PPG SIGNAL FEATURE EXTRACTION

```
 import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft
from scipy.signal import find_peaks
def extract_features(signal, fs):
"""
Extracts basic features from a signal.
Parameters:
signal (array-like): The 1D signal from which features will be extracted.
fs (int): Sampling frequency of the signal.
Returns:
dict: A dictionary with the extracted features.
"""
features = {}
# Time-domain features
features['mean'] = np.mean(signal)
features['std_dev'] = np.std(signal)
features['rms'] = np.sqrt(np.mean(np.square(signal)))
features['zero_crossings'] = np.count_nonzero(np.diff(np.sign(signal)))
features['energy'] = np.sum(np.square(signal))
# Frequency-domain features using FFT
n = len(signal)
freqs = np.fft.fftfreq(n, d=1/fs)
fft_values = fft(signal)
```

```
fft_magnitude = np.abs(fft_values)

# Spectral centroid (center of mass of the spectrum)

spectral_centroid = np.sum(freqs * fft_magnitude) / np.sum(fft_magnitude)

features['spectral_centroid'] = spectral_centroid

# Spectral bandwidth (spread of the spectrum)

spectral_bandwidth = np.sqrt(np.sum(((freqs - spectral_centroid)**2) * fft_magnitude) /
np.sum(fft_magnitude))

features['spectral_bandwidth'] = spectral_bandwidth

# Spectral entropy (measuring complexity of the signal)

fft_magnitude_norm = fft_magnitude / np.sum(fft_magnitude)

spectral_entropy = -np.sum(fft_magnitude_norm * np.log2(fft_magnitude_norm + 1e-10)) # Adding
small value to avoid log(0)

features['spectral_entropy'] = spectral_entropy

return features

def plot_signal_and_fft(signal, fs):

"""

Plots the time-domain signal and its FFT (frequency-domain).

Parameters:

signal (array-like): The 1D signal to plot.

fs (int): Sampling frequency of the signal.

"""

n = len(signal)

t = np.arange(0, n) / fs

# Plot the time-domain signal

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)

plt.plot(t, signal)

plt.title("Time Domain Signal")

plt.xlabel("Time [s]")
```

```
plt.ylabel("Amplitude")
# Plot the frequency-domain (FFT)
freqs = np.fft.fftfreq(n, d=1/fs)
fft_values = fft(signal)
fft_magnitude = np.abs(fft_values)
plt.subplot(1, 2, 2)
plt.plot(freqs[:n // 2], fft_magnitude[:n // 2]) # Only plot the positive frequencies
plt.title("Frequency Domain (FFT)")
plt.xlabel("Frequency [Hz]")
plt.ylabel("Magnitude")
plt.tight_layout()
plt.show()
# Example usage:
if __name__ == "__main__":
# Generate a sample signal (e.g., sine wave with noise)
fs = 1000 # Sampling frequency in Hz
t = np.arange(0, 1, 1/fs) # Time vector (1 second duration)
signal = np.sin(2 * np.pi * 50 * t) + 0.5 * np.random.randn(len(t)) # 50 Hz sine wave with noise
# Extract features from the signal
features = extract_features(signal, fs)
# Print the extracted features
print("Extracted Features:")
for key, value in features.items():
print(f"{key}: {value}")
# Plot the signal and its FFT
plot_signal_and_fft(signal, fs)
Extracted Features:
mean: -0.03138678114215478
std_dev: 0.880290695604582
```

rms: 0.880850066014906

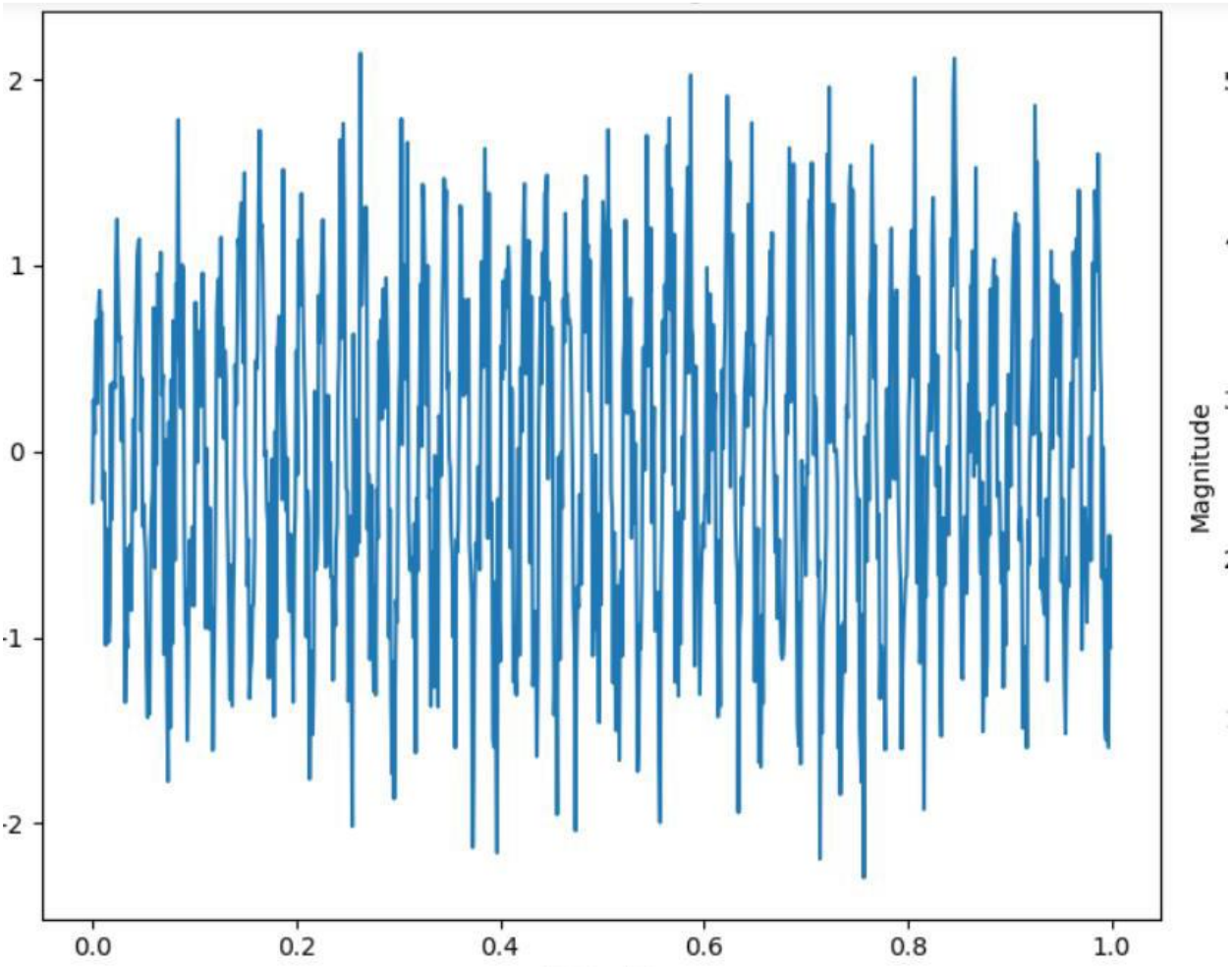
zero_crossings: 206

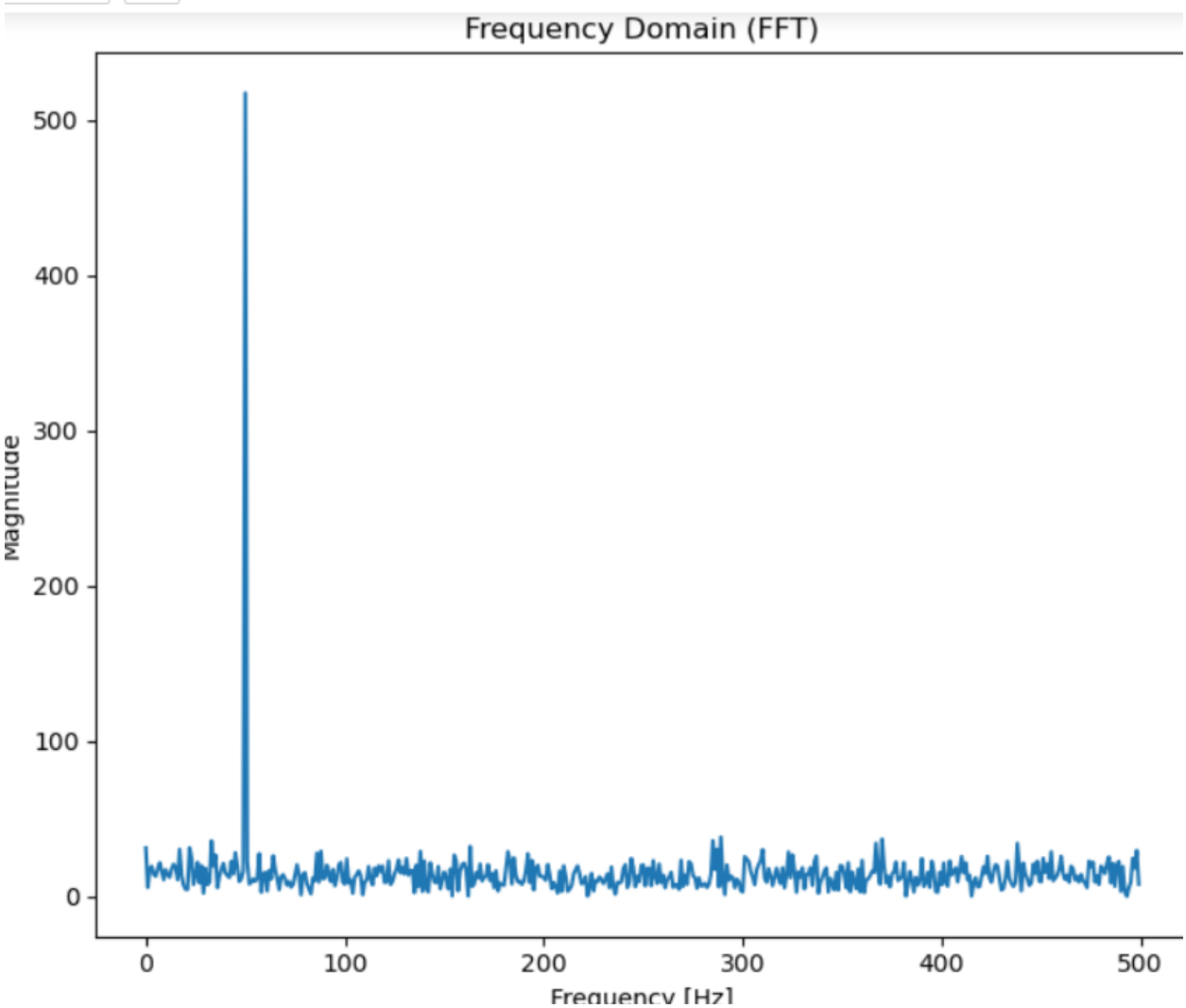
energy: 775.8968387984643

spectral_centroid: -0.3118651195886744

spectral_bandwidth: 280.02264836961155

spectral_entropy: 9.512349362246521



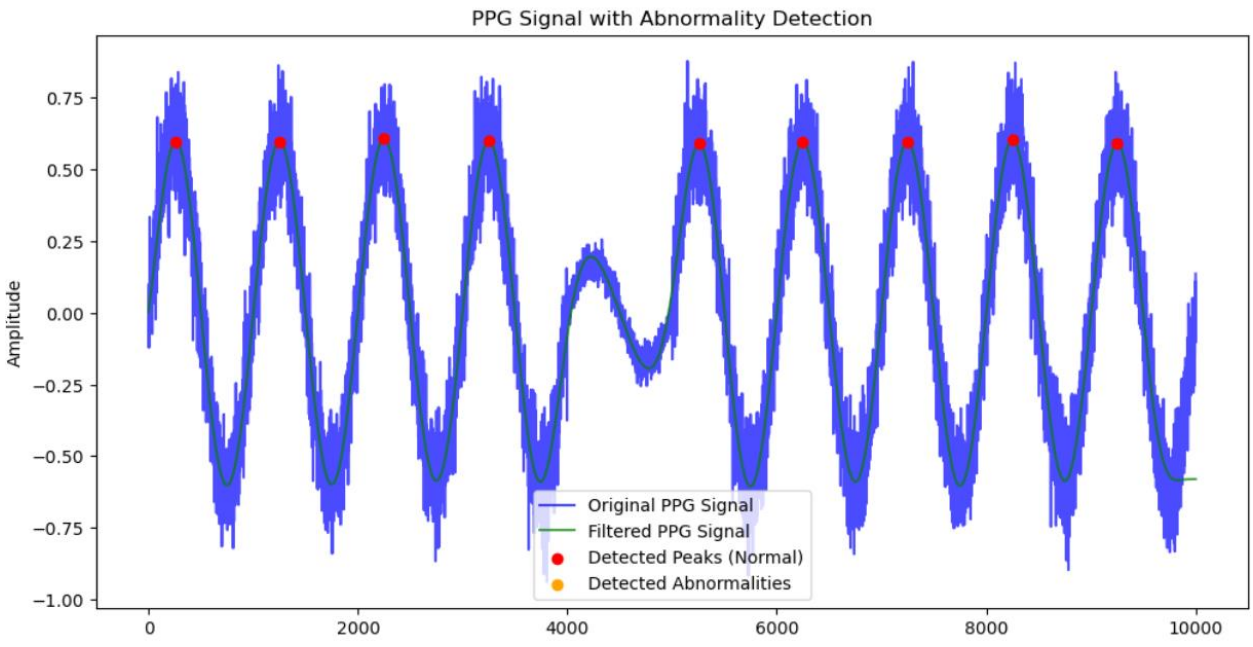


PPG SIGNAL ABNORAMILY DETECTION

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks, butter, filtfilt
```

```
from scipy.stats import zscore

# Low-pass filter to remove noise

def butter_lowpass(cutoff, fs, order=4):

nyquist = 0.5 * fs

normal_cutoff = cutoff / nyquist

b, a = butter(order, normal_cutoff, btype='low', analog=False)

return b, a

def butter_lowpass_filter(data, cutoff, fs, order=4):

b, a = butter_lowpass(cutoff, fs, order)

y = filtfilt(b, a, data)

return y

# Detect abnormality based on outlier detection (Z-score)

def detect_abnormality_ppg(ppg_signal, fs, peak_threshold=0.3, zscore_threshold=3.0):

# Step 1: Preprocess the PPG signal using a low-pass filter

cutoff_freq = 2.0 # Cutoff frequency in Hz (suitable for PPG signals)

filtered_signal = butter_lowpass_filter(ppg_signal, cutoff_freq, fs)

# Step 2: Detect peaks (heartbeats) in the filtered signal

peaks, _ = find_peaks(filtered_signal, height=peak_threshold, distance=int(fs*0.4)) # distance=~0.4s for
PPG peaks

# Step 3: Analyze peak intervals (time between beats)

peak_intervals = np.diff(peaks) / fs # Convert intervals to seconds

# Check for abnormal intervals (outliers in peak intervals)

interval_zscore = zscore(peak_intervals)

abnormal_intervals = np.where(np.abs(interval_zscore) > zscore_threshold)[0]

# Step 4: Check for outliers in the signal amplitude using Z-score

signal_zscore = zscore(filtered_signal)

abnormal_amplitudes = np.where(np.abs(signal_zscore) > zscore_threshold)[0]

# Step 5: Detect abnormal patterns

abnormal_peaks = np.concatenate([peaks[abnormal_intervals], abnormal_amplitudes])
```

```
abnormal_peaks = np.unique(abnormal_peaks) # Remove duplicates

# Plot the results

plt.figure(figsize=(12, 6))

plt.plot(ppg_signal, label='Original PPG Signal', color='blue', alpha=0.7)

plt.plot(filtered_signal, label='Filtered PPG Signal', color='green', alpha=0.7)

plt.scatter(peaks, filtered_signal[peaks], color='red', label='Detected Peaks (Normal)', zorder=5)

plt.scatter(abnormal_peaks, filtered_signal[abnormal_peaks], color='orange', label='Detected
Abnormalities', zorder=5)

plt.title("PPG Signal with Abnormality Detection")

plt.xlabel("Time (samples)")

plt.ylabel("Amplitude")

plt.legend()

plt.show()

return abnormal_peaks

# Example usage:

if __name__ == "__main__":

# Simulated PPG signal with some abnormality

fs = 1000 # Sampling frequency (Hz)

t = np.arange(0, 10, 1/fs) # Time vector (10 seconds)

# Generate a normal PPG signal (a sine wave with noise)

normal_ppg = 0.6 * np.sin(2 * np.pi * 1.0 * t) + 0.1 * np.random.randn(len(t)) # 1 Hz heartbeat, noise
added

abnormal_ppg = np.copy(normal_ppg)

# Simulate an abnormality (drop in amplitude at around 4 seconds)

abnormal_ppg[int(4*fs):int(5*fs)] = abnormal_ppg[int(4*fs):int(5*fs)] * 0.3 # Amplitude drop

# Detect abnormalities in the PPG signal

abnormal_peaks = detect_abnormality_ppg(abnormal_ppg, fs)
```


PPG Signal with Abnormality Detection
Amplitude
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2000
4000
6000
8000
10000
Original PPG Signal
Filtered PPG Signal
Detected Peaks (Normal)
Detected Abnormalities