
Heron: Gaussian Process Surrogate Modelling

Release 0.2.6.dev22+g9cb6bac

Daniel Williams

Sep 27, 2019

1	Heron	3
1.1	Features	3
2	Installation	5
2.1	Stable release	5
2.2	Source installation	5
3	Usage	7
4	Tutorials	9
4.1	Introduction to Gaussian processes	9
4.2	Covariance Functions	12
5	Pre-supplied Models	17
5.1	George-based models	17
5.2	Implementing new models	18
6	Indices and tables	21
	Index	23

Heron is a Python package for producing surrogate models for computationally intensive functions, such as numerical relativity waveforms.

This version of heron is implemented slightly differently to older versions, and should allow for a greater degree of flexibility for using different GP libraries for the modelling, and to fit into analysis pipelines better than the earlier development versions.

Warning: This documentation is still being written, and you may find a few places either where documentation is missing, or where the formatting isn't good. If you spot something which has clearly been missed in the documentation please open an issue on the git repository.

The `heron` package is a python library for using Gaussian Process Regression (GPR) to emulate functions which are expensive to

It was originally built for producing a surrogate model for numerical relativity waveforms from binary black hole coalescences, but the code should be sufficiently general to allow other surrogate models to be built.

In order to handle very large models, `heron` can use the `george` python package to generate the underlying Gaussian Process, which can handle very large models thanks to its use of a hierarchical matrix inverter.

1.1 Features

- Single-valued function surrogate production from multivalued inputs
- Handling very large datasets.

2.1 Stable release

To install the latest stable release of Heron you can use `pip`:

```
$ pip install heron
```

This should always install the latest stable release of heron, though it may still be sensible to run this command inside a virtual environment.

If you don't have `pip` installed, this [Python installation guide](#) can guide you through the process.

2.2 Source installation

Alternatively, if you want to make your own changes to the code, or test code between releases you can install from source. The Heron source can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/transientlunatic/heron
```

Or download the [tarball](#):

```
$ curl -OL https://github.com/transientlunatic/heron/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ pip install .
```


CHAPTER 3

Usage

In order to run one of Heron's built-in models you'll need to import that model as well as Heron itself, for example:

```
import heron
from heron.models.georgebased import HeronHodlr

import numpy as np
```

We also imported numpy for convenience.

This will load latest version of the NR-trained 7-dimensional Heron model. We now need to set the generator up to produce waveforms. Currently we need to tell the model the total mass of the system, but this

```
generator = HeronHodlr()
```

Two different types of waveform can be requested from the model: the mean waveform (and its variance), or individual waveform samples.

In order to produce a mean waveform we need to provide the model with the intrinsic properties of the system, that is, the mass ratio, and the spin parameters. If any parameters are omitted from the dictionary they're set to zero.

```
waveform = generator.mean(times=np.linspace(-0.02, 0.02), p={"mass ratio": 0.3})
```

The output of the mean method is two waveforms (one each for the plus and cross polarisations).

The `data` attribute of each waveform contains the mean strain data, while the `variance` attribute contains the variance on this mean waveform.

Alternatively, Heron can return individual function draws. These may not look especially similar to what you would expect out of a normal waveform model, but used collectively they can allow the calculation of various statistics.

```
samples = generator.distribution(samples=100, times=np.linspace(-0.02, 0.02), p={
    ↪ "mass ratio": 3})
```

This produces 100 waveform samples drawn from the model, at the same model configuration as the previous mean waveform.

4.1 Introduction to Gaussian processes

Consider a regression problem with a set of data

$$D = (\vec{x}_i, y_i), i \in 1, \dots, n$$

which is composed of n pairs of inputs, \vec{x}_i ,

which are vectors which describe the location of the datum in parameter space, which are the inputs for the problem, and y_i , the outputs. The outputs may be noisy; in this work I will only consider situations where the noise is additive and Gaussian, so

$$y_i(\vec{x}_i) = f(\vec{x}_i) + \epsilon_i, \quad \text{for } \epsilon_i \sim \mathcal{N}(0, \sigma^2) \quad (4.1)$$

where σ is the standard deviation of the noise, and f is the (latent) generating function of the data.

This regression problem can be addressed using *Gaussian processes*:

A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution [cite:gpr.book.rw](#).

Where it is more conventional to consider a prior over a set of, for example, real values, such as a normal distribution, the Gaussian process forms a prior over the functions, f from equation [ref:eq:gp:additive-noise](#), which might form the regression fit to any observed data. This assumes that the values of the function f behave as

$$p(\vec{f} | \vec{x}_1, \vec{x}_2, \dots, \vec{x}_n) = \mathcal{N}(0, K) \quad (4.2)$$

where K is the covariance matrix of \vec{x}_1 and \vec{x}_2 , which can be calculated with reference to some *covariance function*, k , such that $K_{ij} = k(\vec{x}_i, \vec{x}_j)$. Note that I have assumed that the [abbr:gp](#) is a *zero-mean* process; this assumption is frequent within the literature. While this prior is initially untrained it still contains information about our preconceptions of the data through the form of the covariance function. For example, whether or not we expect the fit to be smooth, or periodic. Covariance functions will be discussed in greater detail in section [ref:sec:gp:covariance](#).

By providing training data we can use Bayes theorem to update the Gaussian process, in the same way that the posterior distribution is updated by the addition of new data in a standard Bayesian context, and a posterior on the set of all possible functions to fit the data is produced. Thus, for a vector of test values of the generating function \vec{f}_* , the joint posterior $p(\vec{f}, \vec{f}_* | \vec{y})$, given the observed outputs \vec{y} can be found by updating the abbr:gp prior on the training and test function values $p(\vec{f}, \vec{f}_*)$ with the likelihood $p(\vec{y} | \vec{f})$:

$$p(\vec{f}, \vec{f}_* | \vec{y}) = \frac{p(\vec{f}, \vec{f}_*)p(\vec{y} | \vec{f})}{p(\vec{y})}. \quad (4.3)$$

Finally the (latent) training-set function values, \vec{f} can be marginalised out:

$$p(\vec{f}_* | \vec{y}) = \int p(\vec{f}, \vec{f}_* | \vec{y}) \vec{f} = \frac{1}{p(\vec{y})} \int p(\vec{y} | \vec{f}) p(\vec{f}, \vec{f}_*) \vec{f} \quad (4.4)$$

We can take the mean of this posterior in the place of the “best fit line” which other techniques produce, and then use the variance to produce an estimate of the uncertainty of the prediction.

Both the prior $p(\vec{f}, \vec{f}_*)$ and the likelihood $p(\vec{y} | \vec{f})$ are Gaussian:

$$p(\vec{f}, \vec{f}_*) = \mathcal{N}(\vec{0}, K^+), \quad \text{and} \quad p(\vec{y} | \vec{f}) = \mathcal{N}(\vec{f}, \sigma^2 I) \quad (4.5)$$

with

$$K^+ = \begin{bmatrix} K_{\vec{f}, \vec{f}} & K_{\vec{f}, \vec{f}_*} \\ K_{\vec{f}_*, \vec{f}} & K_{\vec{f}_*, \vec{f}_*} \end{bmatrix}, \quad (4.6)$$

and I the identity matrix.

This leaves the form of the marginalised posterior being analytical:

$$p(\vec{f}_* | \vec{y}) = \mathcal{N} \left(K_{\vec{f}_*, \vec{f}} (K_{\vec{f}, \vec{f}} + \sigma^2 I)^{-1} \vec{y}, K_{\vec{f}_*, \vec{f}_*} - K_{\vec{f}, \vec{f}_*} (K_{\vec{f}, \vec{f}} + \sigma^2 I)^{-1} K_{\vec{f}, \vec{f}_*} \right). \quad (4.7)$$

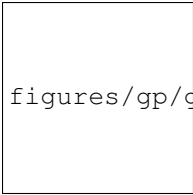
Figures ref:fig:gp:training-data to ref:fig:gp:posterior-best show visually how a one-dimensional regressor can be created using an abbr:gp method, starting from a abbr:gp prior and (noisy) data.

The mean and variance of this posterior distribution can be used to form a regressor for the data, D , with the mean taking the role of a “line-of-best-fit” in conventional regression techniques, while the variance describes the goodness of that fit.

A graphical model of a abbr:gp is shown in figure ref:fig:gp:chain-diagram which illustrates an important property of the abpl:gp model: the addition (or removal) of any input point to the abbr:gp does not change the distribution of the other variables. This property allows outputs to be generated at arbitrary locations throughout the parameter space.

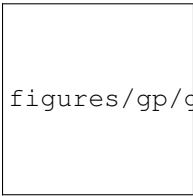
Gaussian processes trained with N training data require the ability to both store and invert an $N \times N$ matrix of covariances between observations; this can be a considerable computational challenge.

Gaussian processes can be extended from the case of a single-dimensional input predicting a single-dimensional output to the ability to predict a multi-dimensional output from a multi-dimensional input cite:2011arXiv1106.6251A,Alvarez2011a,Bonilla2007.




figures/gp/gp-training-data.pdf

Fig. 1: [Step 1] An example of raw training data (containing additive Gaussian noise) which is suitable for training a Gaussian process. In this example the input data (x -axis) are 1-dimensional, although GPs are also capable of handling multi-dimensional data. Here the generating function is plotted as a grey line.



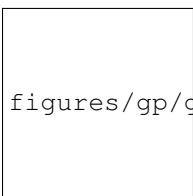
figures/gp/gp-example-prior-draws.pdf

Fig. 2: [Step 2] We choose a covariance function for the Gaussian process, in this case an exponential-quadratic covariance function. The Gaussian process containing no data and this covariance matrix forms our prior probability distribution. Here 50 draws from the prior distribution are plotted.



figures/gp/gp-example-posterior-draws.pdf

Fig. 3: [Step 3] The trained Gaussian process can be sampled multiple times to produce multiple different potential fitting functions. Here 50 draws from the Gaussian process posterior are displayed.



figures/gp/gp-posterior-meancovar.pdf

Fig. 4: [Step 4] We can then take the mean and the covariance of the Gaussian process, and produce a single “best-fit” with confidence intervals. Again, the original generating function for the data is shown as a grey line.

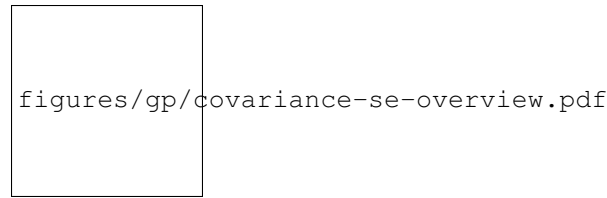


Fig. 5: The **squared exponential** covariance function (defined in equation ref:eq:gp:kernels:se). The panel on the left depicts the value of the kernel as a function of $r = (|\vec{x} - \vec{x}'|)$, at a number of different length scales ($l = 0.25, 0.5, 1.0$) while the panel on the right contains draws from Gaussian processes with se covariance with the same length scales as the left panel.

4.2 Covariance Functions

The covariance function defines the similarity of a pair of data points, according to some relationship with suitable properties. The similarity of input data is assumed to be related to the similarity of the output, and therefore the more similar two inputs are the more likely their outputs are to be similar.

As such, the form of the covariance function represents prior knowledge about the data, and can encode understanding of effects such as periodicity within the data.

A stationary covariance function is a function $f(\vec{x} - \vec{x}')$, and which is thus invariant to translations in the input space.

If a covariance function is a function of the form $f(|\vec{x} - \vec{x}'|)$ then it is isotropic, and invariant under all rigid motions.

A covariance function which is both stationary and isotropic has the property that it can be expressed as a function of a single variable, $r = |\vec{x} - \vec{x}'|$ is known as a abbr:rbf. Functions of the form $k : (\vec{x}, \vec{x}') \rightarrow \mathbb{C}$, for two vectors $\vec{x}, \vec{x}' \in \mathcal{X}$ are often known as *kernels*, and I will frequently refer interchangeably to covariance functions and kernels where the covariance function has this form.

For a set of points $\vec{x}_i | i = 1, \dots, n$ a kernel, k can be used to construct the gram matrix, $K_{i,j} = k(x_i, x_j)$. If the kernel is also a covariance function then K is known as a *covariance matrix*.

For a kernel to be a valid covariance function for a abbr:gp it must produce a positive semidefinite covariance matrix K . Such a matrix, $K \in \mathbb{R}^{n \times n}$ must satisfy $\vec{x} K \vec{x} \geq 0$ for all $\vec{x} \in \mathbb{R}^n$.

4.2.1 Example covariance functions

One of the most frequently encountered covariance functions in the literature is the abbr:se covariance functions cite:gpr.book.rw. Perhaps as a result of its near-ubiquity this kernel is known under a number of similar, but confusing names (which are often inaccurate). These include the *exponential quadratic*, *quadratic exponential*, *squared exponential*, and even *Gaussian* covariance function.

The reason for this is its form, which closely resembles that of the Gaussian function:

$$k_{SE}(r) = \exp\left(-\frac{r^2}{2l^2}\right) \quad (4.8)$$

for r the Euclidean distance of a datum from the centre of the parameter space, and l is a scale factor associated with the axis along which the data are defined.

The squared exponential function imposes strong smoothness constraints on the model, as it is infinitely differentiable.

The scale factor, l in ref:eq:gp:kernels:se, also known as its *scale-length* defines the size of the effect within the process. This characteristic length-scale can be understood cite:adler1976,gpr.book.rw in terms of the number of times the abbr:gp should cross some given level (for example, zero). Indeed, for a abbr:gp with a covariance function

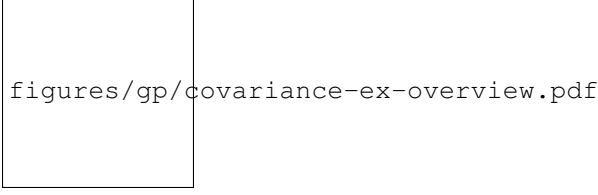


Fig. 6: The **exponential** covariance function (defined in equation ref:eq:gp:kernels:exp). The panel on the left depicts the value of the kernel as a function of $r = (|\vec{x} - \vec{x}'|)$, at a number of different length scales ($l = 0.25, 0.5, 1.0$) while the panels on the right contain draws from Gaussian processes with exponential covariance with the same length scales as the left panel.

k which has well-defined first and second derivatives the expected number of times, N_u the process will cross a value u is

$$\mathbb{E}(N) = \frac{1}{2\pi} \sqrt{-\frac{k''(0)}{k(0)}} \exp\left(-\frac{u}{2k(0)}\right) \quad (4.9)$$

A zero-mean abbr:gp which has an abbr:se covariance structure will then cross zero $1/(2\pi l)$ times on average.

Examples of the squared exponential covariance function, and of draws from a Gaussian process prior which uses this covariance function are plotted in figure ref:fig:gp:covariance:overviews:se for a variety of different scale lengths.

For data which is not generated by a smooth function a suitable covariance function may be the exponential covariance function, k_{EX} , which is defined

$$k_{\text{EX}} = \exp\left(-\frac{r}{l}\right), \quad (4.10)$$

where r is the pairwise distance between data and l is a length scale, as in equation ref:eq:gp:kernels:se.

Examples of the exponential covariance function, and of draws from a Gaussian process prior which uses this covariance function are plotted in figure ref:fig:gp:covariance:overviews:ex for a variety of different scale lengths.

For data generated by functions which are smooth, but not necessarily infinitely differentiable we may turn to the Matérn family of covariance functions, which take the form

$$k_{\text{Mat}}(r) = \frac{1}{2^{\nu-1}\Gamma_{\nu}} \left(\frac{\sqrt{2\nu}}{l}\right)^{\nu} K_{\nu}\left(\frac{\sqrt{2\nu}}{l}r\right), \quad (4.11)$$

for K_{ν} the modified Bessel function of the second kind, and Γ the gamma function. As with the previous two covariance functions l is a scale length parameter, and r the distance between two data. A abbr:gp which has a Matérn covariance function will be $(\lceil \nu \rceil - 1)$ -times differentiable.

While determining an appropriate value of ν during the training of the abbr:gp is possible, it is common to select a value *a priori* for this quantity. $\nu = 3/2$ and $\nu = 5/2$ are common choices as K_{ν} can be determined simply, and the covariance functions are analytic.

The case with $\nu = 3/2$, commonly referred to as a Matérn-3/2 kernel then becomes

$$k_{\text{M32}}(r) = \left(1 + \frac{\sqrt{3}d}{l}\right) \exp\left(-\frac{\sqrt{3}d}{l}\right). \quad (4.12)$$

Examples of this covariance function, and example draws from a abbr:gp using it as a covariance function are plotted in figure ref:fig:gp:kernels:m32.

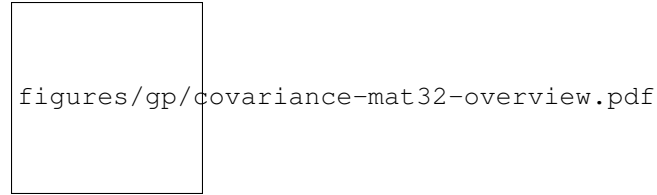


Fig. 7: The **Matérn-3/2** covariance function (defined in equation `ref:eq:gp:kernels:mat`, with $\nu = 3/2$). The panel on the left depicts the value of the kernel as a function of $r = (|\vec{x} - \vec{x}'|)$, at a number of different length scales ($l = 0.25, 0.5, 1.0$) while the panels on the right contain draws from Gaussian processes using a Matérn-3/2 covariance with the same length scales as the left panel.

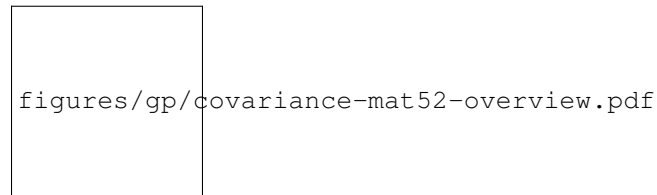


Fig. 8: The **Matérn-5/2** covariance function (defined in equation `ref:eq:gp:kernels:mat`, with $\nu = 5/2$). The panel on the left depicts the value of the kernel as a function of $r = (|\vec{x} - \vec{x}'|)$, at a number of different length scales ($l = 0.25, 0.5, 1.0$) while the panels on the right contain draws from Gaussian processes using Matérn-5/2 covariance functions with the same length scales as the left panel.

Similarly, the Matérn-5/2 is the case where $\nu = 5/2$, taking the form

$$k_{M52}(r) = \left(1 + \frac{\sqrt{5}d}{l} + \frac{5d^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}d}{l}\right). \quad (4.13)$$

Again, examples of this covariance function, and example draws from a `abbr:gp` using it as a covariance function are plotted in figure `ref:fig:gp:kernels:m52`.

Data may also be generated from functions with variation on multiple scales. One approach to modelling such data is to use a `abbr:gp` with **rational quadratic** covariance. This covariance function represents a scale mixture of `abbr:rbf` covariance functions, each with a different characteristic length scale. The rational quadratic covariance function is defined as

$$k_{RQ}(r) = \left(1 + \frac{r^2}{2\alpha l^2}\right)^{-\alpha}, \quad (4.14)$$

where α is a parameter which controls the weighting of small-scale compared to large-scale variations, and l and r are the overall length scale of the covariance and the distance between two data respectively. Examples of this function, at a variety of different length scales and α values, and draws from `abpl:gp` which use these functions are plotted in figure `ref:fig:gp:kernels:rq`.

This summary of potential covariance functions for use with a `abbr:gp` is far from complete (see `cite:gpr.book.rw` for a more detailed list). However, these four can be used or combined to produce highly flexible regression models, as they can be added and multiplied as normal functions.

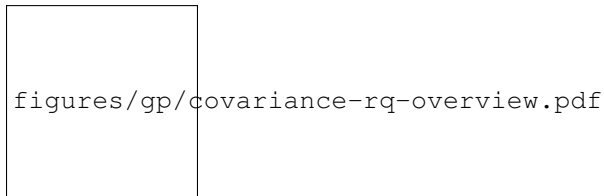


Fig. 9: The **rational quadratic** covariance function (defined in equation 4.14). The panel on the left depicts the value of the kernel as a function of $r = |\vec{x} - \vec{x}'|$, at a number of different length scales ($l = 0.25, 0.5, 1.0$) while the panel on the right contains draws from Gaussian processes with rational quadratic covariance with the same length scales as the left panel.

Pre-supplied Models

This package contains a number of different pre-baked waveform models, and the data which is required to reproduce them. It's also fairly easy to use the existing framework to implement a new model, using the same training data as pre-supplied models, or using new training data.

5.1 George-based models

A number of models implemented in *Heron* make use of the *George* Gaussian process library which implements a number of simplifications to make the inversion of the covariance matrix required for GPR predictions more tractable.

The main model produced this way is *HeronHODLR*, which implements a fully-spinning BBH waveform model which is trained on waveform data from the Georgia Tech waveform catalogue.

All of the george-based models are contained in the *heron.models.georgebased* module.

5.1.1 HeronHODLR: A spinning, NR-trained waveform model

The *HeronHODLR* model implements a surrogate model for gravitational waveforms from binary black hole events with arbitrary spin parameters between a mass ratio of 1 and 8.

class `heron.models.georgebased.HeronHodlr`
Produce a BBH waveform generator using the `Hodlr` method.

Methods

<code>bilby(self, time, mass_1, mass_2, ...)</code>	Return a waveform from the GPR in a format expected by the Bilby ecosystem
<code>build(self[, mean, white_noise, tol])</code>	Construct the GP object
<code>distribution(self, p, times[, samples, ...])</code>	Return the mean waveform and the variance at a given location in the BBH parameter space.

Continued on next page

Table 1 – continued from previous page

<code>eval(self)</code>	Prepare the model to be evaluated.
<code>log_evidence(self, k, n)</code>	Evaluate the log-evidence of the model at a hyperparameter location k .
<code>mean(self, p, times)</code>	Return the mean waveform at a given location in the BBH parameter space.
<code>train(self)</code>	Prepare the model to be trained.

5.1.2 Heron2DHodlrIMR

This model is a 2D prototype waveform model trained on phenomenological sample waveforms. In contrast to the full *HeronHODLR* model, this model models only non-spinning waveforms between mass ratios of 1 and 10.

class `heron.models.georgebased.Heron2dHodlrIMR`

Produce a BBH waveform generator using the Hodlr method with IMRPhenomPv2 training data.

Methods

<code>bilby(self, time, mass_1, mass_2, ...)</code>	Return a waveform from the GPR in a format expected by the Bilby ecosystem
<code>build(self[, mean, white_noise, tol])</code>	Construct the GP object
<code>distribution(self, p, times[, samples, ...])</code>	Return the mean waveform and the variance at a given location in the BBH parameter space.
<code>eval(self)</code>	Prepare the model to be evaluated.
<code>log_evidence(self, k, n)</code>	Evaluate the log-evidence of the model at a hyperparameter location k .
<code>mean(self, p, times)</code>	Return the mean waveform at a given location in the BBH parameter space.
<code>train(self)</code>	Prepare the model to be trained.

5.2 Implementing new models

All models implemented in the *heron* package are built on top of the *Model* class, which provides a number of useful methods to assist in creating a Gaussian process model.

class `heron.models.Model`

This is the factory class for statistical models used for waveform generation.

A model class must expose the following methods: - *distribution* : produce a distribution of waveforms at a given point in the parameter space - *mean* : produce a mean waveform at a given point in the parameter space - *train* : provide an interface for training the model

All of the methods which are provided by this class are intended to be treated as *private* methods, which other classes build upon, and which aren't intended to be accessed directly.

To build a new model we can begin by inheriting the *Model* class.

```
class NewModel(Model):
    pass
```

This will give your new model access to the various methods which are needed to produce waveform outputs from a model.

As a minimum your new model must contain three methods so that *heron* can interact with it properly.

5.2.1 `distribution()`

This method should return the parameters of the waveform distribution at a given location in parameter space, i.e. a vector representing the mean and variance at each requested location. The function should have the following signature

```
distribution(self, p, times, *args, **kwargs)
```

With *p* being a dictionary of coordinates in the parameter space, and *times* being a list or array of times at which the waveform distribution should be produced.

5.2.2 `mean()`

This method should return only the mean waveform at a given location in the parameter space. It should have the signature

```
mean(p, times, *args, **kwargs)
```

Where *p* is a dictionary of coordinates in the parameter space, and *times* is a list or array of times at which the waveform distribution should be produced.

5.2.3 `train()`

This method defines the correct way to “train” the model, in order to determine the optimal values of the hyperparameters for the model (using an empirical Bayesian approach). This method should, at the minimum, set `self.training` to `True` for the model, and `self.evaluate` to `False`, in order to mark the model as being in a training state, and not suitable for evaluation.

If this method conducts the entire training process you may then switch these flags to set `self.evaluate` to `True` and `self.training` to `False` before the method completes its execution. Alternatively you should define an `eval()` method to do this, and any other cleaning-up which should be done after training. If *heron* encounters a model in training state when attempting to evaluate it, it will first attempt to run the `eval()` method on the model.

In addition to the `Model` class, a number of additional helper classes exist within *heron*, mainly to help with the construction of gravitational wave models.

5.2.4 Gravitational wave-specific classes

The `heron.models.gw.BBHSurrogate` should be inherited in a class if it is designed to emulate binary black hole waveforms. This class provides metadata related to the intrinsic parameters of these systems. For simpler models which don’t include spin effects you should use the `heron.models.gw.BBHNonSpinSurrogate` class instead.

For time-domain strain models you should have your model class inherit the `heron.models.gw.HofTSurrogate` class. This provides interfaces to the waveform model which are particular to a time-domain model.

CHAPTER 6

Indices and tables

- `genindex`
- `modindex`
- `search`

H

Heron2dHodlrIMR (class in *heron.models.georgebased*), [18](#)

HeronHodlr (class in *heron.models.georgebased*), [17](#)

M

Model (class in *heron.models*), [18](#)