**GWpy**: A Python package for gravitational-wave astrophysics

Duncan M. Macleod\textsuperscript{a,}\textsuperscript{*}, Joseph S. Areeda\textsuperscript{b}, Scott B. Coughlin\textsuperscript{a,}\textsuperscript{d}, Thomas J. Massinger\textsuperscript{c}, Alexander L. Urban\textsuperscript{e}

\textsuperscript{a} Cardiff University, Cardiff CF24 3AA, UK
\textsuperscript{b} California State University Fullerton, Fullerton, CA 92831, USA
\textsuperscript{c} LIGO, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
\textsuperscript{d} Center for Interdisciplinary Exploration & Research in Astrophysics (CIERA), Northwestern University, Evanston, IL 60208, USA
\textsuperscript{e} Louisiana State University, Baton Rouge, LA 70803, USA

**Article history:**
Received 15 March 2020
Received in revised form 29 October 2020
Accepted 5 January 2021

**Keywords:**
Gravitational waves
Python
Software

**ABSTRACT**

GWpy is a Python software package that provides an intuitive, object-oriented interface through which to access, process, and visualise data from gravitational-wave detectors. GWpy provides a number of new utilities for studying data, as well as an improved user interface for a number of existing tools. The ease-of-use, along with extensive online documentation and examples, has resulted in widespread adoption of GWpy as a basis for Python software development in the international gravitational-wave community.

© 2021 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

**Code metadata**

- Current code version: 1.0.0
- Permanent link to code/repository used for this code version: https://github.com/ElsevierSoftwareX/SOFTX_2020_127
- Legal Code License: GPL-3.0-or-later
- Code versioning system used: git
- Software code languages, tools, and services used: Python, astropy \( \geq 1.1.1 \), dqsgodb2, gwdatafind, gwsc \( \geq 0.4.0 \), h5py \( \geq 1.3 \), ligo-segments \( \geq 1.0.0 \), ligotimegps \( \geq 1.2.1 \), matplotlib \( \geq 1.2.0 \), numpy \( \geq 1.7.1 \), python-dateutil, scipy \( \geq 0.12.1 \), six \( \geq 1.5 \), tqdm \( \geq 4.10.0 \)
- If available Link to developer documentation/manual: https://gwpy.github.io/docs/1.0.0/
- Support email for questions: See https://github.com/gwpy/gwpy/blob/v1.0.0/CONTRIBUTING.md

1. Motivation and significance

In recent years, the Advanced Laser Interferometer Gravitational-Wave Observatory [1] and Advanced Virgo [2] instruments have made the first detections of gravitational waves (GWs), including the first direct observation of a binary black hole merger [3], and the first joint GW-electromagnetic (EM) observation of a binary neutron star [4]. All of the detections made to date required vast amounts of computational data analysis, not only to extract the signals and their parameters from detector data, but also to study the detectors themselves and characterise their behaviour.

The Python programming language [5] has become a critical component of nearly every facet of computational GW science, including detector control and automation [6], calibration [7], detector characterisation [8–10], and data analysis [11–13]. However, many packages in these areas have been developed independently of the others, resulting in mismatching/multiple APIs for basic operations.

GWpy is a Python package that provides an intuitive, object-oriented user interface to the basic building blocks for data analysis. Its purpose is to simplify data input/output (I/O), signal processing, tabular data filtering, and visualisation. GWpy’s unified I/O system in particular has greatly simplified access to and processing of both ‘raw’ instrumental and processed data, as well as trivialising comparisons of algorithms that previously stored data in incompatible formats.

\textsuperscript{*} Corresponding author.
E-mail address: macleoddm@cardiff.ac.uk (Duncan M. Macleod).
Since its first alpha release in 2014, GWpy has grown to provide a software basis for single-person investigations and large-scale data processing workflows, as well as a number of other newly developed software packages. It is now a key component in the automated data processing environment of LIGO, including detector performance monitoring [14], low-latency event processing [15], and parameter estimation [16], and was critical in the data-quality investigations that validated GW150914, the first direct detection of gravitational waves from a binary black hole merger [17].

This article does not present a complete record of all capabilities of GWpy, this is presented online at https://gwpy.github.io/docs/stable/.

2. Software description

GWpy is implemented in pure Python (i.e. no compiled extension modules), relying heavily on a number of established scientific programming packages [18–22] as well as custom GW data analysis libraries [23–27]. GWpy is designed to simplify the typically complex data analysis tasks that are common across various areas of GW research, including I/O, signal processing, and visualisation.

2.1. Software architecture

The GWPy library interface is structured around a small number of class objects that represent the data structures common in GW data processing, as described in Table 1. Each of these objects is furnished with a suite of class and instance methods that provide the user with a complete interface for all operations.

2.1.1. Array structures

GWpy's high-level array structures (TimeSeries, FrequencySeries, and Spectrogram) are implemented in a common hierarchy as subclasses of the astropy.units.Quantity object [28], itself a subclass of numpy.ndarray [29]. This structure provides direct access to the optimised array functions from NumPy as well as physical unit handling from Astropy.

For all array classes, GWpy adds metadata attributes that describe the source of the original data (if appropriate) and the sampling along a specific physical axis (time or frequency, typically); see Table 2 for descriptions of each attribute.

The index metadata are typically only stored as the starting index value (**x0**) and the step size (**dx**), with a full index array (**xindex**) only evaluated (via a property method) when specifically requested by the user. This allows for a minimal memory overhead of the indexing, whilst not requiring the user to manually evaluate the index if they need it. Arbitrary index arrays can be stored by directly setting the **xindex** attribute. For two-dimensional arrays, the index metadata for the second axis are stored in the **y0**, **dy**, and **yindex** attributes.

Additionally, the TimeSeriesList and TimeSeriesDict objects provide additional functionality for collections of time-domain data.

2.1.2. Tabular data

GWpy’s EventTable class is a subclass of the astropy.table.Table object, providing customisations specific to the typical domain use case of storing parameters for groups of time-domain events. These are typically either transient noise bursts (glitches) or astrophysical GW events.

2.1.3. Data-quality data

GWpy’s DataQualityFlag class represents the time-domain metadata associated with GW detector operational state or the quality of the recorded data. Each DataQualityFlag is comprised of two segmentlist [26] objects, representing times when the relevant flag was known and active respectively. For full details on data-quality flags see [30].

2.2. Software functionalities

As described above, each of the class objects is provided with all relevant functionality provided as class or instance methods. In this section we describe the unified I/O and visualisation interfaces common to all classes, as well as signal processing methods typically used to transform time-domain data into other forms.

2.2.1. Unified input/output

Astropy provides an infrastructure for unified input/output via the astropy.io module that is only applied in that package to the astropy.table.Table class object. GWpy leverages this infrastructure to provide common read() and write() methods for all class objects, enabling reading from and writing to all common GW file formats, see Table 3 for a reduced list.

2.2.2. Remote data access

GWpy also provides an intuitive remote-data access system for downloading time-domain data directly into a TimeSeries object. This is split into two processing methods that serve public data from the Gravitational-Wave Open Science Center [32] (GWOSC) and proprietary data from LIGO data archives respectively.

For public data access, where only GW strain data are typically available, the user need only supply the interferometer prefix (e.g. ‘H1’ for LIGO Hanford Observatory), and the start and stop times of their interval of interest.1 GWpy then uses the gwosc [25] library to identify the remote URLs of data files containing that fulfil the request, downloads them to a temporary location, reads the data, then removes the temporary files.

For proprietary data, where hundreds of thousands of data channels are available, the user must supply the name of the channel along with the timing interval. GWpy will then query the local data archive service (if running directly at a LIGO-operated computing centre), or one or all of an ordered list of remote data access services, in either case returning only the requested data to the user.

---

1 Timing parameters can be given either as GPS times (**float**), or as human-readable UTC date strings.

---

### Table 1

<table>
<thead>
<tr>
<th>Data structure</th>
<th>GWpy class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D time-domain data</td>
<td>gwpy.timeseries.TimeSeries</td>
</tr>
<tr>
<td>1D frequency-domain data</td>
<td>gwpy.frequencieseries.FrequencySeries</td>
</tr>
<tr>
<td>2D time–frequency maps</td>
<td>gwpy.spectrogram.Spectrogram</td>
</tr>
<tr>
<td>data tables</td>
<td>gwpy.table.EventTable</td>
</tr>
<tr>
<td>data-quality flags</td>
<td>gwpy.segments.DataQualityFlag</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>str</td>
<td>name of these data</td>
</tr>
<tr>
<td>channel</td>
<td>Channel</td>
<td>source of these data</td>
</tr>
<tr>
<td>x0</td>
<td>Quantity</td>
<td>X-axis value of starting data point</td>
</tr>
<tr>
<td>dx</td>
<td>Quantity</td>
<td>X-axis step between data points</td>
</tr>
</tbody>
</table>
### 2.2.3. Signal processing

Many research applications require transforming the ‘raw’ time-domain data recorded at an observatory into the frequency domain, or another format in order to study the features of the data. The `TimeSeries` object leverages the SciPy [22] signal processing library `scipy.signal` to provide instance methods for time-domain signal processing, including: calculating the Fourier transform of data `.fft()`, estimating the coherence between two series `.coherence()`, estimating the Power or Amplitude Spectral Density `.psd()`, `.asd()`, and generating a Spectrogram of overlapping spectral density estimates `.spectrogram()`, `.spectrogram2()`.

Additionally, GWpy provides an implementation of the Q transform [35], used to generate multi-resolution time–frequency maps of data `.qtransform()`. The `FrequencySeries` object provides an `.ifft()` instance method to calculate the inverse Fourier transform.

### 2.2.4. Visualisation

Each of the GWpy class objects includes a visualisation interface supported by Matplotlib [20]. For most objects this is provided as a plot() instance method that decomposes the object into the relevant arrays required by matplotlib, renders those arrays as required, and returns a formatted `matplotlib` figure.Figure 3 demonstrates this functionality.

### 3. Illustrative examples

The following examples are reproduced from the online documentation at https://gwpy.github.io/docs/1.0.0/examples/.

#### 3.1. Example 1: estimating amplitude spectral density from public LIGO data

This example demonstrates downloading public GW event data associated with GW150914, estimating the amplitude spectral density of the strain data, and visualising these in a figure (see Fig. 1).

**Listing 1:** Example code to generate an ASD plot with GWpy.

```python
from gwpy.timeseries import TimeSeries
data = TimeSeries.fetch_open_data('H1',
    "2015-09-14 09:50:30", 
    "2015-09-14 09:51:00", 
    sample_rate=16384,
)asd = data.asd(fftlength=4)
plot = asd.plot( 
    color='gwy:ligo-hanford',
xscale='log', ylim=(20, 4000),
ylabel='Strain noise, $[Hz^{-1/2}]$",
ylabel=r'Coherence', xlabel='Frequency', ylabel='Amplitude Spectral Density', yscale='linear', ylim=(2e-24, 5e-21),
plot.show()
```

For a full description of the supported averaging methods, see the documentation for `TimeSeries.psd()`.

#### 3.2. Example 2: estimating the coherence between two data channels

This example demonstrates accessing proprietary LIGO instrumental data and estimating the coherence between an accelerometer signal and the calibrated GW strain data (see Fig. 2).

**Listing 2:** Example code to estimate the coherence between two timeseries with GWpy.

```python
from gwpy.timeseries import TimeSeriesDict
data = TimeSeriesDict.get(
    ['H1:GDS-CALIB_STRAIN', 'H1:PEM-CS_PSL_PERISCOPE_X_DQ'],
    1126260017, 1126260617,
)hoft = data['H1:GDS-CALIB_STRAIN']
acc = data['H1:PEM-CS_PSL_PERISCOPE_X_DQ']
coh = hoft.coherence(acc, fftlength=2, overlap=1)
plot = coh.plot( 
    xscale='log', yscale='log',
ylabel='Coherence', yscale='linear', ylim=[0, 1],
plot.show()
```

### 4. Impact

GWpy is now a widely used library in GW data analysis, with more than 40 downstream dependants [36] and more than 250 stars [37], including LALInference [38], PyCBC [11], and Bilby [16].

The intuitive object-oriented interface has significantly reduced the overhead of repetitive tasks common to the majority of GW data analysis pipelines, allowing research scientists – often junior researchers or post-graduate students new to scientific programming – to concentrate on implementing new scientific techniques, rather than reimplementing and validating banal pre-processing tasks.

GWpy has specifically enabled creation of, or enhancements to, two widely-used web-based services. LIGO Data Viewer Web (LDVW) [39] is a browser-based application that enables data visualisation through simple web forms. This application now uses GWpy as the backend for the majority of its available products. The LIGO Summary Pages [14] are an automatically-updating view of the performance of the GW network, including ASDs, ASD spectrograms, sensitive distance trends and transient glitch maps. This system generates $\mathcal{O}(1000)$ figures of merit, updating every 30 min (on average) for each LIGO observatory, all of which are generated using GWpy as the backend for data handling and visualisation.

These two services together have enabled all members of the the LIGO Scientific Collaboration (LSC) and the Virgo Collaboration to see up-to-date information on detector network performance, and reproduce and generate their own figures of merit with identical look-and-feel, greatly increasing the comparability of data. The ease by which data can be accessed and processed, enabled by GWpy, was critical to the validation of...
GW150914 [17], the first detection of gravitational waves, and subsequent detections [40,41].

5. Conclusion

GWpy is a Python package that provides the basic building blocks for a growing number of GW data analysis workflows. It provides a user-friendly, object-oriented interface that trivialises multi-format data I/O, signal processing, and visualisation in a way that significantly reduces the overhead for researchers when developing scientific analysis software. GWpy has been critical to the success of Advanced Laser Interferometer Gravitational-Wave Observatory [1] (aLIGO) by enabling creation and operation of console- and web-based tools that have accelerated data-quality investigations and understanding of GW detector data.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank all members of the LIGO Scientific Collaboration and the Virgo collaboration, and the wider scientific community, for all of the constructive feedback, bug reports, and feature requests relating to GWpy.

DMM was supported by the European Union Horizon 2020 Framework Programme, UK under grant agreement no. 663380-CU073.
References