stingray Documentation

Release 0.1.dev226

Stingray Developers

April 05, 2016

Contents

Part I

User Documentation

Stingray at a glace

CHAPTER 1

Overview

CHAPTER 2

Installation

CHAPTER 3

Getting Started

Core modules

CHAPTER 4

Lightcurve

Definition of `Lightcurve`.

`Lightcurve` is used to create light curves out of photon counting data or to save existing light curves in a class that's easy to use.

class `stingray.lightcurve.`**`Lightcurve`**(*time*, *counts*, *input_counts=True*)

Make a light curve object from an array of time stamps and an array of counts.

Parameters

time: iterable

A list or array of time stamps for a light curve

counts: iterable, optional, default None

A list or array of the counts in each bin corresponding to the bins defined in `time` (note: **not** the count rate, i.e. counts/second, but the counts/bin).

input_counts: bool, optional, default True

If True, the code assumes that the input data in 'counts' is in units of counts/bin. If False, it assumes the data in 'counts' is in counts/second.

Attributes

time: numpy.ndarray	The array of midpoints of time bins
counts: numpy.ndarray	The counts per bin corresponding to the bins in `time`.
countrate: numpy.ndarray	The counts per second in each of the bins defined in `time`.
ncounts: int	The number of data points in the light curve.
dt: float	The time resolution of the light curve.
tseg: float	The total duration of the light curve.
tstart: float	The start time of the light curve.

static **`make_lightcurve`**(*toa*, *dt*, *tseg=None*, *tstart=None*)

Make a light curve out of photon arrival times.

Parameters

toa: iterable

list of photon arrival times

dt: float

time resolution of the light curve (the bin width)

tseg: float, optional, default None

The total duration of the light curve. If this is `None`, then the total duration of the light curve will be the interval between the arrival between the first and the last photon in `toa`.

Note: If tseg is not divisible by dt (i.e. if tseg/dt is not an integer number), then the last fractional bin will be dropped!

tstart: float, optional, default None

The start time of the light curve. If this is None, the arrival time of the first photon will be used as the start time of the light curve.

Returns

lc: `Lightcurve` object

A light curve object with the binned light curve

rebin_lightcurve(*dt_new*, *method='sum'*)

Rebin the light curve to a new time resolution. While the new resolution need not be an integer multiple of the previous time resolution, be aware that if it is not, the last bin will be cut off by the fraction left over by the integer division.

Parameters

dt_new: float

The new time resolution of the light curve. Must be larger than the time resolution of the old light curve!

method: {"sum" | "mean" | "average"}, optional, default "sum"

This keyword argument sets whether the counts in the new bins should be summed or averaged.

Returns

lc_new: `Lightcurve` object

The `Lightcurve` object with the new, binned light curve.

CHAPTER 5

Powerspectrum

class stingray.powerspectrum.**Powerspectrum**(*lc=None*, *norm='rms'*)

Make a Periodogram (power spectrum) from a (binned) light curve. Periodograms can be Leahy normalized or fractional rms normalized. You can also make an empty Periodogram object to populate with your own fourier-transformed data (this can sometimes be useful when making binned periodograms).

Parameters

lc: lightcurve.Lightcurve object, optional, default None

The light curve data to be Fourier-transformed.

norm: {"leahy" | "rms"}, optional, default "rms"

The normaliation of the periodogram to be used. Options are "leahy" or "rms", default is "rms".

Attributes

norm: {"leahy" | "rms"}	the normalization of the periodogram
freq: numpy.ndarray	The array of mid-bin frequencies that the Fourier transform samples
ps: numpy.ndarray	The array of normalized squared absolute values of Fourier amplitudes
df: float	The frequency resolution
m: int	The number of averaged powers in each bin
n: int	The number of data points in the light curve
nphots: float	The total number of photons in the light curve

classical_significances(*threshold=1*, *trial_correction=False*)

Compute the classical significances for the powers in the power spectrum, assuming an underlying noise distribution that follows a chi-square distributions with 2M degrees of freedom, where M is the number of powers averaged in each bin.

Note that this function will *only* produce correct results when the following underlying assumptions are fulfilled: (1) The power spectrum is Leahy-normalized (2) There is no source of variability in the data other than the periodic signal to be determined with this method. This is important! If there are other sources of (aperiodic) variability in the data, this method will *not* produce correct results, but instead produce a large number of spurious false positive detections! (3) There are no significant instrumental effects changing the statistical distribution of the powers (e.g. pile-up or dead time)

By default, the method produces (index,p-values) for all powers in the power spectrum, where index is the numerical index of the power in question. If a `threshold` is set, then only powers with p-values *below* that threshold with their respective indices. If `trial_correction` is set to True, then the threshold will be corrected for the number of trials (frequencies) in the power spectrum before being used.

Parameters
threshold : float

The threshold to be used when reporting p-values of potentially significant powers. Must be between 0 and 1. Default is 1 (all p-values will be reported).

trial_correction : bool

A Boolean flag that sets whether the `threshold` will be correted by the number of frequencies before being applied. This decreases the threshold (p-values need to be lower to count as significant). Default is False (report all powers) though for any application where `threshold` is set to something meaningful, this should also be applied!

Returns
pvals : iterable

A list of (index, p-value) tuples for all powers that have p-values lower than the threshold specified in `threshold`.

`compute_rms`(*min_freq*, *max_freq*)
Compute the fractional rms amplitude in the periodgram between two frequencies.

Parameters
min_freq: float

The lower frequency bound for the calculation

max_freq: float

The upper frequency bound for the calculation

Returns
rms: float

The fractional rms amplitude contained between min_freq and max_freq

`rebin`(*df*, *method='mean'*)
Rebin the periodogram to a new frequency resolution df.

Parameters
df: float

The new frequency resolution

Returns
bin_ps = Periodogram object

The newly binned periodogram

`rebin_log`(*f=0.01*)
Logarithmic rebin of the periodogram. The new frequency depends on the previous frequency modified by a factor f:

dnu_j = dnu_{j-1}*(1+f)

Parameters
f: float, optional, default 0.01

parameter that steers the frequency resolution

Returns
binfreq: numpy.ndarray

the binned frequencies

binps: numpy.ndarray

the binned powers

nsamples: numpy.ndarray

the samples of the original periodogramincluded in each frequency bin

class stingray.powerspectrum.**AveragedPowerspectrum**(*lc*, *segment_size*, *norm='rms'*)

Make an averaged periodogram from a light curve by segmenting the light curve, Fourier-transforming each segment and then averaging the resulting periodograms.

Parameters

lc: lightcurve.Lightcurve object OR

iterable of lightcurve.Lightcurve objects The light curve data to be Fourier-transformed.

segment_size: float

The size of each segment to average. Note that if the total duration of each Lightcurve object in lc is not an integer multiple of the segment_size, then any fraction left-over at the end of the time series will be lost.

norm: {"leahy" | "rms"}, optional, default "rms"

The normaliation of the periodogram to be used. Options are "leahy" or "rms", default is "rms".

Attributes

norm: {"leahy" | "rms"}	the normalization of the periodogram
freq: numpy.ndarray	The array of mid-bin frequencies that the Fourier transform samples
ps: numpy.ndarray	The array of normalized squared absolute values of Fourier amplitudes
df: float	The frequency resolution
m: int	The number of averaged periodograms
n: int	The number of data points in the light curve
nphots: float	The total number of photons in the light curve

CHAPTER 6

Pulsar

Basic pulsar-related functions and statistics.

stingray.pulse.pulsar.**fftfit**(*prof*, *template=None*, ***fftfit_kwargs*)

Align a template to a pulse profile.

Parameters

phase : array

The phases corresponding to each bin of the profile

prof : array

The pulse profile

template : array, default None

The template of the pulse used to perform the TOA calculation. If None, a simple sinusoid is used

Returns

mean_amp, std_amp : floats

Mean and standard deviation of the amplitude

mean_phase, std_phase : floats

Mean and standard deviation of the phase

Other Parameters

fftfit_kwargs : arguments

Additional arguments to be passed to error calculation

stingray.pulse.pulsar.**fftfit_error**(*phase*, *prof*, *template*, *p0*, ***fftfit_kwargs*)

Calculate the error on the fit parameters from FFTFIT.

Parameters

phase : array

The phases corresponding to each bin of the profile

prof : array

The pulse profile

template : array

The template of the pulse used to perform the TOA calculation

p0 : list

The initial parameters for the fit

Returns

mean_amp, std_amp : floats

Mean and standard deviation of the amplitude

mean_phase, std_phase : floats

Mean and standard deviation of the phase

Other Parameters

nstep : int, optional, default 100

Number of steps for the bootstrap method

stingray.pulse.pulsar.**fftfit_fun**(*profile*, *template*, *amplitude*, *phase*)

Function to be minimized for the FFTFIT method.

From Taylor (1992).

Parameters

profile : array

The pulse profile

template : array

A pulse shape template, of the same length as profile.

amplitude, phase : float

The amplitude and phase of the template w.r.t the real profile.

Returns

fftfit_chisq : float

The chi square-like statistics of FFTFIT

stingray.pulse.pulsar.**fold_detection_level**(*nbin*, *epsilon=0.01*, *ntrial=1*)

Return the detection level for a folded profile.

See Leahy et al. (1983).

Parameters

nbin : int

The number of bins in the profile

epsilon : float, default 0.01

The fractional probability that the signal has been produced by noise

Returns

detlev : float

The epoch folding statistics corresponding to a probability epsilon * 100 % that the signal has been produced by noise

Other Parameters

ntrial : int

The number of trials executed to find this profile

stingray.pulse.pulsar.**fold_events**(*times*, **frequency_derivatives*, ***opts*)

Epoch folding with exposure correction.

Parameters

times : array of floats

f, fdot, fddot... : float

The frequency and any number of derivatives.

Returns

phase_bins : array of floats

The phases corresponding to the pulse profile

profile : array of floats

The pulse profile

profile_err : array of floats

The uncertainties on the pulse profile

Other Parameters

nbin : int, optional, default 16

The number of bins in the pulse profile

weights : float or array of floats, optional

The weights of the data. It can either be specified as a single value for all points, or an array with the same length as `time`

gtis : [[gti0_0, gti0_1], [gti1_0, gti1_1], ...], optional

Good time intervals

ref_time : float, optional, default 0

Reference time for the timing solution

stingray.pulse.pulsar.**fold_profile_probability**(*stat*, *nbin*, *ntrial=1*)

Calculate the probability of a certain folded profile, due to noise.

Parameters

stat : float

The epoch folding statistics

nbin : int

The number of bins in the profile

Returns

p : float

The probability that the profile has been produced by noise

Other Parameters

ntrial : int

The number of trials executed to find this profile

stingray.pulse.pulsar.**get_TOA**(*prof*, *period*, *tstart*, *template=None*, *additional_phase=0*, ***fftfit_kwargs*)

Calculate the Time-Of-Arrival of a pulse.

Parameters

prof : array

The pulse profile

template : array, default None

The template of the pulse used to perform the TOA calculation, if any. Otherwise use the default of fftfit

tstart : float

The time at the start of the pulse profile

Returns

toa, toastd : floats

Mean and standard deviation of the TOA

Other Parameters

nstep : int, optional, default 100

Number of steps for the bootstrap method

stingray.pulse.pulsar.**phase_exposure**(*start_time*, *stop_time*, *period*, *nbin=16*, *gtis=None*)

Calculate the exposure on each phase of a pulse profile.

Parameters

start_time, stop_time : float

Starting and stopping time (or phase if ‘‘period‘‘==1)

period : float

The pulse period (if 1, equivalent to phases)

Returns

expo : array of floats

The normalized exposure of each bin in the pulse profile (1 is the highest exposure, 0 the lowest)

Other Parameters

nbin : int, optional, default 16

The number of bins in the profile

gtis : [[gti0_0, gti0_1], [gti1_0, gti1_1], ...], optional, default None

Good Time Intervals

stingray.pulse.pulsar.**pulse_phase**(*times*, **frequency_derivatives*, ***opts*)

Calculate pulse phase from the frequency and its derivatives.

Parameters

times : array of floats

The times at which the phase is calculated

***frequency_derivatives: floats**

List of derivatives in increasing order, starting from zero.

Returns

phases : array of floats

The absolute pulse phase

Other Parameters

ph0 : float

The starting phase

to_1 : bool, default True

Only return the fractional part of the phase, normalized from 0 to 1

stingray.pulse.pulsar.**stat**(*profile*, *err=None*)

Calculate the epoch folding statistics 'a la Leahy et al. (1983).

Parameters

profile : array

The pulse profile

Returns

stat : float

The epoch folding statistics

Other Parameters

err : float or array

The uncertainties on the pulse profile

stingray.pulse.pulsar.**z2_2_probability**(*z2*, *n=2*, *ntrial=1*)

Calculate the probability of a certain folded profile, due to noise.

Parameters

z2 : float

A Z^2_n statistics value

n : int, default 2

The n in Z^2_n

Returns

p : float

The probability that the Z^2_n value has been produced by noise

Other Parameters

ntrial : int

The number of trials executed to find this profile

stingray.pulse.pulsar.**z2_n_detection_level**(*n=2*, *epsilon=0.01*, *ntrial=1*)

Return the detection level for the Z^2_n statistics.

See Buccheri et al. (1983), Bendat and Piersol (1971).

Parameters

n : int, default 2

The n in Z^2_n

epsilon : float, default 0.01

The fractional probability that the signal has been produced by noise

Returns

detlev : float

The epoch folding statistics corresponding to a probability epsilon * 100 % that the signal has been produced by noise

Other Parameters

ntrial : int

The number of trials executed to find this profile

stingray.pulse.pulsar.**z_n**(*phase*, *n=2*, *norm=1*)

Z^2_n statistics, a‘ la Buccheri+03, A&A, 128, 245, eq. 2.

Parameters

phase : array of floats

The phases of the events

n : int, default 2

The n in Z^2_n.

Returns

z2_n : float

The Z^2_n statistics of the events.

Other Parameters

norm : float or array of floats

A normalization factor that gets multiplied as a weight.

Utilities

CHAPTER 7

io

stingray.io.**check_gtis**(*gti*)

Check if GTIs are well-behaved. No start>end, no overlaps.

Raises

AssertionError

If GTIs are not well-behaved.

stingray.io.**common_name**(*str1*, *str2*, *default=u'common'*)

Strip two strings of the letters not in common.

Filenames must be of same length and only differ by a few letters.

Parameters

str1 : str

str2 : str

Returns

common_str : str

A string containing the parts of the two names in common

Other Parameters

default : str

The string to return if common_str is empty

stingray.io.**contiguous_regions**(*condition*)

Find contiguous True regions of the boolean array "condition".

Return a 2D array where the first column is the start index of the region and the second column is the end index.

Parameters

condition : boolean array

Returns

idx : [[i0_0, i0_1], [i1_0, i1_1], ...]

A list of integer couples, with the start and end of each True blocks in the original array

Notes

From http://stackoverflow.com/questions/4494404/find-large-number-of-consecutive-values-fulfilling-condition-in-a-numpy-array

stingray.io.**create_gti_from_condition**(*time*, *condition*, *safe_interval=0*, *dt=None*)
Create a GTI list from a time array and a boolean mask ("condition").

Parameters
time : array-like

Array containing times

condition : array-like

An array of bools, of the same length of time. A possible condition can be, e.g., the result of lc > 0.

Returns
gtis : [[gti0_0, gti0_1], [gti1_0, gti1_1], ...]

The newly created GTIs

Other Parameters
safe_interval : float or [float, float]

A safe interval to exclude at both ends (if single float) or the start and the end (if pair of values) of GTIs.

dt : float

The width (in sec) of each bin of the time array. Can be irregular.

stingray.io.**create_gti_mask**(*time*, *gtis*, *safe_interval=0*, *min_length=0*, *return_new_gtis=False*, *dt=None*)
Create GTI mask.

Assumes that no overlaps are present between GTIs

Parameters
time : float array

gtis : [[g0_0, g0_1], [g1_0, g1_1], ...], float array-like

Returns
mask : boolean array

new_gtis : Nx2 array

Other Parameters
safe_interval : float or [float, float]

A safe interval to exclude at both ends (if single float) or the start and the end (if pair of values) of GTIs.

min_length : float

return_new_gtis : bool

dt : float

stingray.io.**cross_gtis**(*gti_list*)
From multiple GTI lists, extract the common intervals *EXACTLY*.

Parameters
gti_list : array-like

List of GTI arrays, each one in the usual format [[gti0_0, gti0_1], [gti1_0, gti1_1], ...]

Returns
gtis : [[gti0_0, gti0_1], [gti1_0, gti1_1], ...]

The newly created GTIs

See also:

`cross_two_gtis`
Extract the common intervals from two GTI lists *EXACTLY*

stingray.io.**cross_two_gtis**(*gti0*, *gti1*)
Extract the common intervals from two GTI lists *EXACTLY*.

Parameters
gti0 : [[gti0_0, gti0_1], [gti1_0, gti1_1], ...]

gti1 : [[gti0_0, gti0_1], [gti1_0, gti1_1], ...]

Returns
gtis : [[gti0_0, gti0_1], [gti1_0, gti1_1], ...]

The newly created GTIs

See also:

`cross_gtis`
From multiple GTI lists, extract common intervals *EXACTLY*

stingray.io.**get_btis**(*gtis*, *start_time=None*, *stop_time=None*)
From GTIs, obtain bad time intervals.

GTIs have to be well-behaved, in the sense that they have to pass `check_gtis`.

stingray.io.**get_file_extension**(*fname*)
Get the extension from the file name.

stingray.io.**gti_len**(*gti*)
Return the total good time from a list of GTIs.

stingray.io.**high_precision_keyword_read**(*hdr*, *keyword*)
Read FITS header keywords, also if split in two.

In the case where the keyword is split in two, like

MJDREF = MJDREFI + MJDREFF

in some missions, this function returns the summed value. Otherwise, the content of the single keyword

Parameters
hdr : dict_like

The FITS header structure, or a dictionary

keyword : str

The key to read in the header

Returns
value : long double

The value of the key, or None if something went wrong

stingray.io.**load_events_and_gtis**(*fits_file*, *additional_columns=None*, *gtistring=u'GTI*, *STDGTI'*, *gti_file=None*, *hduname=u'EVENTS'*, *column=u'TIME'*)
Load event lists and GTIs from one or more files.

Loads event list from HDU EVENTS of file fits_file, with Good Time intervals. Optionally, returns additional columns of data from the same HDU of the events.

Parameters
fits_file : str

return_limits: bool, optional

Return the TSTART and TSTOP keyword values

additional_columns: list of str, optional

A list of keys corresponding to the additional columns to extract from the event HDU (ex.: ['PI', 'X'])

Returns
ev_list : array-like

gtis: [[gti0_0, gti0_1], [gti1_0, gti1_1], ...]

additional_data: dict

A dictionary, where each key is the one specified in additional_colums. The data are an array with the values of the specified column in the fits file.

t_start : float

t_stop : float

stingray.io.**load_gtis**(*fits_file*, *gtistring=None*)
Load GTI from HDU EVENTS of file fits_file.

stingray.io.**mkdir_p**(*path*)
Safe mkdir function.

Parameters
path : str

Name of the directory/ies to create

Notes

Found at http://stackoverflow.com/questions/600268/mkdir-p-functionality-in-python

stingray.io.**read_header_key**(*fits_file*, *key*, *hdu=1*)
Read the header key key from HDU hdu of the file fits_file.

Parameters
fits_file: str

key: str

The keyword to be read

Other Parameters
hdu : int

stingray.io.**ref_mjd**(*fits_file*, *hdu=1*)
Read MJDREFF+ MJDREFI or, if failed, MJDREF, from the FITS header.

Parameters
fits_file : str

Returns
mjdref : numpy.longdouble

the reference MJD

Other Parameters
hdu : int

CHAPTER 8

sampledata

stingray.sampledata.**sample_data**()

Import data from .txt file and return a light curve object.

Returns

sample: Lightcurve object

The Lightcurve object with the desired time stamps and counts.

CHAPTER 9

utils

stingray.utils.**is_iterable**(*stuff*)
Test if stuff is an iterable.

stingray.utils.**is_string**(*s*)
Portable function to answer this question.

stingray.utils.**jit**(*fun*)

stingray.utils.**optimal_bin_time**(*fftlen*, *tbin*)
Vary slightly the bin time to have a power of two number of bins.

Given an FFT length and a proposed bin time, return a bin time slightly shorter than the original, that will produce a power-of-two number of FFT bins.

stingray.utils.**rebin_data**(*x*, *y*, *dx_new*, *method=u'sum'*)
Rebin some data to an arbitrary new data resolution. Either sum the data points in the new bins or average them.

Parameters

x: iterable

The dependent variable with some resolution dx_old = x[1]-x[0]

y: iterable

The independent variable to be binned

dx_new: float

The new resolution of the dependent variable x

method: {"sum" | "average" | "mean"}, optional, default "sum"

The method to be used in binning. Either sum the samples y in each new bin of x, or take the arithmetic mean.

Returns

xbin: numpy.ndarray

The midpoints of the new bins in x

ybin: numpy.ndarray

The binned quantity y

stingray.utils.**simon**(*message*, ***kwargs*)
The Statistical Interpretation MONitor.

A warning system designed to always remind the user that Simon is watching him/her.

Parameters

message : string

The message that is thrown

kwargs : dict

The rest of the arguments that are passed to warnings.warn

Project details

CHAPTER 10

Credits

CHAPTER 11

Licence

Part II

Getting help

Part III

Reporting Issues

While using Stingray or going through its codebase, if you find any bug please report it. The preferred way to do that is to create a new issue on the Stingray GitHub issue page. You can also the discuss the issue on our mailing list. Please include an example that demonstrates the issue that will allow the developers to reproduce and fix the problem. You may be asked to also provide information about your operating system and a full Python stack trace; the Stingray developers will walk you through obtaining a stack trace if it is necessary.

Part IV

Contributing

CHAPTER 12

Try the development version

CHAPTER 13

How to make a code contribution

Part V

Developer Documentation

CHAPTER 14

Coding Guidelines

CHAPTER 15

Testing Guidelines

Part VI

Indices and tables

- genindex
- modindex
- search

S

Z