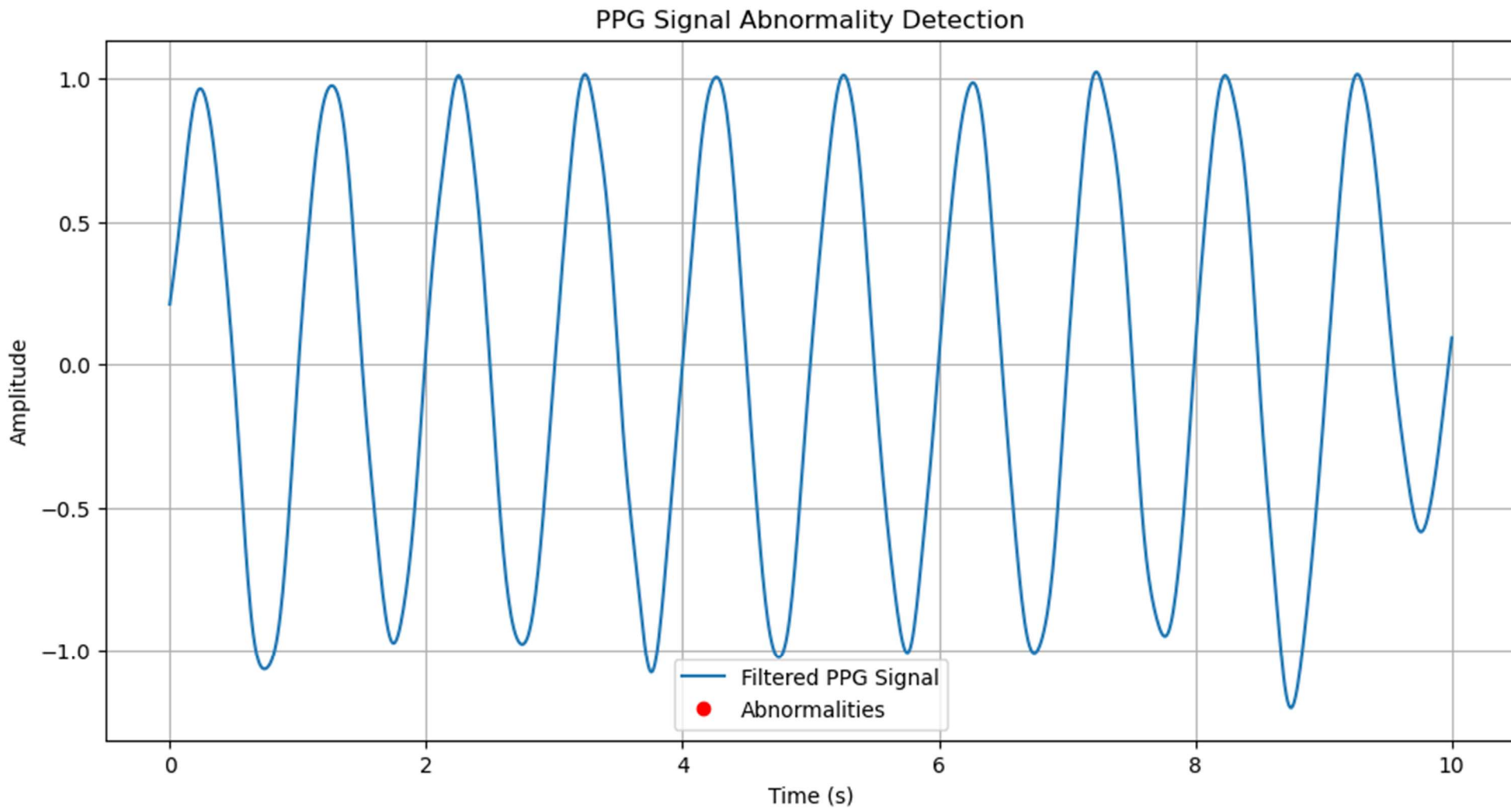
In [1]:

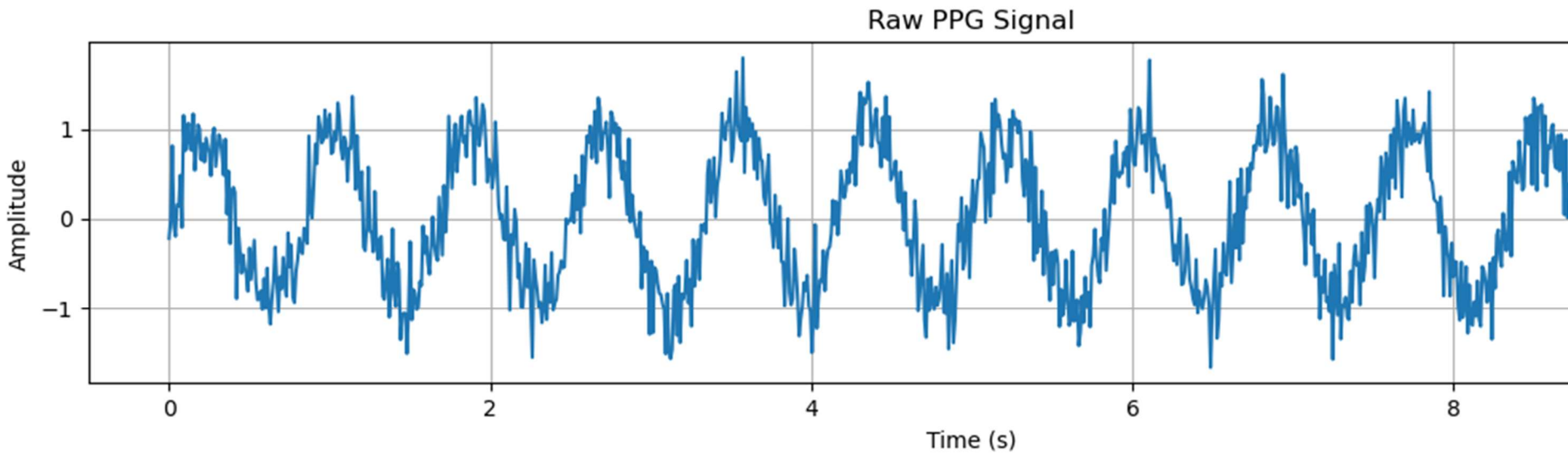
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt

# Simulate or load a PPG signal
# Replace this with your PPG signal data
fs = 100  # Sampling frequency (Hz)
t = np.linspace(0, 10, fs * 10)  # 10 seconds of data
ppg_signal = np.sin(2 * np.pi * 1 * t) + 0.1 * np.random.randn(len(t))  # Simulated signal

# Define a function for bandpass filtering
def bandpass_filter(data, lowcut, highcut, fs, order=4):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    y = filtfilt(b, a, data)
    return y

# Filter the PPG signal
filtered_signal = bandpass_filter(ppg_signal, lowcut=0.5, highcut=5, fs=fs)

# Detect abnormality (e.g., sudden spikes or drops)
def detect_abnormality(signal, threshold=3):
    mean = np.mean(signal)
    std_dev = np.std(signal)
    abnormalities = np.abs(signal - mean) > threshold * std_dev
    return abnormalities

# Find abnormalities
abnormalities = detect_abnormality(filtered_signal)

# Plot the results
plt.figure(figsize=(12, 6))
plt.plot(t, filtered_signal, label='Filtered PPG Signal')
plt.plot(t[abnormalities], filtered_signal[abnormalities], 'ro', label='Abnormalities')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('PPG Signal Abnormality Detection')
plt.legend()
plt.grid()
plt.show()
```

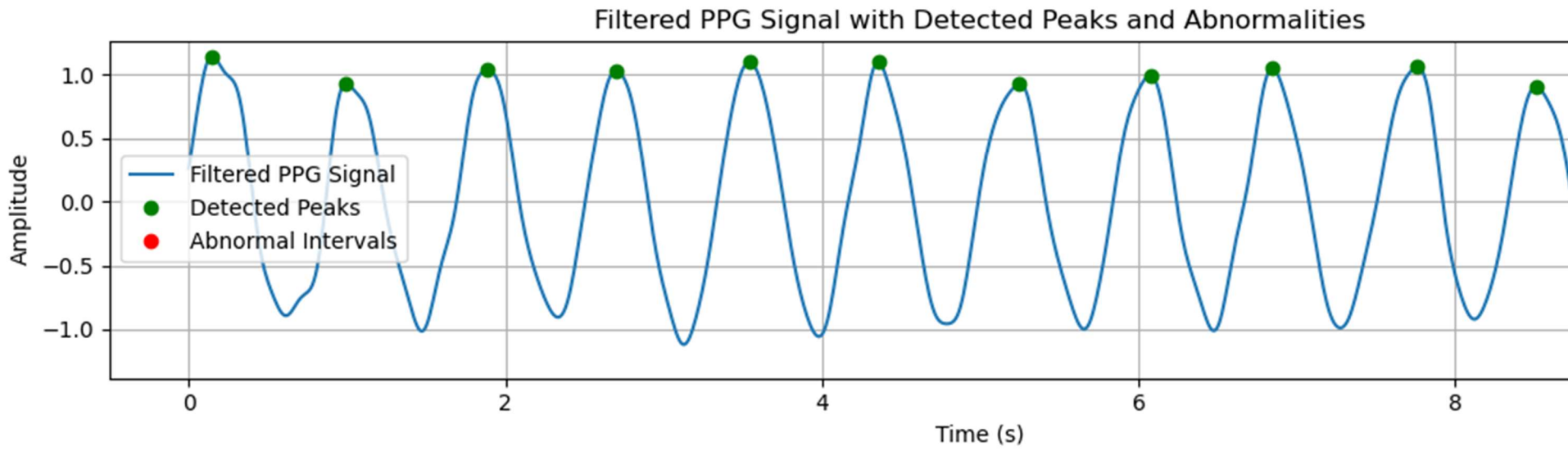


In [2]:

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

# Define a bandpass filter function
def bandpass_filter(data, lowcut, highcut, fs, order=4):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    y = filtfilt(b, a, data)
    return y

# Define a function to detect abnormalities
def detect_abnormalities(signal, fs, threshold_factor=3):
    # Extract heartbeats using peak detection
    peaks, _ = find_peaks(signal, distance=fs//2)  # Assuming heart rate < 120 bpm
    peak_intervals = np.diff(peaks) / fs  # Convert to seconds (time between peaks)

    # Detect abnormal intervals based on deviation from the mean
    mean_interval = np.mean(peak_intervals)
    std_interval = np.std(peak_intervals)
    abnormalities = (np.abs(peak_intervals - mean_interval) > threshold_factor * std_interval)

    abnormal_indices = np.where(abnormalities)[0]
    return peaks, abnormal_indices, peak_intervals

# Simulate or load raw PPG data
fs = 100  # Sampling frequency (Hz)
```

```
t = np.linspace(0, 10, fs * 10)  # 10 seconds of data
raw_ppg_signal = np.sin(2 * np.pi * 1.2 * t) + 0.3 * np.random.randn(len(t))  # Simulated PPG
signal

# Preprocess the signal (filtering)
filtered_ppg_signal = bandpass_filter(raw_ppg_signal, lowcut=0.5, highcut=5, fs=fs)

# Detect abnormalities
peaks, abnormal_indices, peak_intervals = detect_abnormalities(filtered_ppg_signal, fs)

# Plot the raw and filtered signal
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(t, raw_ppg_signal, label='Raw PPG Signal')
plt.title('Raw PPG Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(t, filtered_ppg_signal, label='Filtered PPG Signal')
plt.plot(t[peaks], filtered_ppg_signal[peaks], 'go', label='Detected Peaks')
abnormal_times = t[peaks[:-1]][abnormal_indices]
plt.plot(abnormal_times, filtered_ppg_signal[peaks[:-1][abnormal_indices]], 'ro', label='Abnormal
Intervals')
plt.title('Filtered PPG Signal with Detected Peaks and Abnormalities')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()

# Print the abnormal intervals
for i, idx in enumerate(abnormal_indices):
    print(f"Abnormal Interval {i+1}: Interval = {peak_intervals[idx]:.2f}s, Index = {idx}")
```

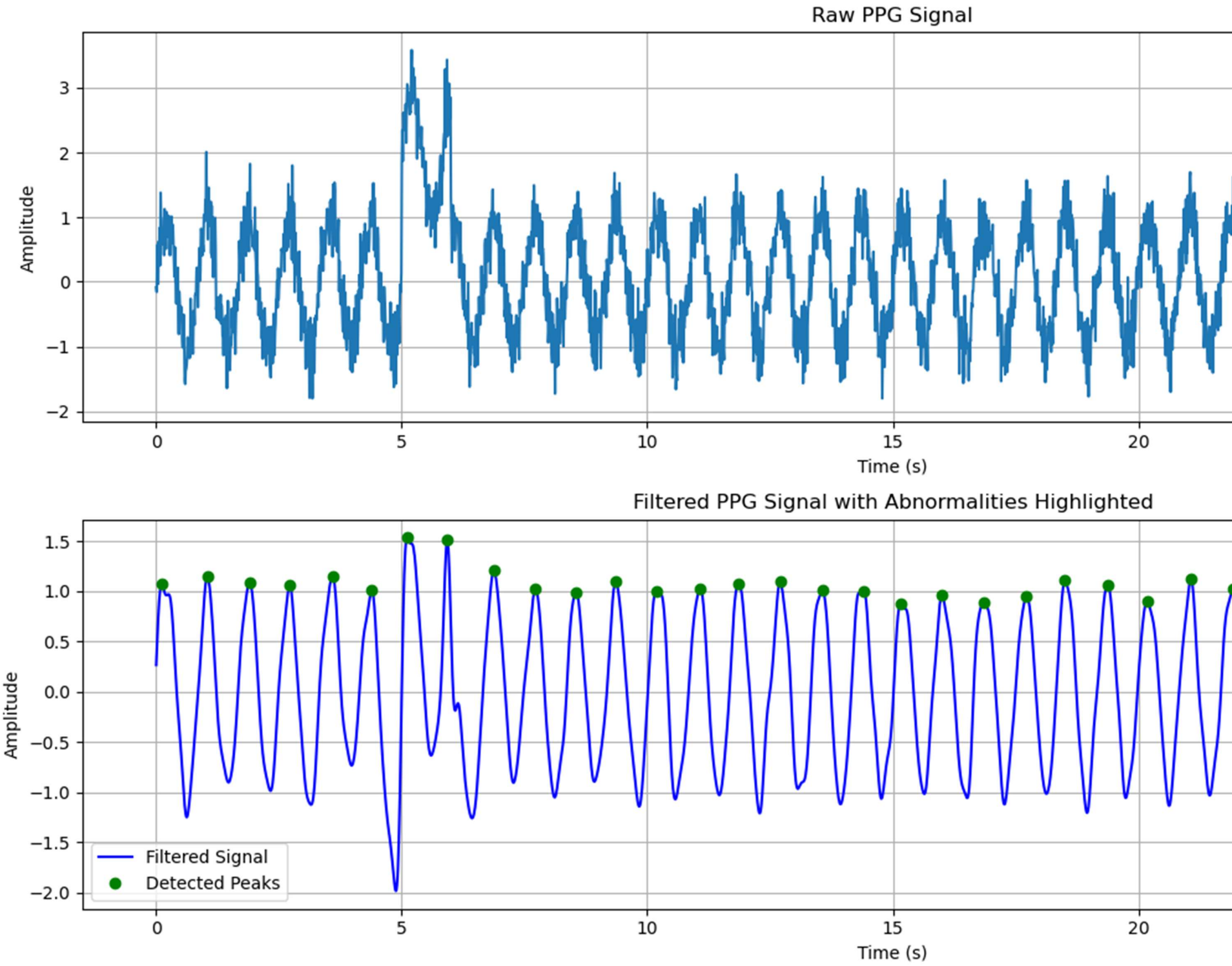




In [3]:

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

# Define a bandpass filter
def bandpass_filter(data, lowcut, highcut, fs, order=4):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    return filtfilt(b, a, data)

# Detect abnormalities
def detect_abnormalities(signal, fs, threshold_factor=3):
    # Find peaks
    peaks, _ = find_peaks(signal, distance=fs//2)  # Assuming heart rate < 120 bpm
    peak_intervals = np.diff(peaks) / fs  # Convert intervals to seconds

    # Mean and standard deviation of intervals
    mean_interval = np.mean(peak_intervals)
    std_interval = np.std(peak_intervals)

    # Identify abnormal intervals
    abnormalities = (np.abs(peak_intervals - mean_interval) > threshold_factor * std_interval)
    abnormal_indices = np.where(abnormalities)[0]

    return peaks, peak_intervals, abnormal_indices
```

```
# Simulate or load raw PPG signal
fs = 100  # Sampling frequency in Hz
t = np.linspace(0, 30, fs * 30)  # 30 seconds of data
raw_signal = np.sin(2 * np.pi * 1.2 * t) + 0.3 * np.random.randn(len(t))  # Simulated PPG signal
raw_signal[500:600] += 2  # Add an artificial abnormality for testing

# Preprocess the signal
filtered_signal = bandpass_filter(raw_signal, lowcut=0.5, highcut=5, fs=fs)

# Detect abnormalities
peaks, peak_intervals, abnormal_indices = detect_abnormalities(filtered_signal, fs)

# Plot the PPG signal and mark abnormalities
plt.figure(figsize=(14, 8))

# Plot raw and filtered signals
plt.subplot(2, 1, 1)
plt.plot(t, raw_signal, label='Raw Signal')
plt.title('Raw PPG Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(t, filtered_signal, label='Filtered Signal', color='blue')
plt.plot(t[peaks], filtered_signal[peaks], 'go', label='Detected Peaks')
for idx in abnormal_indices:
    start = peaks[idx]
    end = peaks[idx + 1]
    plt.axvspan(t[start], t[end], color='red', alpha=0.3, label='Abnormality' if idx == 0 else "")
plt.title('Filtered PPG Signal with Abnormalities Highlighted')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()

# Print abnormal intervals
print("Abnormal Intervals Detected:")
for i, idx in enumerate(abnormal_indices):
    print(f"Interval {i+1}: Start Time = {t[peaks[idx]]:.2f}s, End Time = {t[peaks[idx+1]]:.2f}s,
Duration = {peak_intervals[idx]:.2f}s")
```

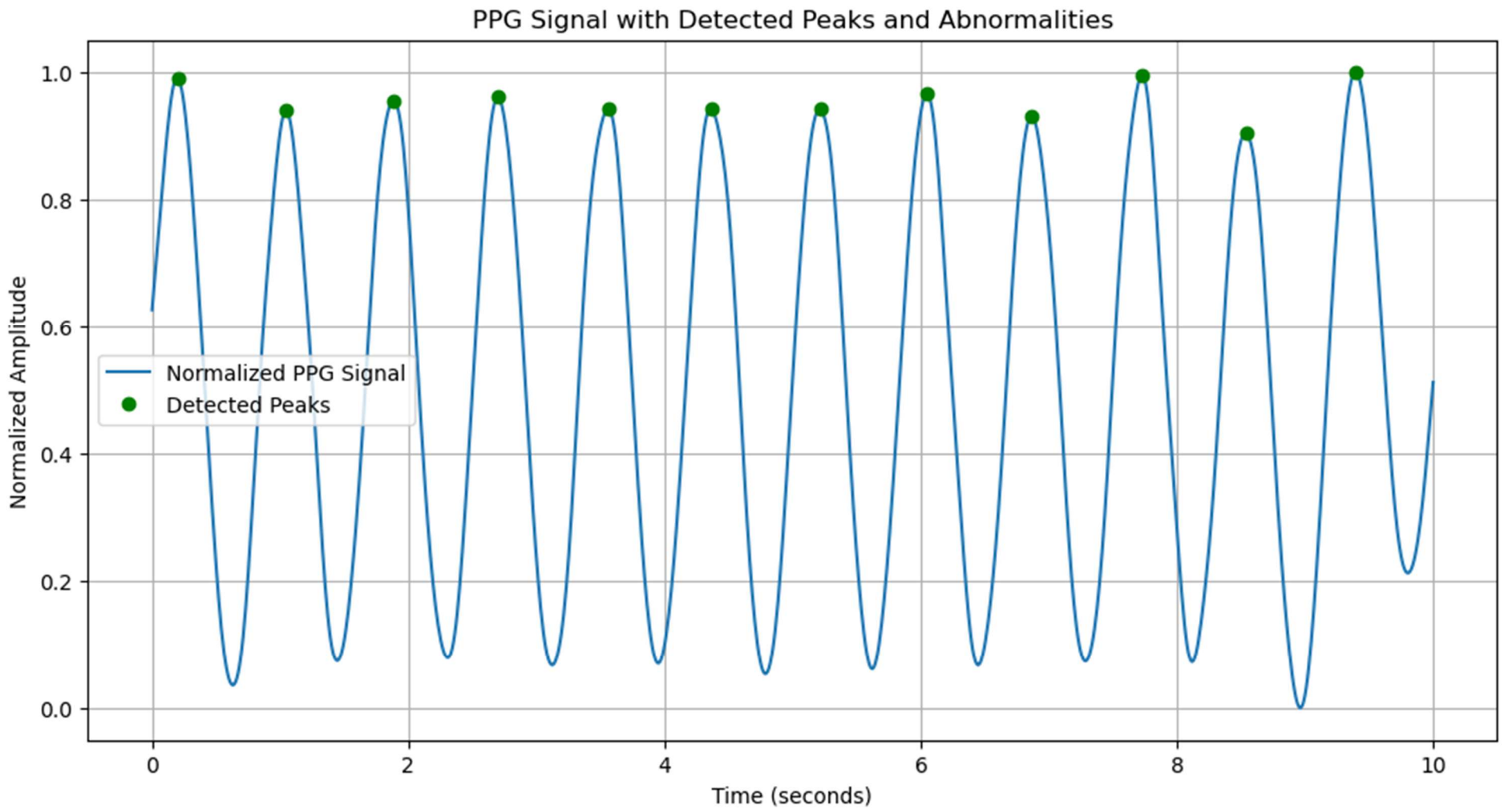


Abnormal Intervals Detected:

In [4]:

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

# Bandpass filter design (0.5 to 5 Hz for heart rate detection)
def bandpass_filter(signal, lowcut, highcut, fs, order=4):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    return filtfilt(b, a, signal)

# Simulating a PPG signal (replace with actual data)
fs = 100  # Sampling rate (Hz)
t = np.linspace(0, 10, fs * 10)  # 10 seconds of data
ppg_signal = 0.6 * np.sin(2 * np.pi * 1.2 * t) + np.random.normal(0, 0.05, len(t))

# Filter the PPG signal
filtered_ppg = bandpass_filter(ppg_signal, 0.5, 5, fs)
```

```
# Normalize the filtered signal
normalized_ppg = (filtered_ppg - np.min(filtered_ppg)) / (np.max(filtered_ppg) -
np.min(filtered_ppg))

# Detect peaks in the PPG signal
peaks, _ = find_peaks(normalized_ppg, distance=fs*0.6)  # Minimum distance of 0.6 seconds
between peaks (HR < 100 BPM)

# Calculate Heart Rate (BPM) and detect abnormalities
ibi = np.diff(peaks) / fs  # Inter-beat interval in seconds
heart_rate = 60 / ibi  # Convert to beats per minute (BPM)
mean_hr = np.mean(heart_rate)
std_hr = np.std(heart_rate)
threshold = 2  # Abnormality threshold (in terms of standard deviation)

# Detect abnormalities
abnormal_indices = np.where(np.abs(heart_rate - mean_hr) > threshold * std_hr)[0]

# Plot the PPG signal with detected peaks and abnormalities
plt.figure(figsize=(12, 6))
plt.plot(t, normalized_ppg, label='Normalized PPG Signal')
plt.plot(t[peaks], normalized_ppg[peaks], 'go', label='Detected Peaks')
for idx in abnormal_indices:
    plt.plot(t[peaks[idx + 1]], normalized_ppg[peaks[idx + 1]], 'ro', label='Abnormality' if idx == 0
else "")
plt.title("PPG Signal with Detected Peaks and Abnormalities")
plt.xlabel("Time (seconds)")
plt.ylabel("Normalized Amplitude")
plt.legend()
plt.grid()
plt.show()

# Print heart rate values and abnormalities
print("Heart Rate Values (BPM):", heart_rate)
if len(abnormal_indices) > 0:
    print("Abnormal Heart Rates Detected:")
    for idx in abnormal_indices:
        print(f"Abnormal HR at Interval {idx + 1}: {heart_rate[idx]:.2f} BPM")
else:
    print("No abnormalities detected.")
```

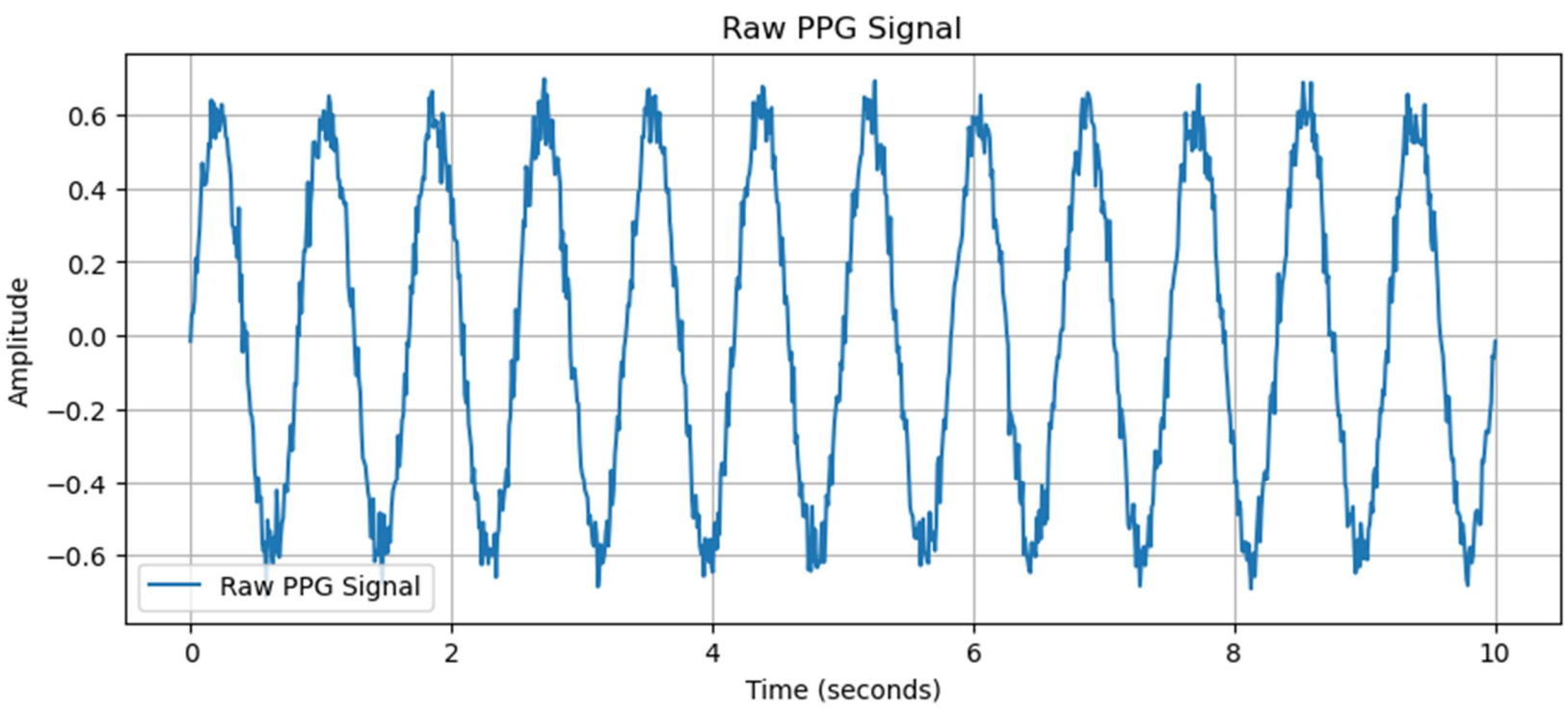


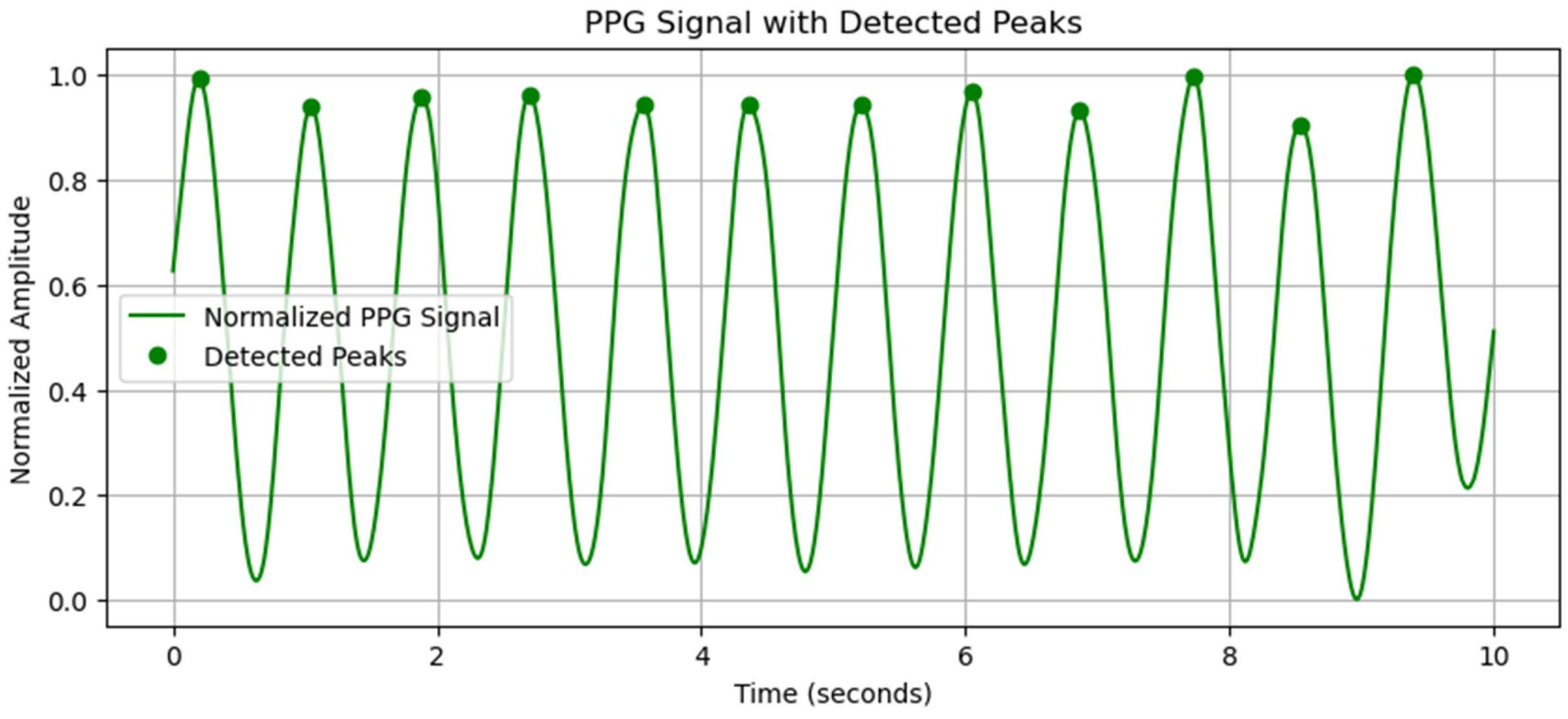
Heart Rate Values (BPM): [71.42857143 71.42857143 73.17073171 69.76744186 75.
70.58823529
 72.28915663 73.17073171 69.76744186 74.07407407 70.58823529]
No abnormalities detected.
In [5]:

```
import numpy as np
import matplotlib.pyplot as plt

# Simulating a PPG signal (replace with actual data)
fs = 100  # Sampling rate (Hz)
t = np.linspace(0, 10, fs * 10)  # 10 seconds of data
ppg_signal = 0.6 * np.sin(2 * np.pi * 1.2 * t) + np.random.normal(0, 0.05, len(t))

# Plot the raw PPG signal
plt.figure(figsize=(10, 4))
plt.plot(t, ppg_signal, label="Raw PPG Signal")
plt.title("Raw PPG Signal")
plt.xlabel("Time (seconds)")
plt.ylabel("Amplitude")
plt.legend()
plt.grid()
plt.show()
from scipy.signal import butter, filtfilt

# Bandpass filter design (0.5 to 5 Hz for heart rate detection)
def bandpass_filter(signal, lowcut, highcut, fs, order=4):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
```

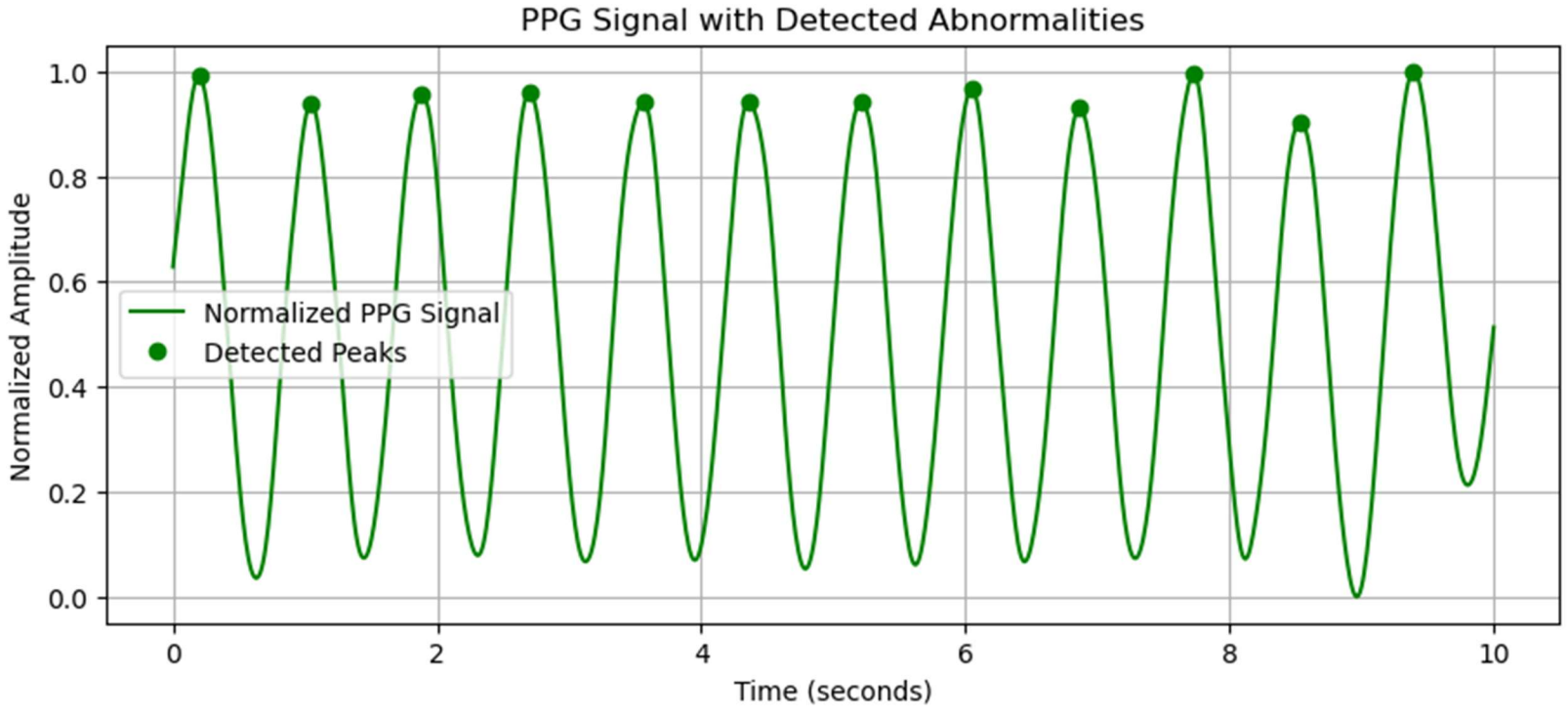
```
    return filtfilt(b, a, signal)

# Filter the PPG signal
filtered_ppg = bandpass_filter(ppg_signal, 0.5, 5, fs)

# Plot the filtered PPG signal
plt.figure(figsize=(10, 4))
plt.plot(t, filtered_ppg, label="Filtered PPG Signal", color="orange")
plt.title("Filtered PPG Signal")
plt.xlabel("Time (seconds)")
plt.ylabel("Amplitude")
plt.legend()
plt.grid()
plt.show()
from scipy.signal import find_peaks

# Detect peaks in the PPG signal
peaks, _ = find_peaks(normalized_ppg, distance=fs * 0.6)  # Minimum distance of 0.6 seconds
between peaks (HR < 100 BPM)

# Plot the PPG signal with detected peaks
plt.figure(figsize=(10, 4))
plt.plot(t, normalized_ppg, label="Normalized PPG Signal", color="green")
plt.plot(t[peaks], normalized_ppg[peaks], 'go', label="Detected Peaks")
plt.title("PPG Signal with Detected Peaks")
plt.xlabel("Time (seconds)")
plt.ylabel("Normalized Amplitude")
plt.legend()
plt.grid()
plt.show()
# Calculate inter-beat intervals (IBIs) and heart rate
ibi = np.diff(peaks) / fs  # Inter-beat interval in seconds
heart_rate = 60 / ibi  # Convert to beats per minute (BPM)

# Define threshold for abnormalities (e.g., 2 standard deviations from the mean)
mean_hr = np.mean(heart_rate)
std_hr = np.std(heart_rate)
threshold = 2  # Threshold multiplier for standard deviation

# Detect abnormal heart rates
abnormal_indices = np.where(np.abs(heart_rate - mean_hr) > threshold * std_hr)[0]

# Plot the PPG signal with abnormalities highlighted
plt.figure(figsize=(10, 4))
plt.plot(t, normalized_ppg, label="Normalized PPG Signal", color="green")
plt.plot(t[peaks], normalized_ppg[peaks], 'go', label="Detected Peaks")
for idx in abnormal_indices:
```

```
    plt.plot(t[peaks[idx + 1]], normalized_ppg[peaks[idx + 1]], 'ro', label="Abnormality" if idx == 0
else "")
plt.title("PPG Signal with Detected Abnormalities")
plt.xlabel("Time (seconds)")
plt.ylabel("Normalized Amplitude")
plt.legend()
plt.grid()
plt.show()

# Print detected abnormalities
if len(abnormal_indices) > 0:
    print("Abnormal Heart Rates Detected:")
    for idx in abnormal_indices:
        print(f"Abnormal HR at Interval {idx + 1}: {heart_rate[idx]:.2f} BPM")
else:
    print("No abnormalities detected.")
```





PPG Signal with Detected Abnormalities

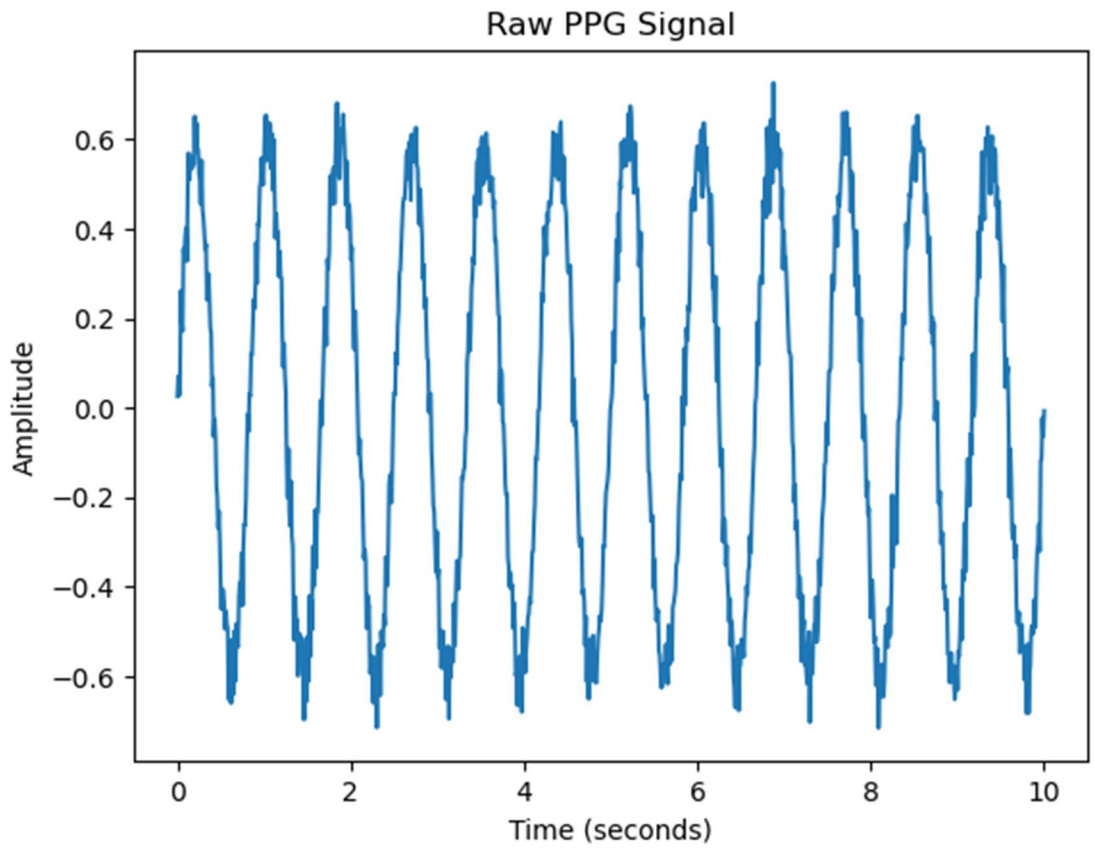


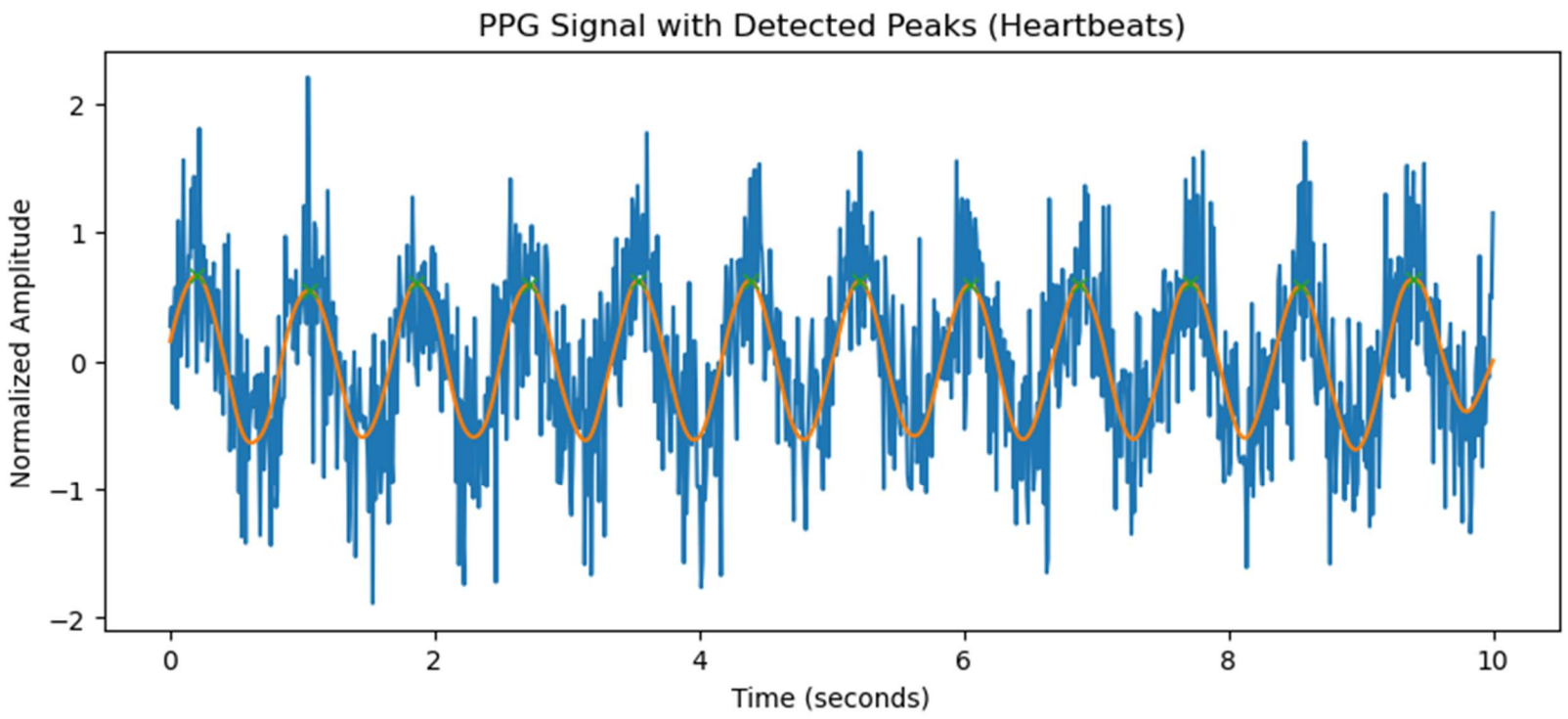
No abnormalities detected.

In [6]:

```
import numpy as np
import matplotlib.pyplot as plt
fs = 100
t = np.linspace(0, 10, fs * 10)
ppg_signal = 0.6 * np.sin(2 * np.pi * 1.2 * t) + np.random.normal(0, 0.05, len(t))
plt.plot(t,ppg_signal)
plt.title("Raw PPG Signal")
plt.xlabel("Time (seconds)")
plt.ylabel("Amplitude")
plt.show()
mean = 0
std = 0.5  # Standard deviation of the noise
noise = np.random.normal(mean, std, ppg_signal.shape)

# Add noise to the signal
noisy_signal = ppg_signal + noise

# Plot the noisy signal
plt.figure(figsize=(10, 4))
```

```
plt.plot(t, noisy_signal)
plt.title("Noisy Signal (ppg_signal + Gaussian Noise)")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
from scipy.signal import find_peaks

# Detect peaks in the PPG signal
peaks, _ = find_peaks(filtered_ppg, distance=fs*0.6)

# Calculate Heart Rate (BPM)
ibi = np.diff(peaks) / fs
heart_rate = 60 / ibi

# Plot the PPG signal with detected peaks
plt.plot(t, filtered_ppg)
plt.plot(t[peaks], filtered_ppg[peaks], "x")
plt.title("PPG Signal with Detected Peaks (Heartbeats)")
plt.xlabel("Time (seconds)")
plt.ylabel("Normalized Amplitude")
plt.show()

print("Heart Rate: ", np.mean(heart_rate), " BPM")
plt.subplot(3,1,1)
plt.plot(t,ppg_signal)
plt.subplot(3,1,2)
plt.plot(t, noisy_signal)
plt.subplot(3,1,3)
plt.plot(t, filtered_ppg)
```

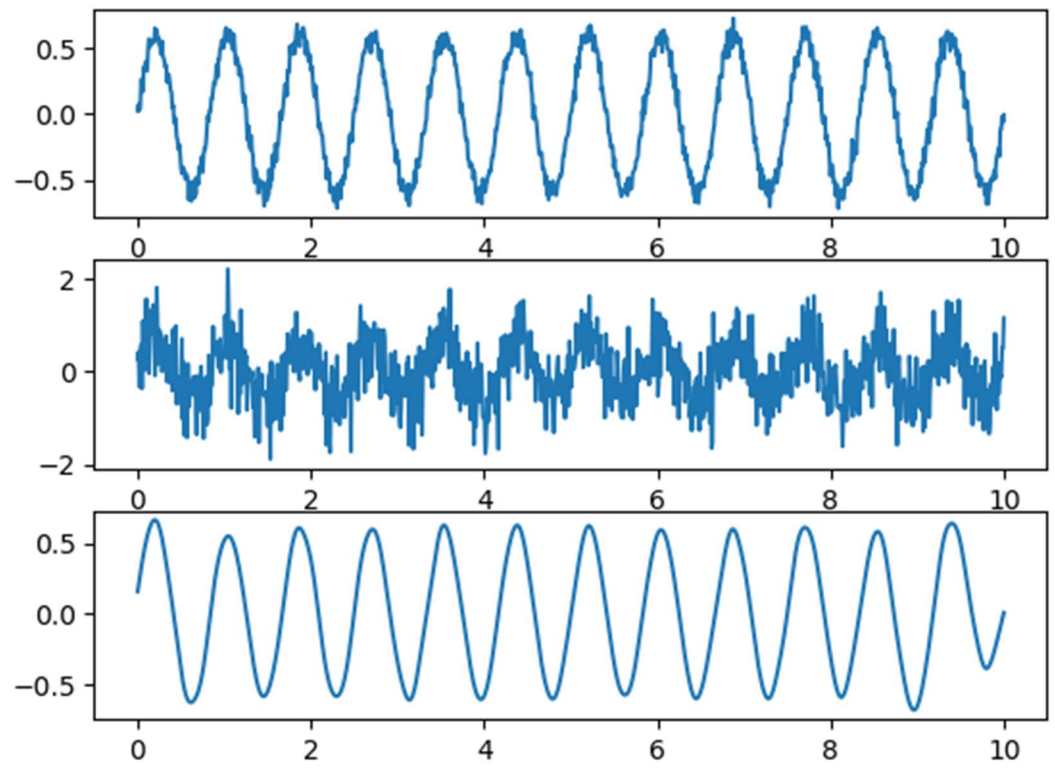




Heart Rate: 71.83862951438275 BPM

Out[6]:

[<matplotlib.lines.Line2D at 0x26bbcec8a70>]



In [2]:

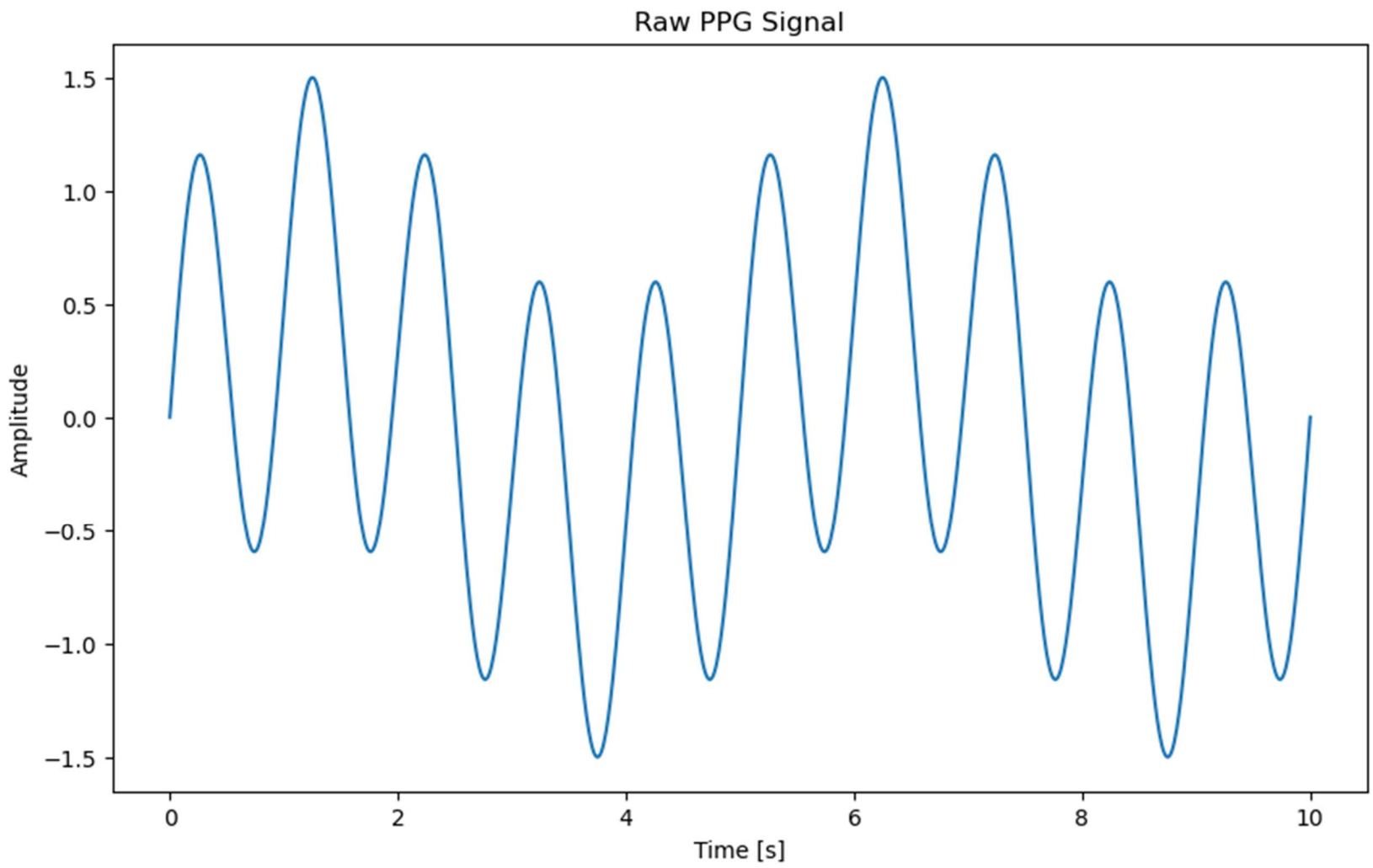
```
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt
```

In [3]:

```
# Simulate a PPG signal (this is just an example, replace with actual data)
time = np.linspace(0, 10, 1000)  # 10 seconds, 1000 samples
ppg_signal = np.sin(2 * np.pi * 1 * time) + 0.5 * np.sin(2 * np.pi * 0.2 * time)  # A synthetic PPG signal
```

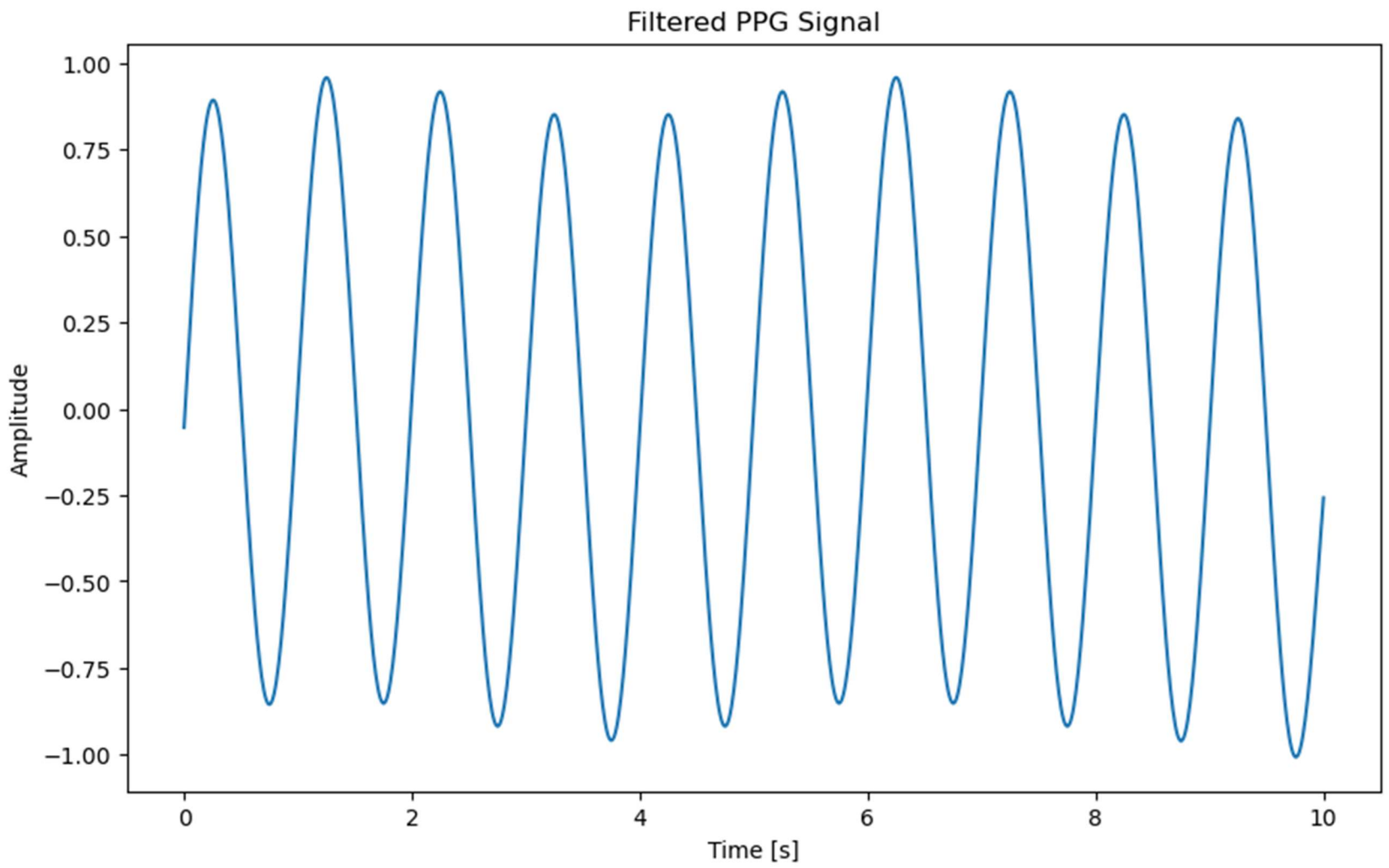
In [4]:

```
# Plot the raw signal
plt.figure(figsize=(10, 6))
plt.plot(time, ppg_signal)
plt.title('Raw PPG Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.show()
```



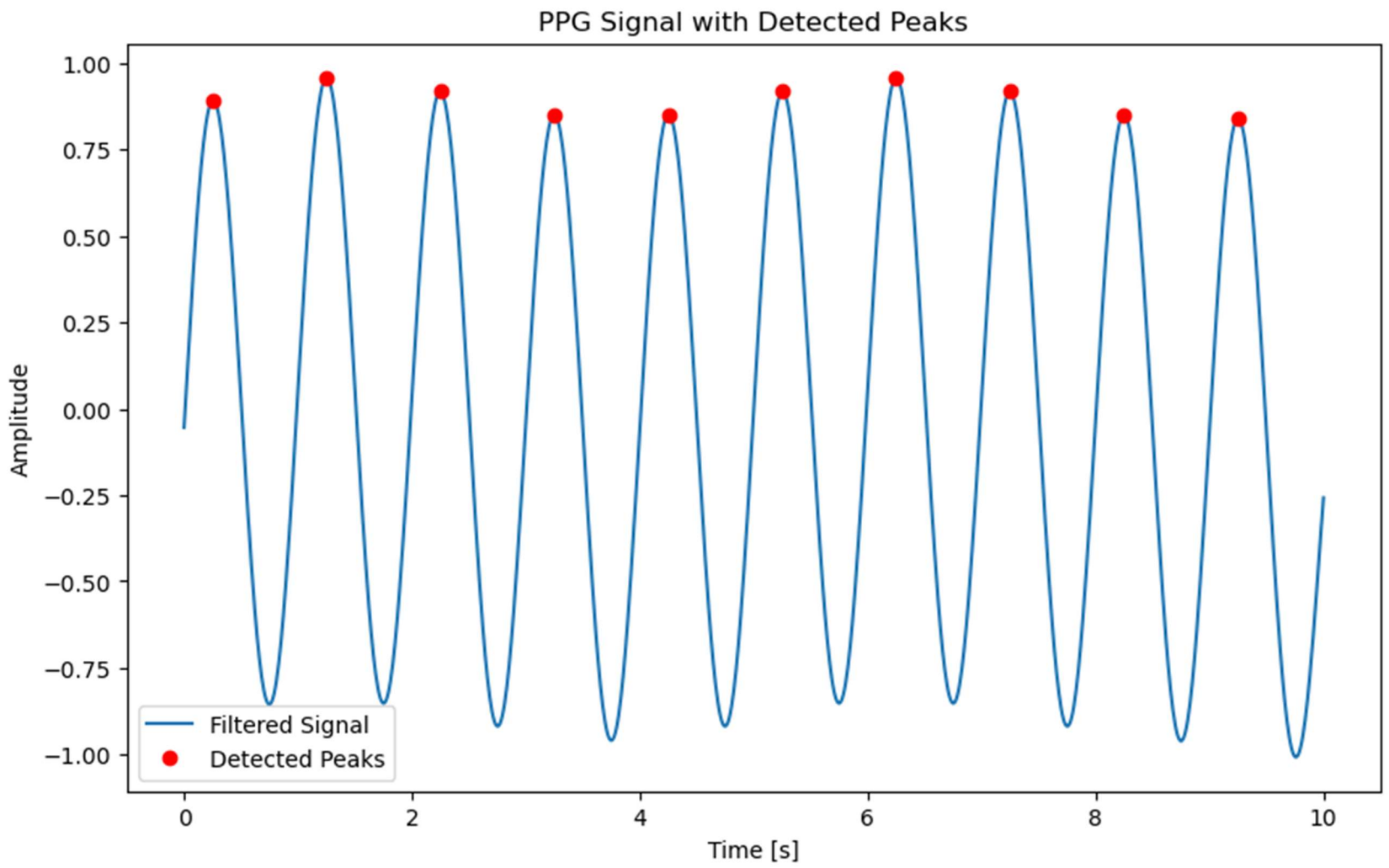
In [5]:

```
# Bandpass filter parameters (you can adjust them)
low_cutoff = 0.5  # Hz (low frequency cutoff)
high_cutoff = 5.0  # Hz (high frequency cutoff)
fs = 100  # Sampling rate (Hz)

# Create the bandpass filter
nyquist = 0.5 * fs
low = low_cutoff / nyquist
high = high_cutoff / nyquist
b, a = signal.butter(1, [low, high], btype='band')

# Apply the filter
filtered_signal = signal.filtfilt(b, a, ppg_signal)

# Plot the filtered signal
plt.figure(figsize=(10, 6))
plt.plot(time, filtered_signal)
plt.title('Filtered PPG Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.show()
```



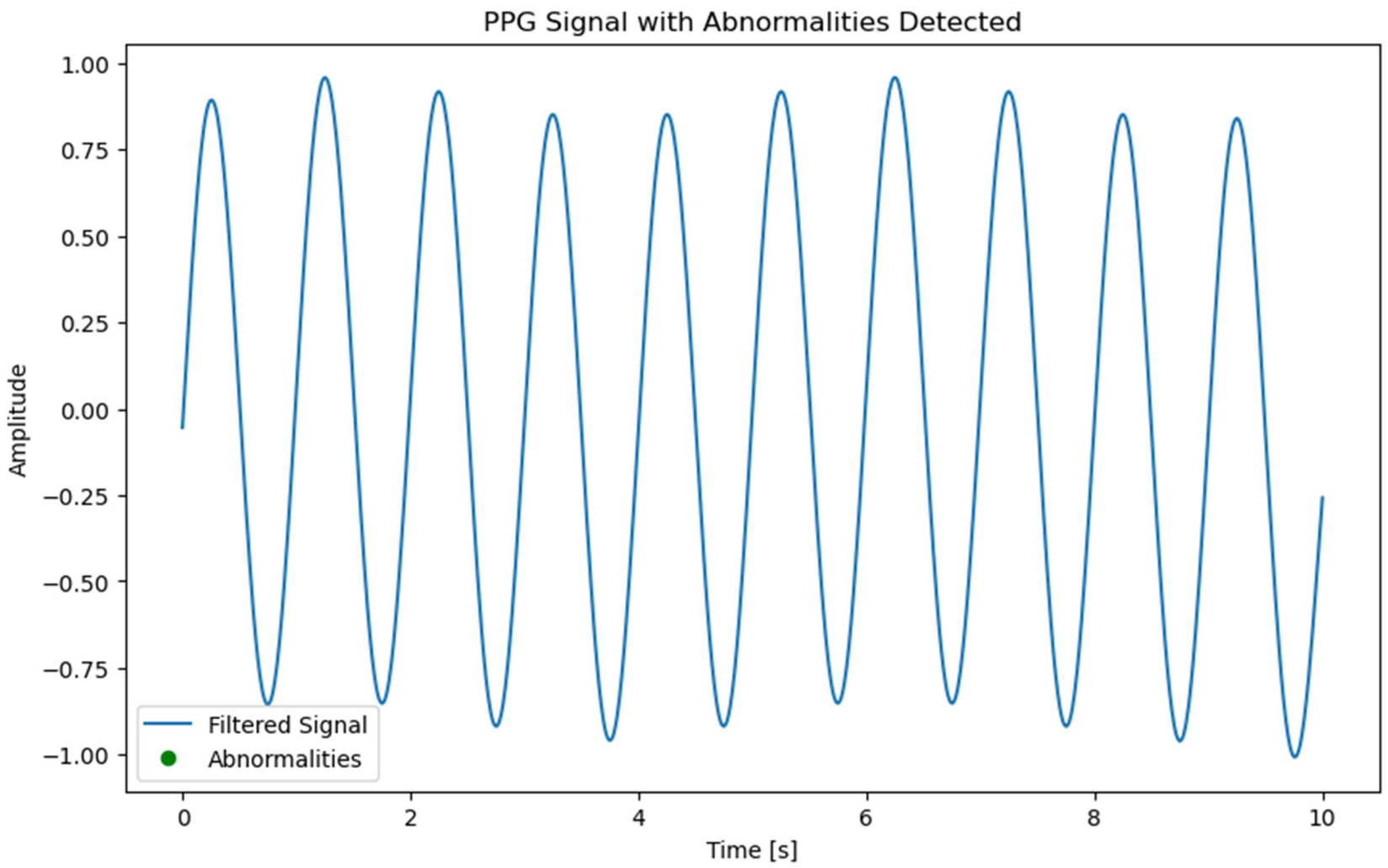
In [6]:

```
# Peak detection
peaks, _ = signal.find_peaks(filtered_signal, height=0.1, distance=50)  # You may need to adjust these parameters

# Plot the filtered signal with detected peaks
plt.figure(figsize=(10, 6))
plt.plot(time, filtered_signal, label='Filtered Signal')
plt.plot(time[peaks], filtered_signal[peaks], 'ro', label='Detected Peaks')
plt.title('PPG Signal with Detected Peaks')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()
plt.show()
```



In [7]:

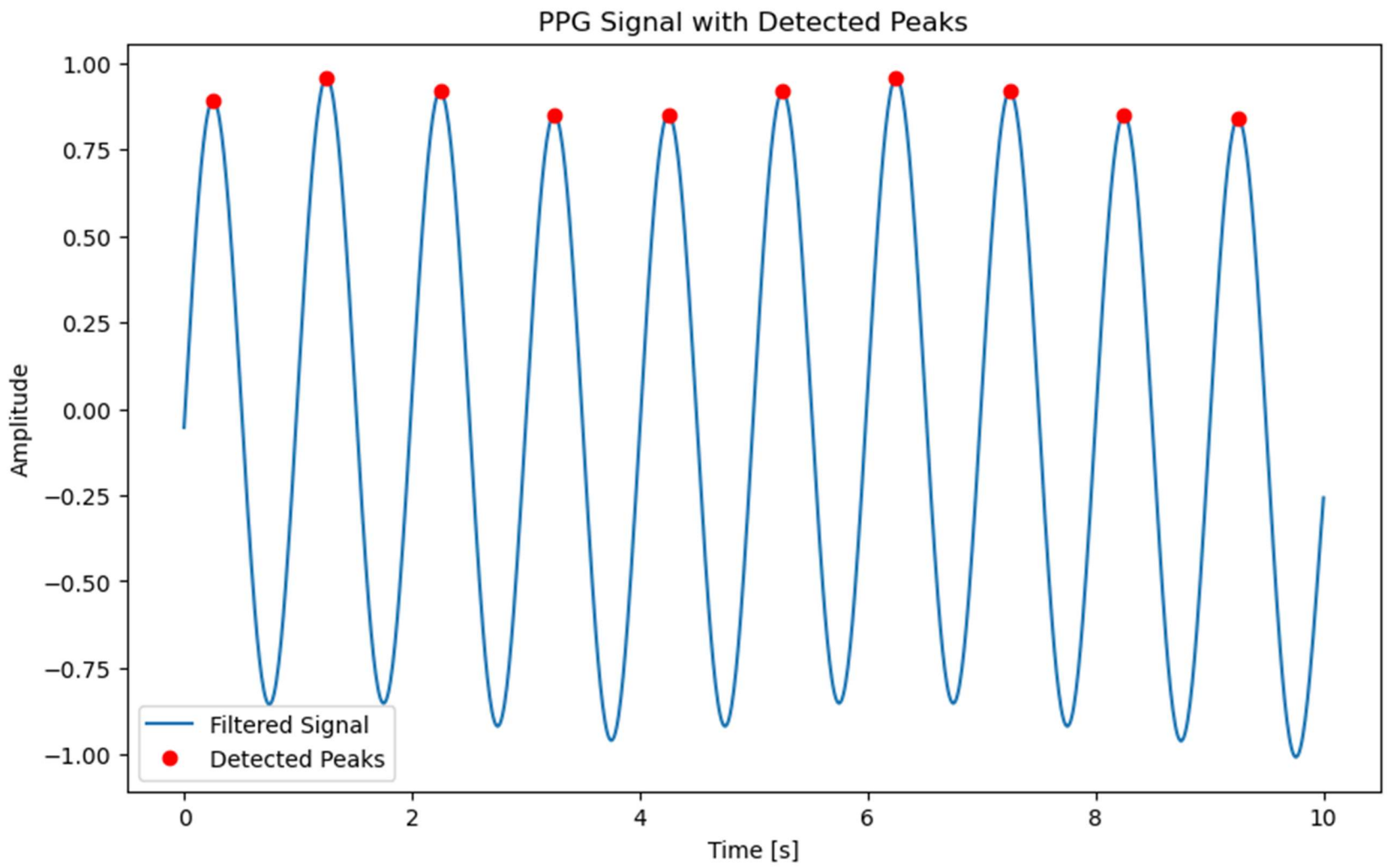
```
# Calculate mean and standard deviation of the filtered signal
mean_signal = np.mean(filtered_signal)
std_signal = np.std(filtered_signal)

# Mark values that are 2 standard deviations above or below the mean as abnormal
abnormalities = np.where(np.abs(filtered_signal - mean_signal) > 2 * std_signal)[0]

# Plot the signal with abnormalities detected
plt.figure(figsize=(10, 6))
plt.plot(time, filtered_signal, label='Filtered Signal')
plt.plot(time[abnormalities], filtered_signal[abnormalities], 'go', label='Abnormalities')
plt.title('PPG Signal with Abnormalities Detected')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()
plt.show()

# Output the abnormality indices
print("Detected abnormality indices:", abnormalities)
```



Detected abnormality indices: []

In [8]:

```
# Detect peaks in the filtered signal
peaks, _ = signal.find_peaks(filtered_signal, height=0.1, distance=50)  # Adjust parameters as needed

# Plot the filtered signal with detected peaks
plt.figure(figsize=(10, 6))
plt.plot(time, filtered_signal, label='Filtered Signal')
plt.plot(time[peaks], filtered_signal[peaks], 'ro', label='Detected Peaks')
plt.title('PPG Signal with Detected Peaks')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()
plt.show()
```



In [9]:

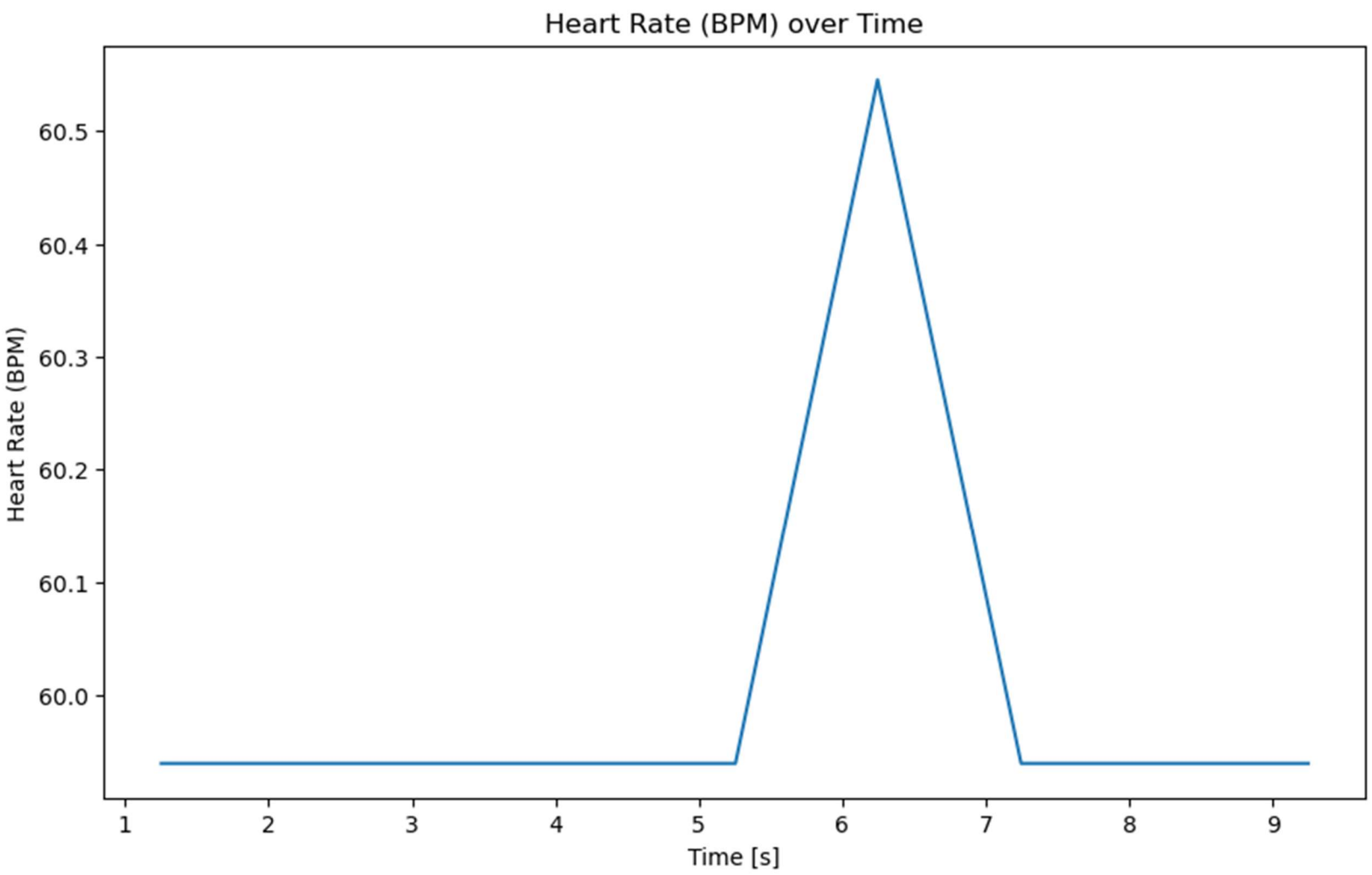
```
# Calculate the times at which the peaks occur
peak_times = time[peaks]


# Calculate the time intervals between successive peaks
peak_intervals = np.diff(peak_times)  # Difference between consecutive peak times
```

In [10]:

```
# Calculate the heart rate for each interval in BPM
heart_rate = 60 / peak_intervals  # Converts intervals to beats per minute
```

In [11]:

```
# Plot the heart rate (BPM) over time
plt.figure(figsize=(10, 6))
plt.plot(peak_times[1:], heart_rate)  # We exclude the first point to match intervals
plt.title('Heart Rate (BPM) over Time')
plt.xlabel('Time [s]')
plt.ylabel('Heart Rate (BPM)')
plt.show()
```



In [12]:

```
# Calculate heart rate (BPM) using the intervals between peaks
heart_rate = 60 / peak_intervals  # Convert time intervals to BPM
```

In [13]:

```
# Define normal heart rate range (can be adjusted based on context)
normal_range_min = 60  # Minimum normal heart rate (BPM)
normal_range_max = 100  # Maximum normal heart rate (BPM)
```

In [14]:

```
# Detect abnormal heart rates (below 60 BPM or above 100 BPM)
abnormal_heart_rates = np.where((heart_rate < normal_range_min) | (heart_rate >
normal_range_max))[0]
```

In [15]:

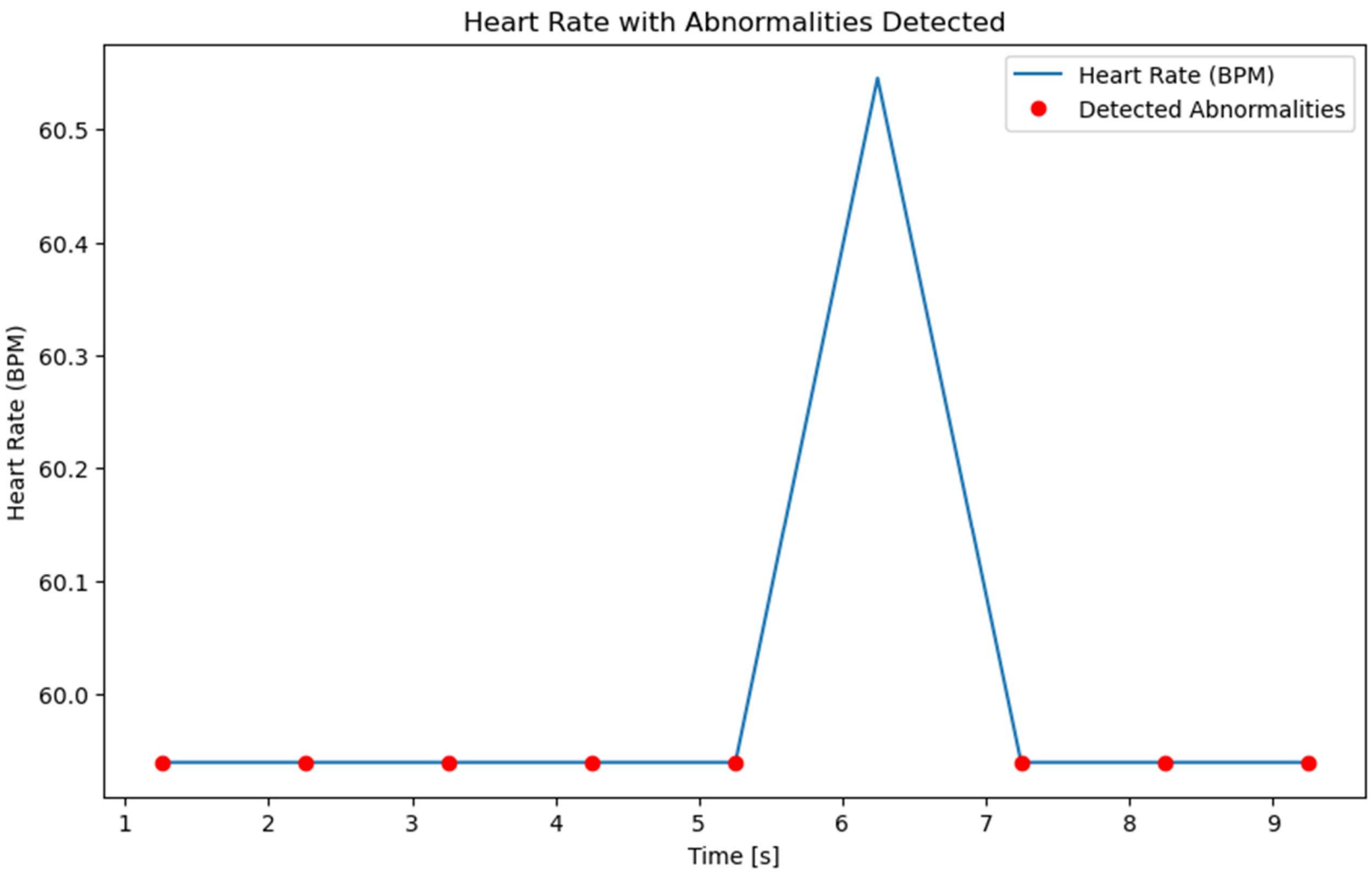
```
# Detect large changes in heart rate (e.g., more than 10 BPM difference between consecutive heart
rates)
irregular_heart_rate_changes = np.where(np.diff(heart_rate) > 10)[0]  # Adjust threshold if
necessary
```

In [16]:

```
# Combine indices of abnormal heart rates and irregular heart rate changes
abnormal_indices = np.unique(np.concatenate([abnormal_heart_rates,
irregular_heart_rate_changes]))
```

In [17]:

```
# Plot heart rate with abnormalities marked
plt.figure(figsize=(10, 6))
plt.plot(peak_times[1:], heart_rate, label='Heart Rate (BPM)')
plt.plot(peak_times[1:][abnormal_indices], heart_rate[abnormal_indices], 'ro', label='Detected
Abnormalities')
plt.title('Heart Rate with Abnormalities Detected')
plt.xlabel('Time [s]')
plt.ylabel('Heart Rate (BPM)')
plt.legend()
plt.show()
```



In [18]:

```
# Output the indices of detected abnormalities
print("Indices of detected abnormalities:", abnormal_indices)
```

```
Indices of detected abnormalities: [0 1 2 3 4 6 7 8]
```

In [19]:

```
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt
```

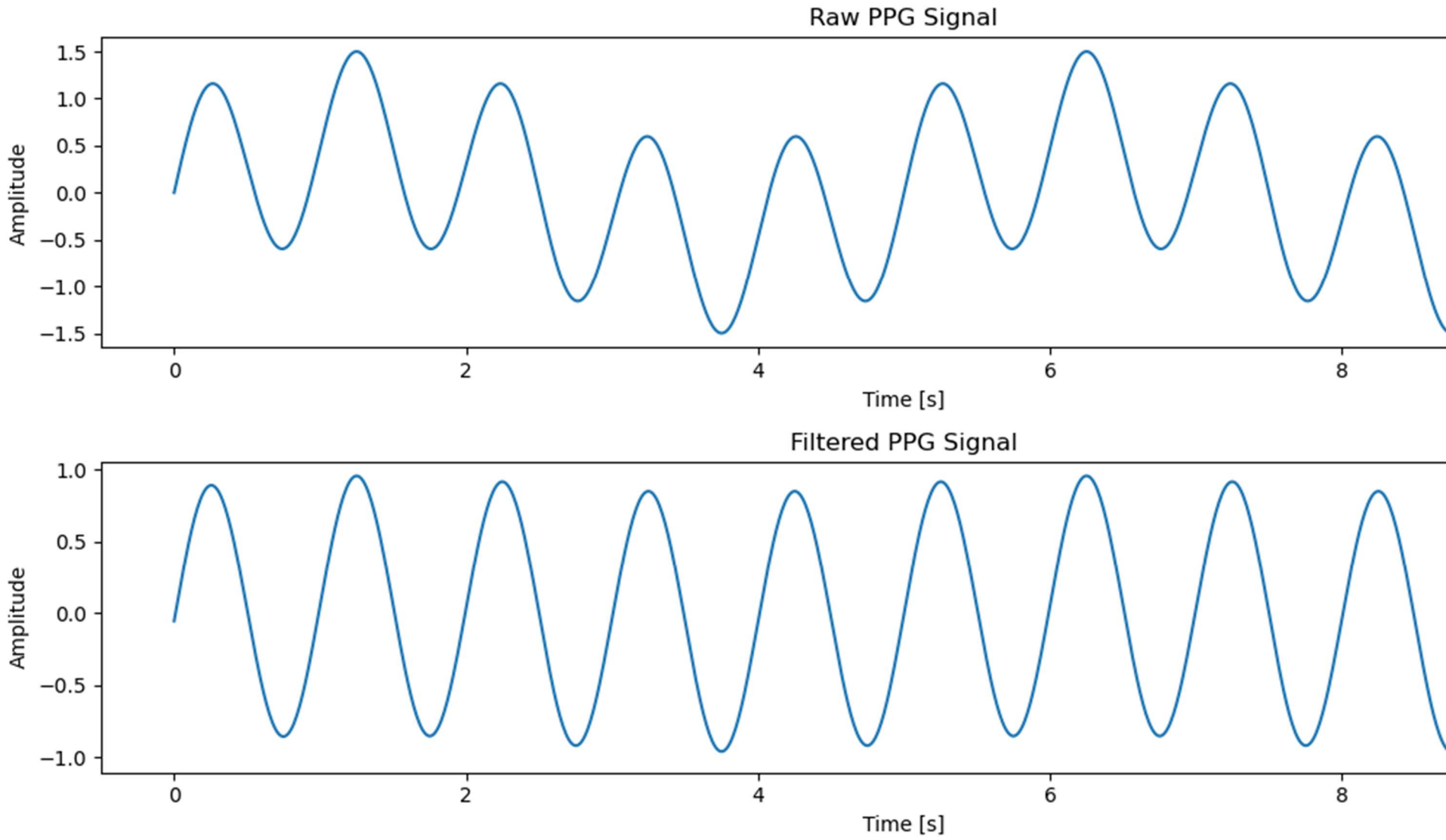
In [20]:

```
# Bandpass filter parameters (you can adjust these)
low_cutoff = 0.5  # Low frequency cutoff (Hz)
high_cutoff = 5.0  # High frequency cutoff (Hz)
fs = 100  # Sampling rate in Hz (adjust if necessary)

# Create the bandpass filter
nyquist = 0.5 * fs
low = low_cutoff / nyquist
high = high_cutoff / nyquist
b, a = signal.butter(1, [low, high], btype='band')

# Apply the bandpass filter to the signal
filtered_signal = signal.filtfilt(b, a, ppg_signal)
```

In [21]:

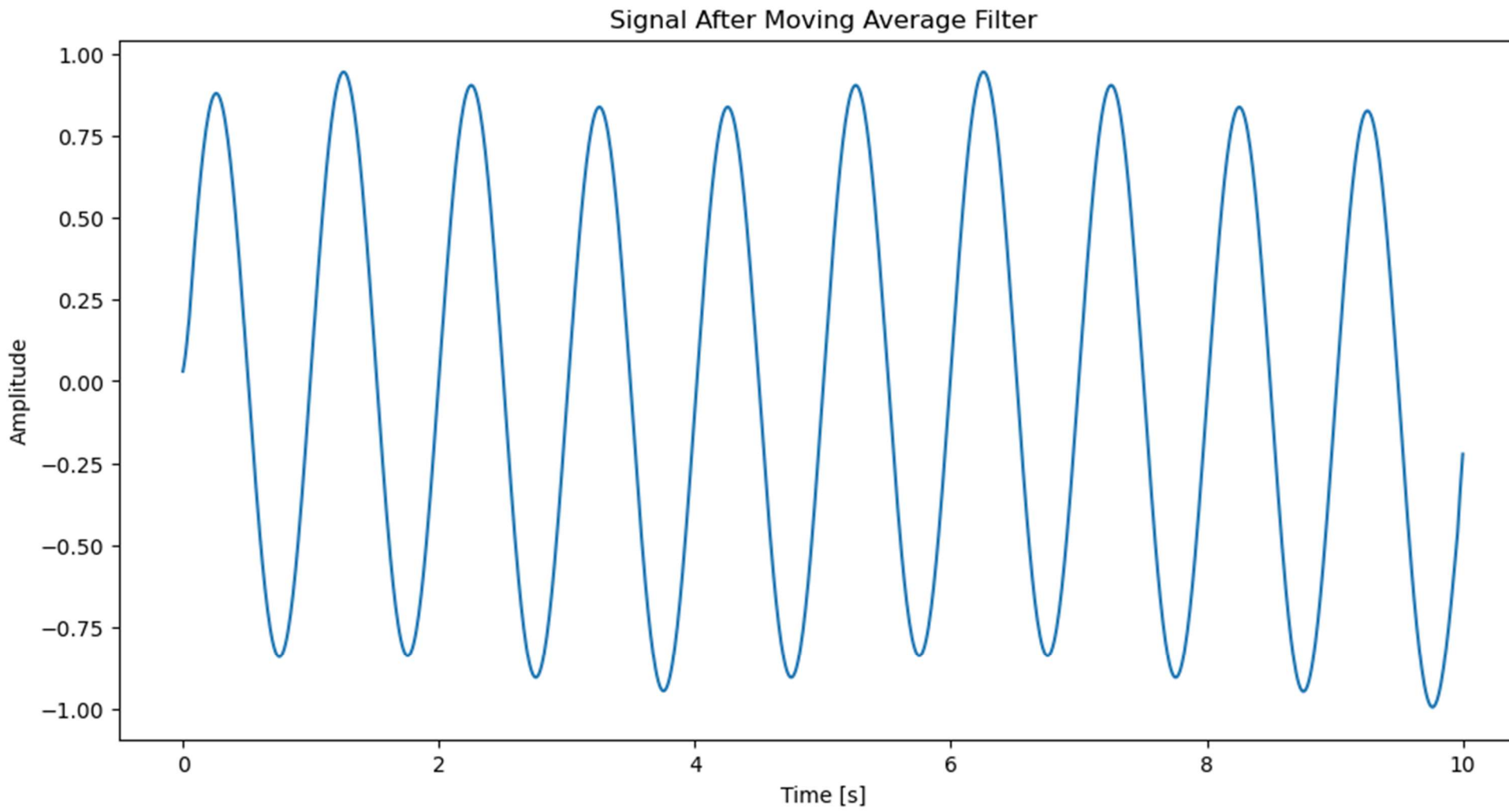
```
# Plot the raw signal and the filtered signal
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(time, ppg_signal)
plt.title('Raw PPG Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
```

```
plt.subplot(2, 1, 2)
plt.plot(time, filtered_signal)
plt.title('Filtered PPG Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')

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



In [22]:

```
# Define the moving average window size (you can adjust this)
window_size = 10  # Number of samples to average

# Apply moving average filter
moving_avg_filtered_signal = np.convolve(filtered_signal, np.ones(window_size)/window_size,
mode='same')

# Plot the moving average filtered signal
plt.figure(figsize=(12, 6))
plt.plot(time, moving_avg_filtered_signal)
plt.title('Signal After Moving Average Filter')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.show()
```


Signal After Moving Average Filter
Amplitude
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
Time [s]