Gravitational Wave Event Detection: Numerical Features in Gravitational Wave Data

Shufan Dong¹

¹Class of 2026, Bronx High School of Science, NY, USA.
dongs1@bxscience.edu

Abstract

This paper presents methodologies for analyzing gravitational wave (GW) data, focusing on time-domain features, event detection, event parameter estimation, and basic statistical analysis. Detailed explanations and Python codes are provided for calculating time-domain features, detecting events, and estimating parameters, followed by summarizing event parameters. The results are contextualized within the framework of ongoing advancements and research in GW astronomy as observed by LIGO, Virgo, and KAGRA collaborations.

Contents

1 Introduction 1
2 Time-Domain Features 2
3 Basic Event Detection & Parameter Estimation 3
4 Basic Statistical Analysis 6
5 Discussion and Conclusion 6

1 Introduction

GW astronomy has opened new platforms for understanding the universe, with instruments like LIGO, Virgo, and KAGRA detecting numerous events such as black holes and neutron star mergers. This research aims to provide tools for the detailed analysis of GW data by focusing on time-domain feature extraction, event detection, and parameter estimation. You can refer to this paper [40] to learn about the preprocessing of these GW data and the libraries imported. The methodologies are illustrated using Python codes, fostering an accessible
approach to data analysis in this field, and while this paper does not introduce the use of machine learning (ML) for event detection, it will be applied later for more precise and accurate detection of events.

2 Time-Domain Features

The function `calc_and_print_time_domain_features` is designed to extract and print key time-domain features from GW data.

```python
def calc_and_print_time_domain_features(data, strain_column, fs):
    peak_amplitude = np.max(data[strain_column])
    min_amplitude = np.min(data[strain_column])
    print("Peak Amplitude ({}): {:.2f}").format(strain_column, peak_amplitude)
    print("Min Amplitude ({}): {:.2f}").format(strain_column, min_amplitude)
    threshold = 0.5 * peak_amplitude
    significant_signal = data[strain_column].abs() > threshold
    signal_duration = significant_signal.sum() * (1/fs)
    print("Signal Duration ({}): {:.2f} s")
    signal_power = np.mean(data[strain_column]**2)
    noise_power = np.mean(data[strain_column].abs() <= threshold)[strain_column]**2
    snr = 10 * np.log10(signal_power / noise_power)
    print("Signal-to-Noise Ratio (SNR) ({}): {:.2f} dB\n")

# Calc features for strain data
print("Calc features for strain data:")
calc_and_print_time_domain_features(data, 'strain', fs)
```

Figure 1: The function accepts three parameters: `data` (a DataFrame containing the signal and time data), `strain_column` (the strain data column), and `fs` (the sampling frequency), and the function calculates the peak and minimum amplitudes of the specified strain column. For computation purposes, a threshold is set at 50% of the peak amplitude, and the duration of significant signals exceeding this threshold is calculated and printed. As a result, the function calculates and prints the signal power, noise power, and Signal-to-Noise Ratio (SNR).

Time-domain features in GW data are crucial because they provide direct insights into the dynamics of astrophysical sources and the propagation of GWs. By analyzing these features, we can extract critical information about the nature and behavior of compact objects, such as black holes and neutron stars, and the environments in which they reside.

For instance, the shape and structure of the GW signal in the time domain can reveal the mass, spin, and orbital dynamics of a binary merger event. Fea-
tures such as chirps, where the frequency and amplitude of the wave increase as the objects spiral closer, are particularly informative. Also, detecting short-lived, transient signals helps identify specific events like black hole mergers and neutron star collisions, and each of them has a unique, discoverable signature in the time domain.

Time-domain analysis allows for the identification of noises, which is essential for improving the signal-to-noise ratio (SNR) and ensuring the accuracy of the detected signals.

Figure 2: The output of the function prints the peak amplitude, minimum amplitude, signal duration, and SNR.

3 Basic Event Detection & Parameter Estimation

The `calc_threshold` function calculates a threshold for event detection based on the standard deviation of the noise in the strain data.

```python
def calc_threshold(data, strain_column, factor=3):
    noise_std = np.std(data[strain_column])
    threshold = factor * noise_std
    return threshold
```

Figure 3: This function calculates a threshold based on the standard deviation of the strain data. The threshold is set to a multiple of this standard deviation and returned. A threshold of approximately 3 is calculated.

The `detect_events` function identifies events in the strain data based on the calculated threshold.
def detect_events(data, strain_column, threshold):
    events = []
    event_start = None

    for i, strain in enumerate(data[strain_column]):
        if abs(strain) > threshold:
            if event_start is None:
                event_start = i
        else:
            if event_start is not None:
                event_end = i
                events.append((event_start, event_end))
                event_start = None

    # Check if an event is ongoing at end of data
    if event_start is not None:
        events.append((event_start, len(data[strain_column]) - 1))

    return events

Figure 4: This function identifies events where the absolute strain exceeds the calculated threshold, and it iterates through the strain data, marking the start and end of events. In the end, detected events are stored as start and end indices in a list.

Event detection is the process of identifying important signals within the GW data that correspond to astrophysical phenomena. Rapid detection enables follow-up observations with electromagnetic (EM) and other observatories, providing critical support to multi-messenger astronomy.

The `estimate_event_params` function calculates parameters for each detected event.
Parameter estimation conveys the importance of determining the physical parameters of the GW source, such as masses, spins, distances, and orbital characteristics. Accurate parameter estimation is vital for interpreting GW observations and understanding the underlying physics.

High-precision parameter estimation allows for stringent tests of general relativity and other gravitational theories. Detailed parameter estimation helps expound the population properties of compact objects, their formation channels, and their role in the cosmos.
4 Basic Statistical Analysis

The `summarize_event_params` function summarizes the parameters of detected events.

```python
def summarize_event_params(event_params):
    if not event_params:  # Check if event_params array is empty
        return {
            'num_events': 0,
            'average_duration': 0,
            'max_duration': 0,
            'average_peak_amplitude': 0,
            'max_peak_amplitude': 0
        }
    durations = [param['duration'] for param in event_params]
    peak_amplitudes = [param['peak_amplitude'] for param in event_params]
    summary = {
        'num_events': len(event_params),
        'average_duration': np.mean(durations),
        'max_duration': np.max(durations),
        'average_peak_amplitude': np.mean(peak_amplitudes),
        'max_peak_amplitude': np.max(peak_amplitudes)
    }
    return summary
```

Figure 7: This function summarizes detected event parameters, and if no events are detected, it returns a summary with zeros. For detected events, it calculates and returns the number of events, average duration, maximum duration, average peak amplitude, and maximum peak amplitude.

Figure 8: This is the summary of the detected events and their corresponding parameters, including total number of events detected, average duration, maximum duration, average peak amplitude, and maximum peak amplitude.

5 Discussion and Conclusion

This paper has detailed the methodologies and Python codes for analyzing time-domain features and detecting events in GW data. The presented functions and their explanations provide a solid framework for handling GW data, enhancing
our ability to extract meaningful insights from GW observations. As GW astronomy advances with new observing runs and enhanced detector sensitivities, these tools will become essential in exploring the cosmos and understanding the underlying physics of these extraordinary events. Later, the implementation of ML will be attempted, and the application of ML may produce expectedly better performance than merely pure Python codes.

References


