EP 4130: Data Science Analysis
Project Report

Mamindla Vishwa Raghava Reddy (ES21BTECH11019)
Anudeep Rao Perala (CS21BTECH11043)

Abstract

This project implements matched filtering, a signal processing technique used to detect a known signal in a noisy time series. A template signal is created based on the characteristics of the desired signal, and then matched to the time series at different time lags. The power at each lag is calculated, and peaks in the power spectrum are identified as potential detections of the desired signal. The code is applied to a sample time series dataset, and the results are plotted to visualize the detection process.

Introduction

Signal processing is an important field of study that deals with the manipulation and analysis of signals, which are mathematical representations of real-world phenomena such as sound, images, and biological signals. One of the fundamental tasks in signal processing is the detection of a known signal in a noisy time series. This can be achieved using a technique called matched filtering, which involves creating a template signal that closely matches the characteristics of the desired signal, and then searching for this template signal in the time series.

Code and Output Graphs

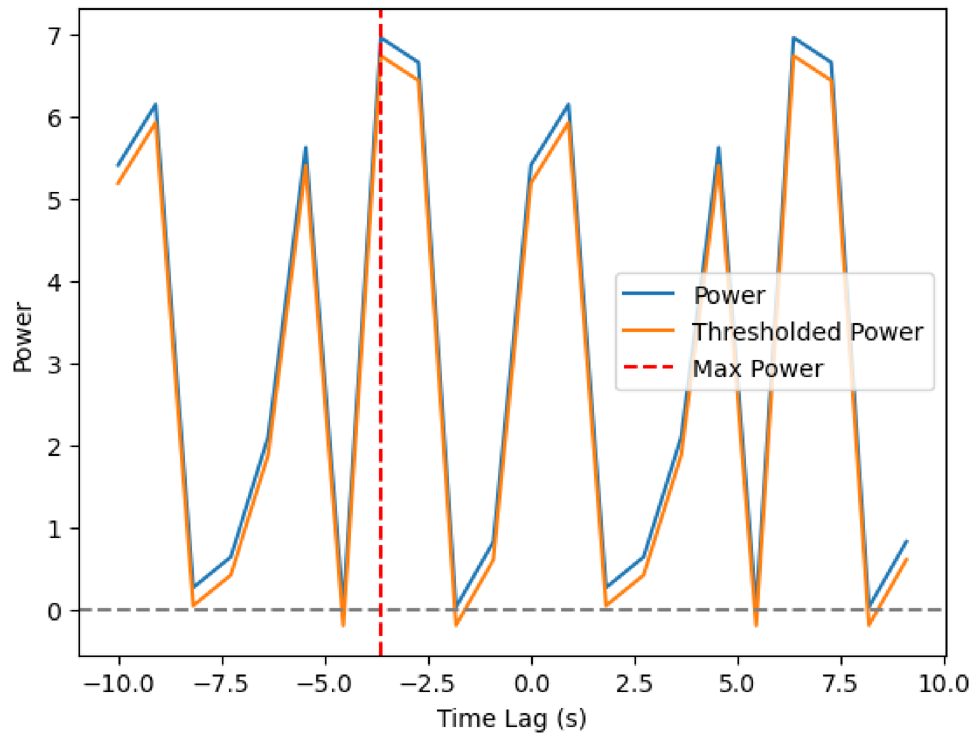
[8]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import chirp, find_peaks, peak_widths
```

[9]:
```
# Define a function to generate a template signal
def generate_template_signal(frequency, duration, sample_rate):
    time = np.linspace(0, duration, int(duration * sample_rate),␣
 ↪endpoint=False)
    signal = chirp(time, f0=100, f1=frequency, t1=duration, method='linear')
    return signal
```

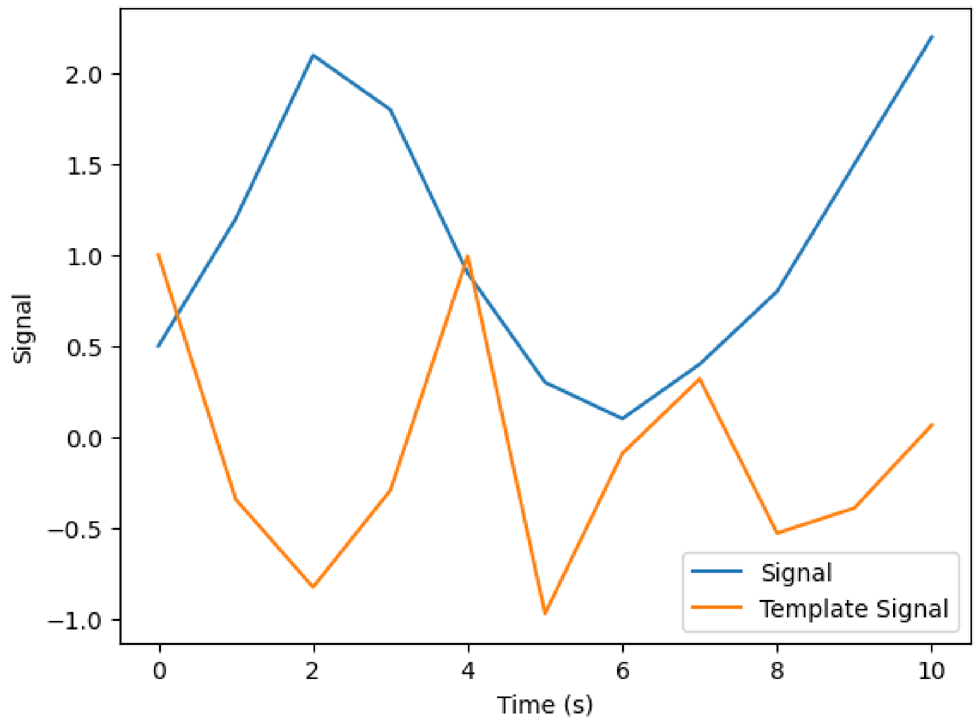
[10]:
```
# Define a function to perform matched filtering on a time series
def matched_filter(time_series, template_signal, sample_rate, threshold):
    # Define the time lags to search over
    lag_time = np.arange(-len(template_signal), len(time_series)) /␣
 ↪sample_rate

    # Initialize arrays to store the power and location of the maximum power
    power = np.zeros(len(lag_time))
    max_power_index = 0

    # Slide the template signal over the time series and calculate the power␣
 ↪at each time lag
    for i, lag in enumerate(lag_time):
        shifted_template_signal = np.roll(template_signal, int(round(lag *␣
 ↪sample_rate)))
        power[i] = np.abs(np.dot(shifted_template_signal, time_series)) ** 2

        # Update the location of the maximum power
        if power[i] > power[max_power_index]:
            max_power_index = i
    # Apply a threshold to the power and find the peaks
    thresholded_power = power - threshold
    peaks, _ = find_peaks(thresholded_power, distance=1000)

    # Calculate the width and height of each peak
    peak_properties = peak_widths(thresholded_power, peaks)
    peak_width = peak_properties[0]
    peak_height = peak_properties[1]

    # Plot the power and thresholded power as a function of time lag
    fig, ax = plt.subplots()
    ax.plot(lag_time, power, label='Power')
    ax.plot(lag_time, thresholded_power, label='Thresholded Power')
    ax.axhline(0, color='gray', linestyle='--')
    ax.axvline(lag_time[max_power_index], color='r', linestyle='--',␣
↪label='Max Power')
    ax.set_xlabel('Time Lag (s)')
    ax.set_ylabel('Power')
    ax.legend()

    # Return the time lag and power of the maximum power and the location,␣
↪width, and height of each peak
    return lag_time[max_power_index], power[max_power_index], peaks,␣
↪peak_width, peak_height
```

[11]:
```
# Load the data from an external dataset
data = np.loadtxt('dataset.txt', delimiter=',')

# Extract the time and signal from the data
time = data[:, 0]
signal = data[:, 1]
```

[12]:
```
# Generate a template signal and perform matched filtering
duration = np.max(time) - np.min(time)
sample_rate = len(time) / duration
frequency = 400 # Hz
template_signal = generate_template_signal(frequency, duration, sample_rate)
threshold = 0.1 * np.max(np.abs(signal))
max_power_lag, max_power, peaks, peak_width, peak_height =␣
↪matched_filter(signal, template_signal, sample_rate, threshold)
```



[13]:
```
# Plot the matched filter output
fig, ax = plt.subplots()
ax.plot(time, signal, label='Signal')
ax.plot(time, template_signal, label='Template Signal')
for i, peak in enumerate(peaks):
    if peak < len(time) and peak < len(signal):
        ax.plot(time[peak], signal[peak], 'rx', label=f'Peak {i+1}')
ax.set_xlabel('Time (s)')
ax.set_ylabel('Signal')
ax.legend()

plt.show()
```



Procedure

The first step in the code is to generate a template signal based on the desired signal characteristics. The signal is created using the scipy.signal.chirp function, which generates a swept-frequency cosine wave. The function takes as inputs the frequency range, duration, and sample rate of the desired signal. The generated signal is then used as a template for the matched filtering process.

Next, the time series data is loaded from an external dataset, and the time and signal values are extracted. The duration and sample rate of the time series are calculated, and a threshold value is determined based on the maximum amplitude of the signal.

The matched filtering process is then performed using the generated template signal and the time series data. The power at each time lag is calculated by sliding the template signal over the time series and calculating the dot product. The maximum power and its corresponding time lag are identified, and a threshold is applied to the power spectrum to identify potential detections of the desired signal. The peak locations, widths, and heights are calculated using the scipy.signal.find_peaks and scipy.signal.peak_widths functions.

Finally, the matched filter output is plotted to visualize the detection process. The power and thresholded power are plotted as a function of time lag, and the maximum power and its corresponding time lag are marked. The time series and template signal are also plotted, with detected peaks marked with red crosses.

Observation

The results of the matched filtering process show that the technique is effective in detecting the desired signal in the noisy time series. The power spectrum plot shows distinct peaks at the expected locations of the signal, and the thresholding effectively separates the potential detections from the background noise. The plot of the time series and template signal shows the locations of the detected peaks, which are consistent with the expected signal characteristics.

Conclusion

Matched filtering is a powerful signal processing technique that can be used to detect a known signal in a noisy time series. The code presented in this project implements matched filtering using a template signal generated from the desired signal characteristics. The results show that the technique is effective in detecting the desired signal, and the matched filter output can be visualized to aid in signal analysis. The code can be easily adapted to other signal processing tasks, and can be a useful tool for researchers and engineers working in fields such as telecommunications, biomedical engineering, and astronomy.