



# GWFAST: A Fisher Information Matrix Python Code for Third-generation Gravitational-wave Detectors

Francesco Iacovelli<sup>1,2</sup> , Michele Mancarella<sup>1,2</sup> , Stefano Foffa<sup>1,2</sup> , and Michele Maggiore<sup>1,2</sup> <sup>1</sup> Département de Physique Théorique, Université de Genève, 24 quai Ernest Ansermet, 1211 Genève 4, Switzerland; [Francesco.Iacovelli@unige.ch](mailto:Francesco.Iacovelli@unige.ch)<sup>2</sup> Gravitational Wave Science Center (GWSC), Université de Genève, CH-1211 Genève, Switzerland

Received 2022 August 1; revised 2022 September 8; accepted 2022 September 8; published 2022 October 21

## Abstract

We introduce GWFAST (<https://github.com/CosmoStatGW/gwfast>), a Fisher information matrix Python code that allows for easy and efficient estimation of signal-to-noise ratios and parameter measurement errors for large catalogs of resolved sources observed by networks of gravitational-wave (GW) detectors. In particular, GWFAST includes the effects of the Earth’s motion during the evolution of the signal, supports parallel computation, and relies on automatic differentiation rather than on finite differences techniques, which makes possible the computation of derivatives with accuracy close to machine precision. We also release the library WF4Py (<https://github.com/CosmoStatGW/WF4Py>) implementing state-of-the-art GW waveforms in Python. In this paper we provide a documentation of GWFAST and WF4Py with practical examples and tests of performance and reliability. In the companion paper Iacovelli et al. we present forecasts for the detection capabilities of the second and third generation of ground-based GW detectors, obtained with GWFAST.

*Unified Astronomy Thesaurus concepts:* [Gravitational wave astronomy \(675\)](#); [Gravitational wave detectors \(676\)](#); [Gravitational wave sources \(677\)](#); [Fisher’s Information \(1922\)](#); [Astrostatistics tools \(1887\)](#)

## 1. Introduction

GWFAST is a new, fast and accurate software, capable of computing signal-to-noise ratios (S/Ns) and parameter measurement errors for networks of gravitational-wave (GW) detectors, using the Fisher information matrix (FIM) formalism. This approximates the full posterior probability distribution for the parameters of a GW signal (see, e.g., Cutler & Flanagan 1994; Vallisneri 2008; Rodriguez et al. 2013 for a comprehensive treatment) and is used for forecasts on large catalogs of sources for which a full parameter estimation would be computationally too expensive. The computational cost is the main limitation of present-day forecast studies, especially for the third generation of GW detectors. This is related to two main aspects. The first is the duration of the signal (in particular, for binary neutron stars at ground-based detectors), which requires one to correctly account for the time evolution of the antenna pattern functions and makes the data analysis challenging in terms of computational resources. To our knowledge, the problem of a full Bayesian inference for even a single one of such events is not manageable with techniques and resources used for second-generation detectors. Only recently dedicated approaches have started to be investigated (Smith et al. 2021). The second aspect is the scalability to large catalogs. The study of the reach and parameter estimation capabilities of third-generation (3G) detectors is a key aspect for assessing their scientific potential, and typically requires studying catalogs of tens of thousands of sources. GWFAST is suitable for these applications since it accounts for state-of-the-art waveform models, the effect of the motion of the Earth, and the possibility of parallel evaluations when running on large catalogs. Moreover, it does not rely on finite difference techniques to compute derivatives, but on automatic

differentiation (AD), which is a method that does not suffer from possible inaccuracies arising from the computation of the derivatives, in particular related to the choice of the step size. Hence, we make it publicly available, together with routines to run in parallel. In this paper we provide a documentation, tests to validate the reliability of the code, and some examples. A scheme of the organization of the code is reported in Figure 1. In the companion paper, Iacovelli et al. (2022), we used GWFAST to produce forecasts for the detection capabilities of LIGO–Virgo–KAGRA (LVK) during their forthcoming O4 run, and of 3G ground-based GW detectors, namely, Einstein Telescope (ET) and Cosmic Explorer (CE), based on up-to-date models of the expected population of sources.

This paper is structured as follows. In Section 2 we describe the conventions for the input parameters and the waveform models available in GWFAST, which are a pure Python version of those contained in the LIGO Algorithm Library LAL (LIGO Scientific Collaboration 2018), and compare with their original implementation. The waveform models are also separately released in a ready-to-use version, WF4Py. Moreover, GWFAST implements an interface with LAL, so that all of the waveforms available in this library can be used directly. However, only the use of the pure Python implementation allows us to fully exploit the vectorization capabilities of GWFAST and to use AD, as explained below. In Section 3 we document the two core modules of the software, `signal` and `network`, which allow the user to easily compute S/Ns and FIMs, with various code examples. In particular, in Section 3.3 we provide an overview of how GWFAST deals with the computation of the derivatives of the signal with respect to the parameters. If using the Python implementation of the waveforms, we evaluate those using the AD module of the JAX library (Bradbury et al. 2018), which ensures a fast and accurate computation, while if using the LAL waveforms, the computation is performed using finite difference techniques. In Section 4 we then describe how GWFAST deals with the inversion of the Fisher matrix, to obtain the covariance matrix



Original content from this work may be used under the terms of the [Creative Commons Attribution 4.0 licence](https://creativecommons.org/licenses/by/4.0/). Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

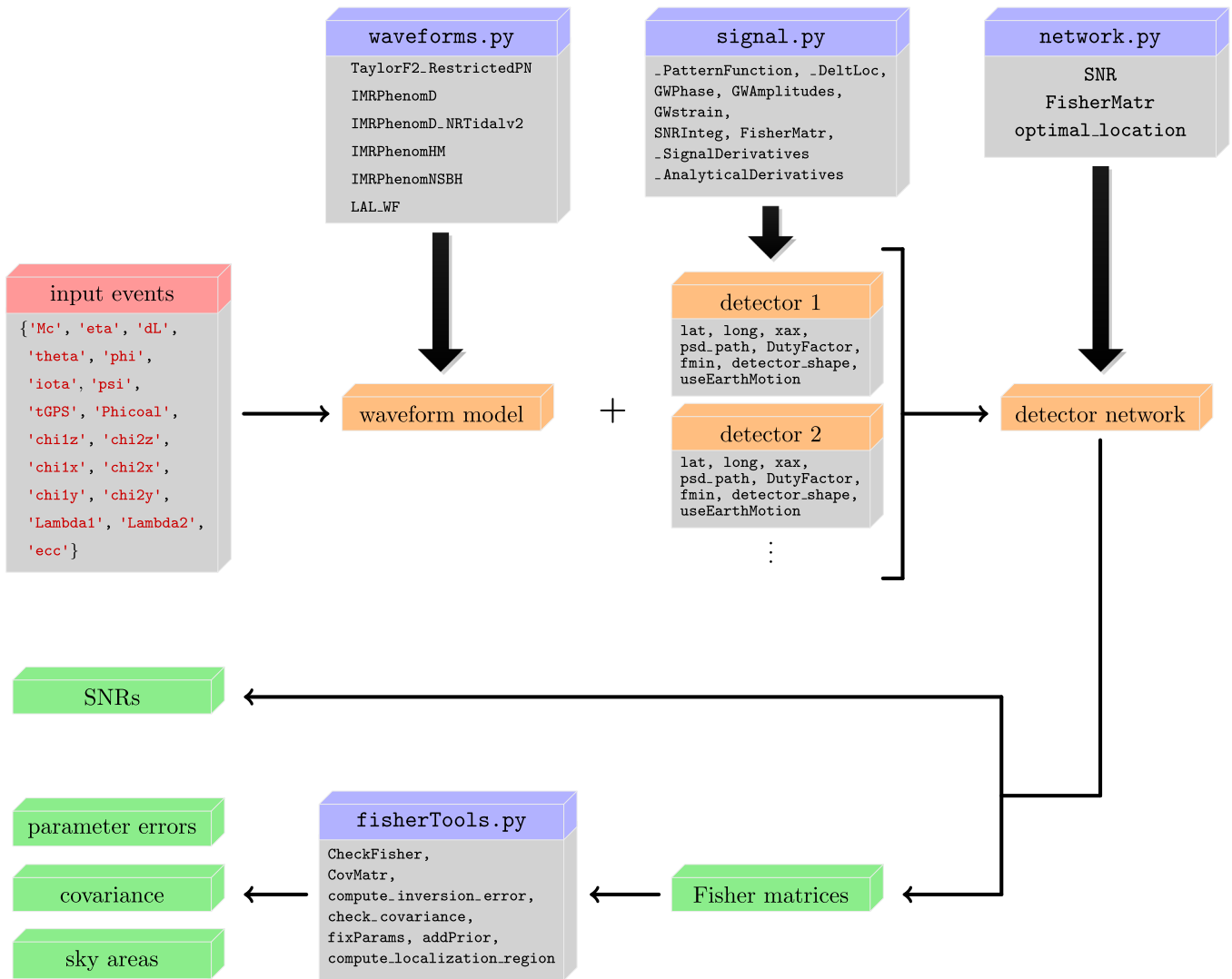


Figure 1. Flowchart of the functioning of GWFAST. See Section 2.1 for the description of the inputs.

and thus the measurement errors, and the various manipulations that can be performed using the module `fisherTools`. Section 5 is devoted to the description of how to run GWFAST on multiple CPUs, so to easily handle huge catalogs of events, through the module `calculate_forecasts_from_catalog.py`. In Section 6, to assess its reliability, we show the application of GWFAST to some of the real events observed by the LIGO and Virgo interferometers during their second and third observing runs, for which a full Bayesian parameter estimation has been performed. Finally, in Section 7 we summarize and conclude this work.

## 2. Signal Modeling

Waveform models give the prediction for a GW signal emitted by a coalescing binary as a function of the parameters of the source. We here give a brief overview of the chosen conventions for these parameters, as well as the waveforms available in GWFAST and their implementation.

### 2.1. Input Parameters

In the most general case, restricting to quasi-circular orbits, the parameters needed to describe the signal emitted by a

coalescing binary system are 15 for a binary black hole (BBH), 17 for a binary neutron star (BNS), and 16 for a neutron star black hole binary (NSBH), i.e.,  $\theta = \{\mathcal{M}_c, \eta, d_L, \theta, \phi, \iota, \psi, t_c, \Phi_c, \chi_{1,x}, \chi_{2,x}, \chi_{1,y}, \chi_{2,y}, \chi_{1,z}, \chi_{2,z}, \Lambda_1, \Lambda_2\}$  (see, e.g., Maggiore 2007), where:  $\mathcal{M}_c$  denotes the detector-frame chirp mass,  $\eta$  is the symmetric mass ratio,  $d_L$  is the luminosity distance to the source,  $\theta$  and  $\phi$  are the sky position coordinates, defined as  $\theta = \pi/2 - \delta$  and  $\phi = \alpha$  (with  $\alpha$  and  $\delta$  R.A. and decl., respectively),  $\iota$  is the inclination angle of the binary orbital angular momentum with respect to the line of sight,  $\psi$  is the polarization angle,  $t_c$  is the time of coalescence,  $\Phi_c$  is the phase at coalescence,  $\chi_{i,c}$  is the dimensionless spin of the object  $i = \{1, 2\}$  along the axis  $c = \{x, y, z\}$ , and  $\Lambda_i$  is the dimensionless tidal deformability of the object  $i$ , which is zero for a BH. GWFAST expresses the chirp mass in units of solar masses,  $M_\odot$ , the luminosity distance in gigaparsecs, the time of coalescence as a GPS time, in seconds, and all of the angles in radians. Also including a small eccentricity in the orbit, there is one more parameter to consider,  $e_0$ , which denotes the eccentricity at a given reference frequency.<sup>3</sup>

<sup>3</sup> The reference frequency is chosen when initializing the waveform model, through the argument `fRef_ecc`. The default is `fRef_ecc = None`, meaning the minimum frequency of the frequency grid is used.

To use GWFAST, the parameters  $\theta$  of the events in the catalog to analyze have to be stored in a dictionary, with the same keys as in the following example:

**Code example 1.** Input parameters for waveform models and FIM calculations in GWFAST.

```
import numpy as np
{ events = { 'Mc':np.array([...]), 'eta':np.array
  ([...]), 'dL':np.array([...]),
  'theta':np.array([...]), 'phi':np.array([...]),
  'iota':np.array([...]),
  'psi':np.array([...]), 'tGPS':np.array([...]), 'Phi-
  coal':np.array([...]),
  'chilz':np.array([...]), 'chi2z':np.array([...]),
  'chilx':np.array([...]), 'chi2x':np.array([...]),
  'chily':np.array([...]), 'chi2y':np.array([...]),
  'Lambda1':np.array([...]), 'Lambda2':np.array([...]),
  'ecc':np.array([...])
}
```

It is also possible to pass some entries with different commonly used parameterizations, namely:

- (a) the sky position coordinates can be given in terms of R. A.,  $\alpha$ , and decl.,  $\delta$ , always in radians, in place of  $\theta$  and  $\phi$ ;
- (b) the time of coalescence can be provided as a Greenwich Mean Sidereal Time (GMST) in days, under the entry name 'tcoal', which takes the place of 'tGPS';
- (c) in the nonprecessing case, one can choose the spin parameters  $\chi_s$ ,  $\chi_a$  instead of  $\chi_{1,z}$ ,  $\chi_{2,z}$ , defined as

$$\chi_s = \frac{1}{2}(\chi_{1,z} + \chi_{2,z}), \quad \chi_a = \frac{1}{2}(\chi_{1,z} - \chi_{2,z}); \quad (1)$$

- (d) it is possible to use the combinations of the tidal deformabilities  $\tilde{\Lambda}$ ,  $\delta\tilde{\Lambda}$  in place of  $\Lambda_1$ ,  $\Lambda_2$ , whose definitions are (Wade et al. 2014)

$$\tilde{\Lambda} = \frac{8}{13}[(1 + 7\eta - 31\eta^2)(\Lambda_1 + \Lambda_2) + \sqrt{1 - 4\eta}(1 + 9\eta - 11\eta^2)(\Lambda_1 - \Lambda_2)], \quad (2a)$$

$$\delta\tilde{\Lambda} = \frac{1}{2} \left[ \sqrt{1 - 4\eta} \left( 1 - \frac{13,272}{1319}\eta + \frac{8944}{1319}\eta^2 \right) (\Lambda_1 + \Lambda_2) + \left( 1 - \frac{15,910}{1319}\eta + \frac{32,850}{1319}\eta^2 + \frac{3380}{1319}\eta^3 \right) \times (\Lambda_1 - \Lambda_2) \right]; \quad (2b)$$

- (e) if using a waveform model that includes the contribution of unaligned spin components (precessing spins) it is possible to substitute the entries  $\iota$ ,  $\chi_{1,x}$ ,  $\chi_{2,x}$ ,  $\chi_{1,y}$ ,  $\chi_{2,y}$ ,  $\chi_{1,z}$ ,  $\chi_{2,z}$  with  $\theta_{JN}$ ,  $\chi_1$ ,  $\chi_2$ ,  $\theta_{s,1}$ ,  $\theta_{s,2}$ ,  $\phi_{JL}$ ,  $\phi_{1,2}$ . These are, respectively, the angle between the total angular momentum and the line of sight,  $\theta_{JN}$ , the magnitudes of the spin vectors,  $\chi_i$ , the angles between the spin vectors and the orbital angular momentum,  $\theta_{s,i}$ , the azimuthal angle of the orbital angular momentum and the total angular momentum,  $\phi_{JL}$ , and the difference in azimuthal angle between the two spin vectors,  $\phi_{1,2}$ .

A summary of the parameters, their physical symbol, and their name in GWFAST is provided in Table 1.

## 2.2. Waveform Models

We use Fourier domain waveform models. In particular, at the time of writing, the code implements some selected

waveform models in Python, and an interface to the LIGO Algorithm Library, LAL, which allows us to use all of the waveforms available in that library.

In particular, the following waveform models are directly available in GWFAST in a Python implementation:

- (i) a restricted post-Newtonian (PN) waveform model (Buonanno et al. 2009; Ajith 2011; Mishra et al. 2016), also with its tidal (Wade et al. 2014) and moderate eccentric (Moore et al. 2016) extensions. This is an inspiral-only waveform, but can still be used to describe signals coming from BNS mergers, whose major contribution to the S/N comes from the inspiral. There is no limitation in the parameters range, except for the eccentricity, which cannot exceed  $e_0 \sim 0.1$  for comparable mass systems;<sup>4</sup>
- (ii) a full inspiral-merger-ringdown waveform model (Husa et al. 2016; Khan et al. 2016), tuned with numerical relativity (NR) simulations, which can efficiently be used to simulate signals coming from BBH mergers, with non-precessing spins up to  $|\chi_z| \sim 0.85$  and mass ratios up to  $q = m_1/m_2 \sim 18$ ;
- (iii) tidal extension of the previous model (Dietrich et al. 2019), which can be used to accurately describe signals coming from BNS mergers. It includes spin terms up to higher order and a filter to terminate the waveform after merger. The validity has been assessed for masses ranging from 1–3  $M_\odot$ , spins up to  $|\chi_z| \sim 0.6$ , and tidal deformabilities up to  $\Lambda_i \simeq 5000$ ;
- (iv) full inspiral-merger-ringdown waveform model (London et al. 2018; Kalaghatgi et al. 2020), which takes into account not only the quadrupole of the signal, but also the subdominant multipoles  $(l, m) = (2, 1), (3, 2), (3, 3), (4, 3)$ , and  $(4, 4)$ , which can be particularly relevant to better describe the signal coming from BBH systems. The calibration range is the same of the IMRPhenomD model;
- (v) full inspiral-merger-ringdown waveform model (Pannarale et al. 2015; Dietrich et al. 2019), which can describe the signal coming from the merger of an NS and a BH, with mass ratios up to  $q \sim 100$ , also taking into account tidal effects and the impact of the possible tidal disruption of the NS.

These waveform models have been translated from their C implementation in the LIGO Algorithm Library (LAL) into a pure Python version. We carefully checked that our Python implementation accurately reproduces the original LAL waveforms, as can be seen on some example events in Figures 2 and 3.<sup>5</sup>

The implementation in Python has two advantages. First, it allows GWFAST to fully exploit the capabilities of this language to vectorize the computation on multiple events at a time, which would be impossible if we had to interact with a code written in C such as LAL. Second, it allows for the possibility of using AD (and in particular the library JAX) to compute derivatives; see Section 3.3.

It is also possible to use the waveform module separately from the rest of the code. For example, in order to generate the

<sup>4</sup> The reference cut frequency of the waveform is set by default to twice the binary innermost stable circular orbit frequency,  $f_{\text{ISCO}} = 1/(2\pi \cdot 6\sqrt{6} GM_{\text{tot}}/c^3)$ . The tidal and eccentric terms can be included through the Booleans `is_tidal` and `is_eccentric`, respectively (see also footnote 1).

<sup>5</sup> Some bigger fluctuations arise when going closer to the merger, but this can be explained by small differences among C and Python in some interpolations needed to compute the ringdown frequency, as we prove below.

**Table 1**  
Summary of the Parameters Used in GWF<sub>AST</sub> to Describe the GW Signal

Parameter Symbol	Parameter Description	Name in GWF <sub>AST</sub>	Units in GWF <sub>AST</sub>	Physical Range
$\mathcal{M}_c$	detector-frame chirp mass	'Mc'	$M_\odot$	$(0, +\infty)$
$\eta$	symmetric mass ratio	'eta'	...	$(0, 0.25]$
$d_L$	luminosity distance	'dL'	Gpc	$(0, +\infty)$
$\theta, \phi/\alpha, \delta^{(a)}$	sky position	'theta', 'phi'/ 'ra', 'dec'	rad	$[0, \pi], [0, 2\pi]/$ $[0, 2\pi], [-\pi/2, \pi/2]$
$\iota^{(e)}$	inclination angle w.r.t. orbital angular momentum	'iota'	rad	$[0, \pi]$
$\psi$	polarization angle	'psi'	rad	$[0, \pi]$
$t_c^{(b)}$	time of coalescence GPS/GMST	'tGPS'/tcoal'	s/day	$[0, +\infty)/[0, 1]$
$\Phi_c$	phase at coalescence	'Phicoal'	rad	$[0, 2\pi]$
$\chi_{i,c}^{(c),(e)}$	spin component of object $i = \{1, 2\}$ along axis $c = \{x, y, z\}$	'chi1x', 'chi1y', 'chi1z' 'chi2x', 'chi2y', 'chi2z'	...	$[-1, 1], (S^2)$
$\Lambda_i^{(d)}$	adimensional tidal deformability of object $i = \{1, 2\}$	'Lambda1', 'Lambda2'	...	$[0, +\infty)$
$e_0$	orbital eccentricity	'ecc'	...	$[0, 1)$
$\chi_s, \chi_a^{(c)}$	symmetric and asymmetric spin components; see Equation (1)	'chiS', 'chiA'	...	$[-1, 1], [-1, 1]$
$\tilde{\Lambda}, \delta\tilde{\Lambda}^{(d)}$	adimensional tidal deformability combinations; see Equation (2a)	'LambdaTilde', 'deltaLambda'	...	$[0, +\infty),$ $(-\infty, +\infty)$
$\theta_{JN}^{(e)}$	inclination angle w.r.t. total angular momentum	'thetaJN'	rad	$[0, \pi]$
$\chi_i^{(e)}$	spin magnitude of object $i = \{1, 2\}$	'chi1', 'chi2'	...	$[0, 1]$
$\theta_{s,i}^{(e)}$	spin tilt of object $i = \{1, 2\}$	'tilt1', 'tilt2'	rad	$[0, \pi]$
$\phi_{JL}^{(e)}$	azimuthal angle between orbital and total angular momentum	'phiJL'	rad	$[0, 2\pi]$
$\phi_{1,2}^{(e)}$	difference in azimuthal angle between the spin vectors	'phi12'	rad	$[0, 2\pi]$

**Note.** The first column reports the symbol used to denote a parameter, the second denotes a brief description of its physical meaning, the third depicts its name in GWF<sub>AST</sub>, the fourth shows the physical units of the parameter adopted in GWF<sub>AST</sub>, and the last column denotes its physical range. Parameters describing the same physical quantities, which thus have to be provided alternatively, are followed by a superscript in the first column, matching the one reported in the list in Section 2.1.  $S^2$  in the  $\chi_{i,c}$  stresses that the three components of a spin vector are not independent, but defined on a sphere, i.e.,  $\chi_{i,x}^2 + \chi_{i,y}^2 + \chi_{i,z}^2 \leq 1$ .

IMRPhenomD waveform amplitude and phase for a given set of events, it is sufficient to run the following sequence of commands:

**Code example 2.** Calculation of waveforms in GWF<sub>AST</sub> and WF4Py. As an illustration, we use the IMRPhenomD model.

```
import waveform as wf
# initialize the waveform, here we choose as an example
IMRPhenomD
mywf = wf.IMRPhenomD()
# compute the cut frequencies
fcut = mywf.fcut(**events)
# initialize the frequency grids from 2 Hz to fcut, with
1000 points per grid
fgrids = np.geomspace(np.full(fcut.shape, 2), fcut,
num = 1000)
# compute the amplitude and phase, respectively
```

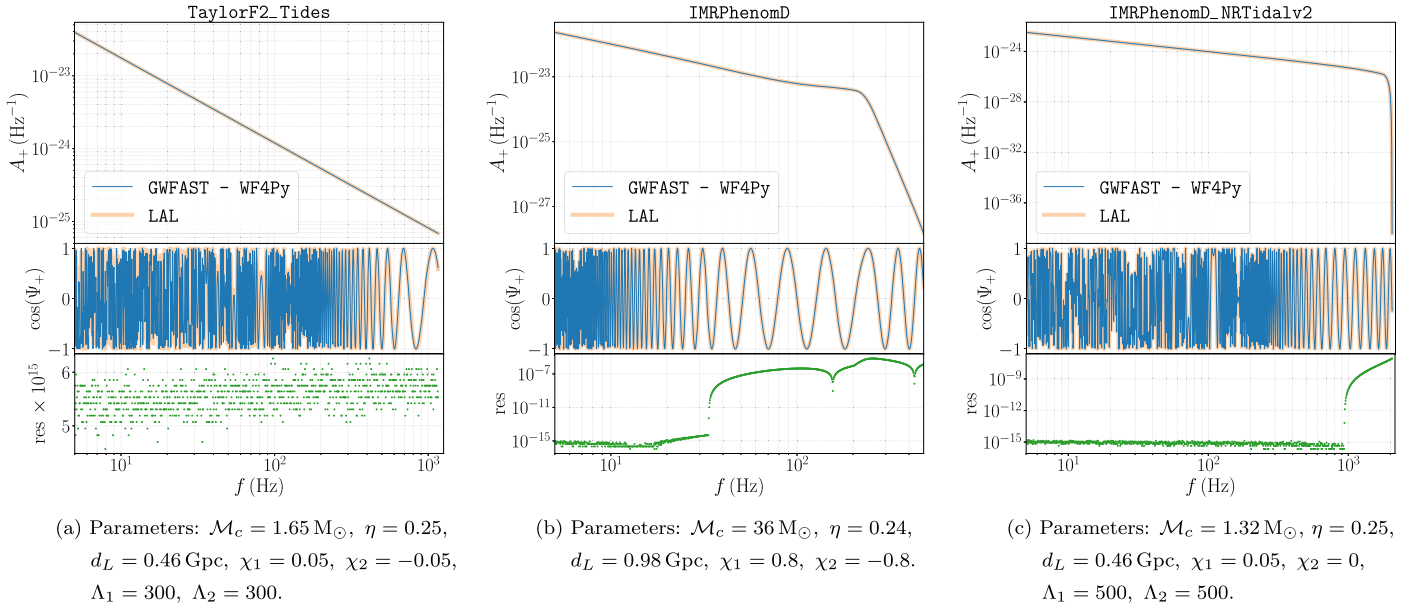
(Continued)

```
myampl = mywf.Ampl(fgrids, **events)
myphase = mywf.Phi(fgrids, **events)
```

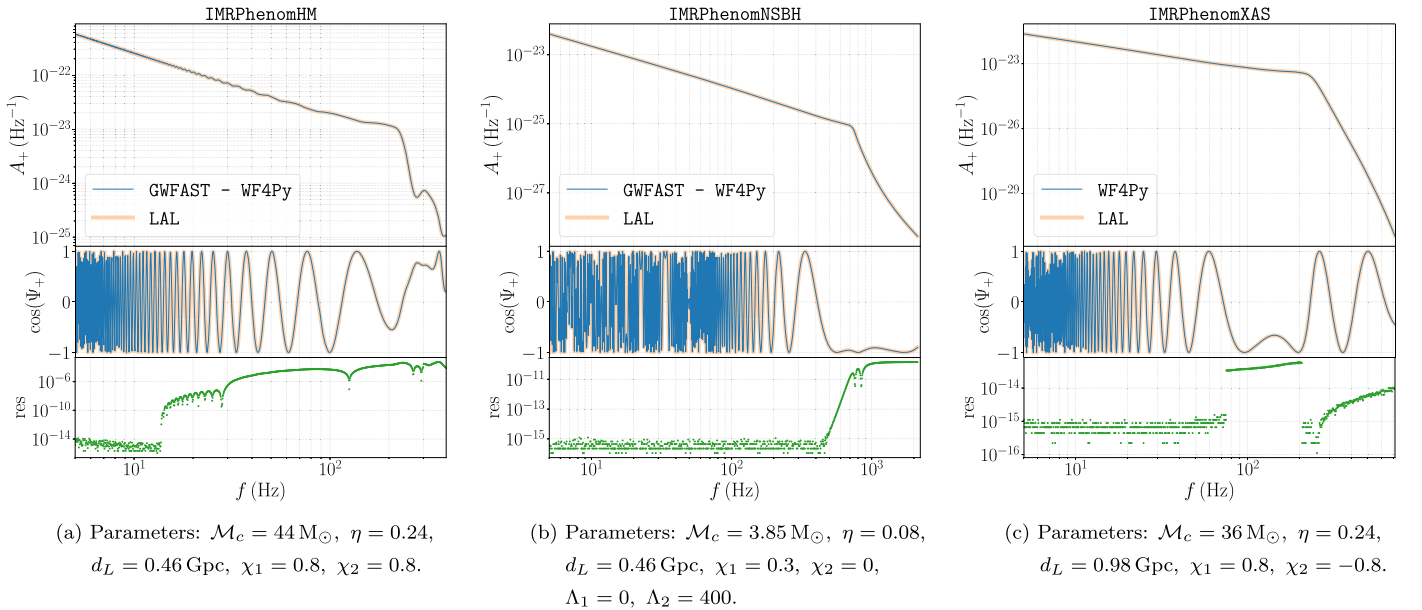
This small piece of code shows that, as GWF<sub>AST</sub>'s waveforms are written in pure Python and fully vectorized, our software does not have to rely on for loops over the events, as in a code interacting with C. Note that the order of the entries in the events dictionary is arbitrary.

All of our waveforms also include a routine to compute the time to coalescence as a function of frequency, needed to take into account Earth's rotation in the computation of the strain, which includes terms up to 3.5 PN order,<sup>6</sup> called tau\_star, and a function to compute the cut frequency for the given waveform, so to properly build the frequency grid, called

<sup>6</sup> As in Buonanno et al. (2009), Equation (3.8b).



**Figure 2.** Comparison of the waveform models TaylorF2\_RestrictedPN (left panel), IMRPhenomD (central panel), and IMRPhenomD\_NRTidalv2 (right panel) obtained from LAL and GWFAST—WF4Py for example events. The upper and central panels of both figures show the “+” GW amplitude and cosine of the phase obtained with the two codes superimposed, while in the lower panel, we report the relative difference (“residual”) among the two amplitudes.



**Figure 3.** The same as in Figure 2 for the waveform models IMRPhenomHM (left panel), IMRPhenomNSBH (central panel), and IMRPhenomXAS (right panel).

fcut, as seen in the above example. Waveform objects in GWFAST contain the attribute `ParNums` giving a dictionary of the form `{'name_of_parameter':position}`, with ‘name\_of\_parameter’ being a string with the parameter name as in Table 1 and `position` being an int corresponding to the position of the parameter in the Fisher matrix.

Apart from their implementation in GWFAST, which includes some features specific for JAX compatibility, we publicly release a pure numpy and ready-to-use version of the waveform models alone, `WF4Py`.<sup>7</sup> The syntax for using waveforms in this library is the same as in the example above. This module further

implements the waveform model `IMRPhenomXAS` (Pratten et al. 2020), which is a full inspiral-merger-ringdown model tuned for the fundamental mode of BBH systems with aligned spins and mass ratios ranging from 1 to 1000, among the last to be released.<sup>8</sup>

<sup>7</sup> <https://github.com/CosmoStatGW/WF4Py>

<sup>8</sup> Note that the comparison of this waveform model with its LAL implementation explains the larger fluctuations in the difference between `WF4Py` and LAL arising close to the merger for the other full inspiral-merger-ringdown waveforms; see Figures 2 and 3. For `IMRPhenomXAS`, the ringdown frequency is computed from an analytical expression rather than an interpolation of a precomputed table, and the adherence of `WF4Py` to the LAL original is close to machine precision, as can be seen from the right panel of Figure 3.

Finally, all waveform models available in LAL can be accessed in GWFAST through the wrapper class `LAL_WF`, which can be used as follows:

**Code example 3.** How to use LAL waveforms in GWFAST. As an illustration, we use the `IMRPhenomXPHM` model.

---

```
myLALwf = wf.LAL_WF('IMRPhenomXPHM', is_tidal = False,
                    is_higherModes = True, is_precessing = True,
                    is_eccentric = False)
```

---

where the first entry has to be a string containing the name of the chosen waveform as in the LAL library<sup>9</sup>—`'IMRPhenomXPHM'` in this example—and the Booleans `is_tidal`, `is_higherModes`, `is_precessing`, and `is_eccentric` are used to specify whether the model includes tidal effects, the contribution of higher-order harmonics, precessing spins, or eccentricity (see also footnote 1), respectively.

### 3. Detector Response and Fisher Matrix

The core modules of GWFAST are `signal` and `network`. The former allows us to model the response of a single detector to a GW signal, while the latter collects multiple detectors constituting a network. Both modules include in particular methods to compute the S/N and the FIM. GWFAST fully accounts for the motion of the Earth during the time the signal remains in the detection band; see Iacovelli et al. (2022) for a detailed discussion.<sup>10</sup>

#### 3.1. Single Detector

A signal object can be initialized from the class `GWSignal` as follows:

**Code example 4.** Initialization of objects characterizing single detectors in GWFAST for a user-specified location, orientation, shape, and path to the PSD file.

---

```
import signal
Virgo = signal.GWSignal(mywf, psd_path = 'path/to/Virgo/
    psd', detector_shape = 'L', det_lat = 43.6, det_
    long = 10.5, det_xax = 115.)
LIGO_L = signal.GWSignal(mywf, psd_path = 'path/to/LIGO/
    L1/psd', detector_shape = 'L', det_lat = 30.6, det_
    long = -90.8, det_xax = 243.)
```

---

where `det_lat` and `det_long` denote the latitude and longitude of the detector in degrees, respectively, `xax` is the angle between the bisector of the detector arms and the east in degrees, and `detector_shape` is its shape, which can be either `'L'` (for L-shaped detectors) or `'T'` (for triangular-shaped detectors). A triangular-shaped detector is defined as three colocated detectors with an opening angle of  $60^\circ$  in a closed-loop configuration.

Other options can be passed when initializing the `GWSignal` object, in particular: the `useEarthMotion` Boolean is used to turn on and off the computation of the

<sup>9</sup> Note that, in case of incorrect input, the code will print a list of all of the available waveform models in the frequency domain.

<sup>10</sup> In particular, this enters in the pattern functions and through a Doppler shift in the phase, and can give a strong contribution in particular for BNS signals, that, for 3G detectors, can stay in band up to  $\mathcal{O}(1)$  day).

effect of Earth's rotation; `fmin` and `fmax` can be used to set the minimum and maximum of the frequency grid (in hertz) and have default values `fmin = 2.0` and `fmax = None` (meaning that the grid extends up to the cut frequency of the waveform); `DutyFactor` can be used to set the duty factor of the detector, i.e., the fraction of time each detector is supposed to be operational, between 0 and 1 (default is `None`, meaning no duty cycle is considered). For triangular-shaped detectors, the duty factor refers to each of the three components of the triangle separately.

The entry `psd_path` is the path to the file containing its amplitude spectral density (ASD) or power spectral density (PSD); the flag `is_ASD` can be used to specify whether the given one is a PSD or ASD. GWFAST contains the following publicly available ASDs in the folder `data/psds/`:

1. the sensitivities from Abbott et al. (2017; last update in January 2020), which can be found at <https://dcc.ligo.org/LIGO-T1500293/public>, in the folder `unofficial_curves_all_dets`;
2. the representative sensitivities of the LIGO and Virgo detectors during the observing runs O1 and O2 (<https://dcc.ligo.org/P1800374/public/>), O3a (<https://dcc.ligo.org/LIGO-P2000251/public>), and O3b (estimated using PyCBC around the times reported in the caption of Figure 2 of Abbott et al. 2021a), in the folder `LVC_O1O2O3`;
3. the sensitivities adopted in Abbott et al. (2020a) for the LIGO, Virgo, and KAGRA detectors during the O3, O4, and O5 runs (<https://dcc.ligo.org/LIGO-T2000012/public>), in the folder `observing_scenarios_paper`;
4. the official ET–D sensitivity curve, from the document `ET-0000A-18.txt` (<https://apps.et-gw.eu/tds/?content=3&r=14065>); and
5. the latest sensitivity curves for CE, used in Srivastava et al. (2022), for various detector configurations (<https://dcc.cosmicexplorer.org/CE-T2000017/public>), in the folder `ce_strain`.

GWFAST also contains some predefined detector configurations in the module `globals`. These are listed, together with their acronyms, in Table 2. The locations of the current detectors are taken from Gossan et al. (2022), while the CE sites are taken from Borhanian (2021) as illustrative examples. Predefined detector configurations can be easily imported from `globals`. In the following, we show how to initialize one signal corresponding to one CE at the Hanford site, with orientation  $\gamma = 0$ :

**Code example 5.** Initialization of objects characterizing single detectors in GWFAST using predefined detector configurations.

---

```
import copy
import os
import gwfastGlobals as glob
# copy the location and orientation of Hanford
CEH_conf = copy.deepcopy(glob.detectors).pop('H1')
# set the detector PSD using the latest curve for CE1
CEH_conf['psd_path'] = os.path.join(glob.detPath,
    'ce_strain', 'cosmic_explorer.txt')
# Set the orientation angle to 0
CEH_conf['xax'] = 0
# Initialise the GWSignal object
CEH = signal.GWSignal(mywf, psd_path = CEH_conf
    ['psd_path'], detector_shape = CEH_conf['shape'],
```

**Table 2**  
Summary of the Positions, Orientations, Angle between Arms, Shapes, and Acronyms of the Detectors Available in GWFAST

Detector	Latitude $\lambda$	Longitude $\varphi$	Orientation $\gamma$	Arms Aperture $\zeta$	Shape	Name in GWFAST
LIGO Hanford, USA	46°5	−119°4	171°	90°	'L'	'H1'
LIGO Livingston, USA	30°6	−90°8	242°7	90°	'L'	'L1'
Virgo, Cascina, IT	43°6	10°5	115°6	90°	'L'	'Virgo'
KAGRA, Hida, JP	36°4	137°3	15°4	90°	'L'	'KAGRA'
LIGO India, Hingoli, IN	19°6	77°0	287°4	90°	'L'	'LIGOI'
ET Sardinia, IT	40°5	9°4	0°	60°	'T'	'ETS'
ET Meuse–Rhine, EU	50°7	5°9	0°	60°	'T'	'ETMR'
CE1 Idaho, USA	43°8	−112°8	−45°	90°	'L'	'CE1Id'
CE2 New Mexico, USA	33°2	−106°5	−105°	90°	'L'	'CE2NM'
CE2 New South Wales, AU	−34°	145°	0°	90°	'L'	'CE2NSW'

**Note.** Using user-defined configurations is straightforward (see the text). The orientation  $\gamma$  denotes the angle between the bisector of the arms (the first arm in the case of a triangle) and East.

(Continued)

```
det_lat = CEH_conf['lat'], det_long = CEH_conf['long'],
det_xax = CEH_conf['xax']
```

Any other user-defined configuration can easily be added as in Code example 4. With the object of type `GWSignal` initialized, the user can easily compute all of the quantities characterizing the signal. In particular, from the `_PatternFunction` function, it is possible to get the pattern functions of the detector; from `GWAmplitudes`, it is possible to compute the '+' and 'x' amplitudes of the signal *at the detector* (i.e., multiplied by the pattern functions and the spherical harmonics), while the full signal strain can be obtained through the function `GWStrain`.

### 3.2. Detector Networks

From more than one `GWSignal` object, one can define a network, which is composed by multiple detectors. The detectors composing the network have to be inserted into a dictionary, which can then be used to initialize an object from the class `DetNet`, characterizing the network:

**Code example 6.** Initialization of the network object in GWFAST from single detector objects.

```
import network
# First collect the signal objects into a dictionary
mySignals = 'V1':Virgo, 'L1':LIGO_L, ...
# Then initialize the network object
myNet = network.DetNet(mySignals)
```

From both the signal and network objects, it is then possible to compute the S/Ns and Fisher matrices for a set of events. The *matched filter* S/Ns in a single detector are computed using the definition

$$S/N_i^2 = 4 \int_{f_{\min}}^{f_{\max}} \frac{|\tilde{h}_{(i)}(f)|^2}{S_{n,i}(f)} df, \quad (3)$$

where  $\tilde{h}_{(i)}(f)$  denotes the GW strain in Fourier domain at the  $i$ th detector, and  $S_{n,i}(f)$  is the noise spectral density of the  $i$ th detector. The network S/N is defined as the sum in quadrature

of the single detectors' S/Ns:

$$S/N^2 = \sum_i S/N_i^2. \quad (4)$$

This can be obtained as simply as:

**Code example 7.** Computation of S/Ns (coded as “SNRs”) in GWFAST both for a single detector and for a network.

```
# If the SNRs in a single detector are needed, e.g., Virgo
SNRsOne = Virgo.SNRInteg(events)
# Instead for the network SNRs
SNRsNet = myNet.SNR(events)
```

The output of the methods above is a `numpy` array of the same length of the number of events.

The FIM elements for a single detector are computed from the definition

$$\Gamma_{ij} = 4 \int_0^\infty df \frac{\partial_i \tilde{h} \partial_j \tilde{h}^*}{S_n(f)}, \quad (5)$$

where  $\partial_i$  denotes the derivative with respect to the parameter  $i$ . The FIM for the network is obtained by summing the individual Fisher matrices (which relies on the fact that different detectors are independent<sup>11</sup>). The FIM for a single detector or a network can be obtained with a single function call:

**Code example 8.** Computation of Fisher matrices in GWFAST both for a single detector and for a network.

```
# If the Fisher matrices for a single detector are needed,
# e.g., Virgo
FisherMatrsOne = Virgo.FisherMatr(events)
# Instead to compute them for the network
FisherMatrsNet = myNet.FisherMatr(events)
```

<sup>11</sup> This assumption is considered valid also for the colocated components of a triangular-shaped detector. In particular, for a triangular configuration, the output for both the S/N and FIM returned by the methods of the class `GWSignal` already includes the contribution of all three independent detectors forming the triangle, with the S/N being summed in quadrature and the FIMs being added.

The FIMs are returned by `GWFAST` in the form of a `numpy` array, treated as an array of matrices in the last dimension. For example, an array of FIMs for five BBH events with nine waveform parameters will have dimension (9, 9, 5). In the case of a network, it might be useful to store the S/Ns and Fisher matrices of the single detectors. This can be done passing to the functions `SNR` and `FisherMatr` the flag `return_all = True`. In this case, the output of both functions is a dictionary, with keys corresponding to the detectors (one key for each arm in the case of a triangular-shaped detector) and a key `'net'` for the network S/Ns and FIMs.

The default parameters for which `GWFAST` computes the FIM, in the quasi-circular, nonprecessing, and nontidal case, are, in order,<sup>12</sup>  $\mathcal{M}_c$  in units of  $M_\odot$ ,  $\eta$ ,  $d_L$  in units of gigaparsecs,  $\theta$ ,  $\phi$ ,  $\iota$ ,  $\psi$ ,  $t_c$  in units of seconds,  $\Phi_c$ , and  $\chi_s$ ,  $\chi_a$ . In the case of precessing spins, the FIM is computed for the full set  $\chi_{1,z}$ ,  $\chi_{2,z}$ ,  $\chi_{1,x}$ ,  $\chi_{2,x}$ ,  $\chi_{1,y}$ ,  $\chi_{2,y}$  in this order, and, in the BNS and NSBH case, the tidal parameters  $\tilde{\Lambda}$  and  $\delta\tilde{\Lambda}$  are also included. In the eccentric case, the parameter  $e_0$  is also included, and appears in the Fisher matrix after both spins and tidal parameters. We chose to use the combinations  $(\chi_s, \chi_a)$  instead of  $(\chi_{1,z}, \chi_{2,z})$  in the nonprecessing case so to have two orthogonal parameters, but the FIM can be as well computed in terms of the latter quantities passing the flag `use_chi_chi2 = True` to the `FisherMatr` function. The choice of the combination  $(\tilde{\Lambda}, \delta\tilde{\Lambda})$  in place of  $(\Lambda_1, \Lambda_2)$  is due to the fact that the parameter  $\tilde{\Lambda}$  is much better constrained than the two dimensionless tidal deformabilities separately, as it is the combination entering at 5 PN order in the inspiral signal. It is also possible to compute the FIM in terms of the combination  $(m_1, m_2)$ , i.e., the two component redshifted masses, in units of  $M_\odot$ , instead of  $(M_c, \eta)$ , by passing to the `FisherMatr` function the flag `use_m1m2 = True`. Finally, if the contribution of precessing spins is included, setting the flag `use_prec_ang = True`, instead of  $\iota$  and  $\chi_{i,c}$ , the FIM will be computed in terms of the parameters  $\theta_{JN}$ ,  $\chi_1$ ,  $\chi_2$ ,  $\theta_{s,1}$ ,  $\theta_{s,2}$ ,  $\phi_{JL}$ ,  $\phi_{1,2}$ , which are more commonly used in the context of parameter estimation of GW events.

As an example, to access the values of the  $(d_L, d_L)$  elements of the FIM for all of the events in the dictionary, the user just has to run:

**Code example 9.** How to access specific FIM elements in `GWFAST`.

```

# The parameters are contained in a dictionary in the
# waveform class
pars = mywf.ParNums
print(FisherMatrsNet[pars['dL'], pars['dL'], :])

```

Both of the classes `GWSignal` and `DetNet` also include a function to compute the optimal coordinates for a signal to be seen by the considered detectors (i.e., the location corresponding to the maximum S/N), as a function of the time of coalescence. This is obtained by maximizing the pattern functions, and can be accessed as:

**Code example 10.** How to compute the optimal location of a binary for a network of detectors in `GWFAST` at `GMST = 0` day.

```

best_theta, best_phi = myNet.optimal_location(0)

```

<sup>12</sup> Note that the ordering of the parameters in the FIM is fixed, and does not depend on how they are sorted in the dictionary containing the events. It can be accessed through the `waveform` class used to initialize the network, through its attribute `ParNums`, as shown in Code example 9.

where the time can be provided both as a GMST or as a GPS time, setting the Boolean `is_tGPS = True`. The syntax to compute the optimal location is equivalent for an object of type `GWSignal`.<sup>13</sup>

### 3.3. Signal Derivatives

The computation of the derivatives of the signal with respect to its parameters is the key ingredient of a Fisher code. In its pure `Python` version, `GWFAST` is based on AD (Margosian 2018) as implemented in the library `JAX` (Bradbury et al. 2018). Differently from finite difference techniques, AD exploits the fact that each code function, however complex, is built up from elementary arithmetical operations and simple functions, whose derivatives are well known; thus, by applying the chain-rule repeatedly, it is possible to automatically compute derivatives of arbitrary order near machine precision, with a number of operations comparable to those of the original function. Having a properly written pure `Python` code, which is a fundamental requirement for this technique to work, it is possible to apply AD to get a fast and accurate evaluation of the GW strain derivatives, in a semianalytic way, despite the complexity of the function, and for multiple events at a time.

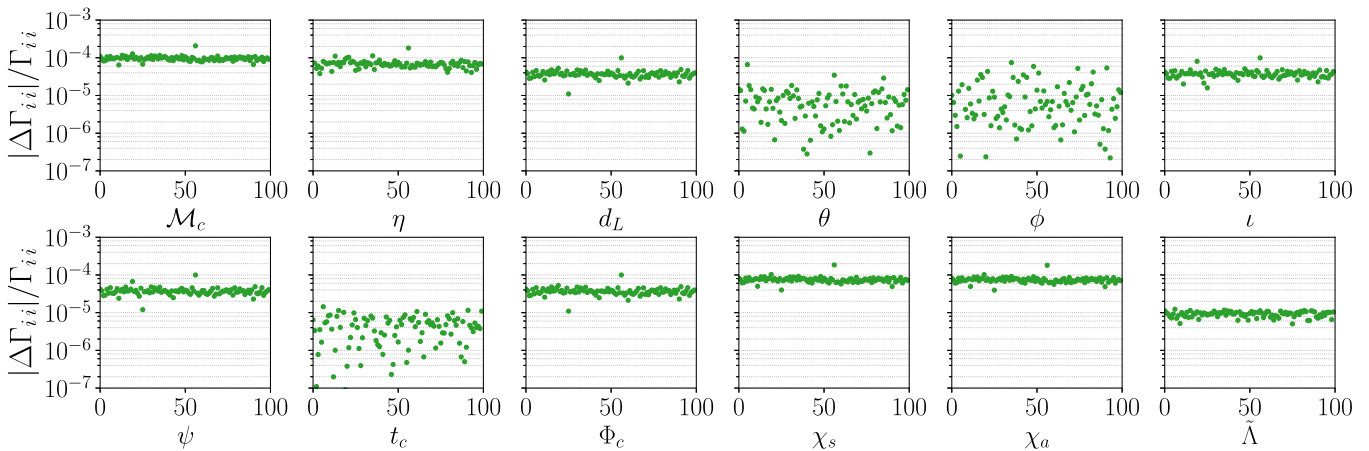
We tested the reliability of `JAX` by comparing the Fisher matrices obtained using the `TaylorF2_RestrictedPN` waveform model with `GWFAST` and an independent code written in `Wolfram Mathematica`, capable of computing derivatives analytically with respect to all parameters. The results for the relative differences of the diagonal elements of the Fisher matrices computed on a sample of 100 events are shown in Figure 4, from which it is possible to see the excellent agreement among the two codes. This test acts on three levels: (1) it proves the absence of bugs, given that the two codes were developed independently, (2) it shows the good behavior of `JAX AD` against an actual analytical computation, and (3) it verifies that integration is correctly performed, given the consistency of the results obtained with two different programming languages, having completely different integration routines.

`GWFAST` also allows the user to compute analytically the derivatives of the signal with respect to many of the parameters, namely  $d_L$ ,  $\theta$ ,  $\phi$ ,  $\iota$ ,  $\psi$ ,  $t_c$ , and  $\Phi_c$  to further speed up the calculation.<sup>14</sup> This can be done passing the flag `computeAnalyticalDeriv = True` to the function `FisherMatr`. We checked that, for these parameters, the analytical results and the result obtained by `JAX` agree at machine precision, i.e.,  $10^{-15}$ . Finally, the FIM for a triangular

<sup>13</sup> Notice that, however, the estimation provided by this function in the case of a network is appropriate only if the detectors have similar characteristics (i.e., PSDs and shape). It is in fact obtained by maximizing the sum in quadrature of the pattern functions, rather than of the full S/Ns, which depends not only on the location of the system, but also on its parameters (which determine the merger frequency), and the detectors' sensitivity curve. Consider, e.g., a network consisting of two detectors: if one of them has better capabilities for observing low-mass systems (i.e., a lower PSD with respect to the other at high frequencies) and the other for high-mass systems (i.e., a lower PSD at low frequencies), a higher S/N will be obtained closer to the optimal location of the former for lighter binaries, and closer to the best location of the latter for heavier ones. Thus, in this case, to estimate the location corresponding to the highest S/N, as a function not only of time, but also of the other parameters of the binary, one has either to perform sampling or maximize the full network S/N for each choice of the binary intrinsic parameters (see, e.g., Section 2.2 of Schutz 2011).

<sup>14</sup> If the waveform model contains the contribution of subdominant modes or precessing spins, the dependence on the parameter  $\iota$  is nontrivial and linked to the modes considered. We thus do not compute the corresponding derivative analytically in this case.





**Figure 4.** Relative difference of the Fisher matrix diagonal elements computed on a sample of 100 events with GWFAST and an independent Wolfram Mathematica code. The labels of the  $x$ -axes refer to the parameters whose diagonal elements are plotted.

detector is computed by using the fact that, for a closed configuration, the sum of the signals is identically zero for geometrical reasons (Freise et al. 2009), which further reduces the computational time by one-third.<sup>15</sup> When instead using waveforms coming from LAL, which are written in C, GWFAST will compute derivatives using the library `numdifftools`,<sup>16</sup> which relies on finite differences techniques. In this case, the computation is performed using the *central differencing scheme* with an adaptive computation of the step size. Both of these choices can be controlled through the arguments `methodNDT` and `stepNDT`, respectively. The finite difference computation can be used alternatively to AD also when exploiting Python waveforms, passing the flag `computeDerivFinDiff = True` to the function `FisherMatr`. Note that, also when using finite differences techniques, derivatives with respect to the parameters  $d_L$ ,  $\theta$ ,  $\phi$ ,  $\iota$ ,  $\psi$ ,  $t_c$ , and  $\Phi_c$  can be performed analytically.

#### 4. Covariance Matrix

In the limit of linear signal (or large S/N) and in presence of Gaussian noise, the inverse of the FIM gives the covariance of the Bayesian posterior probability distribution of the true waveform parameters for a given experiment, assuming a flat prior. The inversion of the matrix can become problematic if the condition number (i.e., the ratio of absolute values of the largest to smallest eigenvalues) is larger than the inverse machine precision. This is the case for highly correlated or nearly degenerate combinations of parameters. In this case, the linear signal approximation might break down in some regions of the likelihood surface (Vallisneri 2008).

##### 4.1. Fisher Matrix Inversion

Tools for obtaining the covariance and analyzing the reliability of the inversion are contained in the module `fisherTools`. Note that all of the functions described here assume that the input FIM is an array of matrices in the last dimension, as described in Section 3.2. The conditioning of the FIM can be checked in GWFAST via the function

<sup>15</sup> We also checked explicitly that the derivatives of the signal in the three arms of a triangular configuration are vanishing (up to machine precision) when computed with GWFAST.

<sup>16</sup> <https://pypi.org/project/numdifftools/>.

`CheckFisher`, which returns the eigenvalues, eigenvectors, and condition number of the matrix. The inversion of the FIM to yield the covariance is done with the function `CovMatr`, as:

**Code example 11.** Computation of covariance matrices from Fisher matrices in GWFAST.

---

```
import fisherTools as fTools
CovMatrsNet, inversion_errors = fTools.CovMatr
(FisherMatrsNet)
```

---

By default, each row and column is normalized to the square root of the diagonal of the FIM before inversion, so that the resulting matrix has adimensional entries with ones on the diagonal and the remaining elements in the interval  $[-1, 1]$  (Harms et al. 2022).<sup>17</sup> The inverse transformation is applied after inversion to yield the inverse of the original matrix. GWFAST also implements a variety of possibilities for the technique used to find the inverse. The Python library for precision arithmetic `mpmath` is used for the inversion. The inversion is not performed if the condition number is larger than a threshold that can be specified by the user via the argument `condNumbMax`. Its default value is  $10^{50}$  (so the code will try to invert every matrix irrespective of the conditioning). The available possibilities are listed below, and can be specified by the argument `invMethodIn`:

- `'inv'`: inverse computed by `mpmath`;
- `'cho'`: inverse computed by means of the Cholesky decomposition, i.e., the (Hermitian, positive-definite) FIM is expressed as a product of a lower triangular matrix and its conjugate transpose, and the latter is inverted. This is the default option in GWFAST;<sup>18</sup>
- `'svd'`: the singular-value decomposition (SVD) of the FIM is used to invert the matrix. In this case, there is the

<sup>17</sup> This transformation is not applied in case the matrix has a zero element on the diagonal.

<sup>18</sup> Note that in some cases, the FIM may be not positive-definite due to the presence of very small eigenvalues that can assume negative values due to numerical fluctuations, in which case the Cholesky decomposition cannot be found. For those matrices, GWFAST resorts by default to an SVD for the inversion, but the error on the inversion in those cases should be carefully checked, and the reliability of the validity of the Fisher approximation may be poor.

additional option of truncating the smallest singular values to the minimum allowed numerical precision, which can help regularize poorly conditioned matrices. This can be required by setting the Boolean `truncate = True`. In this case, for each singular value  $s$ , if the ratio of its absolute value to the absolute value of the largest singular value,  $\max s_i$ , is smaller than a threshold  $\lambda$ , the singular value  $s$  is replaced with  $\lambda \times \max(s_i)$ . The value of the threshold  $\lambda$  can be specified with the argument `svals_thresh`, which is set by default to  $10^{15}$ ;

`'svd_reg'`: the SVD of the FIM is used to invert the matrix, and eigenvalues smaller than the threshold specified by the argument `svals_thresh` are not included in the inversion. This ensures that the error on poorly constrained parameters is not propagated to the other ones (Harms et al. 2022). However, it might result in underestimating the uncertainty for parameters whose eigenvalues are excluded, and the effect should be carefully checked.

`'lu'`: inversion is done by means of the lower–upper decomposition, i.e., the factorization of the FIM into the product of one lower triangular matrix and one upper triangular matrix. This can be a useful option since, as for the Cholesky decomposition, the inversion of a triangular matrix is easier than the one of a full matrix. Differently from the Cholesky decomposition, however, the original matrix does not have to be hermitian and positive-definite, which can make this method more stable against numerical noise for poorly conditioned matrices.

The error on the inversion is computed in GWFAST by the function `compute_inversion_error` with the definition  $\epsilon = \|\Gamma \cdot \Gamma^{-1} - \mathbb{1}\|_{\max} = \max_{ij} |\Gamma \cdot \Gamma^{-1} - \mathbb{1}_{ij}|$ , where  $\mathbb{1}$  denotes the identity matrix,  $\Gamma$  is the FIM, and  $\Gamma^{-1}$  is its inverse as computed by the code:

**Code example 12.** Computation of the inversion errors in GWFAST.

---

```
invErrs = fTools.compute_inversion_error(FisherMatrsNet, CovMatrsNet)
```

---

Two other utilities to check the quality of the inversion are available in GWFAST in the module `fisherTools`. The function `check_covariance` computes the inversion error, and prints the difference between  $\Gamma \cdot \Gamma^{-1}$  and the identity on the diagonal, and the off-diagonal elements of  $\Gamma \cdot \Gamma^{-1}$  exceeding a given threshold specified with the argument `tol`. Second, the function `perturb_Fisher` adds random perturbations to the FIM to a specified decimal (given by the argument `eps`, whose default is  $10^{-10}$ ), and checks if the inversion remains stable.

While the square root of the diagonal elements of the covariance matrix give the expected marginalized  $1\sigma$  errors on the parameters, a useful metric for GW parameter estimation is the sky localization region at some given confidence level. This is computed by (Barack & Cutler 2004; Wen & Chen 2010)

$$\Delta\Omega_{X\%} = -2\pi |\sin\theta| \sqrt{(\Gamma^{-1})_{\theta\theta}(\Gamma^{-1})_{\phi\phi} - (\Gamma^{-1})_{\theta\phi}^2} \times \ln(1 - X/100). \quad (6)$$

The function `compute_localization_region` computes the sky localization region, in square degrees or steradians, according to the previous definition. The desired units can be specified through the `units` key, which can have values `'SqDeg'` and `'Sterad'`, and the confidence level is specified by the optional argument `perc_level` (with default 90%). An example of usage is presented in Code example 13.

#### 4.2. Manipulating the Fisher and Covariance Matrices

The Fisher approach allows us to treat straightforwardly some common situations encountered in parameter estimation, which we summarize here together with a description of their implementation in GWFAST. All functions described in the following belong to the module `fisherTools`:

1. In order to *fix some parameters to their fiducial values*, one has to remove from the FIM (before inverting it) the corresponding rows and columns. This is done with the function `fixParams`, which takes the following arguments (in the order they are listed here): the original matrix, the dictionary specifying the position of each parameter in the FIM (accessible from the waveform object; see the end of Section 2.2 for an explanation), and a list of `string` with names of the parameters to be fixed, with the same names as in Table 1. The function returns the new matrix, and a dictionary of the same form of the input dictionary, with the keys corresponding to the fixed parameters removed, and the remaining rescaled;
2. In order to *add a Gaussian prior on some parameters*, one has to add to the FIM a prior matrix  $P_{ij}$  corresponding to the inverse covariance of the prior. For the moment, GWFAST supports the addition of a diagonal prior matrix. This can be done with the function `addPrior`, which takes as input the original matrix, a list of values to be added on the diagonal of the Fisher (representing thus the inverse covariance of the prior on the corresponding parameter), the dictionary specifying the position of each parameter in the FIM, and the list of names of parameters on which the prior should be added;
3. In order to *marginalize over some parameters*, one has to remove from the covariance matrix (after the inversion of the FIM) the corresponding rows and columns. This can be done again with the function `fixParams` described in the first point.

**Code example 13.** Example of manipulations of the Fisher matrix: fix the spins to their fiducial values, add a Gaussian prior on the angles with standard deviation  $2\pi$ , compute the corresponding covariance, compute the forecasted 90% localization region in square degrees.

---

```
# Fix spins to their fiducial values
FisherMatrsNet_fix_spins, pars_nospin = fTools.fixParams(FisherMatrsNet, pars, ['chilz', 'chi2z'])
# Add Gaussian prior on theta, phi, iota, psi, phicoal
angles = ['theta', 'phi', 'iota', 'psi', 'Phicoal']
priors_vals = np.repeat(1/(2*np.pi**2), len(angles))
FisherMatrsNet_fix_spins_prior = fTools.addPrior(FisherMatrsNet_fix_spins, priors_vals, pars_nospin, angles)
```

---

(Continued)

---

```
# Invert the new FIM
CovMatrsNet_fix_spins_prior,
inversion_errors_fix_spins_prior = fTools.CovMatr
(FisherMatrsNet_fix_spins_prior)
# Compute 90% localization area in square degrees
sky_loc = fTools.compute_localization_region(CovMatrs-
Net_fix_spins_prior, pars_nospin, events['theta'],
perc_level = 90, units = 'SqDeg')
```

---

### 5. Running in Parallel

In addition to the accuracy in the computation of the derivatives, the main advantage of the use of JAX (Bradbury et al. 2018) is that it allows us to vectorize the calculation of the FIM. The typical usage of a code as GWFAST consists of forecasting parameter estimation capabilities for large catalogs of sources, which is clearly a parallel problem. JAX and GWFAST allow us to vectorize the calculation *even on a single CPU*, which can be used in combination with parallelization routines. GWFAST includes the executable `calculate_forecasts_from_catalog.py` that implements such parallelization and is ready to use both on single machines and on clusters.

A catalog has to be stored in `.h5` format in the folder `data/`. A function to save a catalog of the form given in Code example 1 is included in the module `gwfastUtils`:

**Code example 14.** How to save an event catalog in GWFAST.

---

```
from gwfastUtils import save_data
# save events in .h5 format
save_data('file_name.h5', events)
```

---

`calculate_forecasts_from_catalog.py` divides the events in the catalog into batches of size specified by the user with the option `--batch_size`, and splits the calculations assigning a given number of batches to each parallel process. The number of processes is controlled by the option `--npools`. Events in a single batch are computed in vectorized form for each process, which results effectively in a gain of speed of a factor that can be at most equal to the batch size with respect to a nonvectorized implementation. `calculate_forecasts_from_catalog.py` allows both the use of multiprocessing (McKerns et al. 2012) on a single machine and the use of MPI (Dalcin & Fang 2021) on clusters. The usage is as follows:

**Code example 15.** How to run GWFAST on a catalog of events through the script `calculate_forecasts_from_catalog.py`.

---

```
> mkdir my_results
> python calculate_forecasts_from_catalog.py
--fout = my_results --fname_obs FNAME_OBS [--wf_model
WF_MODEL] [--batch_size BATCH_SIZE]
[--npools NPOOLS] [--snr_th SNR_TH] [--idx_in IDX_IN]
[--idx_f IDX_F] [--fmin FMIN]
[--fmax FMAX] [--compute_fisher COMPUTE_FISHER] [--net
NET [NET ...]] [--rot ROT]
[--netfile NETFILE] [--psds PSDS [PSDS ...]] [--mpi MPI]
[--duty_factor DUTY_FACTOR]
```

---

(Continued)

---

```
[--params_fix PARAMS_FIX [PARAMS_FIX ...]] [--lalargs
LALARGS [LALARGS ...]]
[--return_all RETURN_ALL] [--seeds SEEDS [SEEDS ...]]
```

---

The options are as follows:

`fout`: string; path to output folder, which has to exist before the script is launched;

`fname_obs`: string; name of the file containing the catalog *without* the extension `.h5`;

`wf_model`: string; name of the waveform model, default is 'tf2'. Options are: 'tf2', 'tf2\_tidal', 'tf2\_ecc', 'IMRPhenomD', 'IMRPhenomD\_NRTidalv2', 'IMRPhenomHM', and 'IMRPhenomNSBH'. It is also possible to choose all other waveform models available in LAL, by passing 'LAL-wfname', where `wfname` is the name of the chosen waveform in LAL, e.g., 'LAL-IMRPhenomXPHM';

`batch_size`: int, default is 1; size of the batch to be computed in vectorized form on each process;

`npools`: int, default is 1; number of parallel processes;

`snr_th`: float, default is 12.; threshold value for the S/N to consider the event detectable. FIMs are computed only for events with S/N exceeding this value;

`idx_in`: int, default is 0; index of the event in the catalog from which to start the calculation;

`idx_f`: int, default is -1 (meaning all events); index of the event in the catalog from which to end the calculation;

`fmin`: float, default is 2.; minimum frequency of the grid in hertz;

`fmax`: float, default is None; maximum frequency in hertz. If not specified, coincides with the cut frequency of the waveform;

`compute_fisher`: int, default is 1; if 0, only S/Ns are computed; if 1 the code also computes FIMs;

`net`: list of string, default is ['ETS']; the network of detectors chosen. Predefined configurations are passed using the names in Table 2 separated by *single spacing*. Other configurations can be added directly to the dictionary detectors in the module `gwfastGlobals`. Alternatively, one can pass a custom configuration with the option `netfile`;

`psds`: list of string, default is ['ET-0000A-18.txt']; the paths to PSDs of each detector in the network inside the folder `psds/`, separated by *single spacing*;

`netfile`: alternative to the use of `net` and `psds` to configure the detector network; a dictionary containing the configuration can be saved in `.json` format and passed as input. It is possible to save a network configuration as:

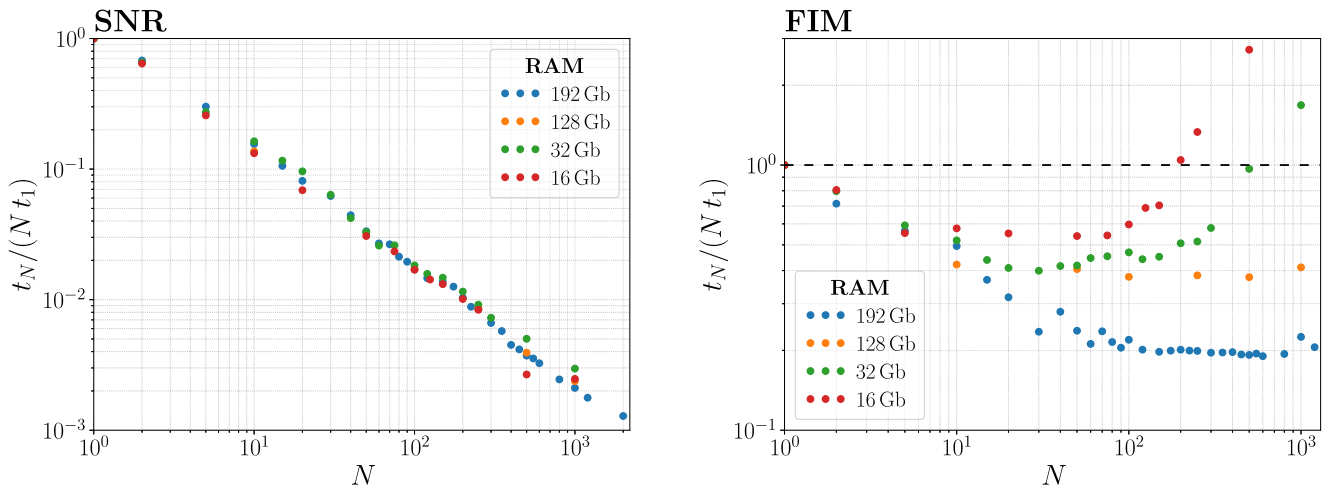
**Code example 16.** How to save a detector network configuration for GWFAST.

---

```
from gwfastUtils import save_detectors
my_network = {'my_detector_1': {'lat': ..., 'long': ...,
'xax': ...,
'shape': ..., 'psd_path': 'path/to/psd'},
'my_detector_2': {...}
}
save_detectors('network_file_name.json', my_network)
```

---

then, send run with `--netfile = network_file_name.json`;



**Figure 5.** Ratio of the time needed to compute S/Ns (left) and Fisher matrices (right) vectorizing on  $N$  events, and the time needed to perform the computation serially, with a `for` loop (equivalent to  $N$  times the time needed for a single evaluation). The different colors refer to the results obtained using machines with different characteristics, as reported in the legend.

`mpi`: int, default is 1; if 0, the code parallelizes using multiprocessing; if 1, it parallelizes using MPI, which is suitable for clusters. In this case, the function should be called accordingly, e.g.,

**Code example 17.** How to parallelize a run using MPI.

```
> mpirun -n 4 python calculate_forecasts_from_catalog.py
... --mpi = 1 --npools = 4
```

`duty_factor`: float  $\in [0, 1]$ , default is 1.; duty factor of the detectors. This is applied separately to each detector in the network (and to each component separately in the case of a triangular configuration);

`params_fix`: list of string, default is []; parameters to fix to the fiducial values, i.e., to eliminate from the FIM;

`rot`: int, default is 1; if 0 the effect of the rotation of the Earth is *not* included in the analysis; if 1 it is included;

`lalargs`: list of string, default is []; specifications of the waveform when using LAL interface. This has to contain 'HM' if the waveform includes the contribution of higher-order modes, 'tidal' if it contains tidal effects, 'precessing' if it includes precessing spins, and 'eccentric' if it includes eccentricity.

`return_all`: int, default is 1; if 1, in case a network of detectors is used, the S/Ns and Fisher matrices of the individual detector are also stored.

`seeds`: list of int, default is []; list of seeds to set for the duty factors in individual detectors, to make the results easily reproducible.

To show the performance gain using vectorization, we report in Figure 5 the ratio  $t/(Nt_1)$  among the time  $t$  needed to compute S/Ns and Fisher matrices on  $N$  events at the same time on the same CPU, and  $N$  times the time  $t_1$  needed to compute the same quantities for 1 event (which is the time needed using a `for` loop). From the left panel, referring to the S/Ns, the impressive gain brought by vectorization is apparent, with an amount of time needed for the computation that stays basically constant while enlarging the batch size, thus effectively being  $N$  times faster than a loop-based computation. Quantifying the advantage from vectorization when computing Fisher matrices is instead more subtle. As is apparent from the

right panel of Figure 5, for  $N \gtrsim 10$ , the behavior has a dependence on the characteristics of the machine used to run the code. Differently from S/Ns, Fisher matrices need much more memory to be allocated during the computation, especially when vectorizing on  $N$  events, and the operations can become much slower on these large arrays, eventually leading to a loss in terms of speed as compared to a serial computation, which instead handles smaller arrays. In any case, there is always an optimal batch size, depending on the machine's characteristics, such that the gain in terms of speed thanks to vectorization can be as large as a factor of  $\sim 5$ .

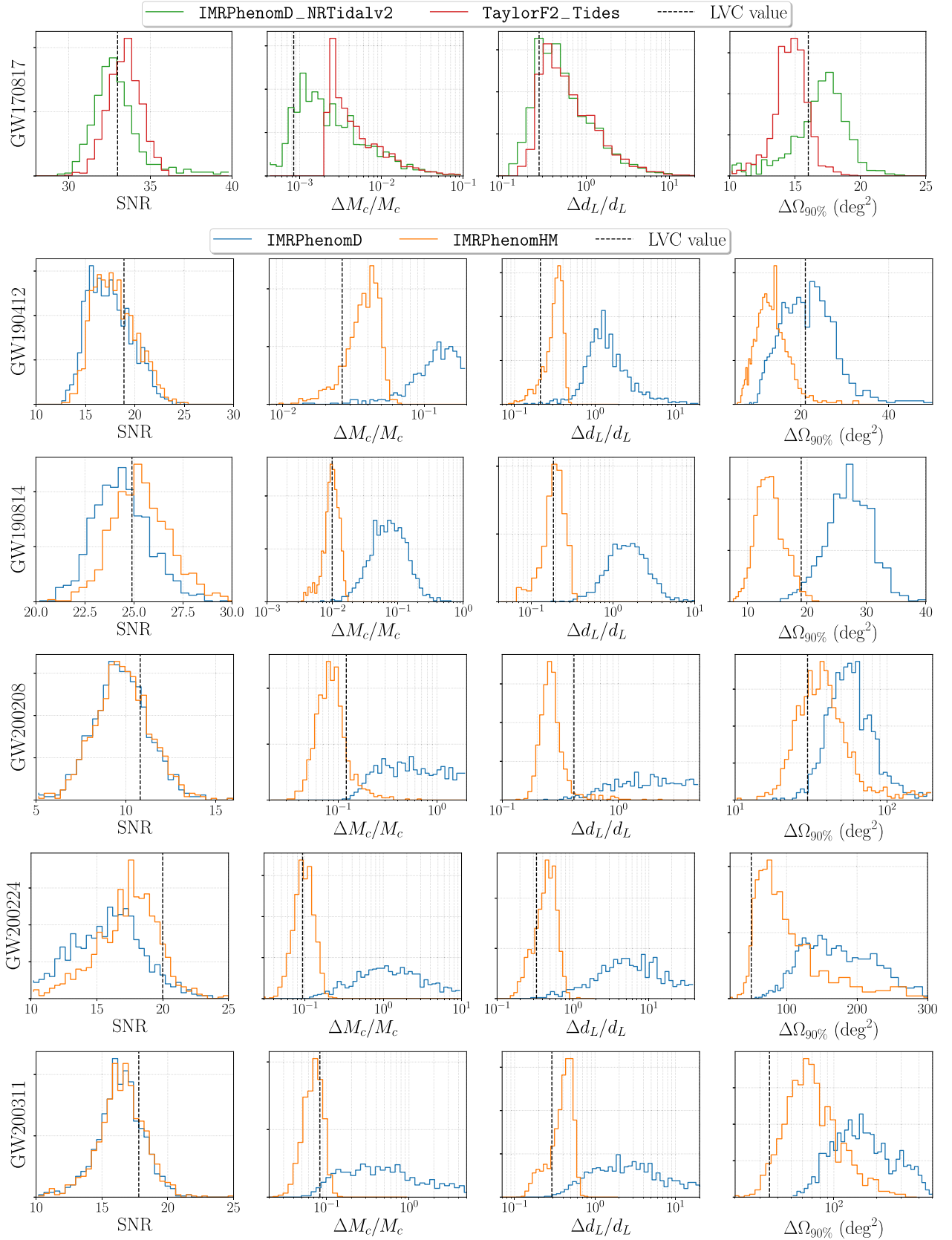
## 6. Comparison with Real Events

As an illustration of the reliability of GWF<sub>AST</sub>, we checked how its predictions compare to the S/Ns and measurement errors associated to real events. This is a stringent test since, for currently available events, many assumptions on which the FIM approximation is based are of course not valid, in particular the Gaussianity of the noise and the limit of high S/N. Here we show that we can still reproduce correctly the order of magnitude of S/Ns and relative errors, taking into account the broad measurement errors of the real events. We selected a subset of the GW events in GWTC-1, GWTC-2, and GWTC-3, with  $S/N \geq 10$  and sky localization  $\Delta\Omega_{90\%} \leq 50 \text{ deg}^2$ .<sup>19</sup> For each selected event, we extracted 1000 samples from the full set of posterior samples<sup>20</sup> for each parameter. We used ASDs obtained averaging the strain of each detector in a window of 1024 s around each event, (see footnote 19) using Welch's method, through the software `GWpy` (Macleod et al. 2021).

The results for GW170817 (BNS), GW190412, GW190814, GW200208\_130117, GW200224\_222234, and GW200311\_115853 (BBHs) are shown in Figure 6, where we compare the errors estimated from GWF<sub>AST</sub> on the set of 1000 samples drawn for each event to the actual measurement errors obtained from the full Bayesian analysis (which can be read

<sup>19</sup> In general, poor sky localization regions are associated to highly non-Gaussian posteriors, at least in the angular variables. Note that the requirement of having a large S/N does not necessarily imply a good localization, for which triangulation is essential. For example, the first event detected by LVC—GW150914—has an S/N of  $\sim 24$  but a rather large localization region of about  $182 \text{ deg}^2$ , having been observed only by the two LIGO detectors.

<sup>20</sup> All of the data is available on GWOSC, <https://www.gw-openscience.org>.



**Figure 6.** Comparison of the results obtained using GWFAST for some selected events from GWTC-2 and GWTC-3: GW170817, GW190412, GW190814, GW200208\_130117, GW200224\_222234, and GW200311\_115853. Each row contains the histogram of the S/N, 90% relative credible intervals on the source-frame chirp mass and luminosity distance, and 90% credible sky area for 1000 samples of the posterior distributions of the events for all of the parameters. Lines of different colors refer to different waveform models, and the vertical dashed lines denote the errors inferred by LVC with a full Bayesian parameter estimation (Abbott et al. 2021b, 2021a).

from Table III of Abbott et al. 2019 for the events belonging to GWTC–1, Table VI of Abbott et al. 2021b for those belonging to GWTC–2, and Table IV of Abbott et al. 2021a for those belonging to GWTC–3). In particular, we show the distribution of the S/Ns, the 90% relative credible intervals on the source-frame chirp mass,  $M_c$ , and luminosity distance,<sup>21</sup> and the size of the 90% credible sky area, and report (the dashed line) the LVK error estimate. For the BNS event GW170817, we performed the analysis with the waveform models `TaylorF2_Tides` and `IMRPhenomD_NRTidalv2`, both including tidal effects, while for the BBH events, we used `IMRPhenomD` and `IMRPhenomHM`, which includes the contribution of higher modes, which are always taken into account in the parameter estimation of the chosen BBH signals (Abbott et al. 2020b, 2020c, 2021a).

We find overall a very good agreement for the S/N distributions for all events, with both the waveform models used in each case, and we also observe that, as expected in the BBH case, `IMRPhenomHM` produces slightly higher S/Ns than `IMRPhenomD`, especially for GW190814. This can be traced to the fact that the mass ratio of this system is large (the primary component has been estimated to have a source-frame mass of about  $23 M_\odot$  and the secondary  $2.6 M_\odot$ ), resulting in a greater relevance of the subdominant modes, as compared to more symmetric binaries (see, e.g., Figure 2 of Puecher et al. 2022).

Regarding GW170817, we find that the agreement of the fractional error on the chirp mass  $M_c$  with the LVK estimate, despite the long inspiral, is better using the full inspiral-merger-ringdown model `IMRPhenomD_NRTidalv2`, which was not included in the first analysis of the system, while the distributions of the fractional error on  $d_L$  are similar, and the distribution of the sky localization  $\Delta\Omega_{90\%}$  is compatible.

For the majority of the BBH systems, we find our estimations on the source-frame chirp mass and luminosity distance errors to be compatible with the values inferred by LVK when using the waveform model including higher-order harmonics, which were indeed included in the analysis. The only exception is the system GW200208\_130117, for which our estimations seem optimistic. This can be understood by the fact that the network S/N for this system (equal to 10.8) is the lowest among the ones considered: for such a value of the S/N, the FIM approach is not guaranteed to work. In any case, even in this case, the sky localization is compatible, and there is always a fraction of cases where also the fractional errors on chirp mass and distance are consistent. As for the sky localization, we find our estimations to include the LVK results for all of the events when using `IMRPhenomHM`, always being on the same order of magnitude and without a clear trend toward higher or lower values for different events.

## 7. Summary

In this article we presented `GWFAST` (<https://github.com/CosmoStatGW/gwfast>), a novel pure Python code for computing S/Ns and FIMs for catalogs of GW events, in a fast, accurate, and user-friendly way. In particular, `GWFAST`:

1. implements a pure Python version of state-of-the-art Fourier domain full inspiral-merger-ringdown waveform

models, suitable for both BBH, BNS, and NSBH systems, namely `IMRPhenomD`, `IMRPhenomD_NRTidalv2`, `IMRPhenomHM`, and `IMRPhenomNSBH`. These are also separately available in the module `WF4Py` (<https://github.com/CosmoStatGW/WF4Py>; which further includes the waveform model `IMRPhenomXAS`), and allow us to exploit vectorization to speed up the computation and to employ AD for computing derivatives. It is also possible to use all waveforms included in LAL, in which case derivatives are computed with finite differences techniques;

2. accounts for the amplitude and phase modulation of the observed GW signal due to Earth’s rotation, which is of fundamental importance at 3G detectors, whose sensitivity curve can extend down to 2 Hz, in particular for BNS systems, which can stay in the detection band for as long as  $\mathcal{O}(1 \text{ day})$ ;
3. is developed to handle networks of detectors, both L-shaped and triangular, and includes 10 predefined locations as well as several sensitivity curves, for both current and planned ground-based detectors, which can also easily be extended;
4. computes, if waveforms in Python are used, derivatives using AD, through the `JAX` package, thus being extremely accurate and fast, and offers the possibility of computing derivatives with respect to many parameters analytically, to further speed up the computation;
5. handles the inversion of the FIM using the `mpmath` library, thus avoiding limitations linked to numerical precision, and includes functions for easily manipulating both the Fisher and covariance matrices, e.g., adding priors or computing localization regions, as well as tools to assess the reliability of the inversion; and
6. can compute S/Ns and Fisher matrices for multiple events at a time *on a single CPU* exploiting Python vectorization, and provides a module for parallelization over multiple CPUs, also suitable for clusters, thus being ideal for large catalogs of sources.

We assessed the reliability of `GWFAST` in computing accurately the signal derivatives, as well as the frequency integral, by comparing with an independent code written in `Wolfram Mathematica`, capable of computing analytical derivatives with respect to all parameters, obtaining excellent agreement. We further compared the predictions for the S/Ns and measurement errors obtained using `GWFAST` on some of the loudest and best localized events detected during the second and third observing runs of the LVK collaboration with the actual results obtained from a full Bayesian parameter estimation, obtaining good agreement. `GWFAST` has been used to produce the results in the companion paper (Iacovelli et al. 2022), where we also discuss its comparison with other existing codes and results (Borhanian 2021; Harms et al. 2022; Pieroni et al. 2022), showing their excellent agreement. Due to its structure and to the use of AD, `GWFAST` is also suitable for extensions of the FIM approximation (Vallisneri 2011; Sellentin et al. 2014; Wang et al. 2022). We are confident that it will constitute a useful tool for assessing the scientific potential of third-generation GW detectors. `GWFAST` is publicly available (<https://github.com/CosmoStatGW/gwfast>). This paper is associated to version v1.0.1, which is archived on Zenodo (Iacovelli & Mancarella 2022). The library `WF4Py` is available at (<https://github.com/CosmoStatGW/WF4Py>).


<sup>21</sup> Computed, for ease of comparison, converting the  $1\sigma$  errors obtained from the inversion of the FIM.

This paper is associated to version v1.0.0, which is archived on Zenodo (Iacovelli 2022).

Our research is supported by the Swiss National Science Foundation, grant 200020\_191957, and by the SwissMap National Center for Competence in Research. The research leading to these results has been conceived and developed within the ET Observational Science Board (OSB).

### ORCID iDs

Francesco Iacovelli  <https://orcid.org/0000-0002-4875-5862>

Michele Mancarella  <https://orcid.org/0000-0002-0675-508X>

Stefano Foffa  <https://orcid.org/0000-0002-4530-3051>

Michele Maggiore  <https://orcid.org/0000-0001-7348-047X>

### References

- Abbott, B. P., Abbott, R., Abbott, T. D., et al. 2019, *PhRvX*, **9**, 031040
- Abbott, B. P., Abbott, R., Abbott, T. D., et al. 2017, *CQGra*, **34**, 044001
- Abbott, B. P., Abbott, R., Abbott, T. D., et al. 2020a, *LRR*, **23**, 3
- Abbott, R., Abbott, T. D., Abraham, S., et al. 2020b, *PhRvD*, **102**, 043015
- Abbott, R., Abbott, T. D., Abraham, S., et al. 2020c, *ApJ*, **896**, L44
- Abbott, R., Abbott, T. D., Acernese, F., et al. 2021a, arXiv:2111.03606
- Abbott, R., Abbott, T. D., Abraham, S., et al. 2021b, *PhRvX*, **11**, 021053
- Ajith, P. 2011, *PhRvD*, **84**, 084037
- Barack, L., & Cutler, C. 2004, *PhRvD*, **69**, 082005
- Borhanian, S. 2021, *CQGra*, **38**, 175014
- Bradbury, J., Frostig, R., Hawkins, P., et al. 2018, JAX: Composable Transformations of Python+NumPy Programs, v0.2.5, <http://github.com/google/jax>
- Buonanno, A., Iyer, B., Ochsner, E., Pan, Y., & Sathyaprakash, B. S. 2009, *PhRvD*, **80**, 084043
- Cutler, C., & Flanagan, E. E. 1994, *PhRvD*, **49**, 2658
- Dalcin, L., & Fang, Y.-L. L. 2021, *CSE*, **23**, 47
- Dietrich, T., Samajdar, A., Khan, S., et al. 2019, *PhRvD*, **100**, 044003
- Freise, A., Chelkowski, S., Hild, S., et al. 2009, *CQGra*, **26**, 085012
- Gossan, S. E., Hall, E. D., & Nissanke, S. M. 2022, *ApJ*, **926**, 231
- Harms, J., Dupletsa, U., Banerjee, B., et al. 2022, arXiv:2205.02499
- Husa, S., Khan, S., Hannam, M., et al. 2016, *PhRvD*, **93**, 044006
- Iacovelli, F. 2022, WF4Py: Gravitational Waves Waveform Models in Pure Python Language, v1.0.0, Zenodo, doi:10.5281/zenodo.7060240
- Iacovelli, F., & Mancarella, M. 2022, gwfast: A Fisher Information Matrix Python Package for GW Detector Networks, v1.0.1, Zenodo, doi:10.5281/zenodo.7060236
- Iacovelli, F., Mancarella, M., Foffa, S., & Maggiore, M. 2022, arXiv:2207.02771
- Kalaghatgi, C., Hannam, M., & Raymond, V. 2020, *PhRvD*, **101**, 103004
- Khan, S., Husa, S., Hannam, M., et al. 2016, *PhRvD*, **93**, 044007
- LIGO Scientific Collaboration 2018, LIGO Algorithm Library—LALSuite, free software (GPL), doi:10.7935/GT1W-FZ16
- London, L., Khan, S., Fauchon-Jones, E., et al. 2018, *PhRvL*, **120**, 161102
- MacLeod, D. M., Areeda, J. S., Coughlin, S. B., Massinger, T. J., & Urban, A. L. 2021, *SoftX*, **13**, 100657
- Maggiore, M. 2007, Gravitational Waves. Vol. 1: Theory and Experiments (Oxford: Oxford Univ. Press)
- Margossian, C. C. 2018, arXiv:1811.05031
- McKerns, M. M., Strand, L., Sullivan, T., Fang, A., & Aivazis, M. A. G. 2012, arXiv:1202.1056
- Mishra, C. K., Kela, A., Arun, K. G., & Faye, G. 2016, *PhRvD*, **93**, 084054
- Moore, B., Favata, M., Arun, K. G., & Mishra, C. K. 2016, *PhRvD*, **93**, 124061
- Pannarale, F., Berti, E., Kyutoku, K., Lackey, B. D., & Shibata, M. 2015, *PhRvD*, **92**, 084050
- Pironi, M., Ricciardone, A., & Barausse, E. 2022, arXiv:2203.12586
- Pratten, G., Husa, S., Garcia-Quiros, C., et al. 2020, *PhRvD*, **102**, 064001
- Puecher, A., Kalaghatgi, C., Roy, S., et al. 2022, arXiv:2205.09062
- Rodriguez, C. L., Farr, B., Farr, W. M., & Mandel, I. 2013, *PhRvD*, **88**, 084013
- Schutz, B. F. 2011, *CQGra*, **28**, 125023
- Sellentin, E., Quartin, M., & Amendola, L. 2014, *MNRAS*, **441**, 1831
- Smith, R., Borhanian, S., Sathyaprakash, B., et al. 2021, *PhRvL*, **127**, 081102
- Srivastava, V., Davis, D., Kuns, K., et al. 2022, *ApJ*, **931**, 22
- Vallisneri, M. 2008, *PhRvD*, **77**, 042001
- Vallisneri, M. 2011, *PhRvL*, **107**, 191104
- Wade, L., Creighton, J. D. E., Ochsner, E., et al. 2014, *PhRvD*, **89**, 103012
- Wang, Z., Liu, C., Zhao, J., & Shao, L. 2022, *ApJ*, **932**, 102
- Wen, L., & Chen, Y. 2010, *PhRvD*, **81**, 082001