DRAFT VERSION DECEMBER 3, 2018
Typeset using LaTeX **twocolumn** style in AASTeX62

stingray: A Modern Python Library For Spectral Timing

DANIELA HUPPENKOTHEN[1]

MATTEO BACHETTI[2]

ABIGAIL L. STEVENS[3,4]

OTHERS!

[1]*DIRAC Institute,*
Department of Astronomy,
University of Washington,
3910 15th Ave NE, Seattle, WA 98195
[2]*INAF-Osservatorio Astronomico di Cagliari,*
via della Scienza 5,
I-09047 Selargius (CA), Italy
[3]*Department of Physics & Astronomy, Michigan State University, 567 Wilson Road, East Lansing, MI 48824, USA*
[4]*Department of Astronomy, University of Michigan, 1085 South University Avenue, Ann Arbor, MI 48109, USA*

Submitted to ApJ

ABSTRACT

This paper describes the design and implementation of stingray, a library in Python built to perform time series analysis and related tasks on astronomical light curves. Its core functionality comprises a range of Fourier analysis techniques commonly used in Spectral Timing, as well as extensions for analysing pulsar data, simulating data sets, and statistical modelling. Its modular build allows for easy extensions and incorporation of its methods into data analysis workflows and pipelines. We aim for the library to be a platform for the implementation of future spectral timing techniques. Here, we describe the overall vision and framework, core functionality, extensions and connections to high-level command-line and graphical interfaces. The code is well-tested, with a test coverage of currently 95%, and is accompanied by extensive API documentation and a set of step-by-step tutorials.

Keywords: methods:statistics

1. INTRODUCTION

Variability is one of the key diagnostics in understanding the dynamics, emission processes and underlying physical mechanisms of astronomical objects. The detection of periodic variations in the radio flux of certain celestial objects has led to the ground-breaking discovery of pulsars (Hewish et al. 1968). Similarly, accurate models of dips in stellar light curves have led to the discovery of thousands of exoplanets (Charbonneau et al. 2000; Henry et al. 2000; Coughlin et al. 2016, e.g.). In high-energy astrophysics, particularly the study of black holes and neutron stars, the scientific developments of recent decades have brought a growing understanding that time and wavelength are intricately linked. Different spectral components react differently to changes in accretion rate and dynamics, leading to time lags, correlated variability and higher-order effects (for a review, see e.g. Uttley et al. 2014). This has led to the study of accretion disks, in particular those of Active Galactic Nuclei (AGN), via reverberation mapping (e.g. Blandford & McKee ????; Bentz 2016), and probes of the accretion disk geometry using the energy-dependence of quasi-periodic oscillations in stellar-mass black holes (Ingram & van der Klis 2015; Ingram et al. 2016). Understanding how the emission at various wavelengths changes with time is crucial for testing and expanding our understanding of General Relativity in the strong gravity limit, the dense matter equation of state and other fundamental questions in astrophysics. In X-

Corresponding author: Daniela Huppenkothen
dhuppenk@uw.edu

rays, there is now a wealth of data sets of variable objects from missions such as the *Rossi X-ray Timing Explorer* (RXTE; Bradt et al. 1993), the *Nuclear Spectroscopic Telescope Array* (*NuSTAR* ; Harrison et al. 2013), and more recently the Neutron Star Interior Composition Explorer (*NICER*; Gendreau et al. 2016). In addition, planned missions such as the Advanced Telescope for High-ENergy Astrophysics (*Athena*) will produce data sets of unprecedented size and complexity, motivating the need for both scalable algorithms as well as the well-tested, open-source software described in this manuscript.

The paper layout is as follows: In Section 2, we very briefly describe the data sets being used in this paper to showcase the implemented methods. In Section 3, we lay out the overall vision, followed by a description of the general package structure. The package's core functionality is shown in more detail in Section 4, where we introduce basic classes for generating light curves and Fourier spectra of various types. In Sections 5, 6 and 7, we present the submodules allowing modelling Fourier products, simulating light curves from stochastic processes, and pulsar analysis, respectively. Section 8 addresses the development process, testing and documentation environments, and Sections 9 and 10 point to connections with a command-line interface and a graphical user interface, respectively, which are being developed concurrently with `stingray`. Finally, in Section 11 we lay out our future development plans. Note that we intentionally omit code examples and specific implementation details in this manuscript in order to preserve long-term accuracy. All code to reproduce the figures in this paper is available online[1], and a full suite of up-to-date tutorials is also available[2].

2. DATA

Throughout the paper, we use real X-ray observations of compact objects to demonstrate the functionality of the software in this package. Here, we give brief introductions into the observations used and the data reduction processes applied before using the resulting event files and time series with `stingray`.

2.1. *GX 339-4*

GX 339–4 is a stellar black hole in a low-mass X-ray binary (Hynes et al. 2003). The black hole has a lower mass limit of $\sim 7 M_\odot$ (Muñoz-Darias et al. 2008) and possibly a near-maximal spin (Ludlam et al. 2015). The system also likely has a low binary orbit inclination; it has been constrained to $37^\circ < i \lesssim 60^\circ$ from optical and X-ray observations (Heida et al. 2017; Zdziarski et al. 1998), and spectral modelling by Wang-Ji et al. (2018) estimates $i \approx 40^\circ$. For this example we use an RXTE Proportional Counter Array (PCA; Jahoda et al. 1996) observation in NASA's High Energy Astrophysics Science Archive Research Center (HEASARC) from the 2010 outburst of GX 339–4 (Yamaoka et al. 2010). We are using observation ID 95409-01-15-06 taken from UT 2010-04-22 23:36:52 to UT 2010-04-23 00:01:10. This observation was taken in 64-channel event mode with 122 μs time resolution (`E_125us_64M_0_1s`). The following filtering criteria were used to obtain Good Time Intervals (GTIs): Proportional Counter Unit (PCU) 2 is on, two or more PCUs are on, elevation angle $> 10^\circ$, and target offset $< 0.02^\circ$. Time since the South Atlantic Anomaly passage was not filtered on.

2.2. *Hercules X-1*

Hercules X-1 (Her X-1) is a well-known persistent X- ray binary pulsar with a period of $P = 1.23$s (Tananbaum et al. 1972) in a binary system with a $\sim 2.2 M_\odot$ stellar companion HZ Herculis (Davidsen et al. 1972; Forman et al. 1972; Bahcall & Bahcall 1972; Reynolds et al. 1997; Leahy & Abdallah 2014) with an orbital period of $P_{\rm orb} = 1.7$ days and super-orbital variations on a ∼35-day timescale (Giacconi et al. 1973; Scott & Leahy 1999; Igna & Leahy 2011). The companion's type varies between late-type A and early-type B with orbital phase (Anderson et al. 1994; Cheng et al. 1995). For this work, we considered two of the several observations of Her X-1 with the *Nuclear Spectroscopic Telescope Array* (*NuSTAR* ; Harrison et al. 2013). The first observation was taken from UT 2012-09-19 to UT 2012-09-20 and was one among several used by (Fürst et al. 2013) to characterize the cyclotron resonance scattering features (CRSFs) in the spectrum of the source. The second was taken from UT 2016-08-20 to UT 2016-08-21 and was used by (Staubert et al. 2017) to detect an inversion of the decay of the CRSF. We downloaded the observation directory for observation IDs 30002006002 and 10202002002 from the HEASARC and used the FTOOL barycorr and the latest (as of Nov. 27, 2018) *NuSTAR* clock correction file on the L2 cleaned science event files (file name ending with `01_cl.evt`) to correct the photon arrival times to the solar system barycenter. For our analysis, we considered photons from 3 to 79 keV at most 50″ from the nominal position of the source, extracted from the two identical Focal Plane Modules A and B (FPMA and FPMB, respectively) onboard the spacecraft. We used a total of 32.67ks of good time intervals (GTIs) in the first observation and 36.56 ks in the second, only selecting intervals longer than 10s.

3. VISION AND GENERAL PACKAGE FRAMEWORK

Despite decades of research, the field of spectral timing in high-energy astrophysics is fragmented in terms of software; there is no commonly accepted, up-to-date framework for the core data analysis tasks involved in (spectral) timing. Code is often siloed within groups, making it difficult to reproduce

[1] https://github.com/StingraySoftware/stingraypaper
[2] https://github.com/StingraySoftware/notebooks

scientific results. Additionally, the lack of fully open-source, well-documented tools constitutes a significant barrier to entry for researchers new to the field, since it effectively requires anyone not part of collaborations with an existing private code base to write their own software from scratch. The NASA library `xronos` is, to our knowledge, the only widely used open-source library in this field, and has several shortcomings. In particular, it performs only a few of the most basic tasks, and it has not been maintained since 2004. Other open-source projects use languages that either require an expensive license (e.g. IDL) or have a very limited scope (e.g. S-Lang). This dearth of well-tested, well-documented software for spectral timing motivated the development of `stingray`[3], a library built entirely in the popular `Python` language and based on `astropy` functionality. `stingray` aims to make many of the core Fourier analysis tools used in timing analysis available to a large range of researchers while providing a common platform for new methods and tools as they enter the field. It includes the most relevant functionality in its core package, while extending that functionality in its subpackages in several ways, allowing for easy modeling of light curves and power spectra, simulation of synthetic data sets and pulsar timing.

Its core idea is to provide time series analysis methods in an accessible, unit-tested way, built as a series of object-oriented modules. In practice, data analysis requirements are varied and depend on the type of data, the wavelength the observation was taken at, and the object being observed. With this in mind, `stingray` does not aim to provide full-stack data analysis workflows; rather, it provides the core building blocks for users to build such workflows themselves, based on the specific data analysis requirements of their source and observation. The modularity of its classes allows for easy incorporation of existing `stingray` functionality into larger data analysis workflows and pipelines, while being easily extensible for cases that the library currently does not cover.

`stingray` separates out core functionality from several more specialized tasks based on those core classes and functions. Constructs related to data products as well as Fourier transforms of the data (e.g. power spectra, cross spectra, time lags, and other spectral timing products) are considered core functionality, as are some utility functions and classes, for example related to GTI calculations.

This core functionality is extended in various ways in currently three subpackages. The `modeling` subpackage (see also Section 5) provides a framework for modelling light curves and Fourier spectra with parametric functions. Based on this framework, it allows users to search efficiently for (quasi-)periodic oscillations in light curves with stochastic variability, and provides convenience functions to aid standard tasks like fitting Lorentzian functions to power spectra.

The subpackage `simulator` (Section 6) provides important functionality to allow efficient simulation of time series from a range of stochastic processes. This includes simulation of light curves from power spectral models as well as the use of transfer functions to introduce time lags and higher-order effects.

Finally, the subpackage `pulsar` implements a range of methods particularly useful for period searches in pulsars. While no other sub-packages are currently under construction, we will implement new sub-packages as required.

`stingray` is formally hosted as part of a GitHub organization called *StingraySoftware*[4], which also hosts the website, documentation, a repository with tutorial notebooks, this manuscript, as well as the two related software packages. `stingray` is designed to be used as a standalone package, but is also at the core of two other software packages currently under development: `HENDRICS` (Bachetti 2015a; see also Section 9) provides pre-built data analysis workflows using `stingray`'s core functionality. These workflows are accessible from the command line and are provided for some common data types and data analysis tasks. `DAVE`[5] (see also Section 10) provides a Graphical User Interface on top of `stingray` to allow for easy interactive exploratory data analysis.

As of v0.1, `stingray` is entirely written in Python, and its core functionality depends exclusively on `numpy` (van der Walt et al. 2011), `scipy` (Jones et al. 2001–) and `astropy` (The Astropy Collaboration et al. 2018), with optional plotting functionality supplied by `matplotlib` (Hunter 2007) . The `modeling` subpackage optionally uses sampling methods supplied by `emcee` (Foreman-Mackey et al. 2013), some functionality implemented in `statsmodels`, and plotting using `corner` (Foreman-Mackey 2016). The `pulse` subpackage optionally allows for just-in-time compilation using `numba` (Lam et al. 2015) for computational efficiency, and for advanced pulsar timing models using `PINT`[6] .

This paper describes `stingray` v0.1, released on 2018-02-12. As with most open-source packages, `stingray` is under continuous development and welcomes contributions from the community.

4. CORE FUNCTIONALITY

`stingray` imports its core functions and classes from the top level package. These classes define the basic data structures such as light curves and cross- as well as power

[3] `stingray` was named partly in homage to the popular 1960s childrens' TV series, from where `stingray`'s motto derives: *Anything can happen in the next half hour (but probably spectral timing)!*

[4] https://github.com/StingraySoftware/

[5] https://github.com/StingraySoftware/dave

[6] https://github.com/nanograv/PINT

spectra that are used in much of the higher-level functionality provided in the sub-packages. Additionally, it incorporates a number of utilities for dealing with GTIs as well as input and output of data sets.

4.1. *The* `Lightcurve` *class*

We expect `stingray` to be used largely on data sets of two forms: (1) event data (i.e. recordings of arrival times of individual photons) or (2) binned light curves (i.e. measurements of brightness in units of flux, magnitude or counts as a function of time).

The majority of methods in `stingray` use binned light curves, which we thus currently consider the default format. The `Lightcurve` class defines a basic data structure to store binned light curves. Its attributes include arrays describing time bins and associated (flux or counts) measurements, the number of data points in the light curve, the time resolution and the total duration of the light curve. For unevenly sampled light curves, the time resolution `dt` will be defined as the median difference between time bin midpoints. Users can pass uncertainties for measurements directly, or pass a string defining the statistical distribution of the data points for automatic calculation. By default, a Poisson distribution is assumed, appropriate for binned event data.

There are two ways to generate a `Lightcurve` object: in the standard case, the instrument has recorded a binned time series of N pairs of time stamps and count (rate) or flux values, $\{t_k, c_k\}_{k=1}^{N}$. In this case, one can simply instantiate a `Lightcurve` object with the keywords `time` and `counts` (and optionally set `use_counts=False` when the input is in units of counts per second). In cases where the native data format is events (e.g. photon arrival times) it is possible to use the static method `Lightcurve.make_lightcurve`, passing the array of events as well as a time resolution `dt` to create a new light curve from the events.

Various operations are implemented for class `Lightcurve`. Custom behaviour of the $+$ and $-$ operators allows straightforward addition and subtraction of light curves from one another. Assuming the light curves have the same time bins, the $+$ and $-$ operators will add or subtract the flux or counts measurements, respectively, and return a new `Lightcurve` object with the results. Other common operations implemented include time-shifting the light curve by a constant factor, joining two light curves into a single object, truncating a light curve at a certain time bin, and input/output operations to read or write objects from/to disk in various formats (HDF5, FITS and ASCII are currently supported). For light curves that do not have consecutive time bins, there is a sorting operations, as well as the option to sort the light curve by the ascending or descending flux or counts.

We provide support for GTIs in many methods and implement rebinning the light curve to a new time resolution exceeding the native resolution of the data (interpolation to a finer resolution is currently not supported). In Figure 1 (left panel), we show an example observation of GX 339-4 as taken with with *RXTE* . `stingray` implements basic methods for plotting (useful for a quick look at the data).

4.2. *The* `Events` *class*

At short wavelengths, data is largely recorded as *photon events*, where arrival times at the detector are recorded for each photon independently, along with a number of other properties of the event (for example an energy channel in which the photon was recorded in, which can be transformed to a rough estimate of the energy of the original photon arriving at the detector).

Even for a single instrument, there are often multiple types of data that can be recorded, resulting in a plethora of data formats and internal schemas for how data is stored within the binary files distributed to the community. `stingray` implements a basic `EventList` class that acts as a container for event data, but does not aim to encompass all data types of all current (and future) instruments. Instead, it aims to abstract away from instrument-specific idosyncrasies as much as possible and remain mission-agnostic. In its basic form, it takes arrays with time stamps and optionally corresponding photon energies as input, and implements a set of basic methods. Similarly to `Lightcurve`, it provides basic input/output (I/O) functionality in the form of `read` and `write` methods as well as a method to join event lists, which can be particularly useful when data is recorded in several independent detector, as is common for several current and future X-ray missions. The `to_lc` method provides straightforward connection to create a `Lightcurve` directly out of an `EventList` object. In return, it is possible to create an `EventList` out of a `Lightcurve` object using the `from_lc`. The latter will create N_i events, each with a time stamp equation to the time bin t_i, where N_i is the number of counts in bin i (event lists are, by their very definition only a useful data product if the light curve used to simulate comes from photon counting data in the first place). It is possible to simulate more physically meaningful photon events from a given light curve and energy spectrum using the `simulate_times` and `simulate_energies` methods (from the `simulator` package, Section 6), which employ a combination of interpolation and rejection sampling to accurately draw events from the given light curve and spectrum.

4.3. *Cross Spectra and Power Spectra*

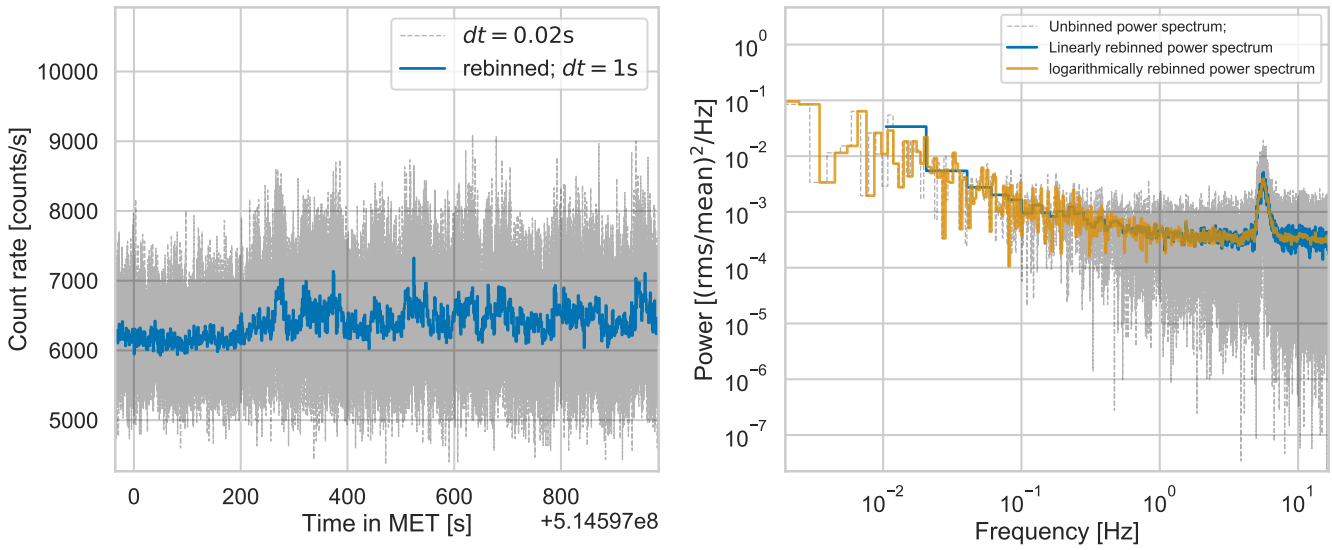


Figure 1. Left panel: A $\sim$ 1ks *RXTE* observation of the black hole X-ray binary GX 339-4. Details of the observation can be found in Section 2.1. In grey, we show the light curve produced by binning the events into 0.02 s bins. The blue line corresponds to the rebinned light curve at $dt = 1.0$ s. Right panel: we show the power spectrum calculated from the light curve in the left panel (grey), as well as a version of the same power spectrum that has been linearly rebinned in frequency (blue), and a version that was logarithmically rebinned (orange).

The cross spectrum and the power spectrum[7] are closely related (for a pedagogical introduction into Fourier analysis, see van der Klis 1989; see also Uttley et al. 2014 for a recent review of spectral timing techniques). Computing the cross spectrum requires two evenly sampled time series $\mathbf{y}_1 = \{y_{1,i}\}_{i=1}^{N}$ and $\mathbf{y}_2 = \{y_{2,i}\}_{i=1}^{N}$ taken simultaneously at exactly the same time intervals $\{t_i\}_{i=1}^{N}$. Under this assumption, one may then compute the discrete Fourier transform of each time series, $\mathcal{F}_1$ and $\mathcal{F}_2$ independently, and multiple the $\mathcal{F}_1$ with $\mathcal{F}_2^*$, i.e. the Fourier transform of $\mathbf{y}_1$ with the *complex conjugate* of the Fourier transform of $\mathbf{y}_2$.

Because the power spectrum is defined as the square of the real part of the Fourier amplitudes of a single, evenly sampled time series, it can be formulated as the special case of the cross spectrum where $\mathbf{y}_1 = \mathbf{y}_2$. In `stingray`, we implement a class `Crossspectrum`, which takes two `Lightcurve` objects as input and internally calculates the complex cross spectrum in one of a number of common normalizations (see below). Because many of the internal calculations are the same, the class `Powerspectrum` is implemented as a subclass of `Crossspectrum`, but takes only a single `Lightcurve` object instead of two.

[7] In the signal processing literature, generally a distinction is made between the power spectrum, which describes the process at the source generating variable time series, and the periodogram, which denotes a realization of said power spectrum, i.e. the time series we actually observe. While the products generated by `stingray` generally fall into the latter category, the astronomy literature usually denotes them by the term power spectrum. We follow this convention here as we do within the software package itself.

There are several popular normalizations for the real part of the cross spectrum as well as the power spectrum implemented in `stingray`: the *Leahy normalization* (Leahy et al. 1983a) is defined such that for simple white noise, the power spectrum will follow a χ^2 distribution with 2 degrees of freedom around a mean value of 2, and the cospectrum—the real part of the cross spectrum—will follow a Laplace distribution centred on 0 with a scale parameter of 1. It is particularly useful for period searches, because the white noise level is well understood and always the same (but be aware that detector effects like dead time can distort the power spectrum in practice; Bachetti et al. 2015). For light curves with complex variability patterns, and especially for understanding how these patterns contribute to the overall variance observed, the *fractional rms normalization* (Belloni & Hasinger 1990; Miyamoto et al. 1992) or the *absolute rms normalization* (Uttley & McHardy 2001) may be more appropriate choices.

The classes `Crossspectrum` and `Powerspectrum` share most of the implemented methods, except where otherwise noted. Both classes include methods to rebin cross- and power spectra. Linear rebinning is implemented analogously to the method in class `Lightcurve`. Additionally, logarithmic binning is implemented in the method `rebin_log` in such a way that the bin width at a given frequency increases a fraction of the previous bin width:

$$d\nu_{i+1} = d\nu_i(1+f) \ ,$$

where f is some constant factor by which the frequency resolution increases, often $f = 0.01$.

Classical period searches are often formulated as outlier detection problems from an expected statistical distribution. Assuming the signal is sufficiently coherent such that all of the signal power is concentrated in one bin, one may calculate the chance probability that an observed power in the spectrum was generated by statistical fluctuations alone. For the white noise case, the equations to accurately calculate a p-value of rejecting the hypothesis that a given outlier in the power spectrum was generated by noise are defined in (Groth 1975), and can be calculated for one or multiple powers in a `Powerspectrum` object using the `classical_significances` method, which enables computation of a (trial-corrected) p-value for a given power in the presence of white noise. Note that the cross spectrum does not follow the same distribution (Huppenkothen & Bachetti 2017), and the recently derived statistical distributions for this case will be implemented in a future version of `stingray`.

In many practical applications, users may wish to average power- or cross spectra from multiple light curve segments in order to suppress statistical noise. This can easily be done with the appropriate classes `AveragedPowerspectrum` and `AveragedCrossspectrum`, which take a `Lightcurve` object or list of `Lightcurve` objects as an input and will compute averaged Fourier products by dividing the light curve into N segments of a given size $\tau_{\rm seg}$. The Fourier spectra (either cross spectra or power spectra) are averaged together. Both are subclasses of `Crossspectrum`, and either inherit or override many of the methods relevant for those classes as well. An example of the kinds of products produced by the functions above is given in Figure 1.

For averaged cross spectra, it is possible to calculate the time lag, defined as

$$\tau_j = \frac{\phi_j}{2\pi\nu_j}$$

for a phase angle ϕ_j derived from the imaginary component of the complex cross spectrum, and a mid-bin frequency ν_j. Similarly, it is possible to calculate the coherence (Vaughan & Nowak 1997; Nowak et al. 1999) from the cross spectrum, defined as

$$c_j = \frac{C_{xy,j}}{C_{x,j}C_{y,j}} \ . \quad (1)$$

Here, $C_{xy,j}$ corresponds to the real part of the unnormalized cross spectrum, and $C_{x,j}$ and $C_{y,j}$ correspond to the analogous squared amplitudes of the power spectrum for each individual light curve.

For long observations with quasi-periodic oscillations (QPOs) spectrograms, more commonly known in the astronomy literature as Dynamic Power Spectra, can be a useful way to track changes in the QPO centroid frequency over time. We have implemented `DynamicalPowerspectrum` as a subclass to `AveragedPowerspectrum` to provide this functionality. Like `AveragedPowerspectrum`, this class takes a `Lightcurve` object and a segment size as input, but instead of averaging the power spectra of each individual segment, it will create a matrix of time bins (one bin for each segment) as columns and Fourier frequencies as rows. Rebinning both along the time and frequency axis is possible. Moreover, the method `trace_maximum` automatically finds the frequency with the highest power in each segment in a given range of frequencies, and traces this maximum over time. An example using data from the source GX 339-4 [[PLEASE CHECK]] is shown in Figure 2.

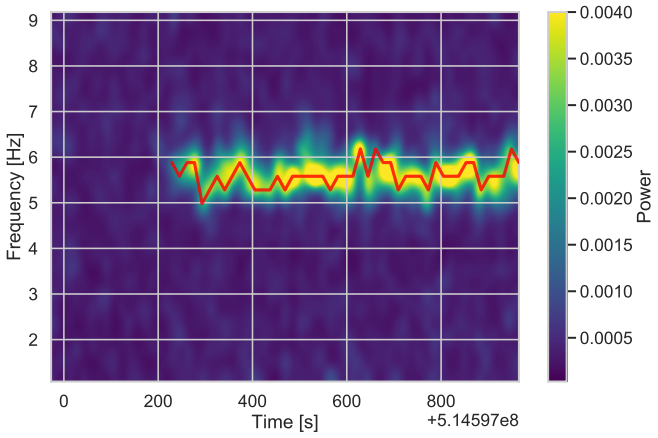


Figure 2. An example of a dynamic power spectrum generated from the GX-339 light curve shown in Figure 1. We generated 63 light curve segments of 16 seconds length with a 0.02 s time resolution and Fourier-transformed each to generate a power spectrum. The dynamic power spectrum here plots each power spectrum as a vertical slice as a function of time, with the colour indicating the fractional rms-normalized power in each bin (yellow are large powers, purple small) . The dynamic power spectrum was clipped to around the range of the QPO, and smoothed using bicubic interpolation to improve clarity. The QPO is clearly visible as a yellow streak, but seems not to be present during the entire observation. In red, we show the frequency with the highest power found in each segment (excluding frequencies below 3 Hz to exclude the low-frequency red noise), using the `trace_maximum` method.

Closely related to the cross spectrum and power spectrum are the crosscorrelation and the autocorrelation, implemented in classes `CrossCorrelation` and `AutoCorrelation`. As their respective Fourier spectra equivalents they take either one (autocorrelation) or two (cross correlation) `Lightcurve` objects as input and computes the correlation between the two light curves or of the single light curve with itself, along with the time lags for which the correlation was produced and the time lag at which the maximum correlation is measured.

It is useful to note that all classes in this section are compatible with GTIs. The classes `Powerspectrum` and

`Crossspectrum` will generate warnings if the observations contain gaps; their averaged versions will take GTIs correctly into account by producing power spectra only from light curve segments for which data is available.

5. THE `MODELING` SUBPACKAGE

Modelling data sets with parametric (often physically motivated) models that map an independent variable (e.g. time or frequency) to one or more dependent variables (e.g. flux, counts or Fourier powers) is a common task in astronomy. Constructing a universal modelling framework is a highly non-trivial task, and better packages exist for general-purpose model building (e.g. STAN, Carpenter et al. 2017), thus `stingray`'s modelling interface restricts itself to models of commonly used spectral-timing products, in particular (averaged) power spectra. While it makes heavy use of the `astropy.modeling.FittableModel` definitions, it uses custom definitions for fitting algorithms motivated by the statistical properties of spectral timing products, which deviate significantly from other data types commonly found in astronomy and thus cannot easily be modelled with standard approaches defined in `astropy`.

The modelling subpackage logically separates out statistical models – likelihoods and posteriors – from the fitting functionality, such that different likelihoods and posteriors can be straightforwardly dropped in and out depending on the data set and problem at hand.In line with the overall philosophy of `stingray`, the modelling subpackage is designed to be modular and easily extensible to specific problems a user might try to solve, while many typical tasks one might do with Fourier products are already built-in. It makes use of the `scipy.optimize` interface for optimization as well as the package `emcee` for Markov Chain Monte Carlo (MCMC) sampling.

5.1. *Statistical Models*

All statistical models are implemented as a subclass of an Abstract Base Class `Likelihood` in module `stingray.posterior`. In its most basic form, each subclass of `Likelihood` takes data in some form (most commonly two arrays, one with the independent and one with the dependent variable) as well as an object of type `astropy.modeling.FittableModel`. The likelihood computes model values for each data point in the array of independent variables and statistically compares these model values with the data points stored in the dependent variable, assuming the particular statistical distribution of the likelihood definition. The result is a single scalar, which can then be e.g. used in an optimization algorithm in order to find a Maximum Likelihood (ML) solution.

For all likelihoods in `stingray`, `stingray` an equivalent subclass of `stingray.modeling.Posterior` available, which uses the `Likelihood` definitions to compute posterior probability densities for the parameters of a model given data. All subclasses of `Posterior` also require definition of a `logprior` method, which calculates the value of the prior probability density of the parameters. Because priors are strongly problem-dependent, they cannot be hard-coded into stingray. Even for relatively straightforward problems such as modelling quasi-periodic oscillations of X-ray binaries, the physical properties and their effect on the data can differ strongly from source to source, indicating that a prior set for Cygnus X-1 may not be appropriate for e.g. GRS 1915+105. Separating out the likelihood and posterior in distinct classes makes it possible to allow the use of the likelihood for maximum likelihood estimation, while requiring priors for estimating the Bayesian posterior probability through e.g. MCMC simulations.

`Loglikelihood` and `Posterior` subclass definitions currently exist within `stingray` for different statistical models useful in the context of astronomical data. `GaussianLogLikelihood` and `GaussianPosterior` implement statistical models for data with normally distributed uncertainties. `GaussianLogLikelihood` will compute what astronomers generally call χ^2, because the likelihood calculated by this statistical model generally follows a χ^2 distribution with $N - P$ degrees of freedom (where N is the number of data points and P the number of free parameters). Note, however, that this is **not** the same as the χ^2 likelihood defined below!

`PoissonLogLikelihood` and `PoissonPosterior` calculate the likelihood and posterior for Poisson-distributed data, respectively. This likelihood is equivalent to what in astronomy is often called the *Cash statistic* (Cash 1979) and is the appropriate likelihood to use for count- or event-type data often found in X-ray astronomy time series and spectra.

`PSDLogLikelihood` and `PSDPosterior` implement the statistical model appropriate for modelling (averaged) power spectra, a χ^2 distribution. We broke with the rule of naming likelihoods and posteriors after the statistical distribution they implement in this case, because as mentioned above, astronomers tend to call the likelihood for normally distributed data χ^2, and this naming helps avoid any confusion. These two classes implement a χ^2_2 distribution for Fourier spectra generated with the `Powerspectrum` class, and a χ^2_{2MK} distribution for power spectra generated with the `AveragedPowerspectrum` class, where M is the number of averaged segments and K is the number of averaged neighbouring frequency bins. Please note that as laid out in Huppenkothen & Bachetti (2017), these distribution are **not** appropriate for use on (averaged) cross spectra. The appropriate distributions for these products are under development for the next version of `stingray`.

Other statistical models can be easily implemented by subclassing the `LogLikelihood` and `Posterior` Abstract Base Classes and using the existing classes as template.

5.2. *General Parameter Estimation and Model Comparison Functionality*

Stingray implements utility functions in order to reduce some of the overhead required for standard parameter estimation and model comparison tasks. In particular, the `parameterestimation` module implements classes and functions to aid users in fitting models to data and estimating the probability distributions of parameters.

The class `ParameterEstimation` provides the basis for more sophisticated, specialized implementations for particular data types. Its core methods are `fit` and `sample`. The former takes an instance of a `LogLikelihood` or `Posterior` subclass and uses minimization algorithms implemented in `scipy.optimize` to find the Maximum Likelihood (ML) or Maximum-A-Posteriori (MAP) solution. The `sample` method uses the Affine-Invariant MCMC sampler implemented in `emcee` (Foreman-Mackey et al. 2013) to generate samples from a posterior distribution passed as an instance of a subclass of `Posterior`. Note that *you should never pass a* `LogLikelihood` *instance into the* `sample` *method*, because sampling from a likelihood is statistically invalid. In addition to these core methods, higher-level functionality implemented in this class includes calculating the Likelihood Ratio Test (LRT) for two different models M_1 and M_2 via the `compute_lrt` method (note the statistical assumptions of the LRT, and where they fail, e.g. Protassov et al. 2002). In addition, the `calibrate_lrt` method allows calibrating the p-value for rejecting the model M_1 via simulations of M_1, using either an MCMC sample (for Bayesian inference and posterior predictive p-values) or the covariance matrix derived from the optimization (both Bayesian and Maximum Likelihood approaches).

Stingray also implements two classes that summarize results of the optimization and sampling procedures in concise, useful ways. The `fit` method returns an instance of class `OptimizationResults`. This contains the most important outputs from the optimizer, but will also behind the scenes calculate a number of useful quantities, including the covariance between parameters (or a numerical approximation for some minimization algorithms), the Akaike and Bayesian Information Criteria (AIC: Akaike 1974; BIC: Schwarz 1978) as well as various summary statistics.

Similarly, an instance of class `SamplingResults` is returned by the `sample` method, which returns the posterior samples calculated by the MCMC sampler, as well as computes a number of helpful quantities using the MCMC chains. It calculates useful diagnostics including the acceptance fraction, the autocorrelation length and the Rubin-Gelman statistic (Gelman & Rubin 1992) to indicate convergence, and infers means, standard deviations and user-defined credible intervals for each parameter.

5.3. *Special Functionality for Fourier Products*

The subclass `PSDParEst` implements a number of additional methods particularly useful for modelling power spectra. One particularly common task is to search for periodic signals (e.g. from pulsars) in a power spectrum, which reduces to finding outliers around an assumed power spectral shape (assuming the signal is *strictly* periodic, and thus all power approximately concentrated in one bin). In the presence of other variability, the probability of observing a certain power P_j at a frequency ν_j under the assumption that no periodic signal is present depends on the shape and parameters of the underlying power spectral model assumed to have generated the data. As Vaughan (2010) show, there is an inherent uncertainty in our inference of the parameters of this power spectral model, which must be taken into account via simulations. `PSDParEst` implements a method `calibrate_highest_outlier`, which finds the k highest outliers (where k is a user-defined number) and calculates the posterior predictive p-value that said outliers cannot be explained by noise alone. It makes heavy use of the method `simulate_highest_outlier`, which uses the `sample` method to derive an MCMC sample and then simulate fake power spectra from that model for a range of plausible parameter values in order to include our model uncertainty in the posterior predictive p-value. For details of the overall procedure, see Vaughan (2010).

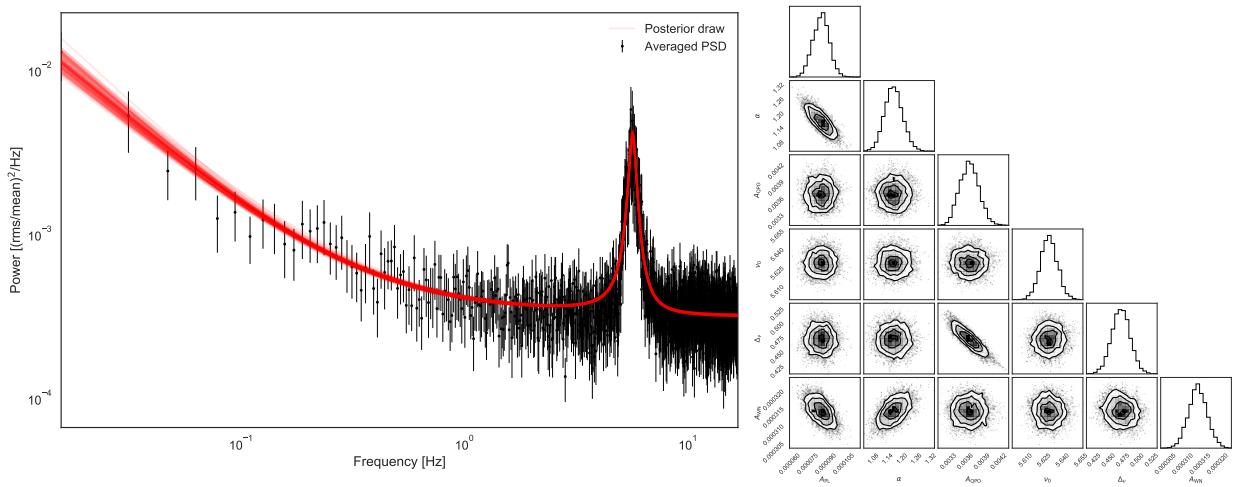
As of this version, the `stingray.modeling` subpackage has no functionality to model higher-order Fourier products. For spectral timing in particular, this would involve being able to read and apply instrument responses to models, as well as being able to interface with the library of spectral models associated with the X-ray spectral fitting tool XSPEC (Arnaud 1996). Providing this functionality is planned for a future release of `stingray`.

6. THE `SIMULATOR` SUBPACKAGE

The `simulator` subpackage contains a number of methods to generate simulated light curves out of known power spectral shapes, and event lists from light curves.

6.1. *Simulating light curves from input power spectra*

The basic `Simulator` object uses the algorithm from Timmer & Koenig (1995) to generate light curves out of a spectral shape. The spectral shape can be input as a spectral power-law index, `astropy.modeling.models` objects, as well as a user-given array of powers. In Figure 4, we present a light curve as generated by a given power spectral shape. The output is a `Lightcurve` object that can be used



(a) plot (b) plot

Figure 3. Left panel: In black, a power spectrum averaged out of 15 light curve segments of 64s each of the GX 339-4 observation, along with draws from the posterior distribution of the power law model plus the Lorentzian QPO model and constant used to represent the data (red). Right panel: corner plot showing the marginal posterior distributions (diagonal) of the six parameters of the model: the amplitude of the power law $A_{\rm PL}$, the power law index α, the amplitude of the Lorentzian $A_{\rm QPO}$, the QPO centroid frequency ν_0, the width of the QPO Δ_ν, and the amplitude of the white noise, $A_{\rm WN}$. The right-hand figure was produced using the package `corner` (Foreman-Mackey 2016).

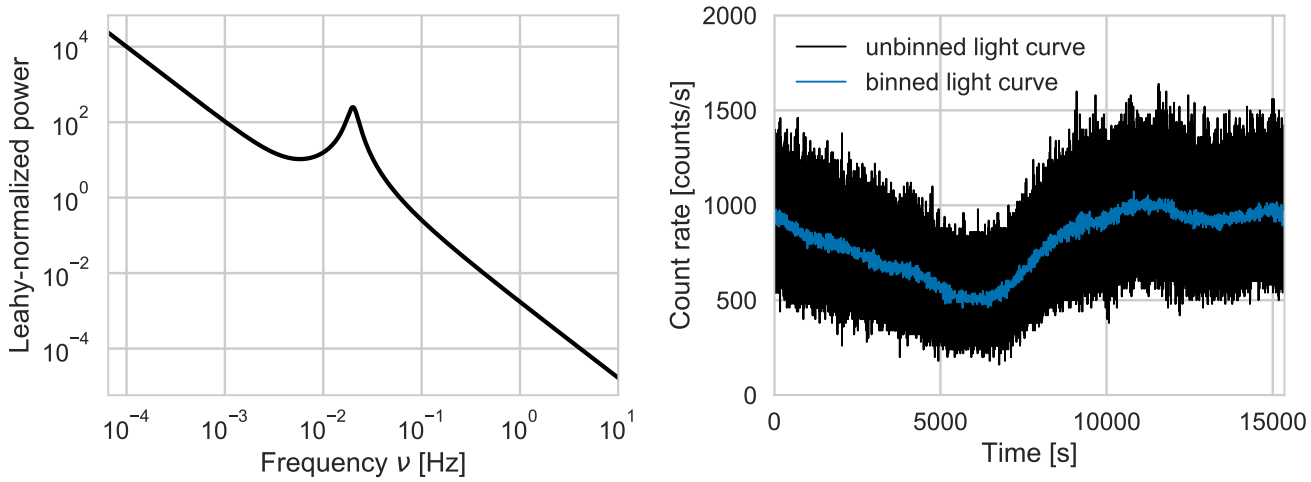


Figure 4. Left: A power spectral shape generated using a compound `astropy.modeling.models` object of a power law and a Lorentzian. Right: A corresponding light curve generated by the `simulator` subpackage with a time resolution of 0.05 s, a total duration of 15 ks, a mean count rate of 40 counts/s and a fractional rms amplitude of 0.2. In blue we show a binned version of the same light curve.

like real data sets, including all functionality related to GTIs, spectral-timing products and modelling.

6.2. *Use transfer functions on light curves*

Most astrophysical signals we receive in our instruments are the mixture of different input signals. Often, a signal emitted in one region can be reflected and re-emitted from another region, or filtered in different ways. Spectral timing studies can decompose the signals and try to understand how the signal is transformed by these phenomena between the emission region and the observer (See Uttley et al. 2014, for a review). This transformation can be encoded in an impulse response function, that describes the response of the system to a delta-function impulse. This is the Fourier transform of another well-known quantity in signal processing, the transfer function (See the textbook by Girod et al. 2001).

The `Simulator` object is capable of generating multiple light curves starting from an initial light curve and multiple input responses, mimicking observations in different energy bands.

6.3. *Simulating event lists from light curves*

The `simulator.base.simulate_times` method is able to simulate event lists from input light curves. It implements the acceptance-rejection method: 1) generate a light curve (and smooth out any Poisson noise if generated through the Timmer & Koenig 1995 method) over the whole observation; normalize it so that the maximum is 1; 2) generate an event, with uniform probability over the observing time; 3) associate to this event a uniform random number $\mathcal{P}$ between 0 and 1; 4) if $\mathcal{P}$ is lower than the normalized light curve at the event time, *accept* the event, otherwise *reject* it. In `stingray`, this procedure is actually done through arrays of events for better performance, using the functionality contained in the `numpy` library.

7. THE `PULSE` SUBPACKAGE

The `stingray` subpackages `pulse` contains the basic operations to perform the search and characterization of pulsed signals for use e.g. in searches of X-ray pulsars.

7.1. *Epoch Folding*

Among the basic algorithms used in pulsar astronomy, one cannot overstate the importance of Epoch Folding (EF). The algorithm consists of cutting the signal at every pulse period and summing all sub-intervals in phase. An alternative way of seeing it, more useful for photon data, is as a *histogram of pulse phases*.

If the period is exactly correct and assuming a stable pulsation, the signal-to-noise ratio will get better approximately with the square root of the number of summed sub-intervals. This is the method used to obtain practically all pulse profiles shown in the literature, as most pulsar signals are orders of magnitude below noise level.

The `pulse.pulsar` submodule contains the functionality to calculate the phase given a simple pulse ephemeris consisting of any number of pulse frequency derivatives, or using a number of methods for the orbit of the pulsar (using the optional dependency `PINT`)Moreover, the module also includes a mechanism to calculate the exposure of single bins in the pulse profile. This is particularly useful for very long-period pulsars where the pulsed period is comparable to the length of the GTIs. The different exposure of pulse bins caused by the absence of signals during GTIs is taken into account in the calculation of the final pulse profile by the folding algorithm, if the user asks for it.

7.2. *Epoch Folding Searches and Z_n^2 Searches*

During a search for pulsations, the first step is usually calculating a power spectrum through a Fast Fourier Transform. However, often pulsations do not leave a clear signature above noise level in the power spectrum, because they are weak or they fall close to bin edges, where the sensitivity is reduced[8]. Even when the signature is clear, the frequency resolution of the power spectrum is often inadequate to measure precisely the pulse frequency. Therefore, an additional statistical analysis is needed.

`stingray` implements two statistical methods for pulsar searches, that can be applied to event lists or light curves (that are treated as event lists with "weights").

The Epoch Folding Search (EFS) method consists of executing the folding at many trial frequencies around the candidate frequency. Once the folding is performed, the following statistics is calculated on the profile:

$$\mathcal{S} = \sum_i \frac{(P_i - \overline{P})^2}{\sigma^2} \tag{2}$$

where P_i are the bins of the profile, $\overline{P}$ is the mean level of the profile, and σ is the standard deviation. $\mathcal{S}$ is the summed squared error of the actual pulsed profile with respect to a *flat* model, and follows a χ^2 distribution.

If there is no pulsation, $\mathcal{S}$ will assume a random value distributed around the number of degrees of freedom $n-1$ (where n is the number of bins in the profile) with a well defined statistical distribution (χ^2_{n-1}) that allows an easy calculation of detection limits. When observing a peak of given height is very unlikely under the null hypothesis (meaning that the probability to obtained this peak by noise is below a certain ϵ), this peak is considered a pulse candidate. If the frequency resolution is sufficiently high, close to the correct frequency, as described by (Leahy et al. 1983b; Leahy 1987), the peak in the epoch folding periodogram has the shape of a sinc^2 function whose width is driven by the length of the observation.

The epoch folding statistic, however, can give the same value for a pulse profile at the correct frequency and, for example, a harmonic that produces a deviation from a Poisson distribution. A more effective method is the Z_n^2 statistics (Buccheri et al. 1983), which is conceptually similar to EF but

[8] This is due to the convolution of the signal with the observing window, that produces a sinc-like response inside the bins of the FFT; periodic signals with the same amplitude are detected with a lower Fourier amplitude if they fall far from the center of the spectral bin (van der Klis 1989)

has high values when the signal is well described by a small number of sinusoidal harmonics:

$$Z_n^2 = \frac{2}{N} \sum_{k=1}^{n} \left[\left(\sum_{j=1}^{N} \cos k\phi_j \right)^2 + \left(\sum_{j=1}^{N} \sin k\phi_j \right)^2 \right], \quad (3)$$

where N is the number of photons, n is the number of harmonics, ϕ_j are the phases corresponding to the event arrival times t_j ($\phi_j = \nu t_j$, where ν is the pulse frequency).

The Z_n^2 statistics defined in this way, far from the pulsed profile, follows a χ_n^2 distribution, where n is the number of harmonics this time. This allows, again, to easily calculate thresholds based on the probability of obtaining a given Z_n^2 by pure noise.

The standard Z_n^2 search calculates the phase of each photon and calculates the sinusoidal functions above for each photon. This is very computationally expensive if the number of photons is high. Therefore, in Stingray, the search is performed by binning the pulse profile first and using the phases of the folded profile in the formula above, multiplying the squared sinusoids of the phases of the pulse profile by a weight corresponding to the number of photons at each phase.

$$Z_n^2 \approx \frac{2}{\sum_j w_j} \sum_{k=1}^{n} \left[\left(\sum_{j=1}^{m} w_j \cos k\phi_j \right)^2 + \left(\sum_{j=1}^{m} w_j \sin k\phi_j \right)^2 \right] \quad (4)$$

Since the sinusoids are only executed on a small number of bins, while the epoch folding procedure just consists of a very fast histogram-like operation, the speedup of this new formula is obvious. Care must be put into the choice of the number of bins, in order to maintain a good approximation even when the number of harmonics is high. We recommend in the documentation to use a number of bins at least 10 times larger than the number of harmonics.

7.3. *Characterization of pulsar behavior*

As seen in Section 7.2, the Z_n^2 or the EF periodograms of a perfectly stable pulsation have the shape of a sinc^2 function. `stingray` has functionality to fit these periodograms with a sinc^2 funtion or alternatively a Gaussian model, and find the mean frequency with high precision.

A significant deviation from the expected shape from these models can happen if the pulsation is not stable. Calculating the *phaseogram* (Figure 5) is an option to investigate how the pulse phase varies in time. The phaseogram in this context consists of a 2D histogram of the phase and arrival times of the pulses. If the pulsation is stable and the pulse frequency was determined with precision, the phaseogram shows vertical stripes corresponding to perfectly aligned pulses. If the frequency is not as precise, the stripes become more and more diagonal. If the pulse has a detectable frequency derivative, these stripes bend with a parabolic shape. If the orbital solution is imperfect, the stripes show specific periodic features [9].

A very precise way to determine the exact pulse ephemeris is out of the scope of `stingray`. Nonetheless, `stingray` has a mechanism to calculate the pulse arrival times (or times of arrival, TOAs) to be analyzed with more specialized software like Tempo, Tempo2 or PINT. We use the same `fftfit` algorithm used for radio pulsars (Taylor 1992), that calculates the cross-correlation between a template profile and the folded profile in the Fourier domain. This is implemented in the `get_TOA` function in `stingray.pulse.pulsar`.

The functionality to plot the phaseogram, interactively change the timing parameters (either pulse parameters or orbital parameters) and adjusting the solution, and calculating the TOAs for use with external programs, is conveniently accessible in HENDRICS (See Section 9) and Figure 5 and in DAVE (Section 10).

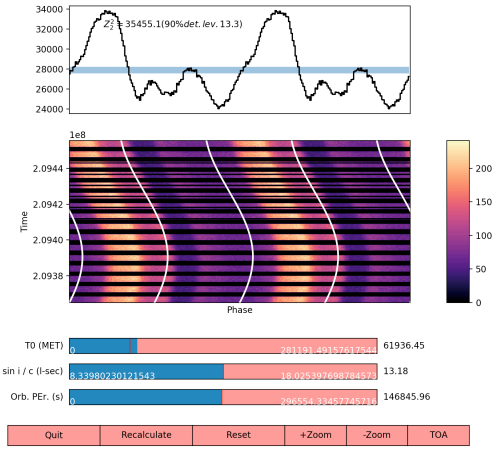


Figure 5. Phaseogram showing the variation of the pulse phase corresponding to an imperfect orbital solution (in this case the time at the ascending node T_0) in a *NuSTAR* observation of Her X-1, executed with Stingray and plotted in a convenient, interactive interface with HENDRICS. The TOA button allows to calculate the TOA for use with Tempo2, PINT or similar programs.

8. DEVELOPMENT AND INTEGRATION ENVIRONMENT

`stingray` is developed entirely in Python3, with backwards compatibility to Python 2.7 where possible through the integration package `six`. Development is version-controlled

[9] See for example https://github.com/matteobachetti/timing-lectures/blob/master/no-binder/Timing_residuals.ipynb

through `git`, and officially hosted through an organization on *GitHub*, where several interconnected repositories related to `stingray` live, including the core library, extension packages `HENDRICS` and `DAVE`, as well as the suite of tutorials and this manuscript. All patches and code are submitted via pull requests to the `stingray` repository, and checked by a maintainer for correctness and adherence to standards of code, documentation and tests. As an *Astropy* Affiliated Package[10], we follow the coding standards as well as community guidelines (including the Code of Conduct) set out by the *Astropy* community. All code within the `stingray` core library is subject to extensive unit testing, with compatibility across platforms as well as different versions of Python and required packages controlled through Continuous Integration services *Travis* (Unix platforms) and *AppVeyor* (Windows). Test coverage is checked using *Coveralls*. All user-facing functions and classes within `stingray` must have documentation in the form of docstrings, compiled and built along with the main documentation pages using *sphinx* and hosted on `readthedocs`[11]. Tutorials are provided in the form of executable *Jupyter* notebooks in a separate repository[12], which can either be run interactively using *Binder* (Project Jupyter et al. 2018) or viewed as part of the documentation.

9. `HENDRICS`: A COMMAND-LINE INTERFACE FOR `stingray`

The `HENDRICS` package[13]—formerly called `MaLTPyNT` (Bachetti 2015b)—builds upon `stingray` by providing a suite of easy-to-execute command-line scripts whose primary use is providing an accurate quick-look (spectral-)timing analysis of X-ray observations, useful for a range of use cases, including exploratory data analysis and quality assessment of larger data analysis pipelines. While its initial development proceeded independently from `stingray`, much of its core functionality since version 3.0 is based on the classes and methods `stingray` provides, and some key functionality has been shifted to `stingray` where appropriate.

Its key distinguishing feature from established command-line interfaces such as FTOOLS is the accurate treatment of gaps in the data (for example due to the Earth's occultation or the South Atlantic Anomaly), as well as its treatment of dead time for certain detectors like *NuSTAR* . Where `stingray` aims to provide flexible building blocks for designing sophisticated spectral-timing analysis workflows, `HENDRICS` provides end-to-end solution for common tasks such as power density and cross spectra, time lags, pulsar searches, colour-colour as well as colour-intensity diagrams, at the cost of loosing some flexibility during the creation of those products. Like `stingray`, `HENDRICS` is an `astropy` affiliated package and aims to build upon and be compatible with functionality provided as part of the `astropy` ecosystem. `HENDRICS` supports a range of output data formats including netCDF4 and ASCII formats, which can then be read into other astronomical data analysis systems such as XSPEC (Arnaud 1996) or ISIS (Houck & Denicola 2000).

`HENDRICS` is in release version 4 as of 2018-02-12, and under active development, utilizing the same continuous integration, testing and code review standards as `stingray`.

10. `DAVE`: EXPLORATORY DATA ANALYSIS IN A GRAPHICAL USER INTERFACE

`DAVE`[14]—the Data Analysis for Variable Events package—is a Graphical User Interface built on top of `stingray` in order to provide users with interactive capabilities for exploratory data analysis of variable time series. Much of the core functionality within `stingray` is available in `DAVE` as well: creation of power spectra, cross spectra, dynamic power spectra and spectral-timing products such as time lags, coherence measurements. In addition, it implements interactive filtering of light curves with respect to energy channels or energies (if a response matrix file is loaded), time ranges, and count rates. Users may compare light curves and power spectra from different energy ranges, and may create auxiliary products such as colour-colour and colour-intensity diagrams that further aid the exploration of the data. An example of the interface is shown in Figure 6. The full interface and its capabilities will be described in a forthcoming publication.

11. FUTURE DEVELOPMENT PLANS

Future plans for `stingray` development are largely aimed at extending current functionality related to Fourier spectra (with e.g. the planned implementation of the statistical distributions relevant to co-spectra; Huppenkothen & Bachetti 2017), and continuing work towards comprehensive spectra-timing capabilities. While reference implementations of higher-order Fourier products such as bispectra, biphase and bicoherence (Maccarone & Schnittman 2005; Maccarone 2013) exist, they require additional extensions to be useful for science. New key features in the next version of `stingray`, based on an existing reference implementation of covariance spectra (Wilkinson & Uttley 2009), will include lag-energy spectra (Vaughan et al. 1994), RMS-energy spectra (Revnivtsev et al. 1999), and excess variance spectra (Vaughan et al. 2003). In addition, while at the moment, there is rudimentary functionality to build spectra-timing products, it is currently not possible to seamlessly work with these products using `stingray`, because `stingray` currently has no native ca-

[10] http://www.astropy.org/affiliated/
[11] https://stingray.readthedocs.io/en/latest/
[12] https://github.com/stingraySoftware/notebooks
[13] https://github.com/stingraySoftware/hendrics
[14] https://github.com/stingraySoftware/dave

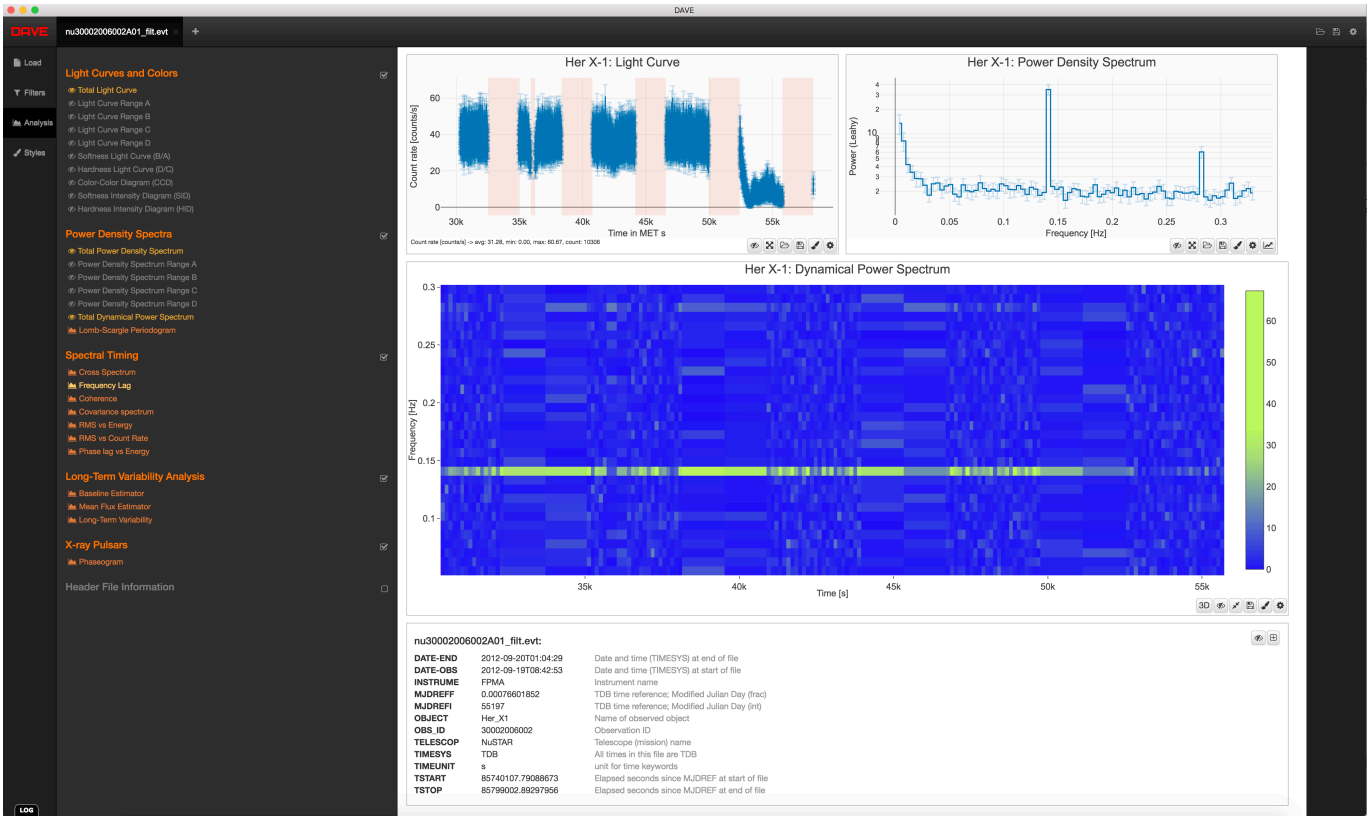


Figure 6. An example of the DAVE graphical user interface for the Her X-1 pulsar data observed with *NuSTAR* : At the top left, the last 30ks of the pulsar light curve with the GTIs clearly marked. In the top right, the averaged power spectrum generated from 109 segments of 256 s duration with a binned time resolution of 1.5 s. In the middle, we present the dynamic power spectrum generated from the same 256 s segments that generated the top right averaged power spectrum. Below, header meta-data is shown for reference. On the left, the menu presents a range of options of figures to plot and compare, including spectral-timing capabilities. All figures are interactive, including panning and zooming, as well as interactive choices of data selection.

pability for energy-spectral modelling. Instead, they would have to be exported (e.g. saved to disk) and then loaded into another software, significantly disrupting workflows and pipeline development. In order to streamline this process, we aim to connect `stingray` with existing packages for modelling X-ray spectra. Here, it will be necessary to connect `stingray` with the extensive suite of physical models implemented in XSPEC, as well as existing spectral fitting codes implemented in *Python*, most notably the open-source package *Sherpa* (Burke et al. 2018).

Data rates from current and future X-ray instruments are increasing at a precipitous rate, pushing memory and processing requirements for even simple tasks like Fast Fourier Transforms of data observed e.g. with *NICER* and *AstroSAT* (Singh et al. 2014) into a regime that is difficult with standard desktop computing architectures. Therefore, the other strong emphasis for the second version of `stingray` will be code and algorithm optimization. Where possible, we will replace existing implementations by high-performance equivalents that take advantage of recent developments in computing (such as GPU-enabled computations and multi-core batch processing), optimize and streamline existing code to minimize computational overhead and memory usage of the classes and functions implemented within `stingray`.

DH acknowledges support from the DIRAC Institute in the Department of Astronomy at the University of Washington. The DIRAC Institute is supported through generous gifts from the Charles and Lisa Simonyi Fund for Arts and Sciences, and the Washington Research Foundation. MB is supported in part by the Italian Space Agency through agreement ASI-INAF n.2017-12-H.0 and ASI-INFN agreement n.2017-13-H.0. We thank Astro Hack Week for providing the venue that started this project and the Lorentz Centre for organizing the workshop that started the collaboration. We thank the Google Summer of Code Program for funding a total 6 students who implemented a large fraction of the various library components over three summers.

REFERENCES

Akaike, H. 1974, IEEE Transactions on Automatic Control, 19, 716, doi: 10.1109/TAC.1974.1100705

Anderson, S. F., Wachter, S., Margon, B., et al. 1994, ApJ, 436, 319, doi: 10.1086/174907

Arnaud, K. A. 1996, in Astronomical Society of the Pacific Conference Series, Vol. 101, Astronomical Data Analysis Software and Systems V, ed. G. H. Jacoby & J. Barnes, 17

Bachetti, M. 2015a, MaLTPyNT: Quick look timing analysis for NuSTAR data, Astrophysics Source Code Library. http://ascl.net/1502.021

—. 2015b, MaLTPyNT: Quick look timing analysis for NuSTAR data, Astrophysics Source Code Library. http://ascl.net/1502.021

Bachetti, M., Harrison, F. A., Cook, R., et al. 2015, ApJ, 800, 109

Bahcall, J. N., & Bahcall, N. A. 1972, ApJL, 178, L1, doi: 10.1086/181070

Belloni, T., & Hasinger, G. 1990, A&A, 227, L33

Bentz, M. C. 2016, AGN Reverberation Mapping, ed. H. M. J. Boffin, G. Hussain, J.-P. Berger, & L. Schmidtobreick (Cham: Springer International Publishing), 249–266. https://doi.org/10.1007/978-3-319-39739-9_13

Blandford, R. D., & McKee, C. F. ????, 255, 419, doi: 10.1086/159843

Bradt, H. V., Rothschild, R. E., & Swank, J. H. 1993, A&AS, 97, 355

Buccheri, R., Bennett, K., Bignami, G. F., et al. 1983, A&A, 128, 245

Burke, D., Laurino, O., dtnguyen2, et al. 2018, sherpa/sherpa: Sherpa 4.10.1, doi: 10.5281/zenodo.1463962. https://doi.org/10.5281/zenodo.1463962

Carpenter, B., Gelman, A., Hoffman, M., et al. 2017, Journal of Statistical Software, Articles, 76, 1, doi: 10.18637/jss.v076.i01

Cash, W. 1979, ApJ, 228, 939, doi: 10.1086/156922

Charbonneau, D., Brown, T. M., Latham, D. W., & Mayor, M. 2000, ApJL, 529, L45, doi: 10.1086/312457

Cheng, F. H., Vrtilek, S. D., & Raymond, J. C. 1995, ApJ, 452, 825, doi: 10.1086/176351

Coughlin, J. L., Mullally, F., Thompson, S. E., et al. 2016, ApJS, 224, 12, doi: 10.3847/0067-0049/224/1/12

Davidsen, A., Henry, J. P., Middleditch, J., & Smith, H. E. 1972, ApJL, 177, L97, doi: 10.1086/181060

Foreman-Mackey, D. 2016, The Journal of Open Source Software, 24, doi: 10.21105/joss.00024

Foreman-Mackey, D., Hogg, D. W., Lang, D., & Goodman, J. 2013, PASP, 125, 306, doi: 10.1086/670067

Forman, W., Jones, C. A., & Liller, W. 1972, ApJL, 177, L103, doi: 10.1086/181061

Fürst, F., Grefenstette, B. W., Staubert, R., et al. 2013, The Astrophysical Journal, 779, 69

Gelman, A., & Rubin, D. B. 1992, Statist. Sci., 7, 457, doi: 10.1214/ss/1177011136

Gendreau, K. C., Arzoumanian, Z., Adkins, P. W., et al. 2016, in Proc. SPIE, Vol. 9905, Space Telescopes and Instrumentation 2016: Ultraviolet to Gamma Ray, 99051H

Giacconi, R., Gursky, H., Kellogg, E., et al. 1973, ApJ, 184, 227, doi: 10.1086/152321

Girod, B., Rabenstein, R., & Stenger, A. 2001, Signals and systems (Wiley). https://books.google.com/books?id=DoseAQAAIAAJ

Groth, E. J. 1975, ApJS, 29, 285, doi: 10.1086/190343

Harrison, F. A., Craig, W. W., Christensen, F. E., et al. 2013, ApJ, 770, 103

Heida, M., Jonker, P. G., Torres, M. A. P., & Chiavassa, A. 2017, ApJ, 846, 132, doi: 10.3847/1538-4357/aa85df

Henry, G. W., Marcy, G. W., Butler, R. P., & Vogt, S. S. 2000, ApJL, 529, L41, doi: 10.1086/312458

Hewish, A., Bell, S. J., Pilkington, J. D. H., Scott, P. F., & Collins, R. A. 1968, Nature, 217, 709, doi: 10.1038/217709a0

Houck, J. C., & Denicola, L. A. 2000, in Astronomical Society of the Pacific Conference Series, Vol. 216, Astronomical Data Analysis Software and Systems IX, ed. N. Manset, C. Veillet, & D. Crabtree, 591

Hunter, J. D. 2007, Computing in Science Engineering, 9, 90, doi: 10.1109/MCSE.2007.55

Huppenkothen, D., & Bachetti, M. 2017, ArXiv e-prints. https://arxiv.org/abs/1709.09666

Hynes, R. I., Steeghs, D., Casares, J., Charles, P. A., & O'Brien, K. 2003, ApJL, 583, L95, doi: 10.1086/368108

Igna, C. D., & Leahy, D. A. 2011, MNRAS, 418, 2283, doi: 10.1111/j.1365-2966.2011.19550.x

Ingram, A., & van der Klis, M. 2015, MNRAS, 446, 3516, doi: 10.1093/mnras/stu2373

Ingram, A., van der Klis, M., Middleton, M., et al. 2016, MNRAS, 461, 1967, doi: 10.1093/mnras/stw1245

Jahoda, K., Swank, J. H., Giles, A. B., et al. 1996, in Proc. SPIE, Vol. 2808, EUV, X-Ray, and Gamma-Ray Instrumentation for Astronomy VII, ed. O. H. Siegmund & M. A. Gummin, 59–70

Jones, E., Oliphant, T., Peterson, P., et al. 2001–, SciPy: Open source scientific tools for Python. http://www.scipy.org/

Lam, S. K., Pitrou, A., & Seibert, S. 2015, in Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, LLVM '15 (New York, NY, USA: ACM), 7:1–7:6. http://doi.acm.org/10.1145/2833157.2833162

Leahy, D. A. 1987, A&A, 180, 275

Leahy, D. A., & Abdallah, M. H. 2014, ApJ, 793, 79, doi: 10.1088/0004-637X/793/2/79

Leahy, D. A., Darbro, W., Elsner, R. F., et al. 1983a, ApJ, 266, 160, doi: 10.1086/160766

Leahy, D. A., Elsner, R. F., & Weisskopf, M. C. 1983b, ApJ, 272, 256, doi: 10.1086/161288

Ludlam, R. M., Miller, J. M., & Cackett, E. M. 2015, ApJ, 806, 262, doi: 10.1088/0004-637X/806/2/262

Maccarone, T. J. 2013, MNRAS, 435, 3547, doi: 10.1093/mnras/stt1546

Maccarone, T. J., & Schnittman, J. D. 2005, MNRAS, 357, 12, doi: 10.1111/j.1365-2966.2004.08615.x

Miyamoto, S., Kitamoto, S., Iga, S., Negoro, H., & Terada, K. 1992, ApJL, 391, L21, doi: 10.1086/186389

Muñoz-Darias, T., Casares, J., & Martínez-Pais, I. G. 2008, MNRAS, 385, 2205, doi: 10.1111/j.1365-2966.2008.12987.x

Nowak, M. A., Dove, J. B., Vaughan, B. A., Wilms, J., & Begelman, M. C. 1999, Nuclear Physics B Proceedings Supplements, 69, 302, doi: 10.1016/S0920-5632(98)00229-1

Project Jupyter, Matthias Bussonnier, Jessica Forde, et al. 2018, in Proceedings of the 17th Python in Science Conference, ed. Fatih Akici, David Lippa, Dillon Niederhut, & M. Pacer, 113 – 120

Protassov, R., van Dyk, D. A., Connors, A., Kashyap, V. L., & Siemiginowska, A. 2002, ApJ, 571, 545, doi: 10.1086/339856

Revnivtsev, M., Gilfanov, M., & Churazov, E. 1999, A&A, 347, L23. https://arxiv.org/abs/astro-ph/9906198

Reynolds, A. P., Quaintrell, H., Still, M. D., et al. 1997, MNRAS, 288, 43, doi: 10.1093/mnras/288.1.43

Schwarz, G. 1978, Ann. Statist., 6, 461, doi: 10.1214/aos/1176344136

Scott, D. M., & Leahy, D. A. 1999, ApJ, 510, 974, doi: 10.1086/306631

Singh, K. P., Tandon, S. N., Agrawal, P. C., et al. 2014, in Proc. SPIE, Vol. 9144, Space Telescopes and Instrumentation 2014: Ultraviolet to Gamma Ray, 91441S

Staubert, R., Klochkov, D., Fürst, F., et al. 2017, Astronomy & Astrophysics, 606, L13, doi: 10.1051/0004-6361/201731927

Tananbaum, H., Gursky, H., Kellogg, E. M., et al. 1972, ApJL, 174, L143, doi: 10.1086/180968

Taylor, J. H. 1992, Philosophical Transactions: Physical Sciences and Engineering, 341, 117

The Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., et al. 2018, ArXiv e-prints. https://arxiv.org/abs/1801.02634

Timmer, J., & Koenig, M. 1995, A&A, 300, 707

Uttley, P., Cackett, E. M., Fabian, A. C., Kara, E., & Wilkins, D. R. 2014, A&A Rv, 22, 72, doi: 10.1007/s00159-014-0072-0

Uttley, P., & McHardy, I. M. 2001, MNRAS, 323, L26, doi: 10.1046/j.1365-8711.2001.04496.x

van der Klis, M. 1989, in NATO Advanced Science Institutes (ASI) Series C, Vol. 262, NATO Advanced Science Institutes (ASI) Series C, ed. H. Ögelman & E. P. J. van den Heuvel, 27

van der Walt, S., Colbert, S. C., & Varoquaux, G. 2011, Computing in Science Engineering, 13, 22, doi: 10.1109/MCSE.2011.37

Vaughan, B., van der Klis, M., Lewin, W. H. G., et al. 1994, ApJ, 421, 738, doi: 10.1086/173686

Vaughan, B. A., & Nowak, M. A. 1997, ApJL, 474, L43, doi: 10.1086/310430

Vaughan, S. 2010, MNRAS, 402, 307, doi: 10.1111/j.1365-2966.2009.15868.x

Vaughan, S., Edelson, R., Warwick, R. S., & Uttley, P. 2003, MNRAS, 345, 1271, doi: 10.1046/j.1365-2966.2003.07042.x

Wang-Ji, J., García, J. A., Steiner, J. F., et al. 2018, ApJ, 855, 61, doi: 10.3847/1538-4357/aaa974

Wilkinson, T., & Uttley, P. 2009, MNRAS, 397, 666, doi: 10.1111/j.1365-2966.2009.15008.x

Yamaoka, K., Sugizaki, M., Nakahira, S., et al. 2010, The Astronomer's Telegram, 2380

Zdziarski, A. A., Poutanen, J., Mikolajewska, J., et al. 1998, MNRAS, 301, 435, doi: 10.1046/j.1365-8711.1998.02021.x