Regression methods in waveform modeling: a comparative study

OPEN ACCESS
IOP Publishing

Classical and Quantum Gravity

Class. Quantum Grav. **37** (2020) 075012 (26pp)

https://doi.org/10.1088/1361-6382/ab693b

Regression methods in waveform modeling: a comparative study

Yoshinta Setyawati[1,2], **Michael Pürrer**[3] **and Frank Ohme**[1,2]

[1] Max Planck Institute for Gravitational Physics (Albert Einstein Institute), Callinstr. 38, D-30167 Hannover, Germany
[2] Leibniz Universität Hannover, D-30167 Hannover, Germany
[3] Max Planck Institute for Gravitational Physics (Albert Einstein Institute), D-14476 Potsdam-Golm, Germany

E-mail: yoshinta.setyawati@aei.mpg.de

Received 25 September 2019, revised 16 December 2019
Accepted for publication 8 January 2020
Published 3 March 2020

Abstract

Theoretical gravitational-wave models of compact-binary mergers need to be accurate, but also fast in order to compare millions of signals in near real time with experimental data. Various regression and interpolation techniques have been employed to build efficient waveform models, but no study has systematically compared the performance of these methods yet. Here we provide such a comparison. For analytical binary-black-hole waveforms, assuming either aligned or precessing spins, we compare the accuracy as well as the computational speed of a variety of regression methods, ranging from traditional interpolation to machine-learning techniques. We find that most methods are reasonably accurate, but efficiency considerations favour in many cases the simpler approaches. We conclude that sophisticated regression methods are not necessarily needed in standard gravitational-wave modeling applications, although machine-learning techniques might be more suitable for problems with higher complexity than what is tested here.

Keywords: gravitational wave, waveform modeling, regression, interpolation, machine learning, precessing, singular-value-decomposition

1. Introduction

The laser interferometric and gravitational-wave (GW) detectors LIGO [1] and Virgo [2] have reported observations of one binary neutron-star (BNS) and ten binary black-hole (BBH) mergers in their first two observing runs [3]. In the third observing run (O3), we expect to observe several tens of signals from compact binary coalescences [4]. The analysis of these GW data is the motivation for our study. The data from the detectors are filtered with many theoretically predicted waveforms with varying binary parameters. These waveform templates are drawn from models of the emitted GWs. The waveform models need to fulfil accuracy and speed requirements so that the parameters of the GW source can be estimated well in a reasonable amount of time.

We highlight two major modeling approaches: analytical and numerical relativity (NR). The basis of analytical models is the Post-Newtonian (PN) expansion [5]. Waveform models in this category are fairly computationally efficient, but the PN approximation breaks down for merger and ringdown part of the signal. The second category is NR. NR waveforms are built by numerically solving Einstein's equations [6–8]. Although these waveforms are known to have exceptional accuracy to model the correct GW signals in general relativity, they require high computational resources and need weeks to months to generate.

Combining the two approaches above, new methods have been developed to model full waveforms. Two major families of this group, namely the effective-one-body (EOB) [9–12] and the *phenomenological* models [13–18] are commonly used in GW analyses. In general, these models start from a reformulation of PN results and calibrate the model to a select number of NR simulations. In this study, we employ `SEOBNRv3` [11] and `IMRPhenomPv2` [17, 18] as two representative models that have been widely used to explore the full parameter space of non-eccentric, precessing BBHs.

Over the past few years, complementary techniques have been developed to build fast surrogates of EOB models and NR waveforms with a much higher computational efficiency. Unlike the previous approaches, these models do not start from PN expansions. They use existing EOB or NR waveforms, decompose, and interpolate them. The NRSurrogate models [12, 19–24] have an exceptional accuracy against the original NR signals, but are more limited in the parameter range and waveform length they cover. Reduced order and surrogate models of EOB waveforms have been crucial to allow EOB models to be used for template bank construction [25] and parameter estimation [26, 27].

In a similar spirit, unique methods have been explored to speed up the waveform generation without compromising accuracy [28–32]. They have shown that advanced mathematical, statistical, and computational techniques are needed to build waveform models optimized for the demands of GW analyses.

We stress that in order to make a relatively small number of computationally expensive waveforms usable for analysis applications that rely on the ability to freely vary all parameters, all waveform models described above crucially rely on some form of *interpolation* or fitting method as part of their construction. Phenomenological and EOB models typically fit free coefficients (often representing unknown, higher-order PN contributions) to a set of NR data. The fits or interpolants are then evaluated over the binary parameter space. Other approaches, such as NR or EOB surrogate models, rely more on data-driven techniques to interpolate the key quantities needed to reconstruct waveforms anywhere in a given parameter-space region. In fact, the interpolation techniques that have recently been employed cover standard methods such as polynomial fits [12, 16, 33], linear interpolation [30, 34], and more complex method such as Gaussian process regression (GPR) [21, 32, 35–37]. Additionally, novel interpolation methods have been developed such as greedy multivariate polynomial fits (GMVP) [29, 31] and tensor-product-interpolation (TPI) [23, 24]. References [12, 16, 23, 24, 29, 31–34] have been used to build waveform models and have been implemented in the analysis of the LIGO data. This study compares various regression methods to investigate their prospect of building GW model with more dimensional parameters.

In this study, we investigate the importance of interpolation and fits in waveform models (which themselves are crucial for GW astronomy), given the accuracy and computational time of various regression methods. We study whether the use of more complicated methods to model the waveforms given the same data preparation and noise reduction is justified in practice. Finally, we compare the performance of machine-learning against various traditional methods. In particular, we explore the prospects of artificial-neural-networks (ANN) as a regression method [38, 39] that has not been widely employed in waveform modeling so far. We focus on BBH systems with spins either aligned with the orbital angular momentum or precessing and provide both theoretical overviews and references to practical tools such as *ready-to-use* algorithms. Our analysis is not only of relevance for current LIGO and Virgo data and their extensions such as the Advanced LIGO A+, Voyager [40], and KAGRA [41], but also for future analysis of GW data by LISA [42] and the third generation instruments such as Einstein Telescope [43] and Cosmic Explorer [44].

The testbed we use is as follows. We compare various methods on waveform data at a fixed point in time as a function of mass ratios and spins. We use two models to generate waveform data: the time-domain model `SEOBNRv3` [11], and the inverse Fourier transform of `IMRPhenomPv2` [17, 18] which is natively given in the frequency domain. Both models were designed for precessing BBH mergers which are described by seven intrinsic parameters: the mass ratio q and the two spin vectors $\vec{\chi}_1$ and $\vec{\chi}_2$ with Cartesian components in the x, y, z directions. `IMRPhenomPv2` models precessing waveforms in a single spin approximation using an effective precession spin parameter.

We consider two classes of training data:

(i) Data on a regular three-dimensional grid describing nonprecessing binaries, $(q, \chi_{1z}, \chi_{2z})$, where $1 \leqslant q \leqslant 10$ and $|\chi_{iz}| \leqslant 1$ for $i = 1, 2$.
(ii) Random uniform data on a full seven-dimensional grid $(q, \vec{\chi}_1$ and $\vec{\chi}_2)$, where $1 \leqslant q \leqslant 2$ and $-1/\sqrt{3} \leqslant \vec{\chi}_i \leqslant 1/\sqrt{3}$ for $i = 1, 2$.

For each case, the regression methods were tested over test sets made up from random uniform test points that were drawn independently of the training set, but covering the same physical domain.

This paper is organized as follows. We prepare the data by defining the waveform and its reference frame and defining waveform data pieces in a precession adapted frame as discussed more detail in section 2.1. We explain the background and the features of traditional methods such as linear interpolation, TPI, polynomial fit, GMVP, and radial basis functions (RBF) as well as machine-learning methods, GPR and ANN in section 2.2. In section 3 we present the results of our study. Finally, a brief conclusion and discussion of future studies are found in section 4. Throughout the manuscript, we employ geometric units with the convention $G = c = 1$.

2. Method

2.1. Waveform data

We generate training and test waveform datasets for various regression methods from two state-of-the art models of the GWs emitted by merging BBHs. We use the phenomenological model `IMRPhenomPv2` [14, 16, 18] and the effective-one-body model `SEOBNRv3` [11, 45, 46]. `IMRPhenomPv2` includes an effective treatment of precession effects, while `SEOBNRv3` incorporates the full two-spin precession dynamics. The models have been independently tuned in the aligned-spin sector to NR simulations.

The GW strain can be written as an expansion into spin-weighted spherical harmonic modes in the inertial frame

$$h(t;\vec{\lambda};\theta,\phi) = \sum_{\ell=2}^{\infty}\sum_{m=-\ell}^{\ell} h_{\mathrm{i}}^{\ell,m}(t;\vec{\lambda})\,{}_{-2}Y_{\ell,m}(\theta,\phi). \tag{1}$$

We can choose to model the waveform modes $h_{\mathrm{i}}^{\ell,m}(t;\theta)$ directly which depend a collection of parameters $\vec{\lambda}$. The spherical harmonics ${}_{-2}Y_{\ell,m}(\theta,\phi)$ for a given (ℓ,m) depend on the direction of emission described by the polar and azimuthal angles θ and ϕ. The two waveform models employed in this study provide approximations to the dominant modes at $\ell=2$. In a precession adapted frame `SEOBNRv3` includes $m=\pm2$ and $m=\pm1$ modes (the negative m modes by symmetry), whereas `IMRPhenomPv2` includes only the $m=\pm2$ modes. For `SEOBNRv3` we directly generate time-domain inertial modes $h_{\mathrm{i}}^{2,m}(t)$, while for `IMRPhenomPv2` we compute the native inertial modes in the Fourier domain $\tilde{h}_{\mathrm{i}}^{2,m}(f)$, and subsequently condition and inverse Fourier transform them to obtain an approximation to the time-domain modes.

To test interpolation methods we work in the setting of the empirical interpolation (EI) method [20, 28]. In this approach we can define an *empirical interpolant* of waveform data piece $X(t;\vec{\lambda})$ (such as, e.g. amplitude or phase of the gravitational waveform) by

$$I_N[X](t;\vec{\lambda}) = \sum_{i=1}^{N} c_i(\vec{\lambda})\mathrm{e}^{\mathrm{i}}(t) = \sum_{j=1}^{N} X(T_j;\vec{\lambda})b^j(t). \tag{2}$$

The first expression is an expansion with coefficients c_i of waveform data in an orthonormal linear basis $\{\mathrm{e}^{\mathrm{i}}(t)\}_{i=1}^{N}$ (e.g. obtained from computing the singular value decomposition [47, 48] for discrete data [23, 24]). A transformation to the basis $\{b^i(t)\}$ results in coefficients which are the waveform data piece X evaluated at empirical node times T_j. The EI basis $\{b^i(t)\}$ and the EI times can be obtained by solving a linear system of equations as discussed in [28]. Here we forgo the basis construction step and just choose EI times manually to select waveform data for accessing regression methods.

We want to transform the inertial frame modes into a more appropriate form, such that data pieces are as simple and non-oscillatory as possible in time and smooth in their parameter dependence on $\vec{\lambda}$. In evaluating the model, we reconstruct the full waveforms by transforming back to the inertial frame. This transformation includes the choice of a precession adapted frame of reference that follows the motion of the orbital plane of the binary. In this frame the waveform modes have a simple structure and are well approximated by non-precessing waveforms. A further simplification in the modes can be achieved by taking out the orbital motion. In addition, we align the waveform and frame following [20] at the same time for different configurations and waveform models. The procedure is comprised of the following steps[4]:

[4] We represent rotations through unit quaternions. Quaternions can be notated as a scalar plus a vector $\mathbf{Q} = q_0 + \mathbf{q} = (q_0, q_1, q_2, q_3)$. A unit quaternion $\mathbf{R} = \mathrm{e}^{\theta\hat{\mathbf{u}}/2}$ generates a rotation through the angle θ about the axis $\hat{\mathbf{u}}$. For calculations we use the `GWFrames` [49] package and notation conventions from [49].

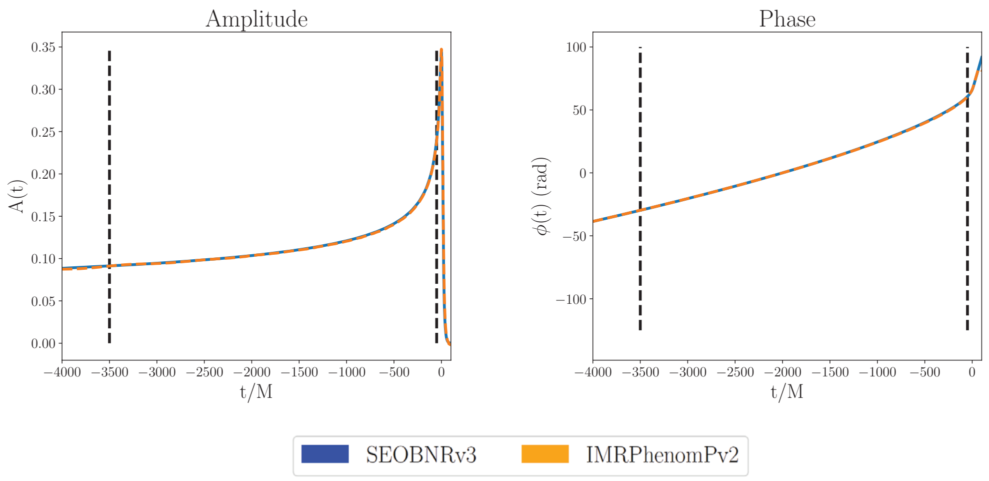

Figure 1. The key quantities of the GW signal of a precessing BBH, here illustrated for a binary with $(q, \chi_{1x}, \chi_{1y}, \chi_{1z}, \chi_{2x}, \chi_{2y}, \chi_{2z}) = (1.99, 0.51, 0.04, 0.03, 0.01, 0.6, 0.1)$. Left: the dimensionless amplitude $A(t)$. Right: the phase $\phi(t)$ (in unit radian). The black dashed lines show the points in time-space, where we perform different interpolation methods ($t = -3500M$ and $t = -50M$).

- We define time relative to the peak of the sum of squares of the inertial frame modes.
- We transform the inertial frame waveform modes $h_{\mathrm{i}}^{\ell,m}(t)$ (dropping the parameter dependence on $\vec{\lambda}$ for now) to the minimally rotating co-precessing frame [50] and thereby obtain the co-precessing waveform modes

$$h_{\mathrm{copr}}^{2,m}(t) = \sum_{m'} h_{\mathrm{i}}^{2,m}(t)\mathcal{D}^{2}_{m',m}\left(\mathbf{R}_{\mathrm{copr}}(t)\right), \tag{3}$$

  where $\mathcal{D}^{\ell}_{m',m}$ are Wigner matrices [49, 51] and $\mathbf{R}_{\mathrm{copr}}(t)$ is the time-dependent unit quaternion which describes the motion of this frame.
- We compute the Newtonian orbital angular momentum unit vector $\hat{\mathbf{l}}_N(t) = \mathbf{R}_{\mathrm{copr}}(t)\,\hat{\mathbf{z}}\,\mathbf{R}^*_{\mathrm{copr}}(t)$, where $\mathbf{Q}^*$ is the conjugate of the quaternion $\mathbf{Q}$ and $\hat{\mathbf{z}} = (0, 0, 1)$. We interpolate $\hat{\mathbf{l}}_N(t)$ to the desired alignement time t_{align}.
- We use the rotor $\mathbf{R_a} = \sqrt{-\hat{\mathbf{l}}_N(t_{\mathrm{align}})\,\hat{\mathbf{z}}}$ that rotates $\hat{\mathbf{z}}$ into $\hat{\mathbf{l}}_{\mathbf{N}}(t_{\mathrm{align}})$ to align the inertial modes at t_{align} and then compute the aligned co-precessing frame modes $\bar{h}^{2,m}_{\mathrm{copr}}(t)$ and quaternion time series $\bar{\mathbf{R}}_{\mathrm{copr}}(t)$, where the bar indicates alignment in time.
- Finally, we rotate around the z-axis to make the phases of the $(2, 2)$ and $(2, -2)$ modes small by applying a fixed Wigner rotation with the rotor $\mathbf{R_z} = \exp(\theta/2\,\hat{\mathbf{z}})\,\bar{\mathbf{R}}_{\mathrm{copr}}$ to obtain $\bar{\bar{h}}^{2,m}_{\mathrm{i}}(t)$ and $\bar{\bar{h}}^{2,m}_{\mathrm{copr}}(t)$.

We choose the following quantities (see figure 1) to test the accuracy and efficiency of interpolation methods: (i) the 'orbital phase' defined as one quarter the averaged GW-phase from the $(\ell, m) = (2, 2)$ and $(2, -2)$ modes in the co-precessing frame

$$\phi(t) := \frac{1}{4}\left(\arg\left[\bar{\bar{h}}^{2,-2}_{\mathrm{copr}}(t)\right] - \arg\left[\bar{\bar{h}}^{2,2}_{\mathrm{copr}}(t)\right]\right), \tag{4}$$

(ii) a linear combination of the $\ell = m = 2$ modes in the co-orbital frame

$$A(t) := \mathrm{Re}\,\bar{\bar{h}}_{+}^{2,2} = \frac{1}{2}\mathrm{Re}\,\left(\bar{\bar{h}}_{\mathrm{coorb}}^{2,2}(t) + \bar{\bar{h}}_{\mathrm{coorb}}^{2,-2^{*}}(t)\right), \tag{5}$$

where the co-orbital modes are defined as

$$h_{\mathrm{coorb}}^{\ell,m}(t) = h_{\mathrm{copr}}^{\ell,m}(t)\mathrm{e}^{\mathrm{i}m\phi(t)}. \tag{6}$$

The rationale for choosing these two quantities is the following: the phasing is usually the quantity that requires the most care in GW-modeling with accuracy requirements of a fraction of a radian over hundreds of waveform cycles. The co-orbital frame mode combinations play the role of a generalized amplitude and are typically smooth and non-oscillatory.

We consider the following waveform training datasets in this study: (i) three-dimensional datasets: several interpolation methods in this study require data on a regular grid. We prepare three-dimensional datasets $(q, \chi_{1z}, \chi_{2z})$ in the mass-ratio $q = m_1/m_2$ and the aligned component spins $\chi_{iz} = \vec{S}_i \cdot \hat{L}_N/m_i^2$ for $i = 1, 2$. We do not include the total mass since it can be factored out from the waveform for GWs emitted from BBHs which are solutions of Einstein's equations in vacuum. The grids have an equal number of points per dimension, ranging from 5 to 11. We choose parameter ranges $1 \leqslant q \leqslant 10$ and $|\chi_{iz}| \leqslant 1$. (ii) The full intrinsic parameter space we consider is seven-dimensional: we include the dimensionless spin vector of each black hole $\chi_i = S_i/m_i^2$ and the mass-ratio q of the binary. Due to the *curse of dimensionality* regular grid methods require a prohibitive amount of data in 7D. For instance, ten points per dimension would require 10^7 waveform evaluations. Therefore, we only produce scattered waveform data in seven dimensions which are drawn from a random uniform distribution in each parameter. Here we choose parameter ranges $1 \leqslant q \leqslant 2$ and $-1/\sqrt{3} \leqslant \vec{\chi}_i \leqslant 1/\sqrt{3}$. For both choices of dimensionality we also generate test data of 2500 points drawn randomly from the respective parameter space.

Waveform data in three and seven dimensions is produced at a total mass of $M = 50M_{\odot}$ with a starting frequency of 20Hz. We align the waveform and frames at $t_{\mathrm{align}} = -2000M$ with the above procedure. We record waveform data from the key quantities at two different times, $t_{\mathrm{target}} = -3500M$ and $-50M$, where we have performed alignment in time such that the mode sum of the waveform amplitudes peaks at $t = 0M$. This choice allows us to independently probe the inspiral and the merger regime. We expect that the waveform data will be very smooth in the inspiral, but more irregular close to merger due to the calibration of internal model parameters to numerical relativity waveforms at a limited number of points in parameter space.

2.2. Regression methods: a general overview

A large number of techniques have been developed to improve the speed and accuracy of generating gravitational waveforms. *A priori*, one would expect that higher speed would go hand-in-hand with less accuracy and less complexity. One frequent question is how to select a method for a specific purpose. Depending on the goals, a choice needs to be made between complex, highly accurate methods with moderate efficiency versus simpler but more efficient methods, and we can choose to trade accuracy for speed.

In this subsection, we discuss various methods and categorize them into two groups. The first group is comprised of traditional interpolation and fitting methods which are based on mathematical techniques and algorithms that are straightforward to implement and easily evaluated. The second group is made up of machine-learning methods which may require a more advanced mathematical and computational background. Methods from the second group are in general more complex and require more computational resources than the first group. Here we give a basic description of these methods, their limitation and provide some references.

2.2.1. Traditional interpolation and fitting methods. The traditional interpolation and fitting methods are either interpolatory, i.e. the approximation is designed such that it exactly includes the data points, or they produce an approximate fit, where a distance function between the data and the model is minimized. Many of these methods rely on polynomials as building blocks to model the data. Some models have a fixed order of approximation, while others let the number of terms be a free parameter. These methods are relatively straightforward to use and do not usually require much computational power.

(i) **Linear interpolation**

Linear interpolation is a straight line approximation that predicts the value of an unknown data point which lies between two known points [52]. This method has been widely used as a standard method to perform interpolation in various studies. If we have several data points, the transition between the adjacent data points is only continuous but not smooth. Since linear interpolation is available as a standard Python package, we include this method to compare to other more complicated techniques. In particular, we investigate the application of multivariate linear interpolation on a regular grid using the regular grid interpolator (RGI) [53, 54] that is available in `scipy` [55].

The mathematical background of linear interpolation can be explained as follows. Assume two known points (x_0, y_0) and (x_1, y_1) and an unknown point (x, y) with $x_0 \leqslant x \leqslant x_1$. This method assumes that the slope between x_0 and x is equal to the slope between x and x_1. Hence, we use the following relation to predict the data point y in one dimension.

$$\begin{aligned} &\frac{y - y_0}{x - x_0} = \frac{y_1 - y}{x_1 - x} \\ &\Leftrightarrow y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0}. \end{aligned} \tag{7}$$

In dimensions $d > 1$, this method requires a regular grid of data points as a training set. Multivariate linear interpolation works as follows. Let $y_i(\vec{x})$ be the data point we want to predict, where $\vec{x}$ denotes the input parameters in d dimensions. Initially, we need to obtain the parameters of the projection of $y_i(\vec{x})$ in $d - 1$ dimensions, followed iteratively by $d - 2$ and so on until we reach one-dimensional case $d = 1$. Once we obtain these projection points, we can employ equation (7) to predict the values of these points in one dimension. Subsequently, we use the predicted values as the known points to predict the result in higher dimensions iteratively. We then repeat the process further to find $y_i(\vec{x})$ in d dimensions. This algorithm involves a small number of multiplications and additions, which are relatively fast.

Since RGI assumes a regular grid, it is affected by *the curse of dimensionality*: the number of training points grows as the power of d. Therefore, we only investigate this method in three dimensions.

Other popular regression methods that we do not consider here are ridge [56], LASSO [57], and Bayesian regression [58]. One reason is that the GW training data is quite well-behaved and does not usually include outliers such that would require special treatment.

(ii) **Tensor product interpolation**

On regular or Cartesian product grids one can use the same univariate interpolation method in each dimension and the grid points can be unequally spaced. This gives rise to

TPI methods. Popular choices for the univariate method are splines [59] and, if the data is very smooth, spectral interpolation [60, 61].

Let us assume that we want to model a waveform quantity $X(t;\vec{\lambda})$ at a particular time $t = t_{\rm i}$. We define the d-dimensional TPI interpolant (where $d = \dim(\vec{\lambda})$) as an expansion in a tensor product of one-dimensional basis functions $\Psi_j(\lambda_j)$,

$$I[X](t_{\rm i};\vec{\lambda}) = \sum_{j_1,\ldots,j_d} a_{j_1,\ldots,j_d}\,(\Psi_{j_1}\otimes\cdots\otimes\Psi_{j_d})\,(\vec{\lambda}). \tag{8}$$

A popular choice for the basis functions are univariate splines, which are piecewise polynomials of degree $k-1$ (order k) with continuity conditions. For instance, cubic splines have degree $k = 4$ and continuous first and second derivatives. The boundaries of the domain require special attention. A simple choice is the natural spline where the second derivative is set to zero at the endpoints. If boundary derivatives are not known it is better to use the so-called 'not-a-knot' boundary condition [59]. This condition is defined by demanding that even the third derivative must be continuous at the first and last knots.

To construct splines in a general manner it is advantageous to introduce basis functions with compact support, so-called *B-splines*. We denote the ith B-spline basis function [59, 62] of order k with the knots vector $\vec{t}$, a nondecreasing sequence of real numbers, evaluated at x by $B_{i,k,t}(x)$. The knots refer to the locations in the independent variable where the polynomial pieces of B-spline basis function are connected. For distinct knots $t_i,\ldots,t_{i+k+1}$, the B-splines can be defined as

$$B_{i,k,t}(x) := (t_{i+k}-t_i)[t_i,\ldots,t_{i+k}](\cdot - x)_+^{k-1}, \tag{9}$$

where $[t_i,\ldots,t_{i+k}]f$ is the *divided difference* [59, 62] of order k of the function f at the sites $t_i,\ldots,t_{i+k}$, and $(x)_+ := \max\{x,0\}$. The B-splines can also be defined in terms of recurrence relations. The definition can be extended to partially coincident knots which are useful for the specification of boundary conditions. B-splines can be shown to form a basis [59] of the spline space for a given order and knots vector. A spline function or spline of degree k with knots $\vec{t}$ can be then defined as an expansion

$$s = \sum_i s_i B_{i,k,t}(x), \tag{10}$$

with real coefficients $\{s_i\}_{i=1}^n$. Given data, a fixed order and knots vector, and a choice of boundary conditions, we can solve the linear system for the spline coefficients s_i. For efficient evaluation we only compute the parts of the B-spline basis functions that are nonzero.

For smooth data, Chebyshev interpolation [60, 61] is a popular choice. Chebyshev polynomials (of the first kind) are defined as the unique polynomials satisfying

$$T_n(\cos(\theta)) = \cos(n\theta) \tag{11}$$

on [–1,1]. In contrast to splines where the polynomial degree is usually low, global high order polynomial interpolation requires a special choice of nodes to be well-conditioned. A good choice are Chebyshev-Gauss-Lobatto nodes (which are defined to be the extrema of the $T_n(x)$ plus the endpoints of the domain)

$$x_k = -\cos\left(\frac{k\pi}{m-1}\right), \quad k = 0,\ldots,m-1. \tag{12}$$

Then we can approximate a function $f(x)$ by an expansion

$$f(x) \approx I[f(x)] := \sum_{k=0}^{m-1} c_k T_k(x). \tag{13}$$

For $f \in C^\infty$ the error of Chebyshev interpolation converges exponentially with the number of polynomials $T_n(x)$.

Tensor product interpolation is a very useful tool for constructing fast *reduced order models* (ROM) or *surrogate models* of time or frequency dependent functions that depend on a moderate number of parameters $\vec{\lambda}$. TPI with splines and Chebyshev polynomials has been used to build several GW models [21, 23, 24, 29] and [63], respectively. TPI is not available in standard Python packages. For TPI spline interpolation we use the Cython [64] implementation in the `TPI` package [65].

(iii) **Polynomial fits**

A polynomial fit is a multiple linear regression model where the independent variables form a polynomial [66]. Different settings of maximum polynomial degrees may cause *underfitting* or *overfitting*, therefore care must be taken in choosing the ansatz.

Assume that we have N training points $(\{\vec{x}_i, y_i\} \in \mathbb{R}^d \times \mathbb{R} | i = 1, \cdots, N)$. Our goal is to find a function or regressor such that each $\vec{x}_i$ yields an output with the lowest error against its function values y_i. We assume that this function $f(\vec{x})$ is expressed by a polynomial of degree k and parameters $\vec{c}$.

In one dimension we have:

$$f(\vec{x}) = c_0 x^k + c_1 x^{k-1} + \cdots + c_{k-1} x + c_k. \tag{14}$$

If we had as many degree of freedom as data points, we could demand:

$$f(x_i) = y_i. \tag{15}$$

In matrix form, equation (15) can be written as:

$$\mathbf{X}\vec{c} = \vec{Y}$$
$$\begin{pmatrix} x_1^k & x_1^{k-1} & \cdots & x_1 & 1 \\ x_2^k & x_2^{k-1} & \cdots & x_2 & 1 \\ \vdots & & \ddots & & \vdots \\ x_N^k & x_N^{k-1} & \cdots & x_N & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_k \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, \tag{16}$$

where $\mathbf{X}$ is the $N \times (k+1)$ Vandermonde matrix. The parameters $\vec{c}$ are obtained by solving equation (16) for the known input and output data, $\mathbf{X}$ and $\vec{Y}$ in the training set. In general, the linear system may be over or under determined such that no unique solution would exist. Instead, we employ the standard discrete least squares fit to minimize the error (see section 10 of [67] and [68]):

$$\Sigma_{j=1}^N |f(x_j) - y_j|^2. \tag{17}$$

Similar to linear interpolation, univariate polynomial interpolation is available in the `scipy` package.

Phillips [66] discusses several methods and provide an overview of multivariate interpolation with polynomials. We employ polynomial fits for multivariate interpolation as in [69] and explained more detail in [70].

(iv) **Greedy multivariate polynomial fit (GMVP)**

London and Fauchon-Jones [31] recently introduced methods that build an interpolant for a given data set by adaptively choosing a small set of analytical basis function from a certain class of functions. In our study here, we test the GMVP procedure described in detail in section II.B of [31].

In this method, a scalar function, f, that is known at discrete points in the d-dimensional parameter space, $\vec{x}_j = \{x_j^1, x_j^2, \ldots, x_j^d\}$, is approximated by a linear sum of analytical basis functions, $\phi_k(\vec{x})$,

$$f(\vec{x}) \approx \sum_k \mu_k\, \phi_k(\vec{x}). \tag{18}$$

Given a set of basis functions, the coefficients μ_k are determined by a 'least-squares' optimal fit to the known function values $f(\vec{x}_j)$. In practice, this is calculated using the pseudoinverse (Moore–Penrose) matrix of $\phi_k(\vec{x}_j)$ (that is, the values of the basis functions at the given location in the parameter space).

In GMVP, the basis functions are chosen to be multivariate polynomials of maximal degree D. In order to prevent overfitting, however, not all possible polynomial terms from the set

$$\phi_k(\vec{x}) \in \left\{ \left(x^1\right)^{\alpha_1} \left(x^2\right)^{\alpha_2} \ldots \left(x^n\right)^{\alpha_d}, \quad \sum_{i=1}^{n} \alpha_i \leqslant D \right\} \tag{19}$$

are included in the basis. Instead, a greedy algorithm [70] iteratively adds the basis functions to (18) that minimize the error

$$\epsilon^2 = \frac{\sum_j \left[f(\vec{x}_j) - \sum_k \mu_k\, \phi_k(\vec{x}_j)\right]^2}{\sum_j \left[f(\vec{x}_j)\right]^2}. \tag{20}$$

This process terminates when the difference in ϵ between two successive iterations becomes smaller than some user-defined tolerance. In order to improve the stability of the algorithm, the maximally allowed multinomial degree D is successively increased, which the authors of [31] refer to as degree tempering.

In our study, we use GMVP with a tolerance of $\epsilon = 5 \times 10^{-4}$ and a maximal multinomial degree of $D = 16$.

(v) **Radial basis functions (RBF)**

Radial basis functions [71] are an approximation for continuous functions, where the predicted outputs depend on the Euclidean distance between the points and a chosen origin. This method is applicable in arbitrary dimensions and does not require a regular grid. We include RBF in this study because this method has been integrated as a standard Python package in `scipy` and used in machine-learning as activation functions in radial basis functions neural networks (see section 2.2.2).

The mathematical background of RBFs is explained as follows. Let N be the number of training points, $\vec{x}_i$ the parameters of each data point, and y_i the data defining the training set $\{(\vec{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R} | i = 1, \ldots, N\}$.

The goal is to find an approximant $s : \mathbb{R}^d \to \mathbb{R}$ to the function $y : \mathbb{R}^d \to \mathbb{R}$ such that $s(\vec{x}_i) = y_i$ (s interpolates y at the chosen points) with the form:

$$s(\vec{x}) = \sum_{i=1}^{N} w_i \varphi(r), \tag{21}$$

where $\vec{x}$ is the vector of independent variables, w_i are the coefficients, r is the Euclidean distance between $\vec{x}$ and $\vec{x}_i$ ($r = \|\vec{x} - \vec{x}_i\|$), and $\varphi(r)$ is known as the *radial basis function*. To obtain the approximant s, we need to solve:

$$\Phi(r)\vec{w} = \vec{Y}, \tag{22}$$

where $\Phi(r) = \{\|\vec{x} - \vec{x_i}\|\}_{x,x_i \in \Xi}$, $\vec{Y} = \{y_i\}_{i=1}^{N}$ and $\vec{w} = \{w_i\}_{i=1}^{N}$. Ξ is a finite subset of $\mathbb{R}^d$ with more than one element [71]. We can solve the linear system for the coefficients and obtain the interpolant. Hence, the computational complexity and thus the training time of RBF is dominated by the computation of vector coefficients $\vec{w}$ that involves matrix inversion and goes as $\mathcal{O}(N^3)$ [72].
The interpolation matrix $\Phi(r)$ has to be nonsingular so that it does not violate the Mairhuber–Curtis theorem [71]. The solution is to choose a kernel function such that $\Phi(r)$ is a semi-definite matrix and therefore nonsingular. One common choice is the multiquadric kernel function $\varphi(r)$ expressed by:

$$\varphi(r) = \sqrt{1 + \left(\frac{r}{\varepsilon}\right)^2}, \tag{23}$$

where ε is the average distance between nodes based on a bounding hypercube as defined in `scipy` [73].

The multiquadric kernel function is commonly applied to scattered data because of its versatility due to its adjustable parameter ε which can improve the accuracy or the stability of the approximation. Buhmann [71] shows that this kernel is also able to approximate smooth functions well so that it useful for approximation. Hence, we employ the *multiquadric* kernel function in this study.

2.2.2. Machine-learning methods. Machine-learning (ML) is the scientific study of computer algorithms and statistics which aims to find patterns or regularities in the data sets. Systems learn from the training data and can predict output values for test data.

Although the distinction is a blur, one major difference between ML and traditional interpolation methods lies in their objectives. In traditional methods, the objective is not only to provide an approximation of an underlying function from which the training data were generated, but also to understand the mathematical process behind the relation of input and output data. In that case, we seek interpolants or fits which often can be found analytically by solving linear systems for the coefficients in the model. Hence, the traditional methods originated from approximation theory and numerical analysis in mathematics. Conversely, in ML, the objective is to recognize patterns from the input-output training set and to construct a model from this data [74, 75]. Although we know that the result follows some mathematical procedures that depend on free parameters, these details are considered to be less important.

(i) **Gaussian process regression (GPR)**
GPR is a unique method that combines statistical techniques and ML. It can predict function values away from training points and can provide uncertainties of the predicted values, which will be useful for certain applications and do not require a regular grid. Compared to traditional methods, GPR requires more knowledge of advanced statistics such as covariance matrices, regression and Bayesian statistics for the optimization strategy.
We provide a summary of GPR as discussed in detail in [76, 77]. We start with the most important assumption in GPR. Any discrete set of function values $y_i = y(\vec{x_i})$ is assumed

to be a realization of a Gaussian process (GP). Assuming the data can be pre-processed to have zero mean, $\mu(\vec{x}) = 0$, the covariance function $k(\vec{x}, \vec{x}')$ fully defines the Gaussian process:

$$y(\vec{x}) \sim GP\Big(\mu(\vec{x}) = 0, k(\vec{x}, \vec{x}')\Big). \tag{24}$$

Assume that we want to predict the value y_* at $\vec{x}_* \in \mathbb{R}^d$ and that we have N numbers of training points, where each point depends on d parameters expressed by $\{(\vec{x}_i, y_i)|i = 1, \ldots, N\}$. The training and test outputs can be written as follows:

$$\begin{bmatrix} \vec{y} \\ y_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right), \tag{25}$$

where $K(X,X)$ denotes the matrix of the covariances evaluated at all pairs of the training points and similarly for $K(X_*,X_*)$, $K(X, X_*)$, and $K(X_*, X)$, σ_n^2 (also called *nugget*) is the variance of the Gaussian (white) *noise* kernel that will be discussed later (see the *hyperparameters*).

Explicitly, in order to predict a single value y_*, we need to compute $K(X,X)$ as the covariance matrix between each point in the training set, $K(X,X_*)$ and its transpose that are vectors and the scalar $K(X_*,X_*)$. In a different form, our main goal is to find the conditional probability expressed by the following distribution:

$$p(y_*|\vec{x}_i, \vec{x}_*, \vec{y}, \vec{\theta}) = \mathcal{N}(\bar{y}_*, \mathrm{var}(y_*)), \tag{26}$$

i.e. the probability of finding the value y_* given the training data $\vec{x}_i$ and $\vec{y}$, the hyperparameters $\vec{\theta}$, and the location $\vec{x}_*$ is a normal distribution with mean $\bar{y}_*$ and variance $\mathrm{var}(y_*)$.

The mean and variance can be shown to be:

$$\bar{y}_* = K(X_*,X)(K(X,X))^{-1}_{ij} y_j \tag{27}$$

$$\mathrm{var}(y_*) = K(X_*,X_*) - K(X_*,X_i)(K(X,X))^{-1}_{ij} K(X_*,X_j). \tag{28}$$

In the equation above, the covariance $K(x_i, x_j)$ is expressed by:

$$K(x_i,x_j) = \sigma_f^2 k(x_i,x_j) + \sigma_n^2 \delta_{ij}, \tag{29}$$

where σ_f and σ_n are hyperparameters, δ_{ij} is the standard Kronecker delta, $k(x_i,x_j) = k(r)$, and r is the distance:

$$r = \sqrt{(\vec{x} - \vec{x}')^T M (\vec{x} - \vec{x}')}. \tag{30}$$

In the following, we discuss the form of M as a diagonal matrix with a tunable length scale in each physical parameter which form part of the hyperparameters.

The hyperparameters

We assume that our training data has some numerical noise σ_n^2 and a scale factor σ_f that can be estimated by optimizing the hyperparameters $\vec{\theta} = \{\sigma_f, \sigma_n, M\}$. For instance, the explicit form of M in the seven-dimensional case is:

$$M = \mathrm{diag}(\ell_q^{-2}, \ell_{\chi_{1x}}^{-2}, \ell_{\chi_{1y}}^{-2}, \ell_{\chi_{1z}}^{-2}, \ell_{\chi_{2x}}^{-2}, \ell_{\chi_{2y}}^{-2}, \ell_{\chi_{2z}}^{-2}), \tag{31}$$

where the ℓ_i are length scales. Rasmussen and Williams [78] describes the length-scale ℓ as the distance taken in the input space before the function value changes significantly. Small values of the lengthscale ℓ imply that the function values change quickly and vice versa. Hence, the lengthscale ℓ describes the smoothness of a function.
To determine the hyperparameters, we can maximizse the marginal log-likelihood:

$$\ln p(y_i|\vec{x}_i, \vec{\theta}) = -\frac{1}{2}\left(y_i (K(X,X))_{ij}^{-1} y_j + \ln |K(X,X)| + N \ln 2\pi \right). \tag{32}$$

Because the log-likelihood may have more than one local optimum, we repeatedly start the optimizer and we choose ten repetitions. For the first run, we set the initial value of each length scale to unity, with bounds of 10^{-5} to 10^5. Furthermore, we set $\sigma_n^2 = 10^{-10}$, where higher σ_n^2 value means that the data are more irregular. The subsequent runs use the allowed values of the hyperparameters from the previous runs until the maximum number of iterations is achieved.
In equation (32), we see that the partial derivatives of the maximum log likelihood can be computed using matrix multiplication. However, the time needed for this computation grows with more data in the training set as $\mathcal{O}(N^3)$. Additionally, we employ algorithm 2.1 of [76], because Cholesky decomposition is about six time faster than the ordinary matrix inversion to compute equation (32). We highlight that although GPR becomes more accurate in predicting the underlying functional form of the data given more training points N, it has complexity $\mathcal{O}(N^3)$ and therefore the method becomes ineffective for large N.
We estimate the posterior distribution of the hyperparameters using Bayes' theorem as follows:

$$p(\vec{\theta}|\vec{x}_i, y_i) \propto p(\theta)\, p(y_i|\vec{x}_i, \vec{\theta}), \tag{33}$$

where we employ a uniform prior distribution $p(\theta)$. Additionally, we use the `sckit-learn` package [77] to optimize the hyperparameters as in the implementation of algorithm 2.1 in [76].
This method is non-parametric because no direct model ansatz is used. Note however that a choice for the covariance function needs to be made.

The covariance functions
In statistics, covariance expresses how likely two random variables change together [79]. Various choices of covariance functions which are usually called kernels $k(\vec{x}, \vec{x}')$ are discussed in more detail in [77] and [76]. In this study, we compare the two most commonly used kernel functions in GPR: the squared exponential kernel and the Matérn kernel explained below.

(a) The squared exponential kernel (SE) is a standard kernel for Gaussian processes:

$$k_{\mathrm{SE}}(r) = \exp\left(\frac{-r^2}{\ell^2}\right), \tag{34}$$

with r defined in equation (30) and ℓ is the length-scale.

(b) The Matérn class of kernels is named after a Swedish statistician, Bertil Matérn and has less smoothness than the SE kernel. The Matérn kernel is given by:

$$k_M(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}r}{\ell}\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}r}{\ell}\right), \tag{35}$$

where K_ν is a modified Bessel function [80], Γ is the gamma function and ν is usually half-integer. Common choices of ν are $k_{\nu=3/2}$ and $k_{\nu=5/2}$.

$$k_{\nu=3/2}(r) = \left(1 + \frac{\sqrt{3}r}{\ell}\right)\exp\left(-\frac{\sqrt{3}r}{\ell}\right), \tag{36}$$

$$k_{\nu=5/2}(r) = \left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right)\exp\left(-\frac{\sqrt{5}r}{\ell}\right). \tag{37}$$

The Matérn kernel is a generalization of the radial basis function kernel. For $\nu = 1/2$, it reduces to exponential kernel and $\nu = \infty$ reduces to the SE kernel. We use the Matérn kernel with $\nu = 3/2$ in our analysis.

(ii) **Artificial neural networks**

Artificial neural networks (ANNs) as computing systems are inspired by emulating the work of brains to learn complex things and to find patterns in biology. In ML algorithms, ANN has been widely used as a framework to perform advanced tasks such as pattern recognition [81], forecasting [82], and many other applications in various disciplines [83]. This framework works analogously to brains: it receives some inputs, processes them, and yields some output [75].

In this study, we employ ANNs or feedforward networks as the simplest neural networks architecture to perform interpolation. The feedforward network with hidden layers can approximate of any function which is known as the universal approximation theorem [75, 84]. This class is called *feedforward* because the information flow from the input to the output and the connection between them does not form a cycle (loop). In our case, the inputs are the waveform's parameters $\vec{\lambda}$ and the output is the predicted value of $A(t_i;\vec{\lambda})$ or $\phi(t_i;\vec{\lambda})$. We define *hidden layer* as a layer between the input and the output of ANN[5].

We employ multi-layer-perceptron (MLP) as one of the simplest architectures to perform function approximation [84, 85]. Figure 2 shows the illustration of the network architecture used in this study.

In figure 2, each layer consists of a finite number of neurons. Each neuron in each layer is connected to the subsequent layer and the previous layer which are generally called *links* or *synapses*. The workflow of MLP is explained as follows:

(a) Define the input as x_{ij}, where i is the index of the layers. Starting at $i = 0$ at the input layer, and j indexes the neurons in a layer. Thus, with $x_{0j},\ j = 1, 2, 3$ corresponds to q, χ_{1z}, χ_{2z} respectively.

(b) The kth neuron of the $(i+1)$th layer receives the value of x_{ij} from the ith layer multiplied by the weight w_{ijk}. These products are then summed over all links from the ith to the $(i+1)$th layer.

[5] In some references, the input layer is counted as the first hidden layer. Here we use the definition of *hidden layer* as a layer between the input and the output layer.

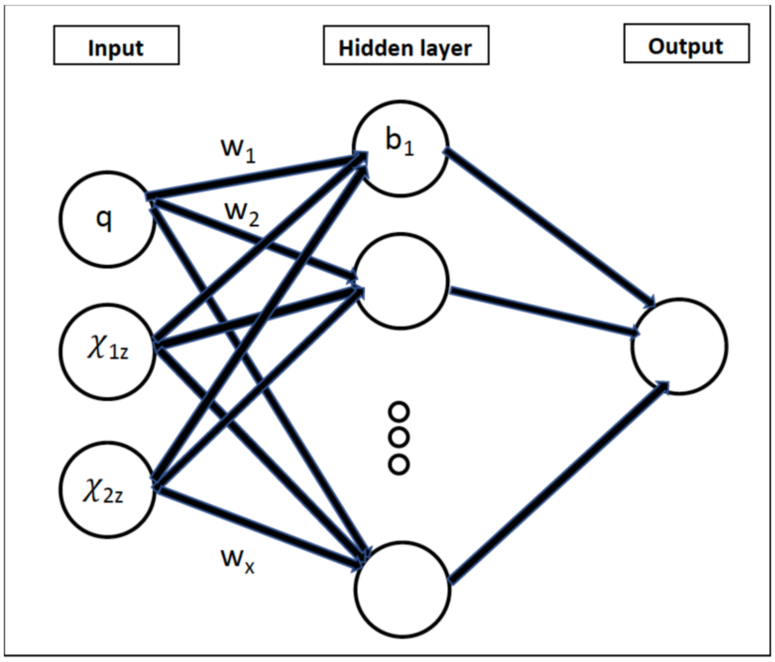

Figure 2. Diagram of ANN architecture used for three-dimensional interpolation in this study. The circles represent the neurons and we indicate weigths w_i along neuron connections and biases b_i. We employ two layers in the hidden layer part of the diagram. The same architecture is used for the seven-dimensional case, where the input contains seven neurons that depend on the seven parameters.

(c) A bias or shift b_{ik} is added to the above value and an activation function σ is applied to the final result. In this study, we use the Rectified Linear Unit (ReLU) [86] because it faster than other functions such as *sigmoid* and *tanh* and it is commonly used in other studies. ReLU is mathematically expressed by the following equation:

$$\sigma(z) = \max(z, 0), \tag{38}$$

and the MLP procedure is expressed by the following relation:

$$x_{i+1,k} = \sigma\left(\sum_j w_{ijk}x_{ij} + b_{ik}\right). \tag{39}$$

We vary the number of neurons in the first hidden layer between 2 to 2000 for the three-dimensional data sets and 2 to 5000 for the seven-dimensional data sets. We then set the number of neurons in the second hidden layer identical to the first hidden layer. The output of the networks is either A or ϕ at a single time. For each network and training data set, we compute mean squared error and the mean absolute error (see [87]) of $A(t)$ and $\phi(t)$, respectively.To train the networks, the training data is separated into several *batches*, where each batch contains the same number of data samples. Each batch is then passed through the networks (see equation (39)). When each data sample in the training set has had an opportunity to pass the networks a single time, this is known as an *epoch*. The number of epochs affects the learning of the networks, i.e. the higher the epoch, the better the learning. In this study, we set our batch size to five and train them through one thousand epochs.The networks compute the *loss functions* during each epoch. The *loss functions* measure the errors or inconsistency between the predicted value and the true data. In this study, we employ the *mean squared error* loss function for $A(t)$ and the *absolute error* for $\phi(t)$ respectively (see [87]).Training neural networks means that we minimize the loss functions so that our predicted values are as close as possible to the true values [88]. To minimize the loss functions, the networks adjust learnable parameters, i.e. the values of the weights and biases of the model. In most cases, the minimization cannot be solved analytically, but can be approached with *optimization algorithms*.During optimization, the network learns the values of weights and biases of the previous epoch and calculates its loss functions. Subsequently, it adjusts the values of weights and biases in the next epoch so that the loss functions become smaller. To minimize the loss functions, we compute the gradient values with respect to the learnable parameters. In this study, we employ *Adam* [89] as the optimization algorithm due its robustness.Following the above procedure, a model is then saved at the end of the run and evaluated through the test data. We then compute the accuracy and execution time of this process similar to other methods. We employ `Keras` [87] and `TensorFlow` [90] to perform this computation.

3. Results

In this section, we show results for accuracy and computational time for different regression methods. We apply methods to the three-dimensional and seven-dimensional data sets defined in section 2.2.

3.1. Three-dimensional case

We investigated the results for aligned spin waveforms with parameters q, χ_{1z}, and χ_{2z}. Training points were given on a regular grid. We placed the same number of points equally spaced to each other for each parameter (see section 2.1). Hence the total number of training points is proportional to the number of training points per dimension cubed. We then varied the number of training points in each dimension from five to eleven which corresponds to a total number of training points of 125 to 1331. We distributed 2500 test points randomly (see section 2.1). These test points are located inside the same domain covered by the training points. Hence, we do not test how well the methods perform for extrapolation.

We calculated relative errors (in percent) for the amplitude $A(t)$:

$$\varepsilon_{re} = \frac{\sum_i^N |A^i_{\text{pred}}(t) - A^i_{\text{true}}(t)|}{\sum_i^N |A^i_{\text{true}}(t)|} \times 100. \tag{40}$$

The phase error is an important diagnostic to measure the accuracy of GW waveform models. Therefore, we consider the absolute phase error (in radians)

$$\varepsilon_{ae} = \frac{1}{N}\sum_i^N |\phi^i_{\text{pred}}(t) - \phi^i_{\text{true}}(t)|. \tag{41}$$

ε_{re} and ε_{ae} are the relative error and the average of the absolute error, respectively, $A_{\text{pred}}(t)$ and $\phi_{\text{pred}}(t)$ are the predicted results of the amplitude and phase regression respectively, and $A_{\text{true}}(t)$ and $\phi_{\text{true}}(t)$ are their true values.

Subsequently, we investigated the computational time taken to evaluate each interpolation method. Here we define the *training time* as the time to compute the interpolant and the *execution time* being the time to compute the 2500 interpolation points following our test set. Furthermore, we define *total time* as the sum between the training time and the execution time, i.e. the entire process to perform interpolation for 2500 points. The comparison results in the early inspiral ($t = -3500M$) are shown in figure 3, whereas the results at $t = -50M$ are shown in figure 4. We now discuss the results shown the results for different regression methods.

(i) **Traditional interpolation and fitting methods & GPR**

We expect that the key quantities for two waveform models, `SEOBNRv3` and `IMRPhenomPv2` agree quite well in the early inspiral. The error in $A(t)$ and $\phi(t)$, decreases with more training points for both models. This result is expected as we populate our parameter space with more points located on a regular grid.

For both quantities, we find that errors for different methods are similar between waveform models. GPR errors show a dependence on the kernel choice. We first consider the amplitude errors. For `SEOBNRv3` the errors fall off in a similar way for either choice of kernel, whereas for `IMRPhenomPv2` the error is much higher for the SE kernel compared to the Matérn kernel. This is likely due to the higher level of noise in the `IMRPhenomPv2` data due to the inverse Fourier transformation.

The SE kernel assumes a higher degree of smoothness in the data than the Matérn kernel. Similarly, we find for either waveform model that the SE kernel shows a higher phase error than the Matérn kernel.

(ii) **Artificial neural networks**

We now discuss errors for ANNs as indicated by the filled circles in figure 3. Here we compare the results of the double layer MLP with various numbers of neurons. By design, the double layer MLP consists of one input layer, two hidden layers, and one output layer. We set the number of inputs as the dimensionality of the parameter space and only produce a single output. In the aligned spin case, our inputs are the parameters q, χ_{1z}, and χ_{2z} and output is either $A(t)$ or $\phi(t)$. For the hidden layers, we varied the number of neurons between 2 and 2000 in the first hidden layer, and set an equal number of neurons for the second hidden layer.

Thus, we obtained a set of errors as we modified the number of neurons in the *hidden layers* for a fixed number of training points N per dimension. In figure 3, we only show the results of the smallest errors for each training set. In this plot, different colors of the circles correspond to different numbers of neurons as indicated by the color bar. We note that the ANN with the smallest error may not be the fastest one.

Regarding the computational time, the training time obviously grows with the number of neurons per layer. However, we argue that there is no guarantee that many neurons yield smaller error than fewer neurons. In fact, too many neurons lead to overfitting and too few neurons lead to underfitting. We could reduce overfitting by activating the *Dropout* function in `Keras`, *Dropout* removes the result from a selected number of neurons randomly. However, we prefer to not include an additional stochastic element and do not include *Dropout* in this study.

Next, we compare execution times. Execution time is relatively similar between the GPR, RBF, TPI, and ANN methods. Other traditional methods such as linear, polynomial fit and GMVP, and linear interpolation are faster.

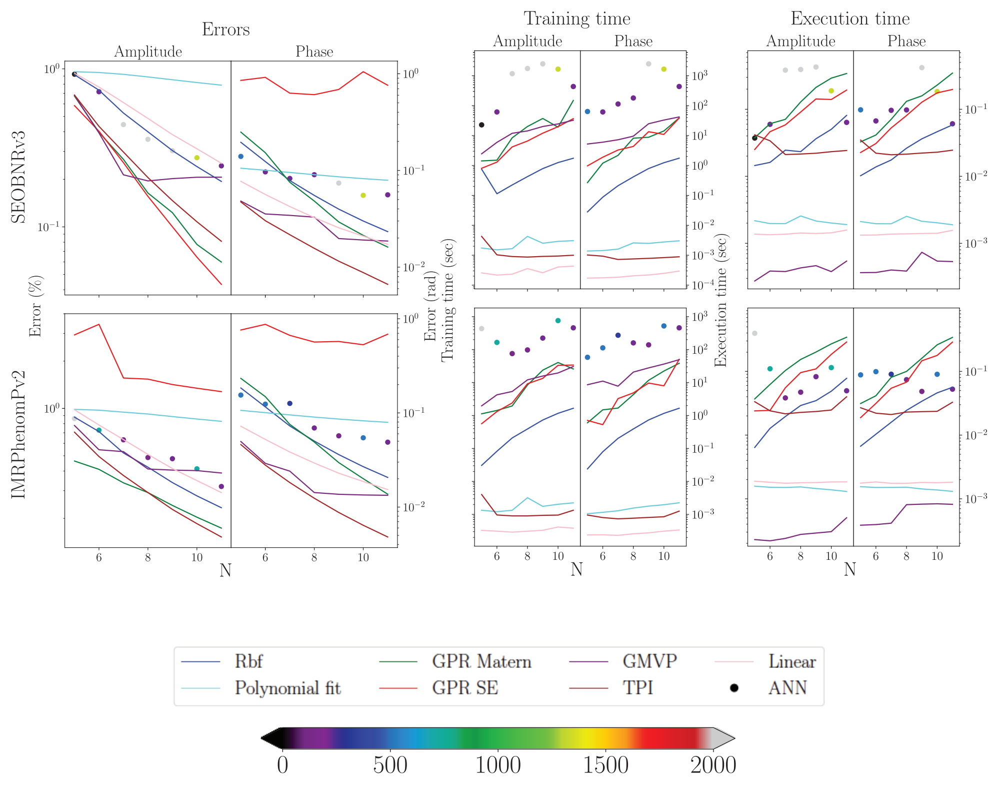

Figure 3. The three-dimensional interpolation results at $t = -3500M$. Top: SEOBNRv3, bottom: IMRPhenomPv2. The x-axes show the number of training points in each dimension, N, and the y−axes show the errors, training, and indicated execution time as on the labels of the panels. Left: errors of the amplitude and phase respectively, middle: training time in (seconds), and right: execution time (seconds). Different colors represent different interpolation methods as shown in the shared legend. The colored circles show ANN results, where different colors represent the number of neurons per layer in a double layer ANN as shown in the corresponding color bar.

To ensure a fair comparison between all methods, we explored the performance on the same machines (2x Intel Xeon E5-2698 v4) with 20 CPU cores, 256 Gigabytes of RAM, and 1x HDD (1TB, 6Gbps) of storage.

Due to the limited scope of our study, we only investigate results for the double layer ANN. This leaves tuning parameters and architectures to be explored in future studies. A possible way to reduce training and execution times is to use on GPUs instead of CPUs.

Finally, we discuss results for training times. The training time for RBF and GPR rise proportionally with the number of training points. In RBF, this is caused by the least-squares-fit computation that takes a longer time with more training points. For GPR, the training time goes as $\mathcal{O}(N^3)$ with N the number of training points as explained in section 2.2. Polynomial fit, TPI and linear interpolation do not depend strongly on the size of the training set and their training time is relatively fast.

For both models, ANN yields comparable errors and execution times as other interpolation methods, but generally with longer training time than other methods. Several methods have

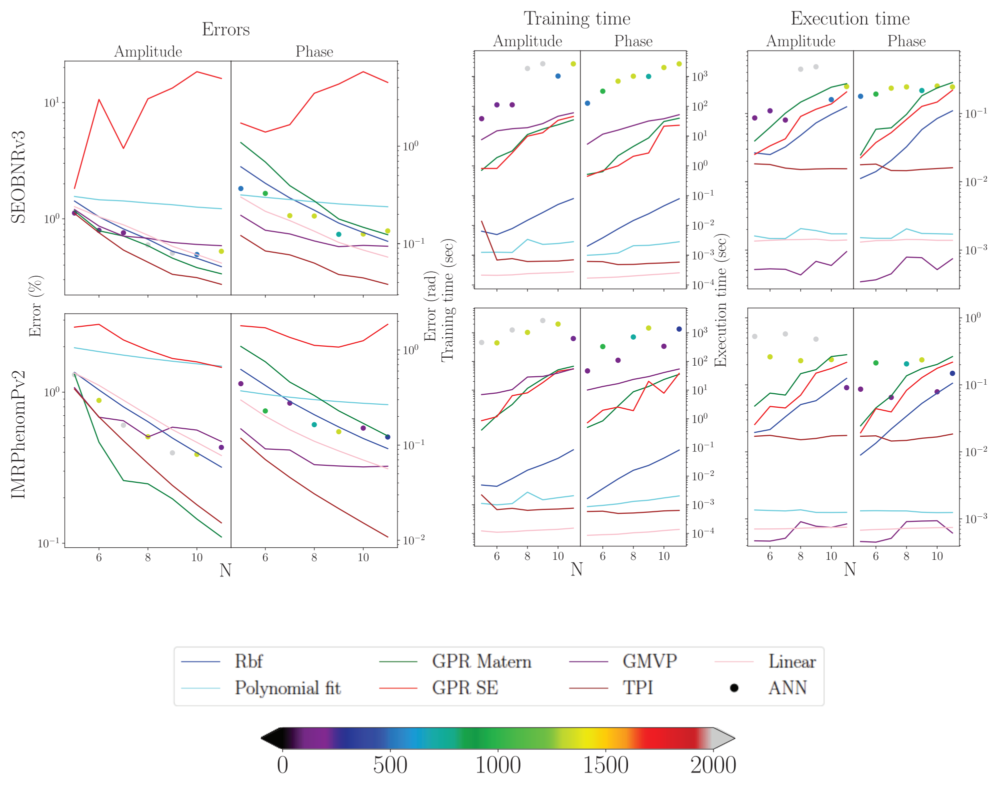

Figure 4. The three-dimensional interpolation results at $t = -50M$. Top: `SEOBNRv3`, bottom:`IMRPhenomPv2`. The x-axes show the number of training points in each dimension, N, and the $y-$axes show the errors, training, and indicated execution time as on the labels of the panels. Left: errors of the amplitude and phase respectively, middle: training time in (seconds), and right: execution time (seconds). Different colors represent different interpolation methods as shown in the shared legend. The colored circles show ANN results, where different colors represent the number of neurons per layer in a double layer ANN as shown in the corresponding color bar.

execution times that are independent of the size of the training set for a fixed order of approximations. This includes TPI, linear interpolation, polynomial fit, and ANNs.

Combining all the results at $t = -3500M$ and at $t = -50M$, we found that the errors are generally larger in noisy data. We also found that the methods with longer training time do not always yield a better result than the methods with less training time (see figure 4).

Using too many neurons in the hidden layers may cause problems such as overfitting. It occurs when the networks have too much capacity to process information such that the amount of information in the training set is not enough to train the networks [91]. Hence, the number of neurons must be set such that there are not too few or not too many. The selection however, depend on the architecture of the networks and the *hyperparameters*.

3.2. Seven-dimensional case

In seven dimensions, we distribute the training points randomly in each dimension. The main reason for this placement is to avoid the *curse of dimensionality* as explained in the previous

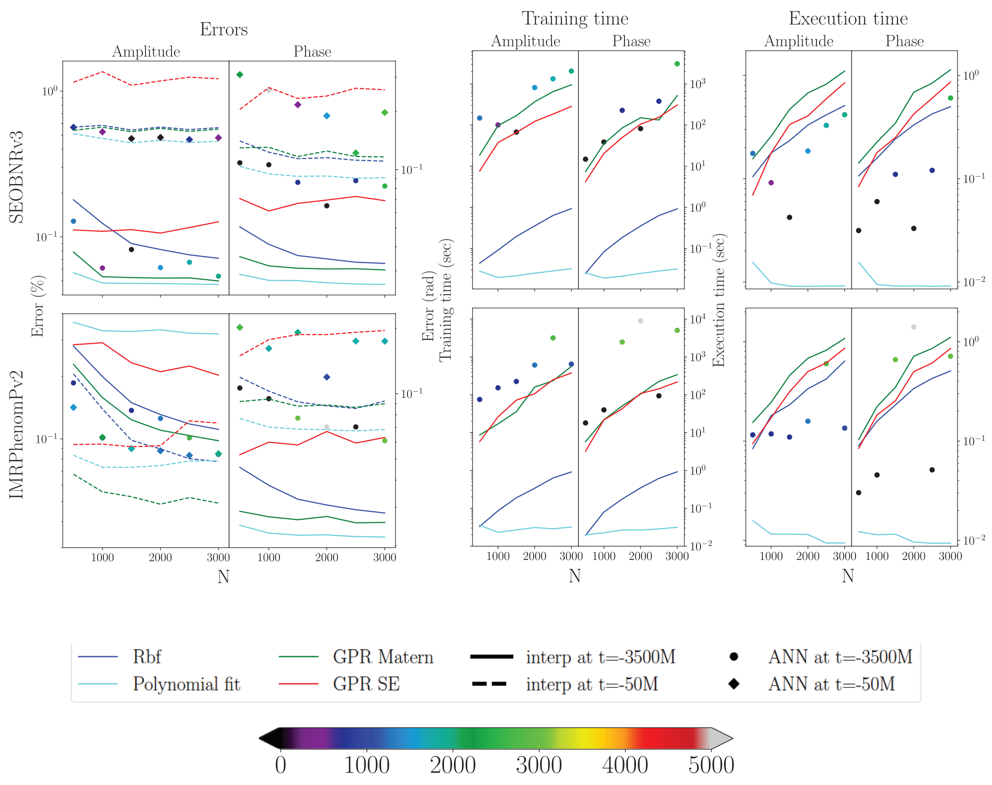

Figure 5. The seven-dimensional interpolation results. Top:`SEOBNRv3`, bottom: `IMRPhenomPv2`. The x-axes show the number of training points N and the y-axes shows the errors, training, and execution time as shown on the plot. Left: errors of the amplitude ($A(t)$) and phase ($\phi(t)$) respectively, middle: training time in unit seconds, and right: execution time in unit seconds. The solid lines show the results at $t = -3500M$ and the dashed lines for $t = -50M$. Different colors represent different interpolation methods as shown in the shared legend. The colored circles correspond to the results of ANN at $t = -3500M$ and the colored diamonds for $t = -50M$. Different colors represent a different number of neurons on a double layer ANN as shown in the corresponding color bar.

section. Similarly to the three-dimensional case, we investigate training sets of different sizes, from 500 to 3000 points. As discussed in section 2.1, the seven-dimensional case has a narrower range of mass ratio ($1 \leqslant q \leqslant 2$) than the three-dimensional one ($1 \leqslant q \leqslant 10$) and full-spin range.

We construct a single test set with 2500 points distributed randomly and located within the parameter ranges. Some of the test points may be outside the domain covered by the training points. This means that our results may contain a small extrapolation.

Since TPI and linear interpolation require regular grid training points, we do not include them in our analysis. For other methods, we employed the same settings (kernels, hyperparameters, degree) as in the three-dimensional case.

We built the architecture of ANN in a similar way as before. The results of the seven-dimensional case for different interpolation methods ($t = -3500M$ and $t = -50M$) are shown in figure 5.

Table 1. Summary of features of the methods used in this study. We present the advantages, the disadvantages and the scaling complexity for each method. For linear interpolation, TPI, RBF, and GPR the (training time) depends on the number of training points N (and polynomial degree k). Other methods have different complexity scalings that affect their training time.

Methods	Advantages	Disadvantages	Training time
Linear (RGI)	Standard `scipy`	Needs regular grid	$\mathcal{O}(N)$
TPI	Robust and high accuracy	Needs regular grid	$\mathcal{O}(N^k)$
GMVP	Irregular grid fast execution time	Complex	#basis function #error tolerance
Polynomial fit	Irregular grid simple and fast	Runge's phenomenon only univariate in `scipy`	$\mathcal{O}(N)$ and #polynomial degree
RBF	`scipy` irregular grid	high computational complexity	$\mathcal{O}(N^3)$
GPR	irregular grid can predict uncertainty	Depends on the choice of kernel and hyperparameters complex	$\mathcal{O}(N^3)$
ANN	Irregular grid flexible architecture choices	Complex	#neurons #hidden layers

We observed that errors of `SEOBNRv3` are not significantly different than the corresponding three-dimensional results. Furthermore, the errors of this model at $t = -50M$ are higher than at $t = -3500M$ in a similar way as in three dimensions.

Surprisingly, the relative amplitude errors for `IMRPhenomPv2` (top left plots) in the late inspiral are smaller than in the early inspiral in contrast to `SEOBNRv3`. The $A(t)$ quantity of `IMRPhenomPv2` is smoother at $t = -50M$ than at $t = -3500M$. We emphasize that both models, `SEOBNRv3` and `IMRPhenomPv2` have comparable amplitude values at $t = -50M$ and at $t = -3500M$.

In the early inspiral ($t = -3500M$), both waveforms agree well, similar to the three-dimensional case. Hence, the percent errors are not significantly different as shown in the same plot.

The phase errors were computed as absolute errors (see equation (41)). We find that the phase errors for `SEOBNRv3` and `IMRPhenomPv2` are comparable. Furthermore, the late inspiral errors are higher than the early inspiral as the data fluctuates more. In figure 5, we observe a similar behavior for the training time as in three dimensions, where higher training time was found for GPR, ANN, and RBF. This is caused by the same factors as explained in the three-dimensional case. For the execution time (right panel), we found that the more complex methods take longer time than the simpler methods. For RBF and GPR this is due to their dependence on the size of the training set. Interestingly, the execution time for ANNs is faster than GPR and RBF. This is because ANN picks the optimum weights and biases during the training and its execution time does not depend on the number of training points in the data.

We remind the reader that we set the parameter space of the seven-dimensions analysis narrower in mass ratio than the three-dimensions. Hence, the errors should not be compared directly to the three-dimensional case. For the same parameter ranges, the seven dimensional case yields errors up to 100 times larger for the $A(t)$ and 15 times larger for the $\phi(t)$. The order of accuracy does not significantly change, where the best accuracy in this range is obtained by polynomial interpolation.

Overall, we found that in some cases, a simple method such as polynomial fit yields lower errors and performs faster than the more complex methods.

4. Discussion and conclusion

Various approximation methods play important roles in building gravitational waveform models. Methods with high accuracy, low complexity, and fast computational time are needed for current and future applications. In this paper, we presented a comparative study of interpolation, fitting and regression methods applied to precessing and aligned BBH systems. Precessing BBH model depends on seven key intrinsic parameters $(q, \vec{\chi}_1, \vec{\chi}_2)$, whereas the aligned model depends on three parameters $(q, \chi_{1z}, \chi_{2z})$.

We generated the data sets in the time domain using two waveform models: `SEOBNRv3` (originally built in the time domain) and the inverse Fourier transform of `IMRPhenomPv2` (originally built in frequency domain). The full waveforms were transformed into a precession adapted frame where we extracted two quantities: amplitude $A(t)$ and phase $\phi(t)$ as explained in section 2.1 to perform a comparative study. For each key quantity, we picked two points in time, $t = -3500M$ in the inspiral for the smoother data set and $t = -50M$ near merger for the more irregular data. We employed this procedure on different numbers of training sets and used different approximation methods.

We split approximation methods into two categories: traditional methods and ML methods (see section 2.2). The traditional methods consist of linear interpolation, polynomial fits, radial basis function, GMVP, and TPI. Since linear interpolation and TPI package require a regular grid, we do not include them in the seven dimensional analysis. Furthermore, we investigated ML methods such as GPR and ANN. For GPR, we compared two kernel functions: the square exponential kernel and the Matérn kernel. In our results, we do not include GPR uncertainties in our comparison. We only consider the posterior mean of GPR SE and Matérn as shown by the red and green curves in figures 3–5. The SE kernel is C^∞ which makes GPRs constructed with it assume that the data has high smoothness. However, this assumption is not realistic as many physical processes contain noise components in the data. If the smoothness of the data is not well understood, we recommend that the Matérn kernel be used and its smoothness parameter ν should be included in hyperparameter optimization. For ANN, we focused on networks with two hidden layers and varied the number of their neurons.

We computed the relative errors for $A(t)$ and the absolute errors for $\phi(t)$. To validate the result, we generated 2500 test points distributed randomly within the same parameter space. The comparison results of different methods in accuracy, training time and execution time (in second) are presented in section 3.

We found that most methods perform better with more training data. Furthermore, we compared the performance of the same method in a set of smoother data and a set of more irregular data. In general, we found that approximation methods perform better in smoother data as expected. We recommend to use preprocessing methods to improve the smoothness of the data where possible which should increase the accuracy of regression results. This preparation is crucial as any methods perform well with smoother data sets. Different accuracies are attained by different methods in handling the irregularities in the data. We give a brief summary of different methods in table 1.

For lower dimensions, simpler methods such as linear interpolation and TPI provide good accuracy and speed. However, these methods need a regular grid and therefore are less useful for high dimensional data sets as explained above. For this situation, we found that polynomial fits are one of the simplest methods that offers a good combination between accuracy and speed. Furthermore, polynomial fits have been used widely and can be coded manually making it reliable and easy. The computational timing of polynomial fits depends on the number of parameters and the maximum polynomial degree. Another method that can perform approximation of scattered data sets is GMVP. GMVP which is based on polynomials can perform very well by setting error tolerance on its algorithm. For lower dimensionality, GMVP is computationally cheap. However, as the number of parameters rise, the computational time to compute the interpolant with the same error tolerance grows significantly higher. Therefore, we do not include this method in our analysis for the seven-dimensional case.

RBF and GPR are promising methods for scaterred data points. RBF has been integrated in a standard `scipy` package, making it easy for users. GPR computes the uncertainty of the predicted values. This feature is useful for future applications and cannot be found in other methods. Furthermore, GPR has been integrated in `sckit-learn` package [77]. Both RBF as GPR have the freedom to choose suitable kernel functions and hyperparameters. However, their speed depends on the number of training points cubed $\mathcal{O}(N^3)$. Hence, these methods become inefficient for larger data set.

A simple ANN can be used to perform regression for scattered data points. Similar to GPR, this method is more complex and depends on the choice of architecture and hyperparameters. We showed that the the three-dimensional result of ANN requires a longer training time with relatively comparable accuracy to other methods. We argue that such complexity is less needed for lower dimensional parameter and users should use a more simpler methods that provide good accuracy and speed. However, ANN is highly versatile to solve problems in higher dimensions and is promising to be explored further.

One might expect that methods with higher complexity perform better than methods with lower complexity. We find that this is not always the case. A more complicated method does not guarantee that the results are always better or faster. We find that simpler methods may yield smaller errors than more complex methods and perform faster in many cases. Hence, we suggest that one should critically evaluate the performance of approximation methods and understand the features of the method that are necessary for the data of interest. Simpler methods that perform better or at least equal to more complicated methods should be used as the first choice to avoid unecessary complexity.

Acknowledgments

The authors would like to thank to Lionel London, Scott Field, Stephen Green, Chad Galley, Christopher Moore, Zoheyr Doctor, Rory Smith, Ed Fauchon-Jones, and Lars Nieder for useful discussions. Computations were carried out on the Holodeck cluster of the Max Planck Independent Research Group 'Binary Merger Observations and Numerical Relativity'. This work was supported by the Max Planck Society's Independent Research Group Grant.

ORCID iDs

Yoshinta Setyawati https://orcid.org/0000-0003-3718-4491
Frank Ohme https://orcid.org/0000-0003-0493-5607

References

[1] Aasi J *et al* 2015 Advanced LIGO *Class. Quant. Grav.* **32** 074001
[2] Acernese F *et al* 2014 Advanced Virgo: a second-generation interferometric gravitational wave detector *Class. Quant. Grav.* **32** 024001
[3] Abbott B P *et al* 2019 Gwtc-1: a gravitational-wave transient catalog of compact binary mergers observed by ligo and virgo during the first and second observing runs *Phys. Rev.* X **9** 031040

[4] The LIGO Scientific Collaboration 2019 LIGO third observing time (O3) (https://dcc.ligo.org/public/0152/G1801056/004/G1801056-v4.pdf)
[5] Blanchet L 2014 Gravitational radiation from post-newtonian sources and inspiralling compact binaries *Living Rev. Relativ.* **17** 2
[6] Campanelli M, Lousto C O, Marronetti P and Zlochower Y 2006 Accurate evolutions of orbiting black-hole binaries without excision *Phys. Rev. Lett.* **96** 111101
[7] Pretorius F 2005 Evolution of binary black-hole spacetimes *Phys. Rev. Lett.* **95** 121101
[8] Baker J G, Centrella J, Choi D-I, Koppitz M and van Meter J 2006 Gravitational-wave extraction from an inspiraling configuration of merging black holes *Phys. Rev. Lett.* **96** 111102
[9] Damour T 2001 Coalescence of two spinning black holes: an effective one-body approach *Phys. Rev.* D **64** 124013
[10] Damour T, Jaranowski P and Schäfer G 2008 Effective one body approach to the dynamics of two spinning black holes with next-to-leading order spin-orbit coupling *Phys. Rev.* D **78** 024009
[11] Babak S, Taracchini A and Buonanno A 2017 Validating the effective-one-body model of spinning, precessing binary black holes against numerical relativity *Phys. Rev.* D **95** 024010
[12] Bohé A *et al* 2017 Improved effective-one-body model of spinning, nonprecessing binary black holes for the era of gravitational-wave astrophysics with advanced detectors *Phys. Rev.* D **95** 044028
[13] Santamaría L *et al* 2010 Matching post-Newtonian and numerical relativity waveforms: systematic errors and a new phenomenological model for nonprecessing black hole binaries *Phys. Rev.* D **82** 064016
[14] Khan S, Husa S, Hannam M, Ohme F, Pürrer M, Forteza X J and Bohé A 2016 Frequency-domain gravitational waves from nonprecessing black-hole binaries. II. A phenomenological model for the advanced detector era *Phys. Rev.* D **93** 044007
[15] Hannam M, Husa S, González J A, Sperhake U and Brügmann B 2008 Where post-Newtonian and numerical-relativity waveforms meet *Phys. Rev.* D **77** 044020
[16] Husa S, Khan S, Hannam M, Pürrer M, Ohme F, Forteza X J and Bohé A 2016 Frequency-domain gravitational waves from nonprecessing black-hole binaries. I. New numerical waveforms and anatomy of the signal *Phys. Rev.* D **93** 044006
[17] Schmidt P, Hannam M and Husa S 2012 Towards models of gravitational waveforms from generic binaries: a simple approximate mapping between precessing and nonprecessing inspiral signals *Phys. Rev.* D **86** 104063
[18] Hannam M, Schmidt P, Bohé A, Haegel L, Husa S, Ohme F, Pratten G and Pürrer M 2014 Simple model of complete precessing black-hole-binary gravitational waveforms *Phys. Rev. Lett.* **113** 151101
[19] Blackman J, Field S E, Galley C R, Szilágyi B, Scheel M A, Tiglio M and Hemberger D A 2015 Fast and accurate prediction of numerical relativity waveforms from binary black hole coalescences using surrogate models *Phys. Rev. Lett.* **115** 121102
[20] Blackman J *et al* 2017 Numerical relativity waveform surrogate model for generically precessing binary black hole mergers *Phys. Rev.* D **96** 024058
[21] Doctor Z, Farr B, Holz D E and Pürrer M 2017 Statistical gravitational waveform models: what to simulate next? *Phys. Rev.* D **96** 123011
[22] Varma V, Field S E, Scheel M A, Blackman J, Kidder L E and Pfeiffer H P 2019 Surrogate model of hybridized numerical relativity binary black hole waveforms *Phys. Rev.* D **99** 064045
[23] Pürrer M 2014 Frequency-domain reduced order models for gravitational waves from aligned-spin compact binaries *Class. Quantum Grav.* **31** 195010
[24] Pürrer M 2016 Frequency domain reduced order model of aligned-spin effective-one-body waveforms with generic mass ratios and spins *Phys. Rev.* D **93** 064041
[25] Abbott B P *et al* 2016 GW150914: first results from the search for binary black hole coalescence with advanced LIGO *Phys. Rev.* D **93** 122003
[26] Veitch J *et al* 2015 Parameter estimation for compact binaries with ground-based gravitational-wave observations using the lalinference software library *Phys. Rev.* D **91** 042003
[27] Veitch J, Mandel I, Aylott B, Farr B, Raymond V, Rodriguez C, van der Sluys M, Kalogera V and Vecchio A 2012 Estimating parameters of coalescing compact binaries with proposed advanced detector networks *Phys. Rev.* D **85** 104045
[28] Field S E, Galley C R, Hesthaven J S, Kaye J and Tiglio M 2014 Fast prediction and evaluation of gravitational waveforms using surrogate models *Phys. Rev.* X **4** 031006

[29] Blackman J *et al* 2017 A surrogate model of gravitational waveforms from numerical relativity simulations of precessing binary black hole mergers *Phys. Rev.* D **95** 104023
[30] Setyawati Y, Ohme F and Khan S 2019 Enhancing gravitational waveform models through dynamic calibration *Phys. Rev.* D **99** 024010
[31] London L and Fauchon-Jones E 2018 On modeling for Kerr black holes: Basis learning, QNM frequencies, and spherical-spheroidal mixing coefficients *Class. Quantum Grav.* **36** 235015
[32] Lackey B, Pürrer M, Taracchini A and Marsat S 2018 Surrogate model for an aligned-spin effective one body waveform model of binary neutron star inspirals using Gaussian process regression *Phys. Rev.* D **100** 024002
[33] Buonanno A, Pan Y, Pfeiffer H P, Scheel M A, Buchman L T and Kidder L E 2009 Effective-one-body waveforms calibrated to numerical relativity simulations: Coalescence of nonspinning, equal-mass black holes *Phys. Rev.* D **79** 124028
[34] Vinciguerra S, Veitch J and Mandel I 2017 Accelerating gravitational wave parameter estimation with multi-band template interpolation *Class. Quantum Grav.* **34** 115006
[35] Williams D, Heng I S, Gair J, Clark J A and Khamesra B 2019 A precessing numerical relativity waveform surrogate model for binary black holes: a Gaussian process regression approach (arXiv:1903.09204)
[36] Moore C J, Berry C P L, Chua A J K and Gair J R 2016 Improving gravitational-wave parameter estimation using gaussian process regression *Phys. Rev.* D **93** 064001
[37] Moore C J, Berry C P L, Chua A J K and Gair J R 2016 Fast methods for training gaussian processes on large datasets *R. Soc. Open sci.* **3** 160125
[38] Rebei A *et al* 2018 Fusing numerical relativity and deep learning to detect higher-order multipole waveforms from eccentric binary black hole mergers *Phys. Rev.* D **100** 044025
[39] Chua A J K, Galley C R and Vallisneri M 2019 Reduced-order modeling with artificial neurons for gravitational-wave inference *Phys. Rev. Lett.* **122** 211101
[40] The LIGO Scientific Collaboration 2018 LIGO instrument white paper: LIGO A+, Cosmic Explorer and Voyager https://dcc.ligo.org/LIGO-T1500290/public
[41] Aso Y, Michimura Y, Somiya K, Ando M, Miyakawa O, Sekiguchi T, Tatsumi D and Yamamoto H 2013 Interferometer design of the KAGRA gravitational wave detector *Phys. Rev.* D **88** 043007
[42] Bender P *et al* 2020 Laser Interferometer Space Antenna: a cornerstone mission for the observation of gravitational waves *Technical Report ESA-SCI(2000) 11*
[43] Punturo M *et al* 2010 The third generation of gravitational wave observatories and their science reach *Class. Quantum Grav.* **27** 084007
[44] Abbott B P *et al* 2017 Exploring the sensitivity of next generation gravitational wave detectors *Class. Quantum Grav.* **34** 04400
[45] Pan Y, Buonanno A, Taracchini A, Kidder L E, Mroué A H, Pfeiffer H P, Scheel M A and Szilágyi B 2014 Inspiral-merger-ringdown waveforms of spinning, precessing black-hole binaries in the effective-one-body formalism *Phys. Rev.* D **89** 084006
[46] Taracchini A *et al* 2014 Effective-one-body model for black-hole binaries with generic mass ratios and spins *Phys. Rev.* D **89** 061502
[47] Golub G H and Van Loan C F 1996 *Matrix Computations* 3rd edn (Baltimore, MD: Johns Hopkins)
[48] Demmel J W 1997 *Applied Numerical Linear Algebra* (Philadelphia, PA: SIAM)
[49] Boyle M 2013 Angular velocity of gravitational radiation from precessing binaries and the corotating frame *Phys. Rev.* D **87** 104006
[50] Boyle M, Owen R and Pfeiffer H P 2011 A geometric approach to the precession of compact binaries *Phys. Rev.* D **84** 124011
[51] Wigner E P 1959 *Group Theory and its Application to the Quantum Mechanics of Atomic Spectra* (New York: Academic)
[52] Garrido J M 2015 *Introduction to Computational Models with Python* vol 1, 1st edn (London: Chapman and Hall)
[53] Buchner J 2015 Regular grid interpolator GitHub (https://pypi.org/project/regulargrid)
[54] SCIPY 2015 Scipy RGI (SciPy 0.16.1) (https://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.interpolate.RegularGridInterpolator.html)
[55] Virtanen P *et al* 2019 SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python (arXiv:1907.10121)
[56] Birkes D and Dodge Y 1993 *Alternative Methods of Regression* 1st edn (New York: Wiley)

[57] Hastie T, Tibshirani R and Wainwright M 2015 *Statistical Learning with Sparsity and Generalizations* 1st edn (Boca Raton, FL: CRC Press)
[58] Wakefield J 2013 *Bayesian and Frequentist Regression Methods* (*Springer Series in Statistics*) (New York: Springer)
[59] de Boor C 2001 *A Practical Guide to Splines* (Berlin: Springer)
[60] Boyd J P 2000 *Chebyshev and Fourier Spectral Methods* (New York: Dover)
[61] Canuto C, Hussaini M Y, Quarteroni A and Zang T A 2006 *Spectral Methods, Fundamentals in Single Domains* (Berlin: Springer)
[62] Quarteroni A, Sacco R and Saleri F 2000 *Numerical Mathematics* (Berlin: Springer)
[63] Lackey B D, Bernuzzi S, Galley C R, Meidam J and Van Den Broeck C 2017 Effective-one-body waveforms for binary neutron stars using surrogate models *Phys. Rev.* D **95** 104036
[64] Behnel S, Bradshaw R, Citro C, Dalcin L, Seljebotn D S and Smith K 2011 Cython: The best of both worlds *Computi. Sci. Eng.* **13** 31–9
[65] Pürrer M and Blackman J 2018 Tensor product interpolation package for python (TPI). GitHub https://github.com/mpuerrer/TPI
[66] Phillips G M 2003 *Interpolation and Approximation by Polynomials* 1st edn (New York: Springer)
[67] Quarteroni A, Sacco R and Saleri F 2007 *Numerical Mathematics* vol 1–10, 2nd edn (Berlin: Springer)
[68] Press W, Flannery B, Teukolsky S and Vetterling W 1992 *Numerical Recipes in C: the Art of Scientific Computing* volume 1, 2 (Cambridge: Cambridge University Press)
[69] Chad Galley 2018 *rompy Bitbucket* (https://bitbucket.org/chadgalley/rompy/src/master/)
[70] Field S E, Galley C R, Herrmann F, Hesthaven J S, Ochsner E and Tiglio M 2011 Reduced basis catalogs for gravitational wave templates *Phys. Rev. Lett.* **106** 221102
[71] Buhmann M D 2004 *Radial Basis Function: Theory and Implementaions. The Pitt Building, Trumpington Street* (Cambridge: Cambridge University Press)
[72] Roussos G and Baxter B 2005 Rapid evaluation of radial basis functions *J. Comput. Appl. Math.* **180** 51–70
[73] Scipy 2018 Scipy RBF (https://github.com/scipy/scipy/blob/v1.4.1/scipy/interpolate/rbf.py)
[74] Müller A and Guido S 2016 *Introduction to Machine Learning with Python* (Sebastopol, CA: O'Reilly Media)
[75] Goodfellow I, Bengio Y and Courville A 2017 *Deep Learning* (Cambridge, MA: MIT Press)
[76] Rasmussen C and Williams C 2006 *Gaussian Processes for Machine Learning* (Cambridge, MA: MIT Press)
[77] Pedregosa F *et al* 2011 Scikit-learn: machine learning in python *J. Mach. Learn. Res.* **12** 2825–30
[78] Rasmussen C and Williams C 1996 Gaussian processes for regression *Advances in Neural Information Processing Systems* pp 514–20
[79] Jackson J E 2003 *A User's Guide to Principal Components* vol 1, 3rd edn (New York: Wiley)
[80] Abramowitz M and Stegun I 1965 *Handbook of Mathematical Functions* (New York: Dover)
[81] Egmont-Petersen M, de Ridder D and Handels H 2002 Image processing with neural networks: a review *Pattern Recognit.* **35** 2279–301
[82] Zhang G, Patuwo B E and Hu M Y 1998 Forecasting with artificial neural networks: the state of the art *Int. J. Forecast.* **14** 35–62
[83] Abiodun O, Jantan A, Omolara A, Dada K, Mohamed N and Arshad H 2018 State-of-the-art in artificial neural network applications: a survey *Int. J. Forecast.* **4** e00938
[84] Hornik K, Stinchcombe M and White H 1989 Multilayer feedforward networks are universal approximators *Neural Netw.* **2** 359–66
[85] Sonoda S and Murata N 2017 Neural network with unbounded activation functions is universal approximator *Appl. Comput. Harmon. Anal.* **43** 233–68
[86] Glorot X, Bordes A and Bengio Y 2011 Deep sparse rectifier neural networks *J. Mach. Learn. Res.* **15** 315–23 http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.
[87] Chollet F 2015 keras (version 2.2.0) (https://github.com/fchollet/keras)
[88] da Silva I N, Spatti D H, Flauzino R A, Liboni L H B and dos Reis Alves S F 2017 *Artificial Neural Networks: a Practical Course* 1st edn (Berlin: Springer)
[89] Kingma D and Ba J 2017 Adam: a methods for stochastic optimization (arXiv:1412.6980)
[90] Abadi M *et al* 2016 Tensorflow: A system for large-scale machine learning *12th Symp. on Operating Systems Design and Implementation* pp 265–283
[91] Heaton J 2008 *Introduction to Neural Networks with Java* 2nd edn (Chesterfield: Heaton Research, Inc.)