunakan.

In [101…
```
from scipy.signal import butter, filtfilt
```

Lalu, membuat fungsi lowpass_filter() yang merupakan blok anti-aliasing dan denoising supaya hasil decimation bebas artefak frekuensi tinggi dan tidak berubah bentuk helaan detak jantung aslinya.

In [102…
```
# Fungsi lowpass_filter
def lowpass_filter(signal, cutoff, sampling_rate, order):
    nyquist = 0.5 * sampling_rate  # Nyquist frequency
    normal_cutoff = cutoff / nyquist  # Frekuensi normalisasi (cutoff / nyquist)

    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    filtered_signal = filtfilt(b, a, signal)  # meneerapkan filter ke sinyal
    return filtered_signal
```

In [103…
```
# Daftar Parameter
durasi = 98  # Durasi berdasarkan 3 digit terakhir NIM
sampling_rate = 150  # Hz
noise_level = 0.98 # Berdasarkan 2 digit terakhir NIM
heart_rate = 80  # BPM
```

```
random_state = 40311 # Hz
cutoff = 2  # frekuensi cutoff untuk lowpass filter (Hz)
```

In [107…
```
# Simulasi sinyal PPG
ppg_signal = nk.ppg_simulate(duration=durasi, sampling_rate=sampling_rate, heart

np.random.seed(random_state)
noise = np.random.normal(0, noise_level, len(ppg_signal))
ppg_noisy = ppg_signal + noise
```

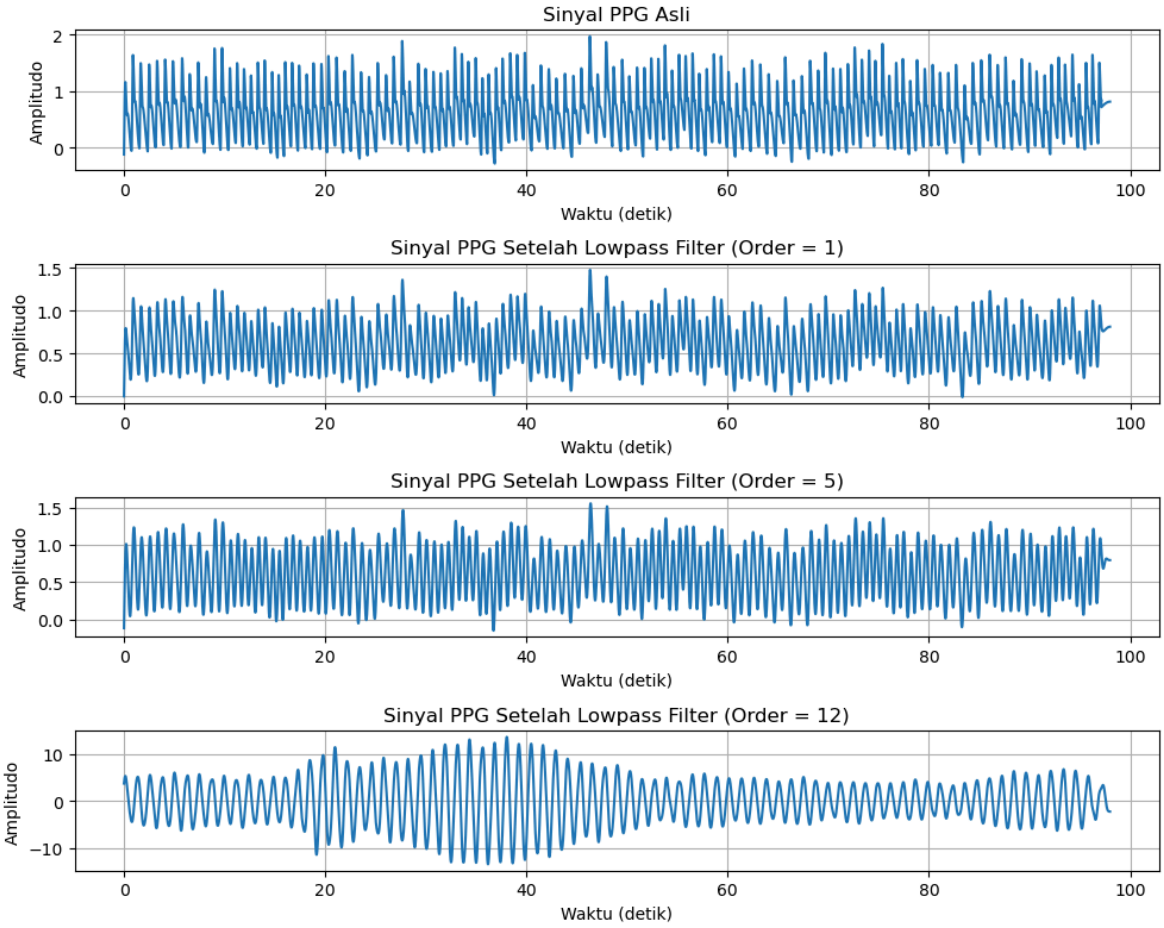
Kita gunakan fungsi lowpass_filter dengan parameter oder menggunakan 3 nilai yaitu 1, 3, dan 12.

In [108…
```
# Menerapkan lowpass filter dengan berbagai order
orders = [1, 5, 12]
filtered_ppgs = [lowpass_filter(ppg_signal, cutoff, sampling_rate, order)for ord
```

In [109…
```
# waktu untuk plotting
time = [i / sampling_rate for i in range(len(ppg_signal))]
```

Kemudian, plot sinyal.

In [110…
```
# Visualisasi
plt.figure(figsize=(10, 8))

# Plot sinyal asli
plt.subplot(4, 1, 1)
plt.plot(time, ppg_signal)
plt.title("Sinyal PPG Asli")
plt.xlabel("Waktu (detik)")
plt.ylabel("Amplitudo")
plt.grid(True)

# Plot sinyal yang telah difilter untuk setiap order
for i, order in enumerate(orders):
    plt.subplot(4, 1, i+2)
    plt.plot(time, filtered_ppgs[i])
    plt.title(f"Sinyal PPG Setelah Lowpass Filter (Order = {order})")
    plt.xlabel("Waktu (detik)")
    plt.ylabel("Amplitudo")
    plt.grid(True)

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



Berdasarkan grafik di atas, dapat disimpulkan bahwa perubahan nilai order sangat berpengaruh terhadap kemampuan filter dalam menghilangkan noise dan mempertahankan sinyal yang diinginkan. Semakin tinggi nilai order, maka semakin bagus filter dalam menghilangkan noise tetapi beresiko menghilangkan fluktuasi-fluktuasi kecil yang ada. Sebaliknya, semakin rendah nilai order, maka kemampuan filter untuk menghilangkan noise akan semakin kecil juga, tetapi flutuasi-fluktuasi penting kemungkinan dipertahankan.

2. Eksperimen Filter Band-Pass

Pada soal ini akan dilakukan eksperimen untuk merancang filter band-pass menggunakan signal.butter pada sinyal respirasi. Mula-mula, definisikan library yang akan digunakan.

In [111…
```
from scipy.signal import freqz
```

Kemudian di sini saya mendefinisikan fungsi filter band_pass.

In [112…
```
#  filter band-pass
def bandpass_filter(signal, low_cutoff, high_cutoff, sampling_rate, order=4):
    nyquist = 0.5 * sampling_rate
    normal_low_cutoff = low_cutoff / nyquist  # Normalisasi frekuensi rendah
    normal_high_cutoff = high_cutoff / nyquist  # Normalisasi frekuensi tinggi
    b, a = butter(order, [normal_low_cutoff, normal_high_cutoff], btype='bandpas
```

```
    filtered_signal = filtfilt(b, a, signal)
    return filtered_signal
```

Berikutnya mendefinisikan parameter-parameter berdasarkan soal yang diberikan.

In [113…
```
# Parameter
durasi = 98
sampling_rate = 100  #  Hz
noise_level = 0.98
respiratory_rate = 18/60  # Hz
random_state = 40311
```

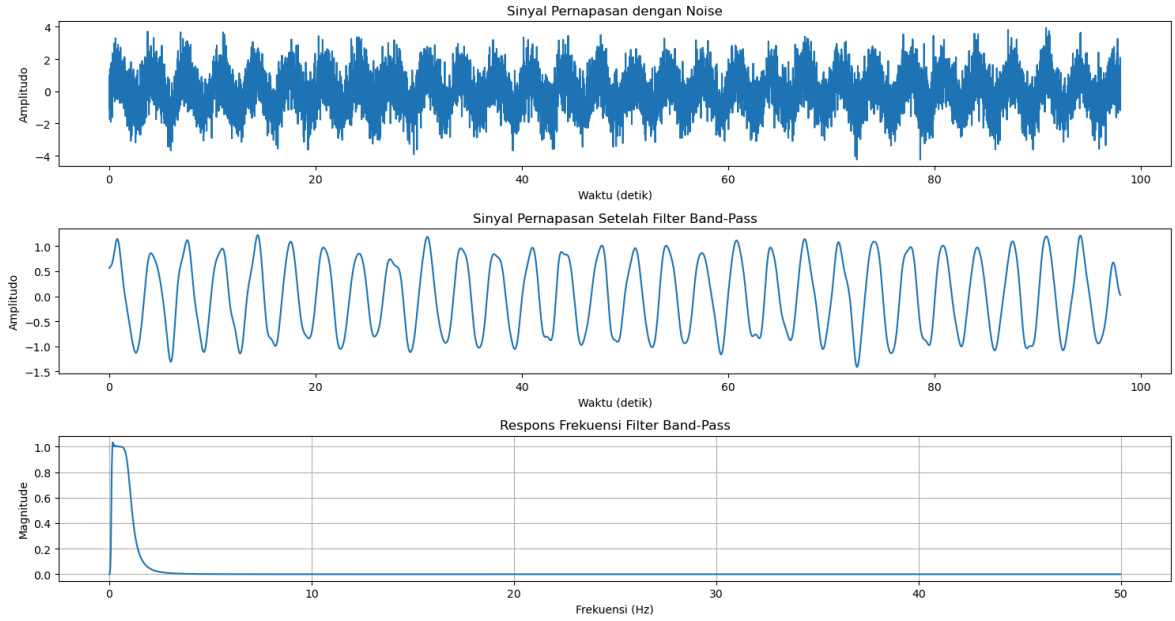
Lalu, menggabungkan sinyal respirasi dengan noise. Di sini, nilai untuk low_cutoff adalah 0.1 Hz dan high_cutoff adalah 0.5 Hz. Kemudian kita terapkan filter pada sinyal.

In [114…
```
# Menggabungkan sinyal pernapasan dengan noise
np.random.seed(random_state)
time = np.linspace(0, durasi, durasi * sampling_rate)
respiratory_signal = np.sin(2 * np.pi * respiratory_rate * time)
noise = np.random.normal(0, noise_level, len(time))
respiratory_noisy = respiratory_signal + noise

# Nilai cutoff untuk filter band-pass
low_cutoff, high_cutoff = 0.1, 1.0

# Menerapkan filter band-pass
filtered_signal = bandpass_filter(respiratory_noisy, low_cutoff, high_cutoff, sa
```

Kemudian, lakukan plot untuk grafik.

In [115…
```
# Visualisasi hasil
plt.figure(figsize=(15, 8))

# Plot sinyal pernapasan dengan noise
plt.subplot(3, 1, 1)
plt.plot(time, respiratory_noisy)
plt.title("Sinyal Pernapasan dengan Noise")
plt.xlabel("Waktu (detik)")
plt.ylabel("Amplitudo")

# Plot sinyal pernapasan setelah filter band-pass
plt.subplot(3, 1, 2)
plt.plot(time, filtered_signal)
plt.title("Sinyal Pernapasan Setelah Filter Band-Pass")
plt.xlabel("Waktu (detik)")
plt.ylabel("Amplitudo")

# Plot respons frekuensi dari filter
b, a = butter(4, [low_cutoff / (0.5 * sampling_rate), high_cutoff / (0.5 * sampl
w, h = freqz(b, a, worN=2000)
plt.subplot(3, 1, 3)
plt.plot(0.5 * sampling_rate * w / np.pi, np.abs(h))
plt.title("Respons Frekuensi Filter Band-Pass")
plt.xlabel("Frekuensi (Hz)")
plt.ylabel("Magnitude")
plt.grid()
```

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



Berikut cutoff yang digunakan: Low_cutoff = 0.1 High_cutoff = 1.0

Alasan pemilihan cutoff ini adalah karena filtering melewatkan osilasi napas (0.1–1.0 Hz), menekan drift lambat (<0.1 Hz), dan menghapus noise di atas 1 Hz. Sehingga didapatkan sinyal pernapasan pada grafik terlihat jelas tanpa gangguan. Sebelumnya saya sudah melakukan eksperimen dengan Low_cutoff = 0.1 dan High_cutoff = 0.5 -> yang merupakan rentang frekuensi pernapasan manusia, namun hasil grafik yang didapatkan kurang baik.

3. Filter Audio Sederhana

Pada soal ini, kita diminta untuk membuat filter audio sederhana dengan menggunakan bandpass, dimana audionya merupakan rekaman suara dengan noise (disini saya menggunakan noise dari kipas angin). Pertama, kita definisikan library-library yang akan digunakan.

In [116…
```
from scipy.io import wavfile
from scipy.signal import butter, filtfilt
```

Kemudian definisikan fungsi bandpass_filter dan compute_fft. Fungsi bandpass_filter akan mengambil sinyal x dan menyaring hanya komponen frekuensi antara lowcut (batas bawah) dan highcut (batas atas) dengan filter Butterwort digital berorde order, lalu menghilangkan distorsi lewat pemroses maju mundur (filtfilt). Kemudian fungsii compute_fft akan mengubah sinyal waktu x menjadi domain frekuensi menggunakan FFT, mengembalikan 2 vektor yaitu frekuensi dan magnitudo.

In [117…
```
# Fungsi
def bandpass_filter(x, lowcut, highcut, fs, order=4):
    nyq = 0.5 * fs
    b, a = butter(order, [lowcut/nyq, highcut/nyq], btype='band')
```

```
    return filtfilt(b, a, x)

def compute_fft(x, fs):
    N = len(x)
    freqs = np.fft.rfftfreq(N, d=1/fs)
    mag   = np.abs(np.fft.rfft(x))
    return freqs, mag
```

Lalu, di bagian ini, kita menginputkan audionya (format WAV). Selain itu kita akan mendefinisikan bandpass, dimana akan didefinisikan 3 rentang frekuensi audioi yang akan dipertahankan ketika proses filterisasi. Ini adalah rentang frekuensi yang umum digunakan dalam pemroses audio. Kemudian setelah rentang frekuensi didefinisikan, kita akan menerapkan filter band-pass untuk setiap rentang BP1, BP2, dan BP3.

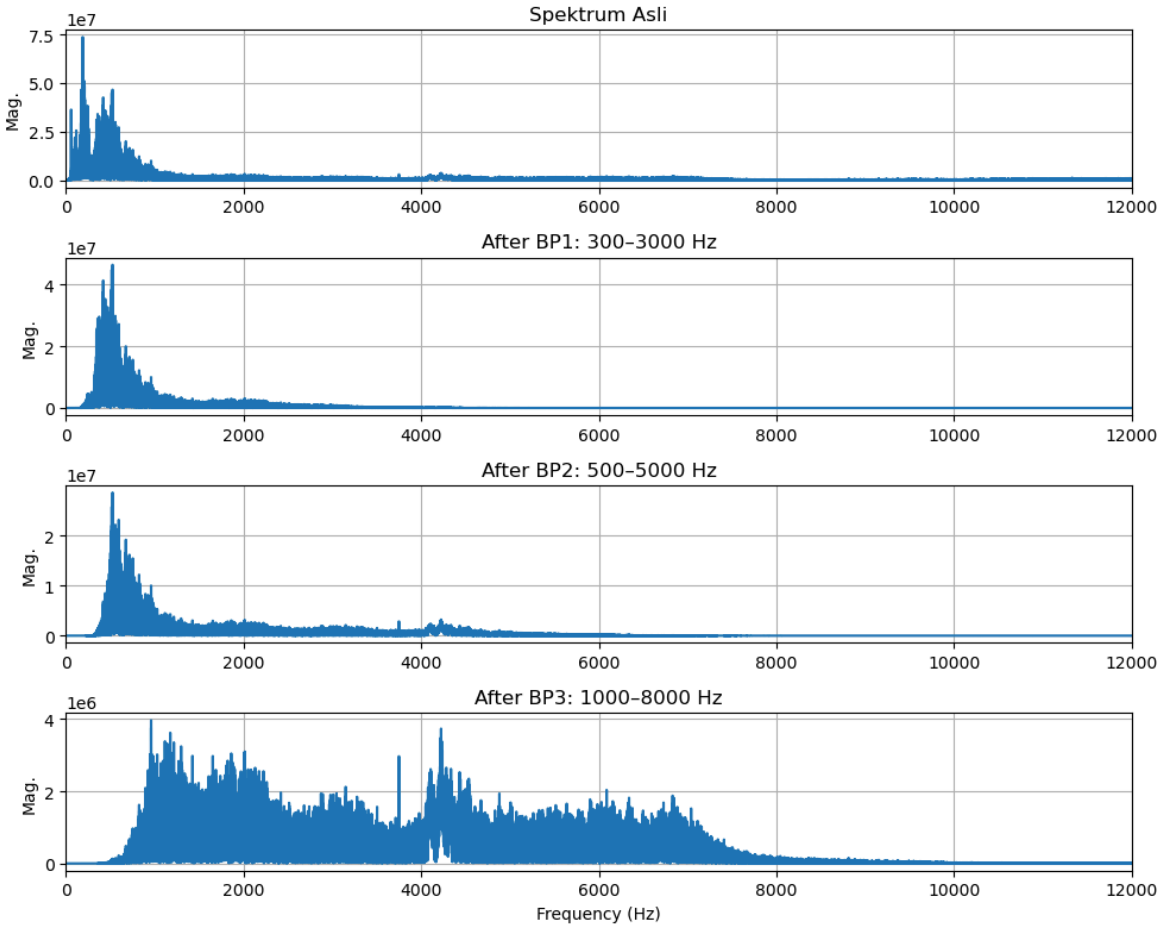
In [118…

```
# Baca file WAV
fs, audio = wavfile.read('sample_DSP.wav')   # fs ≈ 44100
audio = audio.astype(float)

# Definisikan bandpass
bandpasses = {
    'BP1': (300, 3000),    # fokus pita vokal dasar
    'BP2': (500, 5000),    # lebar menengah
    'BP3': (1000, 8000),   # pita tinggi (sibilan)
}

# Terapkan Filter
filtered = {}
for name, (low, high) in bandpasses.items():
    filtered[name] = bandpass_filter(audio, low, high, fs, order=4)
```

Berikutnya, kita lakukan plot grafik terhadap spektrum frekuensi sebelum difilter dan setelah difilter (3 grafik setelah difilter -> BP1, BP2, BP3).

In [119…

```
# Plot Spektrum
plt.figure(figsize=(10, 8))

# Spektrum asli
freqs, mag = compute_fft(audio, fs)
plt.subplot(4,1,1)
plt.plot(freqs, mag)
plt.title('Spektrum Asli')
plt.xlim(0,12000); plt.ylabel('Mag.'); plt.grid(True)

# Spektrum setelah tiap bandpass
i = 2
for name, sig in filtered.items():
    f, m = compute_fft(sig, fs)
    plt.subplot(4,1,i)
    plt.plot(f, m)
    plt.title(f'After {name}: {bandpasses[name][0]}-{bandpasses[name][1]} Hz')
    plt.xlim(0,12000); plt.ylabel('Mag.'); plt.grid(True)
    i += 1

plt.xlabel('Frequency (Hz)')
plt.tight_layout()
plt.show()
```



Pada grafik spektrum asli, energi paling besar ada di bawah 500 Hz, yang merupakan campuran suara kipas dan nada dasar suara manusia di audio dan masih ada sisa sampai sekitar 12 kHz. Pada BP1 (300–3000 Hz), semua “dengung” rendah di bawah 300 Hz dan desis tinggi di atas 3000 Hz hilang, sehingga tinggal suara vokal utama (300–1000 Hz) dan sedikit konsonan sampai 3 kHz. Pada BP2 (500–5000 Hz) dilakukan pemotongan frekuensi rendah di bawah 500 Hz, sehingga vokal terasa lebih “tipis” tapi konsonan di 2–5 kHz lebih jelas. Pada BP3 (1000–8000 Hz) terjadi pembuangan semua frekuensi di bawah 1 kHz, jadi nada dasar vokal hilang dan yang tersisa hanya desis dan gema di 1–8 kHz.

Kesimpulannya secara umum adalah, semakin banyak iterasi bandpass berarti filterisasi semakin kencang dalam membuang frekuensi diluar pass-band. Hal ini menyebabkan semakin berkurangnya sinyal di dalam pass-band dan menambah delay. Oleh karena itu, gunakan iterasi secukupnya.

REFERENSI

Link Referensi