`pySEOBNR`: a software package for the next generation of effective-one-body multipolar waveform models

Deyan P. Mihaylov,[1,*] Serguei Ossokine,[1,†] Alessandra Buonanno,[1,2]
Hector Estelles,[1] Lorenzo Pompili,[1] Michael Pürrer,[3,4,1] and Antoni Ramos-Buades[1]

[1]*Max Planck Institute for Gravitational Physics (Albert Einstein Institute), Am Mühlenberg 1, Potsdam 14476, Germany*
[2]*Department of Physics, University of Maryland, College Park, MD 20742, USA*
[3]*Department of Physics, East Hall, University of Rhode Island, Kingston, RI 02881, USA*
[4]*Center for Computational Research, Tyler Hall, University of Rhode Island, Kingston, RI 02881, USA*

(Dated: April 3, 2023)

We present `pySEOBNR`, a Python package for gravitational-wave (GW) modeling developed within the effective-one-body (EOB) formalism. The package contains an extensive framework to generate state-of-the-art inspiral-merger-ringdown waveform models for compact-object binaries composed of black holes and neutron stars. We document and demonstrate how to use the built-in quasi-circular precessing-spin model `SEOBNRv5PHM`, whose aligned-spin limit (`SEOBNRv5HM`) has been calibrated to numerical-relativity simulations and the nonspinning sector to gravitational self-force data using `pySEOBNR`. Furthermore, `pySEOBNR` contains the infrastructure necessary to construct, calibrate, test, and profile new waveform models in the EOB approach. The efficiency and flexibility of `pySEOBNR` will be crucial to overcome the data-analysis challenges posed by upcoming and next-generation GW detectors on the ground and in space, which will afford the possibility to observe all compact-object binaries in our Universe.

METADATA

Current code version	0.1
Permanent link to code / repository	`git.ligo.org/waveforms/software/pyseobnr`
Legal Code License	GNU General Public License
Code versioning system used	`git`
Software languages, tools, and services used	`Python`, `Mathematica`
Compilation requirements and dependencies	`wheel`, `setuptools`, `numpy`
Link to developer documentation / manual	`waveforms.docs.ligo.org/software/pyseobnr`
Support email for questions	`pyseobnr@aei.mpg.de`

Table I. Code metadata

* deyan@aei.mpg.de
† serguei.ossokine@aei.mpg.de

I. MOTIVATION FOR DEVELOPING pySEOBNR

The field of gravitational-wave (GW) astronomy has made significant progress since the first detection of GWs from the merger of a binary black hole in 2015 [1]. The first and second observing runs of the LIGO and Virgo ground-based detectors [2, 3] resulted in only a handful of detections [4, 5], whereas during the third-observing run, over a hundred events were detected [6–10] including GWs from a wider variety of compact-object binaries. The upcoming fourth-observing run, which will also include the KAGRA detector [11, 12], planned upgrades to the existing detectors [13–15] and next-generation GW detectors on the ground (for example the Einstein Telescope [16] and Cosmic Explorer [17, 18]), as well as the space-based detector LISA [19], will result in an even greater detection rate. In order to take full advantage of these important developments, we need accurate models of the GW signals emitted by compact-binary coalescences [20] to infer unique information about gravity, astrophysics, cosmology and fundamental physics.

The golden standard of waveform modeling is performing full numerical-relativity (NR) simulations [21–23]. Over the past decade, a number of NR waveform catalogues have been built, using several different software packages [24–29]. However, their computational cost makes them prohibitively expensive to carry out across the entire binary's parameter space and also limits their duration. For this reason, semi-analytical waveform models which combine analytical predictions for the two-body dynamics and GW radiation with NR data were developed. Presently, there are a number of waveform models available that are built using one of several approaches outlined below.

NR surrogate models are constructed by interpolating a sufficient number of NR waveforms across the desired parameter space [30–39]. This approach has been successful in accurately modeling systems where higher multipoles are important [33], for precessing-spin binaries [32, 34] and for eccentric systems [37]. However, NR surrogate waveforms are limited in their length (unless hybridized to analytical waveforms), and are restricted to those regions of parameter space where enough NR waveforms are available.

The other two dominant approaches to waveform modeling are the phenomenological and effective-one-body (EOB) frameworks. The phenomenological waveform models (`IMRPhenom`) [40–57] combine post-Newtonian (PN) and EOB analytic expressions for the inspiral with NR information for the lead-up to merger and merger-ringdown portions of the waveform. Their computational efficiency has made them a standard choice for data analysis in GW astronomy. The other waveform family routinely used in GW observations is the EOB one, which combines an analytical description of the two-body dynamics based on PN and small-mass ratio perturbation theory with NR data for the strong-field regime [58–62]. EOB models have been developed for a broad range of systems: non-spinning, aligned-spin, precessing-spin, and eccentric binaries. The two most notable EOB families are `SEOBNR` [63–68] and `TEOBResumS` [69–74]. In this work we will focus on the `SEOBNR` family, which has been employed by the LIGO-Virgo-KAGRA (LVK) Collaboration to analyse GW signals of compact-binary coalescences, infer their astrophysical properties, and test the theory of General Relativity (GR).

While waveforms from the `SEOBNR` family have been widely used for GW analyses due to their high accuracy, they are often computationally expensive with nested sampling and MCMC methods [75–77]. This issue has been tackled in several ways. One possible approach is to construct surrogate models [30–34, 66, 78–87]. Another possibility is to use alternative samplers for parameter estimation, such as `RIFT` [88], `parallel Bilby` [89], `VItamin_C` [90], `VARAHA` [91] or others, which have been developed with the goal of decreasing the walltime of analyses. Notably, `DINGO` [92–94], a software package based on machine learning, has also been used for inference studies with `SEOBNR`. However, some events are still extremely challenging to analyse and those approaches alone do not readily scale to a large number of events. The construction of reduced-order or surrogate models to accelerate EOB waveform generation [66, 80–83, 86, 87, 95–97] can be challenging and time-consuming, thus limiting the pace with which new advances can be incorporated into the `SEOBNR` models. All these factors underscore the need to build a new framework for rapid development and generation of new models.

The need to build more accurate waveform models is also driven by the increasing sensitivity of GW detectors. Upcoming observations of the LVK Collaboration, as well as future detectors like LISA, Einstein Telescope and Cosmic Explorer, will provide the opportunity for pioneering discoveries about the nature and origins of compact objects and gravity, fundamental physics and cosmology, provided we are equipped with high-precision and efficient gravitational waveforms, which include all physical effects and can be used in such studies. For example, using accurate and computationally efficient waveform models for binary neutron stars (BNSs) and neutron-star–black-hole (NSBH) binaries, we would be able to probe the internal structure and equation of state of neutron stars, as well as characterize their population and origin, including details about the low-end mass gap of compact objects.

The higher sensitivity of future detectors poses another challenge. In particular, most of these detectors will observe in a lower/wider frequency band, resulting in much longer signals. This necessitates significantly more computationally efficient waveform models. Thus, a combination of Fourier-domain and time-domain modeling should be explored, as well as the use of novel hardware architectures, e.g. GPUs and parallel computing. This in turn requires a new infrastructure and framework for waveform development.

In order to meet the need for faster, more accurate and more physically complete GW models, we have developed a completely new code base, `pySEOBNR`, which implements all tools necessary for the development of the next generation of `SEOBNR` waveform models in a modern and easily maintainable environment. On one hand, the software package provides access to the aligned-spin `SEOBNRv5HM` [98] and precessing-spin `SEOBNRv5PHM` models [99] for use in scientific applications. For example, several different parameter estimation codes are being adapted to use these models, such as `bilby` [77, 100], `DINGO` and `RIFT`. On the other hand, `pySEOBNR` provides a suite of tools for the user to develop new models.

Technical information about `pySEOBNR` can be found in Table I. We choose `python` [101–103] to build the framework, due to its rich scientific ecosystem, maintainability and wide adoption in the GW community [77, 100, 104–106]. In addition, certain parts of the code are built with `cython` [107], a compiled extension to `python`, in order to achieve better efficiency where necessary, while retaining a lot of flexibility. Theoretical results, which are the foundation of the `SEOBNR` family of models, can be provided via `Mathematica` [108] notebooks, which are then parsed and used by `pySEOBNR`. The package is published under the GNU General Public License [109]. Version control is provided through `git` [110, 111], which also offers a platform for streamlined management of code development, issue tracking, and publishing updated releases of the package. The most recent version of `pySEOBNR` is available through the linked `git` repository. Stable versions of `pySEOBNR` are published through the Python Package Index (PyPI) [112] repository (a user may install the latest stable version via `pip install pyseobnr`). In order to facilitate the broader adoption of this new software package, the authors welcome questions, suggestions, and bug reports at the provided electronic mail address.

In this article we review the current state-of-the-art of the `pySEOBNR` package. In Section II we demonstrate its functionality, we describe the necessary inputs and possible options, and we explain how the submodules interact with each other. In Section III we provide code samples that showcase the main use cases of `pySEOBNR`. Finally, in Section IV, we conclude by providing a roadmap for future developments. This article does not represent a complete record of the various functionalities of `pySEOBNR`. For further information about the package, please refer to the full documentation, linked in Table I. The theoretical results on which the waveform models are based can be found in Ref. [113]. For details on the specifics of the models please consult Refs. [98, 114] (aligned-spin model `SEOBNRv5HM` and calibration pipelines), Ref. [99] (precessing-spin model `SEOBNRv5PHM`) and references therein.

II. DESCRIPTION OF THE SOFTWARE PACKAGE

The user interface of `pySEOBNR` is implemented in `python` [103] (more precisely, the package is compatible with Python v3.8 and above) in order to facilitate ease of use and quick adoption by the GW community.

The package depends on a number of established, well-known, and regularly maintained packages under open-software licenses: `numpy` [115], `scipy` [116], `cython` [107], `GSL` [117, 118] , `pygsl` [119] as well as `lal` and `lalsimulation` [120, 121] and others are used in the extensions for waveform generation. In addition, `wolframclient` [122] and `bilby` [123] are used for the extensions that allow the development and calibration of new waveform models. Table I contains the `pySEOBNR` metadata for reference. In this section we present in detail the architecture and submodules of `pySEOBNR`.

`pySEOBNR` has been developed with two main objectives. The first and foremost is to provide an interface for generating state-of-the-art multipolar waveform models to be used in scientific analyses. This functionality can be readily employed for a number of studies and projects related to analysis of GW data. The second objective is to create a new framework to rapidly develop new EOB waveform models. For instance, `pySEOBNR` provides tools such as automatic code generation from `Mathematica` [108] files to incorporate the latest theoretical results, a pipeline to calibrate EOB models against NR simulations as well as many others. Future `SEOBNR` models will be developed within this framework; moreover newly developed theoretical results could easily be accompanied by corresponding releases of the `pySEOBNR` package for timely updates to existing models.

`pySEOBNR` has been designed in a modular fashion in order to make sure that it can be easily extended, developed, and maintained in the future. The codebase has been separated into a number of submodules, each with a clear designation (see Section II B for further information on the submodules). Detailed and informative documentation accompanies all parts of the code. Pre-commit hooks and continuous integration testing are used for code styling, unit and regression tests.

Fig. 1 depicts the overall structure of the `pySEOBNR` package. The bottom-most panels show the primary steps involved in generating aligned- and precessing-spin models, while the panels above them provide details on these operations. The top-most panels showcase the development functionality, including a code parser for analytical information from `Mathematica` and a calibration pipeline to Numerical Relativity and Gravitational Self-Force data. The remaining parts of the diagram show the important steps in construction of EOB models: the parts of the diagram connected by dashed arrows are carried out off-line, during the development process, while the solid lines show the process of generating a waveform model. In physical terms, the dynamics of the binary is computed by numerically integrating the EOB Hamiltonian system with a dissipative radiation-reaction (RR) force that represents the energy loss through radiation of GWs. Subsequently, spherical harmonic modes of the emitted waveform are computed in an inertial reference frame, while taking into account how the free parameters of the EOB model have been calibrated against numerical relativity simulations. Finally, the two GW polarizations are computed from the waveform modes.

A. Software architecture

The principal functionality of `pySEOBNR` is structured around a single class object `GenerateWaveform` and a small number of functions used to generate multipolar gravitational waveforms. The user is able to specify input parameters through options for initialising the class object and can then obtain output through instance methods. In this section we acquaint the reader with the specifics of this class; the variety of options are explained below. For examples of their usage, please refer to Section III.

1. The GenerateWaveform class

One of the two primary ways to generate a waveform using `pySEOBNR` is to make use of the `GenerateWaveform` class and its instance methods. The inputs are provided in a form of a Python dictionary to ensure they can be easily extended. There are 27 possible input parameters, only 2 of them are required (the masses m_1, m_2 of the primary and secondary binary components), while the remaining ones are

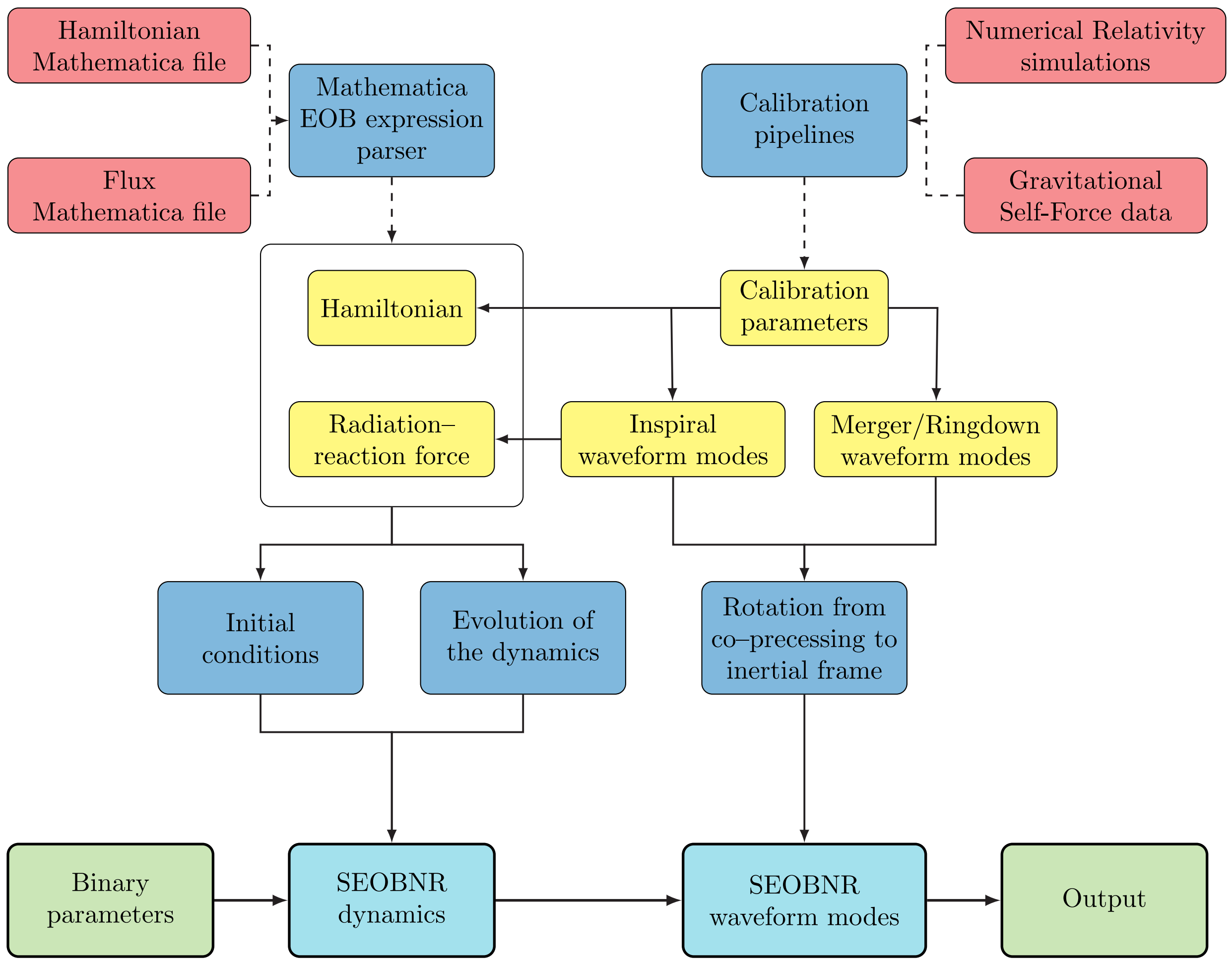
Hamiltonian Mathematica file
Flux Mathematica file
Mathematica EOB expression parser
Numerical Relativity simulations
Gravitational Self-Force data
Calibration pipelines
Hamiltonian
Calibration parameters
Radiation–reaction force
Inspiral waveform modes
Merger/Ringdown waveform modes
Initial conditions
Evolution of the dynamics
Rotation from co–precessing to inertial frame
Binary parameters
SEOBNR dynamics
SEOBNR waveform modes
Output

Figure 1. Diagram of how the separate `pySEOBNR` submodules work and interact with each other. The boxes on the bottom row denote the main sequence of operations executed when generating a (time-domain) waveform. The solid arrows describe how submodules interact to build parts of the waveform, and the dashed lines denote operations which are executed off-line during development and calibration of a new waveform model from scratch.

optional and have default values that are appropriate for most uses, see Table II for details. Of the optional parameters:

- The luminosity `distance` to the GW source d_L is expressed in units of Mpc, with a default value of 100 (the choice of units is owing to the fact that we dealing with extra-galactic sources).
- The `inclination` angle ι is the angle between the line-of-sight and the orbital angular momentum in radians. By default, the binary is viewed head-on.
- The reference frequency `f_ref` is significant for the precessing model, where other quantities (e.g. the spins, or the orbital phase) could be defined at a frequency different from the starting frequency. By default it is equal to `f22_start`.
- The orbital phase (at the reference frequency) `phi_ref` is measured in radians and is 0 by default.
- The time spacing `deltaT` of the waveform is measured in seconds, with a default value of $1/2048$ (used for time-domain outputs and for checking consistency with the Nyquist criterion).
- The maximum frequency `f_max` of the waveform is 1024 Hz by default. If not provided explicitly, its value is computed from `deltaT` as the Nyquist frequency.
- The frequency spacing `deltaF` of the waveform is measured in Hz, with a default value of 0.125 (used for Fourier-domain outputs).

- The mode_array parameter specifies which modes would be computed and returned as output, *in the coprecessing frame*. Only positive$-m$ modes need to be specified, e.g. [(3, 2)] includes both the $(3,2)$ and the $(3,-2)$ modes. By default all modes with $\ell \leq 4$ are selected; at present this includes $(\ell,m) = \{(2,2),(3,3),(3,2),(4,4),(4,3)\}$. In addition, for the models described in this publication, the $(\ell,m) = (5,5)$ mode is available for generation, but is not selected by default.
- approximant directs which model is used to construct the waveform. At present, pySEOBNR includes the SEOBNRv5HM and SEOBNRv5PHM models, while in the future this list will be extended with newly added approximants.
- The conditioning parameter controls the tapering method which would be applied to the waveform before it is returned. Value 1 will taper the beginning of the waveform, as it was done for the SEOBNRv4PHM model in SimInspiralFD(). Value 2 (default) will trigger the standard SimInspiralFD() procedure of adding extra time at the beginning for the purposes of tapering.
- The polarizations_from_coprec option triggers a more efficient computation of the waveform polarizations for the precessing-spin model SEOBNRv5PHM (see Section III C of Ref. [99] for details on this feature).
- The initial_conditions parameter controls whether the model would use "adiabatic" (default) or "postadiabatic" initial conditions (described in further detail in Section II B 3). If "postadiabatic" is selected, the option initial_conditions_postadiabatic_type controls whether the "analytic" (default) or "numeric" post-adiabatic regime is applied for computing the initial conditions.
- Similarly, postadiabatic controls whether the post-adiabatic approximation is used for the solution of the binary inspiral. By default, it is True. See Section II B 3 for details.

In addition, automatic validation of the input parameters is performed at initialization. These include checks that the required parameters (i.e. the component masses) are correctly provided and that the values of the optional parameters are not outside of their allowed values (e.g. the aligned-spin waveform approximant cannot be executed with non-aligned spins).

Once the GenerateWaveform class has been initialised, the user can execute the waveform generator and obtain output through some of the built-in instance methods. Here, it is important to note that the GenerateWaveform class has been designed to be compatible with the standard data analysis package, the LSC's Algorithm Library Suite (LALSuite) [120]. Therefore, within it we reuse standard LALSuite data types and mimic inputs and outputs of main API functions such as SimInspiralChooseTDWaveform() and its Fourier-domain counterpart, which are provided through the lalsimulation sub-package. The available outputs of the waveform generator routine are listed in Table III:

- *time-domain modes*: obtained using the generate_td_modes() method, it produces an array (containing the time domain), and a dictionary containing each of the requested waveform modes (provided through the mode_array input parameter)
- *time-domain polarizations*: obtained using the generate_td_polarizations() method, it will produce two REAL8TimeSeries containing the plus and cross polarization modes of the waveform
- *Fourier-domain polarizations*: obtained using the generate_fd_polarizations() method, it will produce the Fourier transforms of the plus and cross polarization modes as COMPLEX16FrequencySeries.

The output of these class methods are all in SI units: the time domain is provided in seconds, the frequency in Hertz, and the amplitudes of the modes are measured in meters. In Section III we provide code examples for using this functionality.

Name	Description	Default value
`mass1`	Mass of companion 1, in solar masses*	N/A
`mass2`	Mass of companion 2, in solar masses*	N/A
`spin1x`	x-component of dimensionless spin of companion 1	0
`spin1y`	y-component of dimensionless spin of companion 1	0
`spin1z`	z-component of dimensionless spin of companion 1	0
`spin2x`	x-component of dimensionless spin of companion 2	0
`spin2y`	y-component of dimensionless spin of companion 2	0
`spin2z`	z-component of dimensionless spin of companion 2	0
`distance`	Distance to the source, in Mpc	100
`inclination`	Inclination of the source, in radians	0
`f22_start`	Starting waveform generation frequency, in Hz	20
`f_ref`	The reference frequency, in Hz	`f22_start`
`phi_ref`	Orbital phase at the reference frequency, in radians	0
`deltaT`	Time spacing, in seconds	1/2048
`f_max`	Maximum frequency, in Hz	1024 Hz
`deltaF`	Frequency spacing, in Hz	0.125
`mode_array`	Mode content†	`None` (i.e. all modes with $\ell \leq 4$)
`approximant`	Name of the waveform approximant to be used	`"SEOBNRv5HM"`
`conditioning`	Conditioning procedure for the waveform	2
`polarizations_from_coprec`	Whether to generate the polarizations from the co-precessing frame modes	`True`
`initial_conditions`	Mode of initial conditions to be used (adiabatic or post-adiabatic)	`"adiabatic"`
`initial_conditions_postadiabatic_type`	Type of post-adiabatic initial conditions	`"analytic"`
`postadiabatic`	Whether to use the post-adiabatic approximation for the inspiral	`True`
`postadiabatic_type`	Type of post-adiabatic inspiral	`"analytic"`

Table II. Input parameters of the `GenerateWaveform` class. The masses of the primary and secondary components (marked with *) are always required. When specifying the mode content of the output (marked with †), only the positive modes need to be specified, e.g `[(2,2),(2,1)]`. In the future, additional waveform approximant names will be added to this interface as they become part of the `pySEOBNR` package.

2. *Advanced interface: Generating modes and polarizations directly*

Expert users who seek to perform waveform validation have access to a set of specialized functions which can be used to generate the waveform modes or the waveform polarizations.

The function `generate_modes_opt()` may be used for both aligned- and precessing-spin waveforms. For this function, both inputs and outputs are in geometric units. In Section III we provide an example of how to use this endpoint, together with an explanation of its inputs and outputs.

In addition, for precessing waveform models, we provide the function `generate_prec_hpc_opt()` that directly generates the two GW polarizations from the waveform modes in the co-precessing frame, which tracks the motion of the orbital plane in a precessing binary. This function bypasses the construction of the inertial frame modes. This makes the waveform generation significantly faster, especially in the case of low total mass binaries. The user must provide values for the inclination and phase parameters. Since this is significantly faster than the canonical way of generating precessing-spin waveforms, it is also the default approach used for parameter estimation analyses.

B. Primary submodules

In this section we describe the primary submodules responsible for generating waveforms. In particular, we consider modules which contain theoretical EOB expressions that form building blocks of the `SEOBNRv5` models, as well as the infrastructure which allows us to compute the binary inspiral from the Hamiltonian and the RR force, compute the waveform modes, and finally attach the merger-ringdown modes to form the complete IMR waveform.

1. *Hamiltonian*

Consider a binary with masses m_1 and m_2 ($m_1 \geq m_2$) and spins $\mathbf{S}_1$ and $\mathbf{S}_2$. The EOB formalism relies on an effective Hamiltonian (H_{eff}) of a test mass $\mu = m_1 m_2/(m_1 + m_2)$ moving in the Kerr space-time of a body with mass $M = m_1 + m_2$, with deformation parameter $\nu = \mu/M$. The conservative two-body dynamics is obtained from the EOB Hamiltonian:

$$H_{\text{EOB}} = M\sqrt{1 + 2\nu\left(\frac{H_{\text{eff}}}{\mu} - 1\right)}. \tag{1}$$

The dynamical variables on which the value of the Hamiltonian depends are the orbital separation $\mathbf{r}$, the conjugate momentum $\mathbf{p}$, and the spins $\mathbf{S}_{1,2}$. In addition, its value depends on a set of calibration parameters θ, that correspond to yet unknown higher-order PN coefficients. In this section we describe how the EOB Hamiltonian is implemented on `pySEOBNR`.

The `hamiltonian` submodule contains the entire infrastructure for using EOB Hamiltonians. Analytical expressions are converted from `Mathematica` files into `cython` source files which subclass the `Hamiltonian_C` abstract class (or `Hamiltonian_v5PHM_C` for the `SEOBNRv5PHM` precessing-spin model). These files provide the infrastructure to evaluate the EOB Hamiltonian, as well as its Jacobian and Hessian (the derivatives are computed automatically during the conversion from `Mathematica` to `cython` using `wolframclient` as these are more computationally efficient compared to either finite-difference derivatives or automatic derivatives using the `jax` [124] package). In addition, further quantities derived from the EOB Hamiltonian are provided through methods of this class: `xi` computes the tortoise coordinate conversion factor $\xi(r) = p_{r_*}/p_r = \mathrm{d}r/\mathrm{d}r_*$; `auxderivs` provides access to some of the potentials computed as part of computing the value of H_{EOB}, which are necessary for optimising the computational efficiency of the post-adiabatic approximation by using analytic derivatives (see Section II B 3 for details).

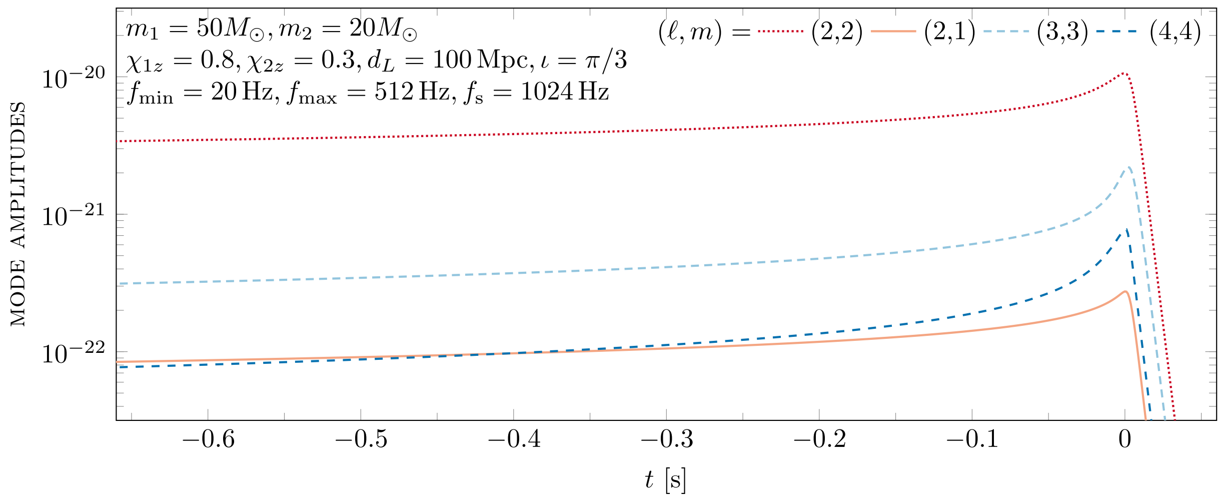

Figure 2. Plot of the waveform modes produced with the `generate_td_modes()` method for an aligned-spin black-hole binary. The parameters used for waveform generation are stated in the figure, with the sampling rate $f_s = 1/$`deltaT`.

The infrastructure provided in `pySEOBNR`will allow for an easy extension of the current Hamiltonians to include more analytical information. In future releases of `pySEOBNR`, the `hamiltonian` submodule will be extended in order to include additional physical effects in the Hamiltonian, for instance tidal corrections, needed for generating waveforms describing binary neutron-star coalescences.

2. *Waveform and radiation-reaction force*

Like the Hamiltonian, RR force is also a fundamental building block of the `SEOBNR` dynamics. In `pySEOBNR`, the RR force is computed as an iterative sum over factorized/resummed PN expressions for the waveform modes [125, 126]:

$$\mathcal{F} = \frac{M\Omega}{16\pi}\frac{\mathbf{p}}{L}\sum_{\ell=2}^{L_{max}}\sum_{m=-\ell}^{\ell} m^2 |d_L h_{\ell m}|^2, \tag{2}$$

where Ω is the angular orbital frequency, L is the magnitude of the orbital angular momentum, $\mathbf{p}$ is the canonical momentum, d_L is the luminosity distance and $h_{\ell m}$ are the gravitational modes far from the source. Due to the large number of terms (in `SEOBNRv5HM` and `SEOBNRv5PHM`, $L_{max} = 8$ is used), the current implementation of the RR force is one area where significant improvements in efficiency could be achieved, which in turn will have an effect on the overall speed of the model.

3. *Dynamics*

For a binary system with aligned or anti-aligned spins, the dynamics is obtained by solving the Hamilton equations:

$$\dot{\mathbf{r}} = \frac{\partial H_{\mathrm{EOB}}}{\partial \mathbf{p}} \tag{3a}$$

$$\dot{\mathbf{p}} = -\frac{\partial H_{\mathrm{EOB}}}{\partial \mathbf{r}} + \mathcal{F} \tag{3b}$$

For a generic-spin system, there are further 6 equations of motion governing the spin evolution:

$$\dot{\mathbf{S}}_{1,2} = \frac{\partial H_{\mathrm{EOB}}}{\partial \mathbf{S}_{1,2}} \times \mathbf{S}_{1,2}. \tag{4}$$

Interestingly, the dynamics may be approximated by a post-adiabatic approach which allows us to compute the evolution of the orbital parameters on a sparse grid in radial domain [68, 72, 127, 128]. The approximation is achieved iteratively and is valid until a short time before merger. The approach requires us to find the solutions of (non-linear) algebraic equations in order to approximate the conjugate momentum to the tortoise radial coordinate p_{r_*} and the orbital angular momentum p_φ:

$$\frac{\mathrm{d}p_\varphi}{\mathrm{d}r}\frac{\partial H_{\mathrm{EOB}}}{\partial p_{r_*}} - \mathcal{F}_\varphi = 0 \tag{5a}$$

$$\frac{\partial H_{\mathrm{EOB}}}{\partial p_{r_*}} + \frac{\partial H_{\mathrm{EOB}}}{\partial r}\frac{\mathrm{d}r}{\mathrm{d}p_{r_*}} - \frac{p_{r_*}}{p_\varphi}\mathcal{F}_\varphi = 0 \tag{5b}$$

The `dynamics` module provides the functionality for computing the binary dynamics of the EOB models. In order to compute the inspiral dynamics up to merger, the code starts by computing the initial conditions for the binary from the user input parameters. The dynamics computation can be performed either by integrating the Hamilton equations numerically, using standard ODE integrators, or alternatively by computing the post-adiabatic approximation to the dynamics, which is more efficient and is therefore the default method in the precessing model `SEOBNRv5PHM`. It should be noted that the post-adiabatic approximation cannot be extended to the entire waveform and therefore the final few orbits need to be computed using the ODE integrator, with the results from both computations merged to provide the full dynamics.

4. *Fits*

In the previous section we have outlined that in order to achieve the desired accuracy of the models, data from NR simulations as well as data from gravitational self-force computations is used to calibrate the model. While the details of the calibration process are presented in Section II C 2, here we discuss how the output of this calibration process is stored and utilised in waveform generation. The calibration pipelines find values of the free parameters of the model [98] which provide the best match between the `SEOBNR` models and the NR simulations. The values of these free parameters are computed, however, at discrete values of the binary parameters (since NR simulations are not available everywhere in parameter space). In order to provide coverage for the entire parameter range of the `SEOBNR` models, the values of these free parameters are fitted using a least-squares method, and the coefficients of these interpolation fits are stored as part of the Fits submodule. In addition, this submodule provides an interface which is used by the dynamics module in order to compute the values of the coefficients when the dynamics is generated. Currently, the fits are implemented as polynomial or radical functions of the physical variables (for details please refer to Appendices A, B, C, and D of Ref. [98]).

In the future, more sophisticated methods will be implemented for utilising the NR and GSF data. In particular, infrastructure and methods for dealing with the uncertainties in the computed values of the free parameters will provide better agreement with NR simulations, and could allow us to translate uncertainties in the provided data into uncertainties of the waveform model itself.

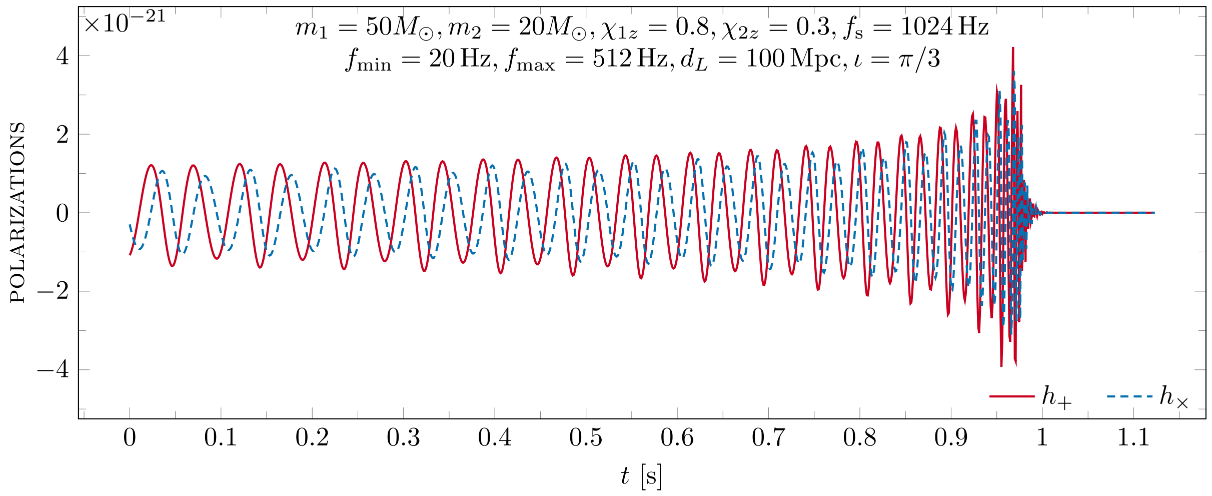

Figure 3. Plot of the waveform polarizations in time domain produced with the `generate_td_polarizations()` method. The parameters used for waveform generation are stated in the figure, with the sampling rate $f_s = 1/$`deltaT`.

C. Auxiliary submodules

In addition to the main submodules needed to generate the waveform, a development-oriented extension to `pySEOBNR` also provides a set of tools whose purpose is to help with the *development* of new waveform models. While these are aimed at advanced users of the package who would like to either build new models or provide improvements to current ones, we give an outline of them here. The most important development tools are the Mathematica expression parser which translates analytical EOB expressions into usable `python/cython` code, and the calibration pipelines which allow us to constrain free parameters in the model to achieve better agreement with NR waveforms. Please note that unlike the much more commonly used waveform generation tools, these modules have not passed a software review process.

1. *Automatic code generation*

`pySEOBNR` contains a parser which can be used to translate `Mathematica` code into python and performant `cython` code leveraging the capabilities of the `sympy` package [129], which can then be used as part of the Hamiltonian and RR force modules. Mathematica files containing analytical EOB Hamiltonian expressions can be parsed using the following command

Method	**LALSuite counterpart**	**Output**
`generate_td_modes()`	`SimInspiralChooseTDModes()`	`times`, `hlm_dict`
`generate_td_polarizations()`	`SimInspiralChooseTDWaveform()`	`hp`, `hc`
`generate_fd_polarizations()`	`SimInspiralChooseFDWaveform()`	`hptilde`, `hctilde`

Table III. Output of the three separate instance methods of the `GenerateWaveform` class for producing the gravitational waveform modes or polarizations. Data types `REAL8TimeSeries` and `COMPLEX16FrequencySeries` follow `LALSuite` conventions [120].

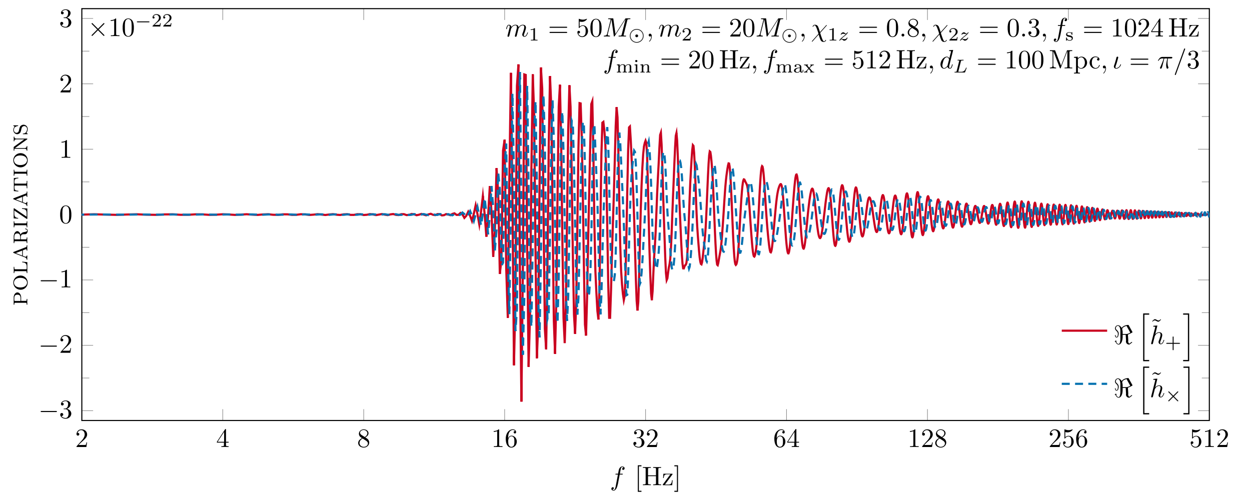

Figure 4. Plot of the waveform polarizations in Fourier domain produced using the `generate_fd_polarizations()` method. The parameters used for waveform generation are stated in the figure, with the sampling rate $f_s = 1/$`deltaT`.

```
python generate_Hamiltonian.py --ham_file [path to Hamiltonian file]
--name [name for this Hamiltonian]
```

The variables of the analytical expressions will be automatically parsed; the variables, constants, and calibration parameters will be identified and translated in the correct fashion to `cython`. The output file is ready to be directly used by the dynamics module without any further preparation. The code generation can be fine-tuned by supplying additional options via `toml` [130, 131] files. In the future, the options provided in this parser will need to be extended in order to cover additional parameters in the analytical Hamiltonians which will be used in future `SEOBNR` waveform models.

2. *Calibration pipelines*

In order to constrain the free coefficients in `SEOBNR` waveform models, we first need to establish a metric for comparison between two waveforms. This is provided through the `metrics` class. Several different metrics are provided, for example: the mismatch between NR and `SEOBNR` waveforms [132], the difference in time to merger between the two waveforms.

The parameter search and estimation is done using Bayesian stochastic sampling, performed via the functionality in the `bilby` package, which is commonly used for data analysis in the GW community. In particular, we use the machine-learning enhanced `nessai` sampler [133], which provides very quick convergence for our class of problems. In order to submit a calibration run, a user may use the commands below:

```
python batch_calibration_setup.py --cases-file [path to NR catalogue]
--prior 4D_tight_a6.prior --sampler nessai --scheduler slurm --queue
hypatia --singularity --singularity-image [path to .sing file]
--calibration-settings calibration_settings.toml bash ./submit_all.sh
```

The output of this process, once completed, will be a set of posterior `JSON` files which can then be used to assign values for each free parameter at each binary parameter point. The interpolations of these parameters are then used by the Fits package in order to provide values for the free parameters of the model during execution.

III. USE CASES

In the previous section, we described in detail the structure of the pySEOBNR package. Here, we show how to use the SEOBNRv5HM and SEOBNRv5PHM models to generate waveforms in several common cases.

A. Generating an aligned-spin gravitational waveform

For the case of a binary with aligned (or anti-aligned) spins, we will demonstrate the use of two different mechanisms. First we consider the most likely use case, such as the one that occurs during GW parameter estimation - the input and output are in physical (SI) units. As shown in listing 1 we use the GenerateWaveform class, which we presented in detail in Section II A 1. We select a binary of total mass $M = 80 M_{\odot}$, with a starting frequency of 20 Hz and an appropriate sampling rate of 1024 Hz. The dimensionless spins of the primary and the secondary binary components are 0.8 and 0.3, respectively. The binary is at a distance of 100 megaparsecs and is inclined at an angle of $\pi/3$.

After the instance has been initialised, we invoke the methods generate_td_modes() and generate_td_polarizations() (see Section II A 1 and Table III) in order to obtain the time-domain modes and polarizations, respectively. For the binary configuration in question, it takes 106 ms to generate the waveform modes, and 113 ms to generate the polarizations. Readers who would like to learn more about the efficiency of the aligned-spin model can refer to [98].

Listing 1. Generating aligned-spin waveform modes and polarizations in physical units with LAL conventions.

```python
import numpy as np
from pyseobnr.generate_waveform import GenerateWaveform

m1, m2 = 50., 20.
s1x, s1y, s1z = 0., 0., 0.8
s2x, s2y, s2z = 0., 0., 0.3

deltaT = 1./1024.
f_min = 20.
f_max = 512.

distance = 100.
inclination = np.pi / 3.
phi_ref = 0.
approximant = "SEOBNRv5HM"

params_dict = {
   "mass1": m1, "mass2": m2,
   "spin1x": s1x, "spin1y": s1y, "spin1z": s1z,
   "spin2x": s2x, "spin2y": s2y, "spin2z": s2z,
   "deltaT": deltaT,
   "f22_start": f_min,
   "phi_ref": phi_ref,
   "distance": distance,
   "inclination": inclination,
   "f_max": f_max,
   "approximant": approximant,
}

waveform_gen = GenerateWaveform(params_dict)
t, hlm = waveform_gen.generate_td_modes()
hp, hc = waveform_gen.generate_td_polarizations()
```

For waveform development and validation it is frequently helpful to instead work in geometric units, e.g. to ease the comparison with NR results. For this we use the function `generate_modes_opt()`, as shown in listing 2.

Listing 2. Using the internal SEOBNRv5HM aligned-spin model generator to obtain the waveform modes and time domain in geometric units.

```python
from pyseobnr.generate_waveform import generate_modes_opt

q = 5.3 # mass ratio
chi_1 = 0.9 # spin of the primary
chi_2 = 0.3 # spin of the secondary
omega0 = 0.0137 # orbital frequency in geometric units with M = 1

t, modes = generate_modes_opt(q, chi_1, chi_2, omega0)
```

Listing 3. Generating precessing-spin waveform modes and polarizations in physical units with LAL conventions.

```python
import numpy as np
from pyseobnr.generate_waveform import GenerateWaveform

m1, m2 = 50., 20.
s1x, s1y, s1z = 0.5, 0., 0.5
s2x, s2y, s2z = 0., 0.5, 0.5

deltaT = 1./1024.
f_min = 20.
f_max = 512.

distance = 1000.
inclination = np.pi / 3.
phi_ref = 0.
approximant = "SEOBNRv5PHM"

params_dict = {
   "mass1": m1, "mass2": m2,
   "spin1x": s1x, "spin1y": s1y, "spin1z": s1z,
   "spin2x": s2x, "spin2y": s2y, "spin2z": s2z,
   "deltaT": deltaT,
   "f22_start": f_min,
   "phi_ref": phi_ref,
   "distance": distance,
   "inclination": inclination,
   "f_max": f_max,
   "approximant": approximant,
}

waveform_gen = GenerateWaveform(params_dict)
t, hlm = waveform_gen.generate_td_modes()
hp, hc = waveform_gen.generate_td_polarizations()
```

B. Generating a precessing-spin gravitational waveform

We can also use the `GenerateWaveform` class to produce waveforms from a binary with precessing spins with minimal changes. Apart from specifying non-zero x- and y-components of the spins, the user needs to specify the precessing-spin `SEOBNRv5PHM` approximant. Readers can use the code in Listing 3 to generate a precessing-spin waveform. For this precessing binary, `pySEOBNR` takes 395 ms to generate the waveform modes, and only 387 ms to generate the polarizations. Readers who would like to learn more about the efficiency of the precessing-spin model are encouraged to refer to [99].

The polarizations in this case can be obtained by using the function `generate_prec_hpc_opt()`, an example of which is shown in listing 4.

Listing 4. Using the `generate_prec_hpc_opt` endpoint to generate the polarizations from the co-precessing frame modes.

```python
from pyseobnr.generate_waveform import generate_prec_hpc_opt

q = 2.0 # mass ratio
chi_1 = np.array([0.5, 0.0, 0.5]) # spin of the primary
chi_2 = np.array([0.0, 0.5, 0.5]) # spin of the secondary
omega0 = 0.01 # orbital frequency in geometric units with M=1

 _, _, model = generate_prec_hpc_opt(
   q,
   chi_1, chi_2,
   omega0,
   debug=True,
   settings={
      "phi_ref": np.pi / 2,
      "inclination": np.pi / 3,
   },
)
```

IV. CONCLUSIONS AND FUTURE DEVELOPMENTS

The `pySEOBNR` package provides a flexible and modern infrastructure for developing waveform models in the `SEOBNR` framework. This Python-based code has allowed us to significantly cut down research and development time and has reduced the amount of boiler-plate code required to build, tune, and implement a working EOB model as compared to the legacy development environment which heavily relied on C/C++ [134, 135] and where models where implemented in C99 [136] in LALSuite [120]. For the sake of compatibility we have retained some of the necessary data types and main interfaces provided by LALSuite. In this publication we have discussed the main capabilities and features of the package, and showcased waveform models for binary black holes with aligned or precessing spins on quasi-circular orbits.

We are planning to significantly enhance the capabilities of `pySEOBNR` in the future. In addition to the quasi-circular aligned- and precessing-spin model, waveforms for eccentric binaries will be provided in order to aid the community-wide effort to survey eccentricity and unveil the origin of the observed compact-binary populations. Moreover, waveforms will be extended for use in tests of General Relativity, which would enable these analyses to be performed using `pySEOBNR`. Additionally, including tidal corrections will allow us to perform inference runs for binary neutron stars and neutron-star–black-hole binaries, which would need to be particularly efficient in the low-mass or large mass-ratio regimes.

Finally, we would like to emphasize that building `pySEOBNR` is a step towards ensuring that the development of new waveform models for LVK and future detectors can proceed using the most modern and sophisticated computing tools and methods. We anticipate that the addition of GPU support and the use of neural networks will make future `SEOBNR` waveform models even more efficient, in turn enabling us to pursue outstanding new science in gravity, fundamental physics, cosmology and astrophysics.

ACKNOWLEDGEMENTS

The authors thank sincerely the LVK team responsible for the review of `pySEOBNR` and the `SEOBNRv5` models: Geraint Pratten, Stanislav Babak, Alice Bonino, Eleanor Hamilton, N. V. Krishnendu, Piero Rettegno, Riccardo Sturani, and Jooheon Yoo. The development work for this software package was carried out on the Hypatia computing cluster at the Max Planck Institute for Gravitational Physics in Potsdam, Germany.

This research has made use of data or software obtained from the Gravitational Wave Open Science Center (gwosc.org), a service of LIGO Laboratory, the LIGO Scientific Collaboration, the Virgo Collaboration, and KAGRA. LIGO Laboratory and Advanced LIGO are funded by the United States National Science Foundation (NSF) as well as the Science and Technology Facilities Council (STFC) of the United Kingdom, the Max-Planck-Society (MPS), and the State of Niedersachsen/Germany for support of the construction of Advanced LIGO and construction and operation of the GEO600 detector. Additional support for Advanced LIGO was provided by the Australian Research Council. Virgo is funded, through the European Gravitational Observatory (EGO), by the French Centre National de Recherche Scientifique (CNRS), the Italian Istituto Nazionale di Fisica Nucleare (INFN) and the Dutch Nikhef, with contributions by institutions from Belgium, Germany, Greece, Hungary, Ireland, Japan, Monaco, Poland, Portugal, Spain. KAGRA is supported by Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan Society for the Promotion of Science (JSPS) in Japan; National Research Foundation (NRF) and Ministry of Science and ICT (MSIT) in Korea; Academia Sinica (AS) and National Science and Technology Council (NSTC) in Taiwan.

A. CODE FOR GENERATING THE PLOTS OF THE WAVEFORM

In order to aid users in getting started and actively using pySEOBNR, in Listing 5 we provide the code for generating the figures of the waveform mode amplitudes (Fig. 2), the time-domain polarizations (Fig. 3), and the Fourier-domain polarizations (Fig. 4).

Listing 5. Generating the plots appearing in Figs. 2, 3, and 4.

```python
import matplotlib.pyplot as plt

plt.figure()

# Create the plot in Figure 2
for mode in [(2, 2), (2, 1), (3, 3), (4, 4)]:
   plt.plot(t, np.abs(hlm[mode]))

plt.yscale('log')
plt.ylim(1e-24, 1e-19)
plt.savefig("figure2.png")

# Create the plot in Figure 3
plt.figure()

plt.plot(t, hp.data.data)
plt.plot(t, hc.data.data)

plt.savefig("figure3.png")

# Create the plot in Figure 4
plt.figure()

plt.plot(f, np.real(hp_f.data.data))
plt.plot(f, np.real(hc_f.data.data))

plt.savefig("figure4.png")
```

[1] B. P. Abbott *et al.* (LIGO Scientific, Virgo), Phys. Rev. Lett. **116**, 061102 (2016), arXiv:1602.03837 [gr-qc].
[2] J. Aasi *et al.* (LIGO Scientific), Class. Quant. Grav. **32**, 074001 (2015), arXiv:1411.4547 [gr-qc].
[3] F. Acernese *et al.* (VIRGO), Class. Quant. Grav. **32**, 024001 (2015), arXiv:1408.3978 [gr-qc].
[4] B. P. Abbott *et al.* (LIGO Scientific, Virgo), Phys. Rev. X **9**, 031040 (2019), arXiv:1811.12907 [astro-ph.HE].
[5] T. Venumadhav, B. Zackay, J. Roulet, L. Dai, and M. Zaldarriaga, Phys. Rev. D **101**, 083030 (2020), arXiv:1904.07214 [astro-ph.HE].
[6] R. Abbott *et al.* (LIGO Scientific, Virgo), Phys. Rev. X **11**, 021053 (2021), arXiv:2010.14527 [gr-qc].
[7] R. Abbott *et al.* (LIGO Scientific, VIRGO), (2021), arXiv:2108.01045 [gr-qc].
[8] R. Abbott *et al.* (LIGO Scientific, VIRGO, KAGRA), (2021), arXiv:2111.03606 [gr-qc].
[9] A. H. Nitz, C. D. Capano, S. Kumar, Y.-F. Wang, S. Kastha, M. Schäfer, R. Dhurkunde, and M. Cabero, Astrophys. J. **922**, 76 (2021), arXiv:2105.09151 [astro-ph.HE].
[10] S. Olsen, T. Venumadhav, J. Mushkin, J. Roulet, B. Zackay, and M. Zaldarriaga, Phys. Rev. D **106**, 043009 (2022), arXiv:2201.02252 [astro-ph.HE].
[11] T. Akutsu *et al.* (KAGRA), Nature Astron. **3**, 35 (2019), arXiv:1811.08079 [gr-qc].
[12] T. Akutsu *et al.* (KAGRA), PTEP **2021**, 05A101 (2021), arXiv:2005.05574 [physics.ins-det].
[13] B. P. Abbott *et al.* (KAGRA, LIGO Scientific, Virgo, VIRGO), Living Rev. Rel. **21**, 3 (2018), arXiv:1304.0670 [gr-qc].
[14] B. P. Abbott *et al.* (LIGO Scientific, Virgo), Astrophys. J. Lett. **882**, L24 (2019), arXiv:1811.12940 [astro-ph.HE].
[15] R. Abbott *et al.* (LIGO Scientific, Virgo), Astrophys. J. Lett. **913**, L7 (2021), arXiv:2010.14533 [astro-ph.HE].
[16] M. Punturo *et al.*, Class. Quant. Grav. **27**, 194002 (2010).
[17] D. Reitze *et al.*, Bull. Am. Astron. Soc. **51**, 035 (2019), arXiv:1907.04833 [astro-ph.IM].
[18] M. Evans *et al.*, (2021), arXiv:2109.09882 [astro-ph.IM].
[19] P. Amaro-Seoane, H. Audley, S. Babak, J. Baker, E. Barausse, P. Bender, E. Berti, P. Binetruy, M. Born, D. Bortoluzzi, *et al.*, arXiv preprint arXiv:1702.00786 (2017).
[20] M. Pürrer and C.-J. Haster, Phys. Rev. Res. **2**, 023151 (2020), arXiv:1912.10055 [gr-qc].
[21] F. Pretorius, Phys. Rev. Lett. **95**, 121101 (2005), arXiv:gr-qc/0507014.
[22] M. Campanelli, C. O. Lousto, P. Marronetti, and Y. Zlochower, Phys. Rev. Lett. **96**, 111101 (2006), arXiv:gr-qc/0511048.
[23] J. G. Baker, J. Centrella, D.-I. Choi, M. Koppitz, and J. van Meter, Phys. Rev. Lett. **96**, 111102 (2006), arXiv:gr-qc/0511103.
[24] K. Jani, J. Healy, J. A. Clark, L. London, P. Laguna, and D. Shoemaker, Class. Quant. Grav. **33**, 204001 (2016), arXiv:1605.03204 [gr-qc].
[25] J. Healy, C. O. Lousto, Y. Zlochower, and M. Campanelli, Class. Quant. Grav. **34**, 224001 (2017), arXiv:1703.03423 [gr-qc].
[26] J. Healy, C. O. Lousto, J. Lange, R. O'Shaughnessy, Y. Zlochower, and M. Campanelli, Phys. Rev. D **100**, 024021 (2019), arXiv:1901.02553 [gr-qc].
[27] M. Boyle *et al.*, Class. Quant. Grav. **36**, 195006 (2019), arXiv:1904.04831 [gr-qc].
[28] J. Healy and C. O. Lousto, Phys. Rev. D **105**, 124010 (2022), arXiv:2202.00018 [gr-qc].
[29] E. Hamilton *et al.*, (2023), arXiv:2303.05419 [gr-qc].
[30] J. Blackman, S. E. Field, C. R. Galley, B. Szilágyi, M. A. Scheel, M. Tiglio, and D. A. Hemberger, Phys. Rev. Lett. **115**, 121102 (2015), arXiv:1502.07758 [gr-qc].
[31] J. Blackman, S. E. Field, M. A. Scheel, C. R. Galley, D. A. Hemberger, P. Schmidt, and R. Smith, Phys. Rev. D **95**, 104023 (2017), arXiv:1701.00550 [gr-qc].
[32] J. Blackman, S. E. Field, M. A. Scheel, C. R. Galley, C. D. Ott, M. Boyle, L. E. Kidder, H. P. Pfeiffer, and B. Szilágyi, Phys. Rev. D **96**, 024058 (2017), arXiv:1705.07089 [gr-qc].
[33] V. Varma, S. E. Field, M. A. Scheel, J. Blackman, L. E. Kidder, and H. P. Pfeiffer, Phys. Rev. D **99**, 064045 (2019), arXiv:1812.07865 [gr-qc].
[34] V. Varma, S. E. Field, M. A. Scheel, J. Blackman, D. Gerosa, L. C. Stein, L. E. Kidder, and H. P. Pfeiffer, Phys. Rev. Research. **1**, 033015 (2019), arXiv:1905.09300 [gr-qc].

[35] D. Williams, I. S. Heng, J. Gair, J. A. Clark, and B. Khamesra, Phys. Rev. D **101**, 063011 (2020), arXiv:1903.09204 [gr-qc].

[36] N. E. M. Rifat, S. E. Field, G. Khanna, and V. Varma, Phys. Rev. D **101**, 081502 (2020), arXiv:1910.10473 [gr-qc].

[37] T. Islam, V. Varma, J. Lodman, S. E. Field, G. Khanna, M. A. Scheel, H. P. Pfeiffer, D. Gerosa, and L. E. Kidder, Phys. Rev. D **103**, 064022 (2021), arXiv:2101.11798 [gr-qc].

[38] T. Islam, S. E. Field, S. A. Hughes, G. Khanna, V. Varma, M. Giesler, M. A. Scheel, L. E. Kidder, and H. P. Pfeiffer, Phys. Rev. D **106**, 104025 (2022), arXiv:2204.01972 [gr-qc].

[39] J. Yoo, V. Varma, M. Giesler, M. A. Scheel, C.-J. Haster, H. P. Pfeiffer, L. E. Kidder, and M. Boyle, Phys. Rev. D **106**, 044001 (2022), arXiv:2203.10109 [gr-qc].

[40] Y. Pan, A. Buonanno, J. G. Baker, J. Centrella, B. J. Kelly, S. T. McWilliams, F. Pretorius, and J. R. van Meter, Phys. Rev. D **77**, 024014 (2008), arXiv:0704.1964 [gr-qc].

[41] P. Ajith *et al.*, Class. Quant. Grav. **24**, S689 (2007), arXiv:0704.3764 [gr-qc].

[42] P. Ajith *et al.*, Phys. Rev. Lett. **106**, 241101 (2011), arXiv:0909.2867 [gr-qc].

[43] L. Santamaria *et al.*, Phys. Rev. D **82**, 064016 (2010), arXiv:1005.3306 [gr-qc].

[44] M. Hannam, P. Schmidt, A. Bohé, L. Haegel, S. Husa, F. Ohme, G. Pratten, and M. Pürrer, Phys. Rev. Lett. **113**, 151101 (2014), arXiv:1308.3271 [gr-qc].

[45] S. Husa, S. Khan, M. Hannam, M. Pürrer, F. Ohme, X. Jiménez Forteza, and A. Bohé, Phys. Rev. D **93**, 044006 (2016), arXiv:1508.07250 [gr-qc].

[46] S. Khan, S. Husa, M. Hannam, F. Ohme, M. Pürrer, X. Jiménez Forteza, and A. Bohé, Phys. Rev. D **93**, 044007 (2016), arXiv:1508.07253 [gr-qc].

[47] L. London, S. Khan, E. Fauchon-Jones, C. García, M. Hannam, S. Husa, X. Jiménez-Forteza, C. Kalaghatgi, F. Ohme, and F. Pannarale, Phys. Rev. Lett. **120**, 161102 (2018), arXiv:1708.00404 [gr-qc].

[48] S. Khan, K. Chatziioannou, M. Hannam, and F. Ohme, Phys. Rev. D **100**, 024059 (2019), arXiv:1809.10113 [gr-qc].

[49] S. Khan, F. Ohme, K. Chatziioannou, and M. Hannam, Phys. Rev. D **101**, 024056 (2020), arXiv:1911.06050 [gr-qc].

[50] T. Dietrich, A. Samajdar, S. Khan, N. K. Johnson-McDaniel, R. Dudi, and W. Tichy, Phys. Rev. D **100**, 044003 (2019), arXiv:1905.06011 [gr-qc].

[51] G. Pratten, S. Husa, C. Garcia-Quiros, M. Colleoni, A. Ramos-Buades, H. Estelles, and R. Jaume, Phys. Rev. D **102**, 064001 (2020), arXiv:2001.11412 [gr-qc].

[52] G. Pratten *et al.*, Phys. Rev. D **103**, 104056 (2021), arXiv:2004.06503 [gr-qc].

[53] C. García-Quirós, M. Colleoni, S. Husa, H. Estellés, G. Pratten, A. Ramos-Buades, M. Mateu-Lucena, and R. Jaume, Phys. Rev. D **102**, 064002 (2020), arXiv:2001.10914 [gr-qc].

[54] H. Estellés, A. Ramos-Buades, S. Husa, C. García-Quirós, M. Colleoni, L. Haegel, and R. Jaume, Phys. Rev. D **103**, 124060 (2021), arXiv:2004.08302 [gr-qc].

[55] H. Estellés, S. Husa, M. Colleoni, D. Keitel, M. Mateu-Lucena, C. García-Quirós, A. Ramos-Buades, and A. Borchers, Phys. Rev. D **105**, 084039 (2022), arXiv:2012.11923 [gr-qc].

[56] H. Estellés, M. Colleoni, C. García-Quirós, S. Husa, D. Keitel, M. Mateu-Lucena, M. d. L. Planas, and A. Ramos-Buades, Phys. Rev. D **105**, 084040 (2022), arXiv:2105.05872 [gr-qc].

[57] E. Hamilton, L. London, J. E. Thompson, E. Fauchon-Jones, M. Hannam, C. Kalaghatgi, S. Khan, F. Pannarale, and A. Vano-Vinuales, Phys. Rev. D **104**, 124027 (2021), arXiv:2107.08876 [gr-qc].

[58] A. Buonanno and T. Damour, Phys. Rev. D **59**, 084006 (1999), arXiv:gr-qc/9811091.

[59] A. Buonanno and T. Damour, Phys. Rev. D **62**, 064015 (2000), arXiv:gr-qc/0001013.

[60] T. Damour, P. Jaranowski, and G. Schaefer, Phys. Rev. D **62**, 084011 (2000), arXiv:gr-qc/0005034.

[61] T. Damour, Phys. Rev. D **64**, 124013 (2001), arXiv:gr-qc/0103018.

[62] A. Buonanno, Y. Chen, and T. Damour, Phys. Rev. D **74**, 104005 (2006), arXiv:gr-qc/0508067.

[63] A. Bohé *et al.*, Phys. Rev. D **95**, 044028 (2017), arXiv:1611.03703 [gr-qc].

[64] R. Cotesta, A. Buonanno, A. Bohé, A. Taracchini, I. Hinder, and S. Ossokine, Phys. Rev. D **98**, 084028 (2018), arXiv:1803.10701 [gr-qc].

[65] S. Ossokine *et al.*, Phys. Rev. D **102**, 044055 (2020), arXiv:2004.09442 [gr-qc].

[66] R. Cotesta, S. Marsat, and M. Pürrer, Phys. Rev. D **101**, 124040 (2020), arXiv:2003.12079 [gr-qc].

[67] A. Ramos-Buades, A. Buonanno, M. Khalil, and S. Ossokine, Phys. Rev. D **105**, 044035 (2022), arXiv:2112.06952 [gr-qc].

[68] D. P. Mihaylov, S. Ossokine, A. Buonanno, and A. Ghosh, Phys. Rev. D **104**, 124087 (2021), arXiv:2105.06983

[gr-qc].
[69] A. Nagar *et al.*, Phys. Rev. D **98**, 104052 (2018), arXiv:1806.01772 [gr-qc].
[70] A. Nagar, G. Pratten, G. Riemenschneider, and R. Gamba, Phys. Rev. D **101**, 024041 (2020), arXiv:1904.09550 [gr-qc].
[71] A. Nagar, G. Riemenschneider, G. Pratten, P. Rettegno, and F. Messina, Phys. Rev. D **102**, 024077 (2020), arXiv:2001.09082 [gr-qc].
[72] R. Gamba, S. Akçay, S. Bernuzzi, and J. Williams, Phys. Rev. D **106**, 024020 (2022), arXiv:2111.03675 [gr-qc].
[73] G. Riemenschneider, P. Rettegno, M. Breschi, A. Albertini, R. Gamba, S. Bernuzzi, and A. Nagar, Phys. Rev. D **104**, 104045 (2021), arXiv:2104.07533 [gr-qc].
[74] D. Chiaramello and A. Nagar, Phys. Rev. D **101**, 101501 (2020), arXiv:2001.11736 [gr-qc].
[75] J. Skilling, Bayesian Analysis **1**, 833 (2006).
[76] J. Veitch *et al.*, Phys. Rev. D **91**, 042003 (2015), arXiv:1409.7215 [gr-qc].
[77] G. Ashton *et al.*, Astrophys. J. Suppl. **241**, 27 (2019), arXiv:1811.02042 [astro-ph.IM].
[78] S. Khan and R. Green, Phys. Rev. D **103**, 064015 (2021).
[79] L. M. Thomas, G. Pratten, and P. Schmidt, Phys. Rev. D **106**, 104029 (2022).
[80] S. E. Field, C. R. Galley, J. S. Hesthaven, J. Kaye, and M. Tiglio, Phys. Rev. X **4**, 031006 (2014), arXiv:1308.3565 [gr-qc].
[81] M. Pürrer, Class. Quant. Grav. **31**, 195010 (2014), arXiv:1402.4146 [gr-qc].
[82] M. Pürrer, Phys. Rev. D **93**, 064041 (2016), arXiv:1512.02248 [gr-qc].
[83] B. D. Lackey, M. Pürrer, A. Taracchini, and S. Marsat, Phys. Rev. D **100**, 024002 (2019), arXiv:1812.08643 [gr-qc].
[84] Z. Doctor, B. Farr, D. E. Holz, and M. Pürrer, Phys. Rev. **D96**, 123011 (2017), arXiv:1706.05408 [astro-ph.HE].
[85] Y. E. Setyawati, M. Pürrer, and F. Ohme, Classical and Quantum Gravity (2020).
[86] B. Gadre, M. Pürrer, S. E. Field, S. Ossokine, and V. Varma, (2022), arXiv:2203.00381 [gr-qc].
[87] L. M. Thomas, G. Pratten, and P. Schmidt, Phys. Rev. D **106**, 104029 (2022), arXiv:2205.14066 [gr-qc].
[88] J. Lange, R. O'Shaughnessy, and M. Rizzo, (2018), arXiv:1805.10457 [gr-qc].
[89] R. J. E. Smith, G. Ashton, A. Vajpeyi, and C. Talbot, Mon. Not. Roy. Astron. Soc. **498**, 4492 (2020), arXiv:1909.11873 [gr-qc].
[90] H. Gabbard, C. Messenger, I. S. Heng, F. Tonolini, and R. Murray-Smith, Nature Phys. **18**, 112 (2022), arXiv:1909.06296 [astro-ph.IM].
[91] V. Tiwari, C. Hoy, S. Fairhurst, and D. MacLeod, (2023), arXiv:2303.01463 [astro-ph.HE].
[92] S. R. Green and J. Gair, Mach. Learn. Sci. Tech. **2**, 03LT01 (2021), arXiv:2008.03312 [astro-ph.IM].
[93] M. Dax, S. R. Green, J. Gair, J. H. Macke, A. Buonanno, and B. Schölkopf, Phys. Rev. Lett. **127**, 241103 (2021), arXiv:2106.12594 [gr-qc].
[94] M. Dax, S. R. Green, J. Gair, M. Pürrer, J. Wildberger, J. H. Macke, A. Buonanno, and B. Schölkopf, (2022), arXiv:2210.05686 [gr-qc].
[95] B. D. Lackey, S. Bernuzzi, C. R. Galley, J. Meidam, and C. Van Den Broeck, Phys. Rev. D **95**, 104036 (2017), arXiv:1610.04742 [gr-qc].
[96] J. Tissino, G. Carullo, M. Breschi, R. Gamba, S. Schmidt, and S. Bernuzzi, (2022), arXiv:2210.15684 [gr-qc].
[97] S. Khan and R. Green, Phys. Rev. D **103**, 064015 (2021), arXiv:2008.12932 [gr-qc].
[98] L. Pompili, A. Buonanno, H. Estellés, M. Khalil, M. van de Meent, D. Mihaylov, S. Ossokine, M. Pürrer, A. Ramos-Buades, A. Kumar Mehta, R. Cotesta, S. Marsat, M. Boyle, L. E. Kidder, H. P. Pfeiffer, M. A. Scheel, H. R. Rüter, N. Vu, R. Dudi, S. Ma, K. Mitman, D. Melchor, S. Thomas, and J. Sanchez, (2023).
[99] A. Ramos-Buades, A. Buonanno, S. Ossokine, H. Estellés, M. Khalil, L. Pompili, and M. Shiferaw, (2023).
[100] I. M. Romero-Shaw *et al.*, Mon. Not. Roy. Astron. Soc. **499**, 3295 (2020), arXiv:2006.00714 [astro-ph.IM].
[101] G. van Rossum and J. de Boer, CWI Quarterly **4**, 283 (1991).
[102] G. Van Rossum and F. L. Drake Jr, *Python tutorial* (Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995).
[103] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual* (CreateSpace, Scotts Valley, CA, 2009).
[104] A. Nitz, I. Harry, D. Brown, C. M. Biwer, J. Willis, T. D. Canton, C. Capano, T. Dent, L. Pekowsky, S. De, M. Cabero, G. S. C. Davies, A. R. Williamson, D. Macleod, B. Machenschalk, F. Pannarale, P. Kumar, S. Reyes, dfinstad, S. Kumar, M. Tápai, S. Wu, L. Singer, veronica villa, S. Khan, S. Fairhurst, K. Chandra, A. Nielsen, S. Singh, and T. Massinger, gwastro/pycbc: v2.1.0 release of pycbc (2023).
[105] `https://git.ligo.org/lscsoft/pyring/`.

[106] G. Carullo, W. Del Pozzo, and J. Veitch, Phys. Rev. D **99**, 123029 (2019), [Erratum: Phys.Rev.D 100, 089903 (2019)], arXiv:1902.07527 [gr-qc].

[107] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, Computing in Science & Engineering **13**, 31 (2011).

[108] W. R. Inc., Mathematica, Version 13.2, champaign, IL, 2022.

[109] Gnu general public license, version 3, `http://www.gnu.org/licenses/gpl.html` (2007), last retrieved 2020-01-01.

[110] `https://git-scm.com`.

[111] S. Chacon and B. Straub, *Pro git* (Apress, 2014).

[112] Python package index - pypi.

[113] M. Khalil, A. Buonanno, H. Estellés, D. Mihaylov, S. Ossokine, L. Pompili, and A. Ramos-Buades, (2023).

[114] M. van de Meent, A. Buonanno, D. Mihaylov, S. Ossokine, L. Pompili, N. Warburton, A. Pound, B. Wardell, L. Durkan, and J. Miller, (2023).

[115] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, Nature **585**, 357 (2020).

[116] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, Nature Methods **17**, 261 (2020).

[117] M. Galassi and et al, Gnu scientific library reference manual (3rd ed.), `http://www.gnu.org/software/gsl/`.

[118] B. Gough, *GNU scientific library reference manual* (Network Theory Ltd., 2009).

[119] `https://github.com/pygsl/pygsl`.

[120] LIGO Scientific Collaboration, LIGO Algorithm Library - LALSuite, free software (GPL) (2018).

[121] K. Wette, SoftwareX **12**, 100634 (2020).

[122] W. Research, wolframclient for python, `https://github.com/WolframResearch/WolframClientForPython` (2019).

[123] G. Ashton, M. Hübner, P. D. Lasky, C. Talbot, K. Ackley, S. Biscoveanu, Q. Chu, A. Divakarla, P. J. Easter, B. Goncharov, F. H. Vivanco, J. Harms, M. E. Lower, G. D. Meadors, D. Melchor, E. Payne, M. D. Pitkin, J. Powell, N. Sarin, R. J. E. Smith, and E. Thrane, The Astrophysical Journal Supplement Series **241**, 27 (2019).

[124] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, JAX: composable transformations of Python+NumPy programs (2018).

[125] T. Damour, B. R. Iyer, and A. Nagar, Phys. Rev. D **79**, 064004 (2009), arXiv:0811.2069 [gr-qc].

[126] Y. Pan, A. Buonanno, R. Fujita, E. Racine, and H. Tagoshi, Phys. Rev. D **83**, 064003 (2011), [Erratum: Phys.Rev.D 87, 109901 (2013)], arXiv:1006.0431 [gr-qc].

[127] A. Nagar and P. Rettegno, Phys. Rev. D **99**, 021501 (2019), arXiv:1805.03891 [gr-qc].

[128] P. Rettegno, F. Martinetti, A. Nagar, D. Bini, G. Riemenschneider, and T. Damour, Phys. Rev. D **101**, 104027 (2020), arXiv:1911.10818 [gr-qc].

[129] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, PeerJ Computer Science **3**, e103 (2017).

[130] T. Preston-Werner, Tom's obvious, minimal language, `https://github.com/toml-lang/toml` (2019).

[131] W. Pearson, toml, `https://github.com/uiri/toml` (2019).

[132] I. Harry, S. Privitera, A. Bohé, and A. Buonanno, Phys. Rev. D **94**, 024012 (2016).

[133] M. J. Williams, J. Veitch, and C. Messenger, Phys. Rev. D **103**, 103006 (2021), arXiv:2102.11056 [gr-qc].

[134] B. W. Kernighan and D. M. Ritchie, *The C programming language* (2006).

[135] ISO, *ISO/IEC 14882:1998: Programming languages — C++* (1998).

[136] ISO, *ISO C Standard 1999*, Tech. Rep. (1999) iSO/IEC 9899:1999 draft.