**Analysis and Feature Extraction from a PPG Signal**

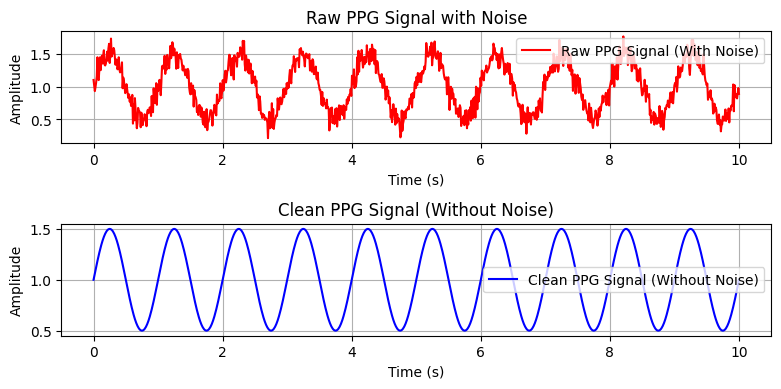
**Purpose:** The purpose of this project is to simulate, process, and analyze a synthetic PPG signal, which is commonly used in wearable devices to measure heart rate, respiratory rate, and heart rate variability (HRV and other physiological parameters. The project demonstrates how to:

* Generate a synthetic PPG signal with noise.
* Clean and filter the signal to remove noise.
* Detect peaks and extract meaningful features from the signal.
* Perform time-domain and frequency-domain analysis
* Compute heart rate, respiratory rate, and heart rate variability (HRV).
* Explore signal processing techniques such as filtering, correlation, convolution, and frequency response analysis.

**Worked Explanations:**

1. **Signal Generation and Preprocessing:**

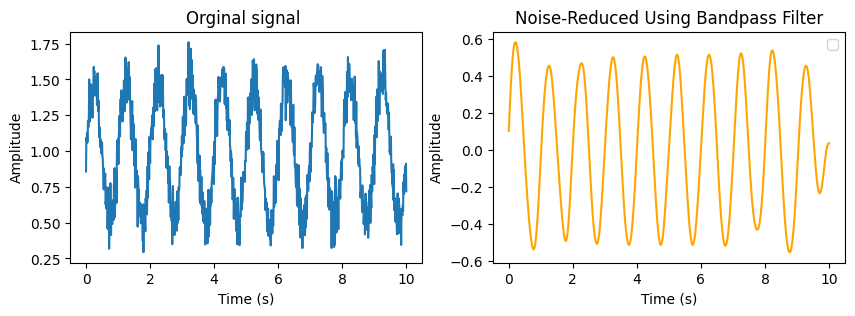
* A synthetic PPG signal is generated using a sinusoidal function with added noise to simulate real-world conditions
* The signal is visualized to understand its characteristics before and after adding noise.



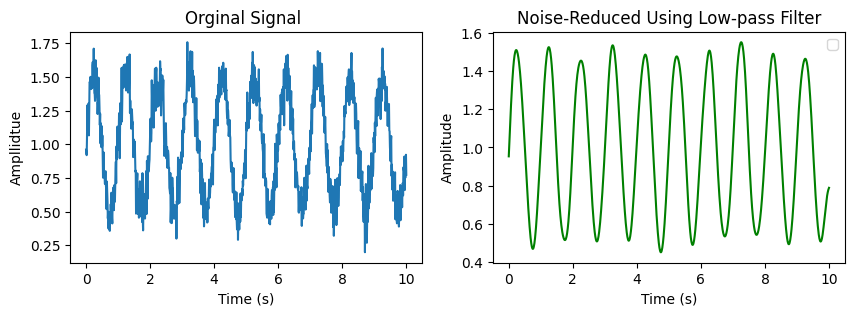
**Figure 01: Raw PPG Signal with noise and without noise**

1. **Noise Reduction and Filtering:**

* A **bandpass filter** (0.5 – 3.0 Hz) is applied to remove unwanted noise and retain only the useful components of the PPG signal.
* A **low-pass filter** is also tested to observe its impact on the signal.



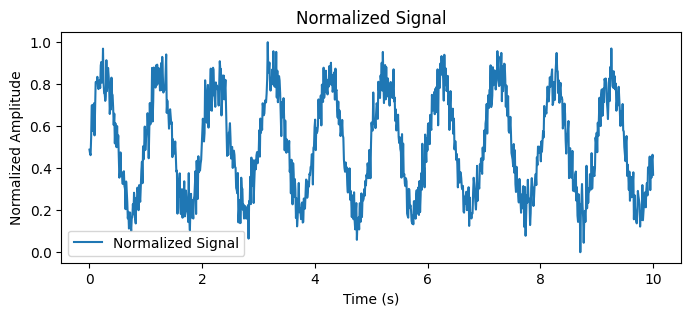
**Figure 02: Noise reduced using bandpass filter**

****

**Figure 03: Noise reduced using bandpass filter**

1. **Signal Normalization:**

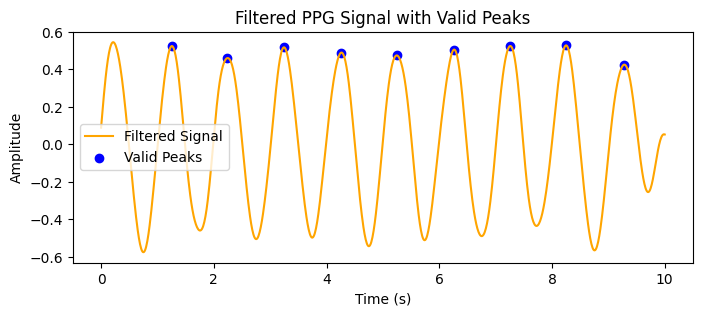
* The PPG signal is normalized to a scale of 0 to 1 to enhance further analysis.



**Figure 04: Normalized Signal**

1. **Peak Detection and Heart Rate Estimation:**

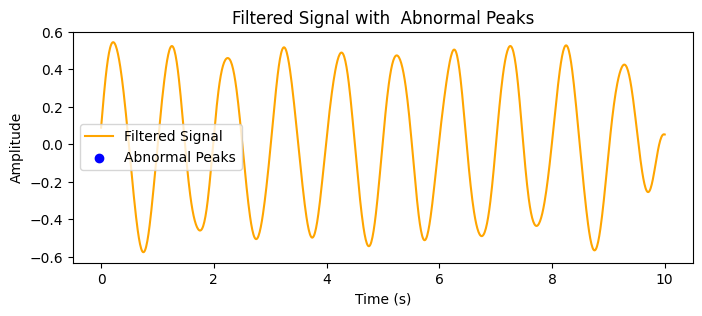
* Peaks are detected using **NeuroKit2’s** PPG find peaks function.
* The **valid peaks** are filtered based on amplitude thresholds.
* **Heart Rate (BPM)** is estimated using the detected peaks



**Figure 05: Valid Peak Detect**

1. **Abnormality Detection:**

* Inter-peak intervals (IBI) are analyzed.
* Abnormal peaks are identified based on deviations from the mean inter-beat interval.
* The percentage of abnormal peaks is calculated to assess irregularities.



**Figure 06: Abnormal Peak**

1. **Feature Extraction:**

* Statistical features of inter-beat intervals:
* **Mean, Standard Deviation (SDNN), Skewness, Kurtosis**
* **Abnormality percentage** in the detected peaks.
* **Frequency-domain analysis using** FFT (Fast Fourier Transform) to identify the dominant frequency in the signal.

**Feature Extraction Results:**

Mean Interval (s): **1.0025**

Standard Deviation of Intervals (s): **0.017139136501002624**

Skewness of Intervals: **0.3538001921438391**

Kurtosis of Intervals: -**1.4223630602082342**

Abnormality Percentage (%): **0.0**

Dominant Frequency (Hz): **1.0**

Heart Rate (BPM): **59.85037406483791**

Respiratory Rate (breaths per minute): **14.962593516209477)**

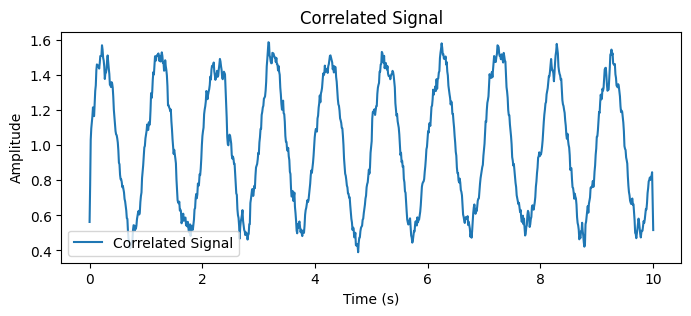
HRV Mean: **1.0025 seconds**

HRV SDNN: **0.017139136501002624 seconds**

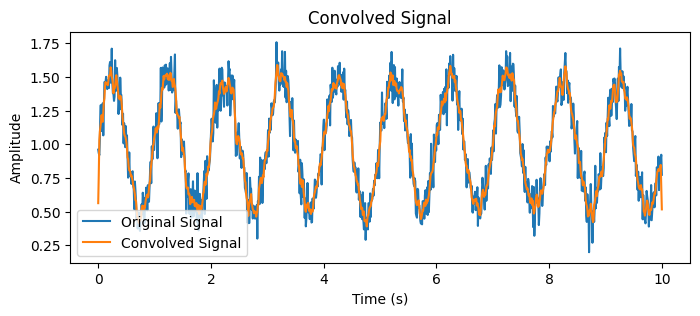
Signal Energy: **123.67222088363499**

1. **Signal Processing Techniques:**

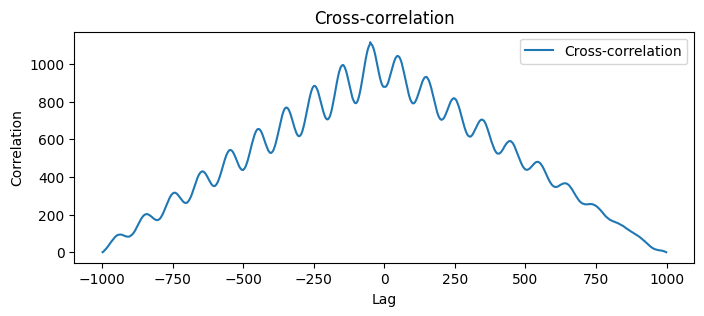
* **Correlation and convolution** are performed using a smoothing kernel to demonstrate signal processing techniques.
* **Cross-correlation** is computed to analyze the relationship between the original and delayed signals.
* **Auto-correlation** is computed to study the signal's self-similarity



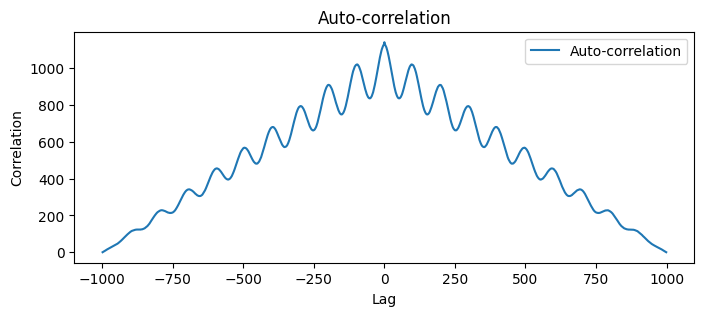
**Figure 07: Correlated Signal**

****

**Figure 08: Convolved Signal**

****

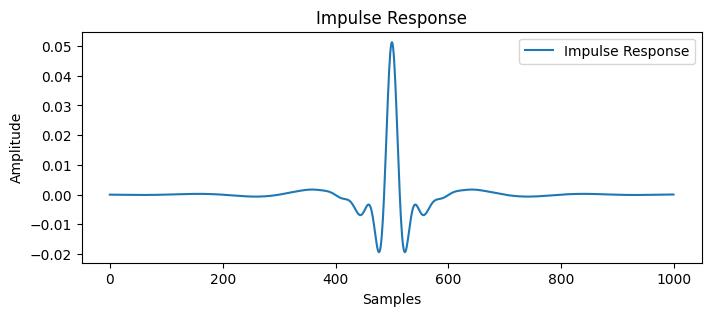
**Figure 09: Cross Correlation**

****

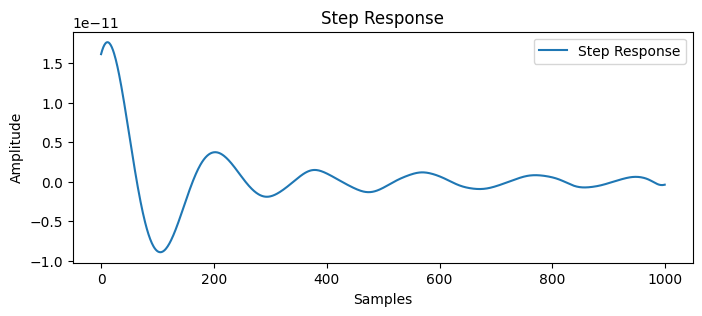
**Figure 10: Auto Cross Correlation**

1. **Filter Analysis:**

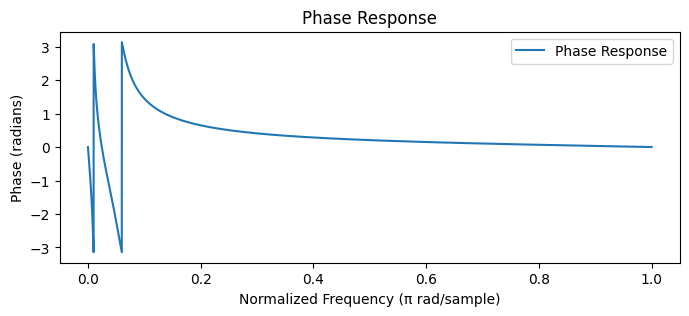
* The impulse response, step response, phase response, and magnitude response of the bandpass filter are computed and visualized.



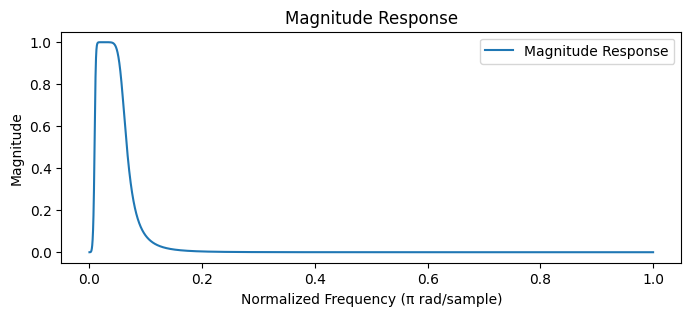
**Figure 11: Impulse Response**



**Figure 12: Step Response**



**Figure 13: Phase Response**



**Figure 14: Magnitude Response**

**Outcomes and Applications:**

* **Heart Rate (BPM) Estimation** – Helps in monitoring cardiovascular health.
* **Respiratory Rate Estimation** – Provides insights into breathing patterns.
* **Detection of Irregular Heartbeats** – Helps in identifying potential arrhythmias or heart abnormalities.
* **Feature Extraction for Machine Learning Models** – Can be used for automated health monitoring applications.
* **Signal Filtering Techniques Comparison** – Evaluates different noise reduction methods.
* **HRV Analysis** – Assesses autonomic nervous system activity and overall heart health.

**Source Code:**

import numpy as np

import neurokit2 as nk

import matplotlib.pyplot as plt

from scipy.signal import butter, filtfilt, correlate

from scipy.stats import skew, kurtosis, mode

# Generate the PPG Signal with Noise

# Generate a 10-second synthetic PPG signal with sinusoidal behavior and random noise.

fs = 100 # Sampling frequency (Hz)

t = np.linspace(0, 10, fs \* 10) # Time vector for 10 seconds

ppg\_signal = 1 + 0.5 \* np.sin(2 \* np.pi \* 1 \* t) + 0.1 \* np.random.randn(len(t)) # PPG signal with noise

# Plot the raw PPG signalplt.figure(figsize=(10, 4))

plt.plot(t, ppg\_signal, label="Raw PPG Signal")

plt.title("Raw PPG Signal with Noise")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.legend()

plt.show()

# Clean PPG signal (without noise)

ppg\_signal\_clean = 1 + 0.5 \* np.sin(2 \* np.pi \* 1 \* t) # PPG signal without noise

# Plot the clean PPG signal

plt.figure(figsize=(10, 4))

plt.plot(t, ppg\_signal\_clean, label="Raw PPG Signal", color='blue')

plt.title("Raw PPG Signal (Without Noise)")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.legend()

plt.grid()

plt.show()

from scipy.signal import butter, filtfilt

# Define bandpass filter

low\_cutoff = 0.5 # Lower cutoff frequency (Hz)

high\_cutoff = 3.0 # Upper cutoff frequency (Hz)

b, a = butter(4, [low\_cutoff / (fs / 2), high\_cutoff / (fs / 2)], btype='band')

# Apply the filter to the signal

filtered\_signal = filtfilt(b, a, ppg\_signal)

# Plot the filtered signal

plt.figure(figsize=(10, 4))

plt.subplot(121)

plt.plot(t, ppg\_signal)

plt.title("Orginal signal ")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.subplot(122)

plt.plot(t, filtered\_signal, color="orange")

plt.title("Noise-Reduced Using Bandpass Filter")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.legend()

plt.show()

# Define low-pass filter

cutoff = 3.0 # Cutoff frequency (Hz)

b, a = butter(4, cutoff / (fs / 2), btype='low')

# Apply the filter to the signal

low\_passed\_signal = filtfilt(b, a, ppg\_signal)

# Plot the low-pass filtered signal

plt.figure(figsize=(10, 4))

plt.subplot(121)

plt.plot(t, ppg\_signal)

plt.title("Orginal Signal ")

plt.xlabel("Time (s)")

plt.ylabel("Ampllidtue")

plt.subplot(122)

plt.plot(t, low\_passed\_signal, color="green")

plt.title("Noise-Reduced Using Low-pass Filter")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.legend()

plt.show()

# Normalize the PPG signal between 0 and 1.

normalized\_signal = (ppg\_signal - np.min(ppg\_signal)) / (np.max(ppg\_signal) - np.min(ppg\_signal)) # Normalization

# Plot the normalized signal

plt.figure(figsize=(10, 4))

plt.plot(t, normalized\_signal, label="Normalized Signal")

plt.title("Normalized Signal")

plt.xlabel("Time (s)")

plt.ylabel("Normalized Amplitude")

plt.legend()

plt.show()

# Detect peaks using nk.ppg\_findpeaks

peaks = nk.ppg\_findpeaks(filtered\_signal, sampling\_rate=fs)["PPG\_Peaks"]

# Filter valid peaks based on amplitude thresholds

valid\_peaks = peaks[(filtered\_signal[peaks] > 0.2) & (filtered\_signal[peaks] < 1.8)] # Valid peaks

# Plot the filtered signal with valid peaks

plt.figure(figsize=(10, 4))

plt.plot(t, filtered\_signal, label="Filtered Signal", color='orange')

plt.scatter(t[valid\_peaks], filtered\_signal[valid\_peaks], color='blue', label="Valid Peaks")

plt.title("Filtered PPG Signal with Valid Peaks")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.legend()

plt.show()

# Detect irregular peak intervals

inter\_peak\_intervals = np.diff(valid\_peaks) / fs # Intervals in seconds

mean\_ibi = np.mean(inter\_peak\_intervals)

std\_ibi = np.std(inter\_peak\_intervals)

# Define thresholds for abnormal intervals (e.g., mean ± 2\*std)

lower\_threshold = mean\_ibi - 2 \* std\_ibi

upper\_threshold = mean\_ibi + 2 \* std\_ibi

abnormal\_intervals = (inter\_peak\_intervals < lower\_threshold) | (inter\_peak\_intervals > upper\_threshold)

# Find indices of abnormal intervals

abnormal\_peaks = valid\_peaks[1:][abnormal\_intervals]

# Plot detected abnormalities on the filtered signal

plt.figure(figsize=(10, 4))

plt.plot(t, filtered\_signal, label="Filtered Signal", color='orange')

# plt.scatter(t[valid\_peaks], filtered\_signal[valid\_peaks], color='green', label="Valid Peaks")

plt.scatter(t[abnormal\_peaks], filtered\_signal[abnormal\_peaks], color='blue', label="Abnormal Peaks")

plt.title("Filtered Signal with Abnormal Peaks")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.legend()

plt.show()

# Feature 1: Statistical features of inter-peak intervals

mean\_interval = np.mean(inter\_peak\_intervals)

std\_interval = np.std(inter\_peak\_intervals)

skewness\_interval = skew(inter\_peak\_intervals)

kurtosis\_interval = kurtosis(inter\_peak\_intervals)

# Feature 2: Abnormality percentage

abnormality\_percentage = len(abnormal\_peaks) / len(valid\_peaks) \* 100

# Feature 3: FFT-based frequency domain analysis

fft\_signal = np.fft.fft(filtered\_signal)

frequencies = np.fft.fftfreq(len(filtered\_signal), 1 / fs)

dominant\_frequency = frequencies[np.argmax(np.abs(fft\_signal[:len(fft\_signal) // 2]))]

# Print extracted features

print("Feature Extraction Results:")

print(f"Mean Interval (s): {mean\_interval}")

print(f"Standard Deviation of Intervals (s): {std\_interval}")

print(f"Skewness of Intervals: {skewness\_interval}")

print(f"Kurtosis of Intervals: {kurtosis\_interval}")

print(f"Abnormality Percentage (%): {abnormality\_percentage}")

print(f"Dominant Frequency (Hz): {dominant\_frequency}")

# Estimate heart rate and respiratory rate from valid peaks.

heart\_rate = 60 / np.mean(np.diff(valid\_peaks) / fs) # Calculate heart rate (BPM)

respiratory\_rate = heart\_rate / 4 # Approximate respiratory rate assuming 4:1 HR:RR ratio

# Print heart rate and respiratory rate

print(f"Heart Rate (BPM): {heart\_rate}")

print(f"Respiratory Rate (breaths per minute): {respiratory\_rate})")

# Perform correlation using the kernel on the PPG signal.

kernel = np.ones(5) / 5 # Smoothing kernel

correlated\_signal = np.correlate(ppg\_signal, kernel, mode='same') # Correlation

# Plot the correlated signal

plt.figure(figsize=(10, 4))

plt.plot(t, correlated\_signal, label="Correlated Signal")

plt.title("Correlated Signal")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.legend()

plt.show()

# Perform convolution on the PPG signal using a smoothing kernel.

convolved\_signal = np.convolve(ppg\_signal, kernel, mode='same') # Convolution

# Plot the convolved signal

plt.figure(figsize=(10, 4))

plt.plot(t, ppg\_signal, label="Original Signal")

plt.plot(t, convolved\_signal, label="Convolved Signal")

plt.title("Convolved Signal")

plt.xlabel("Time (s)")

plt.ylabel("Amplitude")

plt.legend()

plt.show()

plt.show()

# Compute the cross-correlation between the original and delayed signals.

cross\_corr = correlate(ppg\_signal, delayed\_signal, mode='full') # Cross-correlation

lags = np.arange(-len(ppg\_signal) + 1, len(ppg\_signal)) # Lags for correlation

# Plot the cross-correlation

plt.figure(figsize=(10, 4))

plt.plot(lags, cross\_corr, label="Cross-correlation")

plt.title("Cross-correlation")

plt.xlabel("Lag")

plt.ylabel("Correlation")

plt.legend()

plt.show()

# Auto-correlation

# Compute the auto-correlation of the PPG signal.

auto\_corr = correlate(ppg\_signal, ppg\_signal, mode='full') # Auto-correlation

lags = np.arange(-len(ppg\_signal) + 1, len(ppg\_signal)) # Lags for correlation

# Plot the auto-correlation

plt.figure(figsize=(10, 4))

plt.plot(lags, auto\_corr, label="Auto-correlation")

plt.title("Auto-correlation")

plt.xlabel("Lag")

plt.ylabel("Correlation")

plt.legend()

plt.show()

# Design a bandpass filter and compute its impulse response.

impulse = np.zeros\_like(ppg\_signal) # Create an impulse signal

impulse[len(impulse) // 2] = 1 # Set the center to 1

b, a = butter(4, [0.5 / (fs / 2), 3.0 / (fs / 2)], btype='band') # Bandpass filter design

impulse\_response = filtfilt(b, a, impulse) # Compute impulse response

# Plot the impulse response

plt.figure(figsize=(10, 4))

plt.plot(impulse\_response, label="Impulse Response")

plt.title("Impulse Response")

plt.xlabel("Samples")

plt.ylabel("Amplitude")

plt.legend()

plt.show()

# Step Response

# Compute the step response of the bandpass filter.

step = np.ones\_like(ppg\_signal) # Create a step signal

step\_response = filtfilt(b, a, step) # Compute step response

# Plot the step response

plt.figure(figsize=(10, 4))

plt.plot(step\_response, label="Step Response")

plt.title("Step Response")

plt.xlabel("Samples")

plt.ylabel("Amplitude")

plt.legend()

plt.show()

# Phase Response

# Compute and plot the phase response of the filter.

from scipy.signal import freqz

w, h = freqz(b, a, worN=8000) # Frequency response

plt.figure(figsize=(10, 4))

plt.plot(w / np.pi, np.angle(h), label="Phase Response")

plt.title("Phase Response")

plt.xlabel("Normalized Frequency (π rad/sample)")

plt.ylabel("Phase (radians)")

plt.legend()

plt.show()

# Magnitude Response

# Compute and plot the magnitude response of the filter.

plt.figure(figsize=(10, 4))

plt.plot(w / np.pi, np.abs(h), label="Magnitude Response")

plt.title("Magnitude Response")

plt.xlabel("Normalized Frequency (π rad/sample)")

plt.ylabel("Magnitude")

plt.legend()

plt.show()

rr\_intervals = np.diff(valid\_peaks) / fs # RR intervals in seconds

hrv\_mean = np.mean(rr\_intervals)

hrv\_sdnn = np.std(rr\_intervals) # Standard deviation of RR intervals

print(f"HRV Mean: {hrv\_mean} seconds")

print(f"HRV SDNN: {hrv\_sdnn} seconds")

signal\_energy = np.sum(filtered\_signal\*\*2)

print(f"Signal Energy: {signal\_energy}")