Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

GWpy: A Python package for gravitational-wave astrophysics

Duncan M. Macleod [a,*], Joseph S. Areeda [b], Scott B. Coughlin [a,d], Thomas J. Massinger [c], Alexander L. Urban [e]

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

ARTICLE INFO

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.

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
Compilation requirements, operating environments & dependencies	`astropy` ≥ 1.1.1, `dqsegdb2`, `gwdatafind`, `gwosc` ≥ 0.4.0, `h5py` ≥ 1.3, `ligo-segments` ≥ 1.0.0, `ligotimegps` ≥ 1.2.1, `matplotlib` ≥ 1.2.0, `numpy` ≥ 1.7.1, `python-dateutil`, `scipy` ≥ 0.12.1, `six` ≥ 1.5, `tqdm` ≥ 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.

* Corresponding author.
E-mail address: macleoddm@cardiff.ac.uk (Duncan M. Macleod).

https://doi.org/10.1016/j.softx.2021.100657

Table 1
High-level class objects in GWpy.

Data structure	GWpy class
1D time-domain data	`gwpy.timeseries.TimeSeries`
1D frequency-domain data	`gwpy.frequencyseries.FrequencySeries`
2D time–frequency maps	`gwpy.spectrogram.Spectrogram`
data tables	`gwpy.table.EventTable`
data-quality flags	`gwpy.segments.DataQualityFlag`

Table 2
Common array attributes in GWpy.

Attribute	Type	Description
`name`	`str`	name of these data
`channel`	`Channel`	source of these data
`x0`	`Quantity` scalar	X-axis value of starting data point
`dx`	`Quantity` scalar	X-axis step between data points

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 3
A selection of custom formats implemented in GWpy and accessible through the unified I/O interface for the listed class object(s).

Format	Classes	Description
`'gwf'`	`{Time,Frequency}Series`	the common GW data frame format [31]
`'hdf5'`	`{Time,Frequency}Series`	simple HDF5 datasets
`'hdf5.losc'`	`{Time,Frequency}Series`	GWOSC-formated HDF5 datasets [32]
`'ligolw'`	`EventTable`	LIGO Light-Weight XML [33]
`'root'`	`EventTable`	ROOT [34] trees

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()`),[2] 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`. Section 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.

```
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="gwpy:ligo-hanford",
    xscale="log", xlim=(20,4000),
    yscale="log", ylim=(2e-24, 5e-21),
    ylabel=r"Strain␣noise␣[Hz$^{-1/2}$]",
)
plot.show()
```

[2] 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.

```
from gwpy.timeseries import TimeSeriesDict
data = TimeSeriesDict.get(
    ['H1:GDS-CALIB_STRAIN', 'H1:PEM-CS_ACC_PSL_PERISCOPE_X_DQ'],
    1126260017,
    1126260617,
)
hoft = data['H1:GDS-CALIB_STRAIN']
acc = data['H1:PEM-CS_ACC_PSL_PERISCOPE_X_DQ']
coh = hoft.coherence(acc, fftlength=2, overlap=1)
plot = coh.plot(
    xlabel='Frequency␣[Hz]', xscale='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 web 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 access and processed, enabled by GWpy, was critical to the validation of

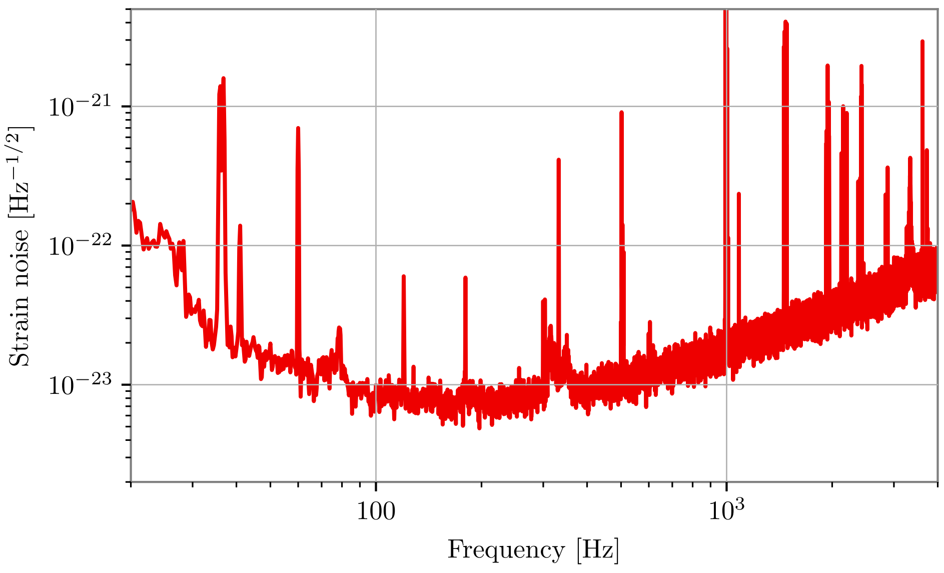

Fig. 1. Output figure for Listing 1.

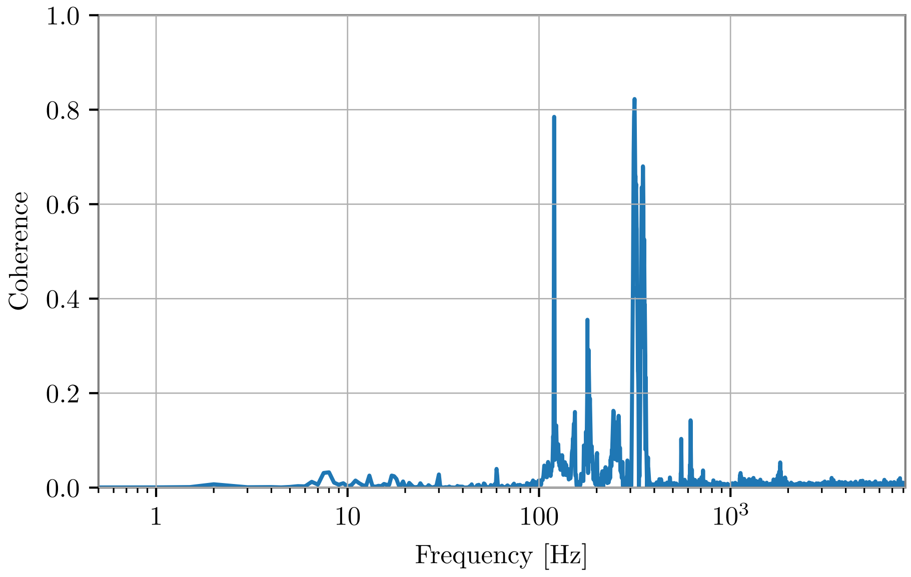

Fig. 2. Output figure for Listing 2.

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

[1] The LIGO Scientific Collaboration. Advanced LIGO. Classical Quantum Gravity 2015;32(7):074001.
[2] Acernese F, et al. Advanced Virgo: a second-generation interferometric gravitational wave detector. Classical Quantum Gravity 2015;32(2):024001.
[3] Abbott BP, et al. Observation of gravitational waves from a binary black hole merger. Phys Rev Lett 2016;116(6):061102.
[4] Abbott BP, et al. GW170817: Observation of gravitational waves from a binary neutron star inspiral. Phys Rev Lett 2017;119(16):161101.
[5] The Python Software Foundation, Python Language Reference, http://www.python.org.
[6] Rollins JG. Distributed state machine supervision for long-baseline gravitational-wave detectors. Rev Sci Instrum 2016;87(9):094502.
[7] Viets AD, et al. Reconstructing the calibrated strain signal in the advanced ligo detectors. Classical Quantum Gravity 2018;35(9):095015.
[8] Essick R, et al. Optimizing vetoes for gravitational-wave transient searches. Classical Quantum Gravity 2013;30(15):155010.
[9] Urban AL, et al. GWDetchar, https://doi.org/10.5281/zenodo.2575786.
[10] Smith JR, et al. Hveto, https://doi.org/10.5281/zenodo.2584615.
[11] Usman SA, et al. The pycbc search for gravitational waves from compact binary coalescence. Classical Quantum Gravity 2016;33(21):215004.
[12] Messick C, et al. Analysis framework for the prompt discovery of compact binary mergers in gravitational-wave data. Phys. Rev. D 2017;95(4):042001.
[13] Pitkin M. CWInPy, https://cwinpy.readthedocs.io/.
[14] Macleod DM, Urban A, et al. GWSumm. https://doi.org/10.5281/zenodo.2647609.
[15] Singer L. GWCelery. https://gw.readthedocs.io/projects/gwcelery/.
[16] Ashton G, et al. Bilby: A user-friendly Bayesian inference library for gravitational-wave astronomy. Astrophys J Suppl 2019;241(2):27.
[17] Abbott BP, et al. Characterization of transient noise in advanced LIGO relevant to gravitational wave signal GW150914. Classical Quantum Gravity 2016;33(13):134001.
[18] Robitaille TP, et al. Astropy: A community python package for astronomy. 2013.
[19] Collette A, et al. h5py. https://www.h5py.org.
[20] Hunter JD. Matplotlib: A 2D graphics environment. Comput Sci Eng 2007;9(3):90–5. http://dx.doi.org/10.1109/MCSE.2007.55.
[21] Oliphant TE. A guide to NumPy, vol. 1. Trelgol Publishing USA; 2006.
[22] Virtanen P, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. Nature Methods 2018 2020;17(3):261–72.
[23] Macleod DM. dqsegdb2. https://doi.org/10.5281/zenodo.2559254.
[24] Macleod DM. GWDataFind. https://gwdatafind.readthedocs.io/.
[25] Macleod DM. `gwosc`. https://doi.org/10.5281/zenodo.1196305.
[26] Cannon K. `ligo-segments`. https://lscsoft.docs.ligo.org/ligo-segments/.
[27] Wette K. SWIGLAL: Python and octave interfaces to the LALSuite gravitational-wave data analysis libraries. SoftwareX 2020;12:100634.
[28] Robitaille TP, et al. Astropy: A community Python package for astronomy. A&A 2013;558:A33.
[29] van der Walt S, Colbert SC, Varoquaux G. The NumPy array: A structure for efficient numerical computation, Comput Sci Eng, 13(2): 22–30.
[30] Fisher RP, et al. DQSEGDB: A time-interval database for storing gravitational wave observatory metadata. SoftwareX 2021. in preparation.
[31] LIGO V. Specification of a common data frame format for interferometric gravitational wave detectors (IGWD) ligo-t970130. 2009, https://dcc.ligo.org/LIGO-T970130/public.
[32] Trovato A. GWOSC: Gravitational wave open science center. In: Proceedings of the New Era of Multi-Messenger Astrophysics — PoS(Asterics2019). Trieste, Italy: SISSA Medialab; 2020, p. 082.
[33] Cannon K. `python-ligo-lw`. https://git.ligo.org/kipp.cannon/python-ligo-lw/.
[34] Brun R, Rademakers F. ROOT: An object oriented data analysis framework. In: New Computing Techniques in Physics Research V. Proceedings, 5th International Workshop, AIHENP '96, Lausanne, Switzerland, September 2-6, 1996. Nucl Instrum Methods A 1997;389:81–6. http://dx.doi.org/10.1016/S0168-9002(97)00048-X.
[35] Brown JC. Calculation of a constant q spectral transform. J Acoust Soc Am 1998;89(1):425–34.
[36] GWpy Dependents. https://github.com/gwpy/gwpy/network/dependents.
[37] GWpy Stargazers. https://github.com/gwpy/gwpy/stargazers.
[38] Veitch J, et al. Parameter estimation for compact binaries with ground-based gravitational-wave observations using the LALInference software library. Phys. Rev. D 2015;91(4):042003.
[39] Areeda JS, et al. Ligodv-web: Providing easy, secure and universal access to a large distributed scientific data store for the LIGO scientific collaboration. A&C 2017;18:27–34.
[40] Abbott BP, et al. Effects of data quality vetoes on a search for compact binary coalescences in advanced LIGO's first observing run. Classical Quantum Gravity 2018;35(6):065010.
[41] Nuttall LK. Characterizing transient noise in the LIGO detectors. Phil Trans R Soc A 2018;376(2120):20170286.