MaLTPyNT Documentation

Release 1.0.5

Matteo Bachetti

September 05, 2015

Contents

The MaLTPyNT (Matteo's Libraries and Tools in Python for NuSTAR Timing) suite is designed for the **quick-look timing analysis** of NuSTAR data. It treats properly orbital gaps (e.g., occultation, SAA passages) and performs the standard aperiodic timing analysis (power density spectrum, lags, etc.), plus the **cospectrum**, a proxy for the power density spectrum that uses the signals from two detectors instead of a single one (for an explanation of why this is important in NuSTAR, look at Bachetti et al., *ApJ*, **800**, 109 -arXiv:1409.3248). The output of the analysis, be it a cospectrum, a power density spectrum, or a lag spectrum, can be fitted with Xspec, Isis or any other spectral fitting program.

Despite its main focus on NuSTAR, the software can be used to make standard spectral analysis on X-ray data from, in principle, any other satellite (for sure XMM-Newton and RXTE). Input files can be any event lists in FITS format, provided that they meet certain minimum standard. Also, light curves in FITS format or text format can be used. See the documentation of *MPlcurve* for more information.

CHAPTER 1

Preliminary notes

1.1 MaLTPyNT vs FTOOLS (and together with FTOOLS)

1.1.1 vs POWSPEC

MaLTPyNT does a better job than POWSPEC from several points of view:

- **Good time intervals** (GTIs) are completely avoided in the computation. No gaps dirtying up the power spectrum! (This is particularly important for NuSTAR, as orbital gaps are always present in typical observation timescales)
- The number of bins used in the power spectrum (or the cospectrum) need not be a power of two! No padding needed.

1.1.2 Clarification about dead time treatment

MaLTPyNT **does not supersede** nulccorr. If one is only interested in frequencies below ~0.5 Hz, nulccorr treats robustly various dead time components and its use is recommended. Light curves produced by nulccorr can be converted to MaLTPyNT format using `MPlcurve --fits-input <lcname>.fits`, and used for the subsequent steps of the timing analysis.

Note: Improved livetime correction in progress!

In the next release MaLTPyNT 2.0, `MPexposure` pushes livetime correction to timescales below 1 s, allowing livetime-corrected timing analysis above 1 Hz. The feature is under testing

1.2 License and notes for the users

This software is released with a 3-clause BSD license. You can find license information in the `LICENSE.rst` file.

If you use this software in a publication, please refer to its Astrophysics Source Code Library identifier:

1. Bachetti, M. 2015, MaLTPyNT, Astrophysics Source Code Library, record ascl:1502.021.

In particular, **if you use the cospectrum**, please also refer to:

2. Bachetti et al. 2015, ApJ , **800**, 109.

I listed a number of **open issues** in the Issues page. Feel free to **comment** on them and **propose more**. Please choose carefully the category: bugs, enhancements, etc.

1.3 Acknowledgements

I would like to thank all the co-authors of the NuSTAR timing paper and the NuSTAR X-ray binaries working group. This software would not exist without the interesting discussions before and around that paper. In particular, I would like to thank Ivan Zolotukhin, Francesca Fornasini, Erin Kara, Felix Fürst, Poshak Gandhi, John Tomsick and Abdu Zoghbi for helping testing the code and giving various suggestions on how to improve it. Last but not least, I would like to thank Marco Buttu (by the way, check out his book if you speak Italian) for his priceless pointers on Python coding and code management techniques.

CHAPTER 2

Getting started

2.1 Installation Instructions

2.1.1 Prerequisites

You'll need a recent python 2.6+ or 3.3+ installation, and the Numpy, Matplotlib, Scipy and Astropy libraries. You should also have a working HEASoft installation to produce the cleaned event files and to use XSpec.

An **optional but recommended** dependency is the netCDF 4 library with its python bindings.

2.1.2 Quick Installation(stable releases)

Run

```
$ pip install maltpynt
```

and that's it!

2.1.3 Installing the Development version

Download

Download the distribution directory:

```
$ git clone git@bitbucket.org:mbachett/maltpynt.git
```

To use this command you will probably need to setup an SSH key for your account (in Manage Account, recommended!). Otherwise, you can use the command

```
$ git clone https://<yourusername>@bitbucket.org/mbachett/maltpynt.git
```

To update the software, just run

```
$ git pull
```

from the source directory (usually, the command gives troubleshooting information if this doesn't work the first time).

Installation

Enter the distribution directory and run

```
$ python setup.py install
```

this will check for the existing dependencies and install the files in a proper way. From that point on, executables will be somewhere in your PATH and python libraries will be available in python scripts with the usual

```
import maltpynt
```

2.2 Tutorial

2.2.1 Preliminary info

This tutorial assumes that you have previous knowledge of timing techniques, so that I don't repeat concepts as Nyquist frequency, the importance of choosing carefully the binning time and the FFT length, and so on. If you are not familiar with these concepts, this paper by Michiel is a very good place to start. Why in the example below I use the cospectrum instead of the PDS, is written in our timing paper.

This software has a modular structure. One starts from cleaned event files (such as those produced by tools like `nupipeline` and possibly barycentered with `barycorr` or equivalent), and produces a series of products with subsequent steps:

1. **event lists** containing event arrival times and PI channel information
2. (optional) **calibrated event lists**, where PI values have been converted to energy
3. **light curves**, choosing the energy band and the bin time
4. (optional) **summed light curves** if we want to join events from multiple instruments, or just from different observing times
5. **power spectrum** and/or **cross spectrum** (hereafter the ''frequency spectra'')
6. **rebinning** of frequency spectra
7. finally, **lags** and **cospectrum**
8. (optional) frequency spectra in XSpec format

Most of these tools have help information that can be accessed by typing the name of the command plus -h or –help:

```
$ MPcalibrate -h
usage: MPcalibrate [-h] [-r RMF] [-o] files [files ...]

Calibrates clean event files by associating the correct energy to each PI
channel. Uses either a specified rmf file or (for NuSTAR only) an rmf file
from the CALDB

positional arguments:
  files              List of files

optional arguments:
  -h, --help         show this help message and exit
  -r RMF, --rmf RMF  rmf file used for calibration
  -o, --overwrite    Overwrite; default: no
```

Some scripts (e.g. `MPreadevents`, `MPlcurve`, `MPfspec`) have a `--nproc` option, useful when one needs to treat multiple files at a time. The load is divided among `nproc` processors, that work in parallel cutting down considerably the execution time.

For I/O, MaLTPyNT looks if the `netCDF4` library is installed. If it's found in the system, files will be saved in this format. Otherwise, the native Python `pickle` format format will be used. This format is *much* slower (It might take some minutes to load some files) and files will be bigger, but this possibility ensures portability. If you don't use netCDF4, you'll notice that file names will have the `.p` extension instead of the `.nc` below. The rest is the same.

2.2.2 Loading event lists

Starting from cleaned event files, we will first save them in `MaLTPyNT` format (a `pickle` or `netcdf4` file). For example, I'm starting from two event lists called `002A.evt` and `002B.evt`, containing the cleaned event lists from a source observed with NuSTAR's `FPMA` and `FPMB` respectively.

```
$ MPreadevents 002A.evt 002B.evt
Opening 002A.evt
Saving events and info to 002A_ev.nc
Opening 002B.evt
Saving events and info to 002B_ev.nc
```

This will create new files with a `_ev.nc` extension (`_ev.p` if you don't use netCDF4), containing the event times and the energy *channel* (`PI`) of each event

2.2.3 Calibrating event lists

Use `MPcalibrate`. You can either specify an `rmf` file with the `-r` option, or just let it look for it in the NuSTAR `CALDB` (the environment variable has to be defined!)

```
$ MPcalibrate 002A_ev.nc 002B_ev.nc
Loading file 002A_ev.nc...
Done.
###############ATTENTION!!####################

Rmf not specified. Using default NuSTAR rmf.

################################################
Saving calibrated data to 002A_ev_calib.nc
Loading file 002B_ev.nc...
Done.
###############ATTENTION!!####################

Rmf not specified. Using default NuSTAR rmf.

################################################
Saving calibrated data to 002B_ev_calib.nc
```

This will create two new files with `_ev_calib.nc` suffix that will contain energy information. Optionally, you can overwrite the original event lists.

2.2.4 Producing light curves

Choose carefully the binning time (option `-b`). Since what we are interested in is a power spectrum, this binning time will limit our maximum frequency in the power spectrum. We are here specifying 2^-8 =0.00390625 for binning time (how to use the `-b` option is of course documented. Use `-h` FMI). Since we have calibrated the event files, we can

also choose an event energy range, here between 3 and 30 keV. Another thing that is useful in NuSTAR data is taking some time intervals out from the start and the end of each GTI. This is mostly to eliminate an increase of background level that often appears at GTI borders and produces very nasty power spectral shapes. Here I filter 100 s from the start and 300 s from the end of each GTI.

```
$ MPlcurve 002A_ev_calib.nc 002B_ev_calib.nc -b -8 -e 3 30 --safe-interval 100 300
Loading file 002A_ev_calib.nc...
Done.
Saving light curve to 002A_E3-30_lc.nc
Loading file 002B_ev_calib.nc...
Done.
Saving light curve to 002B_E3-30_lc.nc
```

To check the light curve that was produced, use the `MPplot` program:

```
$ MPplot 002A_E3-30_lc.nc
```

`MPlcurve` also accepts light curves in FITS and text format. FITS light curves should be produced by the `lcurve` FTOOL or similar, while the text light curves should have two columns: time from the NuSTAR MJDREF (55197.00076601852) and intensity in counts/bin. Use

```
$ MPlcurve --fits-input lcurve.fits
```

or

```
$ MPlcurve --txt-input lcurve.txt
```

respectively.

2.2.5 Joining, summing and "scrunching" light curves

If we want a single light curve from multiple ones, either summing multiple instruments or multiple energy or time ranges, we can use `mp_scrunch_lc`:

```
$ MPscrunchlc 002A_E3-30_lc.nc 002B_E3-30_lc.nc -o 002scrunch_3-30_lc.nc
Loading file 002A_E3-30_lc.nc...
Done.
Loading file 002B_E3-30_lc.nc...
Done.
Saving joined light curve to out_lc.nc
Saving scrunched light curve to 002scrunch_3-30_lc.nc
```

This is only tested in ''safe" situations (files are not too big and have consistent time and energy ranges), so it might give inconsistent results or crash in untested situations. Please report any problems!

2.2.6 Producing power spectra and cross power spectra

Let us just produce the cross power spectrum for now. To produce also the power spectra corresponding to each light curve, substitute `"CPDS"` with `"PDS,CPDS"`. I use rms normalization here, default would be Leahy normalization.

```
$ MPfspec 002A_E3-30_lc.nc 002B_E3-30_lc.nc -k CPDS -o cpds_002_3-30 --norm rms
Beware! For cpds and derivatives, I assume that the files are
ordered as follows: obs1_FPMA, obs1_FPMB, obs2_FPMA, obs2_FPMB...
Loading file 002A_E3-30_lc.nc...
Loading file 002B_E3-30_lc.nc...
Saving CPDS to ./cpds_002_3-30_0.nc
```

2.2.7 Rebinning the spectrum

Now let's rebin the spectrum. If the rebin factor is an integer, it is interpreted as a constant rebinning. Otherwise (only if >1), it is interpreted as a geometric binning.

```
$ MPrebin cpds_002_3-30_0.nc -r 1.03
Saving cpds to cpds_002_3-30_0_rebin1.03.nc
```

2.2.8 Calculating the cospectrum and phase/time lags

The calculation of lags and their errors is implemented in `MPlags`, and needs to be tested properly. For the cospectrum, it is sufficient to read the real part of the cross power spectrum as depicted in the relevant function in `mp_plot.py` (Use the source, Luke!).

2.2.9 Saving the spectra in a format readable to XSpec

To save the cospectrum in a format readable to XSpec it is sufficient to give the command

```
$ MP2xspec cpds_002_3-30_0_rebin1.03.nc --flx2xsp
```

2.2.10 Open and fit in XSpec!

```
$ xspec
XSPEC> data cpds.pha
XSPEC> cpd /xw; setp ener; setp comm log y
XSPEC> mo lore + lore + lore
(...)
XSPEC> fit
XSPEC> pl eufspe delchi
```

etc.

(NOTE: I know, Mike, it's unfolded... but for a flat response it shouldn't matter, right? ;))

CHAPTER 3

Command line interface

3.1 MP2xspec

Save a frequency spectrum in a qdp file that can be read by flx2xsp and produce a XSpec-compatible spectrum file

usage: MP2xspec [-h] [–loglevel LOGLEVEL] [–debug] [–flx2xsp] files [files ...]

`files`

List of files

`-h, --help`

show this help message and exit

`--loglevel <loglevel>`

use given logging level (one between INFO, WARNING, ERROR, CRITICAL, DEBUG; default:WARNING)

`--debug`

use DEBUG logging level

`--flx2xsp`

Also call flx2xsp at the end

3.2 MPcalibrate

Calibrate clean event files by associating the correct energy to each PI channel. Uses either a specified rmf file or (for NuSTAR only) an rmf file from the CALDB

usage: MPcalibrate [-h] [-r RMF] [-o] [–loglevel LOGLEVEL] [–debug] [–nproc NPROC] files [files ...]

`files`

List of files

`-h, --help`

show this help message and exit

`-r <rmf>, --rmf <rmf>`

rmf file used for calibration

`-o, --overwrite`

Overwrite; default: no

`--loglevel <loglevel>`

use given logging level (one between INFO, WARNING, ERROR, CRITICAL, DEBUG; default:WARNING)

`--debug`

use DEBUG logging level

`--nproc <nproc>`

Number of processors to use

3.3 MPcreategti

Create GTI files from a filter expression, or applies previously created GTIs to a file

usage: MPcreategti [-h] [-f FILTER] [-c] [-o] [-a APPLY_GTI] [–safe-interval SAFE_INTERVAL SAFE_INTERVAL] [–loglevel LOGLEVEL] [–debug] files [files ...]

`files`

List of files

`-h, --help`

show this help message and exit

`-f <filter>, --filter <filter>`

Filter expression, that has to be a valid Python boolean operation on a data variable contained in the files

`-c, --create-only`

If specified, creates GTIs withouth applyingthem to files (Default: False)

`-o, --overwrite`

Overwrite original file (Default: False)

`-a <apply_gti>, --apply-gti <apply_gti>`

Apply a GTI from this file to input files

`--safe-interval <safe_interval>`

Interval at start and stop of GTIs used for filtering

`--loglevel <loglevel>`

use given logging level (one between INFO, WARNING, ERROR, CRITICAL, DEBUG; default:WARNING)

`--debug`

use DEBUG logging level

3.4 MPdumpdyn

Dump dynamical (cross) power spectra

usage: MPdumpdyn [-h] [–noplot] files [files ...]

`files`

List of (c)PDS files

`-h, --help`

show this help message and exit

`--noplot`

plot results

3.5 MPfspec

Create frequency spectra (PDS, CPDS, cospectrum) starting from well-defined input ligthcurves

usage: MPfspec [-h] [-b BINTIME] [-r REBIN] [-f FFTLEN] [-k KIND] [–norm NORM] [–noclobber] [-o OUTROOT] [–loglevel LOGLEVEL] [–nproc NPROC] [–back BACK] [–debug] [–save-dyn] files [files ...]

`files`

List of light curve files

`-h, --help`

show this help message and exit

`-b <bintime>, --bintime <bintime>`

Light curve bin time; if negative, interpreted as negative power of 2. Default: 2^-10, or keep input lc bin time (whatever is larger)

`-r <rebin>, --rebin <rebin>`

(C)PDS rebinning to apply. Default: none

`-f <fftlen>, --fftlen <fftlen>`

Length of FFTs. Default: 512 s

`-k <kind>, --kind <kind>`

Spectra to calculate, as comma-separated list (Accepted: PDS and CPDS; Default: “PDS,CPDS”)

`--norm <norm>`

Normalization to use (Accepted: Leahy and rms; Default: “Leahy”)

`--noclobber`

Do not overwrite existing files

`-o <outroot>, --outroot <outroot>`

Root of output file names for CPDS only

`--loglevel <loglevel>`

use given logging level (one between INFO, WARNING, ERROR, CRITICAL, DEBUG; default:WARNING)

`--nproc <nproc>`

Number of processors to use

`--back <back>`

Estimated background (non-source) count rate

`--debug`

use DEBUG logging level

`--save-dyn`

save dynamical power spectrum

3.6 MPlags

Calculate time lags from the cross power spectrum and the power spectra of the two channels

usage: MPlags [-h] [-o OUTROOT] [–loglevel LOGLEVEL] [–noclobber] [–debug] files [files ...]

`files`

Three files: the cross spectrum and the two power spectra

`-h, --help`

show this help message and exit

`-o <outroot>, --outroot <outroot>`

Root of output file names

`--loglevel <loglevel>`

use given logging level (one between INFO, WARNING, ERROR, CRITICAL, DEBUG;default:WARNING)

`--noclobber`

Do not overwrite existing files

`--debug`

use DEBUG logging level

3.7 MPlcurve

Create lightcurves starting from event files. It is possible to specify energy or channel filtering options

usage: MPlcurve [-h] [-b BINTIME] [–safe-interval SAFE_INTERVAL SAFE_INTERVAL] [–pi-interval PI_INTERVAL PI_INTERVAL] [-e E_INTERVAL E_INTERVAL] [-s] [-j] [-g] [–minlen MINLEN] [–ignore-gtis] [-d OUTDIR] [–loglevel LOGLEVEL] [–nproc NPROC] [–debug] [–noclobber] [–fits-input] [–txt-input] files [files ...]

`files`

List of files

`-h, --help`

show this help message and exit

`-b <bintime>, --bintime <bintime>`

Bin time; if negative, negative power of 2

`--safe-interval <safe_interval>`

Interval at start and stop of GTIs used for filtering

`--pi-interval <pi_interval>`

PI interval used for filtering

`-e <e_interval>, --e-interval <e_interval>`

Energy interval used for filtering

`-s, --scrunch`

Create scrunched light curve (single channel)

`-j, --join`

Create joint light curve (multiple channels)

`-g, --gti-split`

Split light curve by GTI

`--minlen <minlen>`

Minimum length of acceptable GTIs (default:4)

`--ignore-gtis`

Ignore GTIs

`-d <outdir>, --outdir <outdir>`

Output directory

`--loglevel <loglevel>`

use given logging level (one between INFO, WARNING, ERROR, CRITICAL, DEBUG; default:WARNING)

`--nproc <nproc>`

Number of processors to use

`--debug`

use DEBUG logging level

`--noclobber`

Do not overwrite existing files

`--fits-input`

Input files are light curves in FITS format

`--txt-input`

Input files are light curves in txt format

3.8 MPplot

Plot the content of MaLTPyNT light curves and frequency spectra

usage: MPplot [-h] files [files ...]

`files`

List of files

`-h, --help`

show this help message and exit

3.9 MPreadevents

Read a cleaned event files and saves the relevant information in a standard format

usage: MPreadevents [-h] [–loglevel LOGLEVEL] [–nproc NPROC] [–noclobber] [-g] [–debug] files [files ...]

`files`

List of files

`-h, --help`

show this help message and exit

`--loglevel <loglevel>`

use given logging level (one between INFO, WARNING, ERROR, CRITICAL, DEBUG; default:WARNING)

`--nproc <nproc>`

Number of processors to use

`--noclobber`

Do not overwrite existing event files

`-g, --gti-split`

Split event list by GTI

`--debug`

use DEBUG logging level

3.10 MPreadfile

Print the content of MaLTPyNT files

usage: MPreadfile [-h] files [files ...]

`files`

List of files

`-h, --help`

show this help message and exit

3.11 MPrebin

Rebin light curves and frequency spectra.

usage: MPrebin [-h] [-r REBIN] [–loglevel LOGLEVEL] [–debug] files [files ...]

`files`

List of light curve files

`-h, --help`

show this help message and exit

`-r <rebin>, --rebin <rebin>`

Rebinning to apply. Only if the quantity to rebin is a (C)PDS, it is possible to specify a non-integer rebin factor, in which case it is interpreted as a geometrical binning factor

`--loglevel <loglevel>`

use given logging level (one between INFO, WARNING, ERROR, CRITICAL, DEBUG; default:WARNING)

`--debug`

use DEBUG logging level

3.12 MPscrunchlc

Sum lightcurves from different instruments or energy ranges

usage: MPscrunchlc [-h] [-o OUT] [–loglevel LOGLEVEL] [–debug] files [files ...]

`files`

List of files

`-h, --help`

show this help message and exit

`-o <out>, --out <out>`

Output file

`--loglevel <loglevel>`

use given logging level (one between INFO, WARNING, ERROR, CRITICAL, DEBUG; default:WARNING)

`--debug`

use DEBUG logging level

3.13 MPsumfspec

Sum (C)PDSs contained in different files

usage: MPsumfspec [-h] [-o OUTNAME] files [files ...]

`files`

List of light curve files

`-h, --help`

show this help message and exit

`-o <outname>, --outname <outname>`

Output file name for summed (C)PDS. Default: tot_(c)pds.nc

CHAPTER 4

API documentation

4.1 maltpynt

4.1.1 maltpynt package

Submodules

maltpynt.mp_base module

A miscellaneous collection of basic functions, that are likely to be used by the other modules.

maltpynt.mp_base.**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** – A string containing the parts of the two names in common

Return type str

Other Parameters **default** (*str*) – The string to return if common_str is empty

maltpynt.mp_base.**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 *mp_check_gtis*.

maltpynt.mp_base.**mp_calc_countrate**(*time*, *lc*, *gtis=None*, *bintime=None*)
Calculate the count rate from a light curve.

Parameters

- **time** (*array-like*) –
- **lc** (*array-like*) –

Returns **countrate** – The mean count rate

Return type float

Other Parameters

- **gtis** (*[[gti0_0, gti0_1], [gti1_0, gti1_1], ...]*)
- **bintime** (*float*) – The bin time of the light curve. If not specified, the minimum difference between time bins is used

maltpynt.mp_base.**mp_check_gtis**(*gti*)
Check if GTIs are well-behaved. No start>end, no overlaps.

Raises `AssertionError` – If GTIs are not well-behaved.

maltpynt.mp_base.**mp_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** – A list of integer couples, with the start and end of each True blocks in the original array

Return type [[i0_0, i0_1], [i1_0, i1_1], ...]

Notes

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

maltpynt.mp_base.**mp_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** – The newly created GTIs

Return type [[gti0_0, gti0_1], [gti1_0, gti1_1], ...]

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.

maltpynt.mp_base.**mp_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*)

maltpynt.mp_base.**mp_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** – The newly created GTIs

Return type [[gti0_0, gti0_1], [gti1_0, gti1_1], ...]

See also:

mp_cross_two_gtis() Extract the common intervals from two GTI lists *EXACTLY*

maltpynt.mp_base.**mp_cross_gtis_bin**(*gti_list*, *bin_time=1*)
From multiple GTI lists, extract the common intervals.

Note: Deprecated *mp_cross_gtis_bin* will be removed, it is replaced by *mp_cross_gtis* because the latter uses a better algorithm.

---

maltpynt.mp_base.**mp_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** – The newly created GTIs

Return type [[gti0_0, gti0_1], [gti1_0, gti1_1], ...]

See also:

mp_cross_gtis() From multiple GTI lists, extract the common intervals *EXACTLY*

maltpynt.mp_base.**mp_detection_level**(*nbins*, *epsilon=0.01*, *n_summed_spectra=1*, *n_rebin=1*)
Detection level for a PDS.

Return the detection level (with probability 1 - epsilon) for a Power Density Spectrum of nbins bins, normalized a la Leahy (1983), based on the 2-dof χ^2 statistics, corrected for rebinning (n_rebin) and multiple PDS averaging (n_summed_spectra)

maltpynt.mp_base.**mp_gti_len**(*gti*)
Return the total good time from a list of GTIs.

maltpynt.mp_base.**mp_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

maltpynt.mp_base.**mp_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.

maltpynt.mp_base.**mp_probability_of_power**(*level*, *nbins*, *n_summed_spectra=1*, *n_rebin=1*)
Give the probability of a given power level in PDS.

Return the probability of a certain power level in a Power Density Spectrum of nbins bins, normalized a la Leahy (1983), based on the 2-dof χ^2 statistics, corrected for rebinning (n_rebin) and multiple PDS averaging (n_summed_spectra)

maltpynt.mp_base.**mp_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*)

maltpynt.mp_base.**mp_ref_mjd**(*fits_file*, *hdu=1*)
Read MJDREFF+ MJDREFI or, if failed, MJDREF, from the FITS header.

Parameters **fits_file** (*str*) –

Returns **mjdref** – the reference MJD

Return type numpy.longdouble

Other Parameters **hdu** (*int*)

maltpynt.mp_base.**mp_root**(*filename*)
Return the root file name (without _ev, _lc, etc.).

Parameters **filename** (*str*) –

maltpynt.mp_base.**mp_sort_files**(*files*)
Sort a list of MaLTPyNT files, looking at *Tstart* in each.

maltpynt.mp_calibrate module

Calibrate event lists by looking in rmf files.

maltpynt.mp_calibrate.**mp_calibrate**(*fname*, *outname*, *rmf_file=None*)
Do calibration of an event list.

Parameters

- **fname** (*str*) – The MaLTPyNT file containing the events
- **outname** (*str*) – The output file

Other Parameters **rmf_file** (*str*) – The rmf file used to read the calibration. If None or not specified, the one given by mp_default_nustar_rmf() is used.

maltpynt.mp_calibrate.**mp_default_nustar_rmf**()
Look for the default rmf file in the CALDB.

The CALDB environment variable has to point to the correct location of the NuSTAR CALDB

Note: The calibration might change in the future. The hardcoded file name will be eventually replaced with a smarter choice based on observing time

maltpynt.mp_calibrate.**mp_read_calibration**(*pis*, *rmf_file=None*)
Read the energy channels corresponding to the given PI channels.

Parameters **pis** (*array-like*) – The channels to lookup in the rmf

Other Parameters **rmf_file** (*str*) – The rmf file used to read the calibration. If None or not specified, the one given by mp_default_nustar_rmf() is used.

maltpynt.mp_calibrate.**mp_read_rmf**(*rmf_file=None*)
Load RMF info.

Note: Preliminary: only EBOUNDS are read.

Parameters **rmf_file** (*str*) – The rmf file used to read the calibration. If None or not specified, the one given by mp_default_nustar_rmf() is used.

Returns

- **pis** (*array-like*) – the PI channels
- **e_mins** (*array-like*) – the lower energy bound of each PI channel
- **e_maxs** (*array-like*) – the upper energy bound of each PI channel

maltpynt.mp_create_gti module

Functions to create and apply GTIs.

maltpynt.mp_create_gti.**mp_apply_gti**(*fname*, *gti*, *outname=None*)
Apply a GTI list to the data contained in a file.

File MUST have a GTI extension already, and an extension called *time*.

maltpynt.mp_create_gti.**mp_create_gti**(*fname, filter_expr, safe_interval=[0, 0], outfile=None*)
Create a GTI list by using boolean operations on file data.

Parameters

- **fname** (*str*) – File name. The file must be in MaLTPyNT format.
- **filter_expr** (*str*) – A boolean condition on one or more of the arrays contained in the data. E.g. '(lc > 10) & (lc < 20)'

Returns **gtis** – The newly created GTIs

Return type [[gti0_0, gti0_1], [gti1_0, gti1_1], ...]

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.
- **outfile** (*str*) – The output file name. If None, use a default root + '_gti' combination

maltpynt.mp_fspec module

Functions to calculate frequency spectra.

maltpynt.mp_fspec.**mp_calc_cpds**(*lcfile1*, *lcfile2*, *fftlen*, *save_dyn=False*, *bintime=1*, *pdsrebin=1*, *outname=u'cpds.p'*, *normalization=u'Leahy'*, *back_ctrate=0.0*, *noclobber=False*)

Calculate the CPDS from a pair of input light curve files.

Parameters

- **lcfile1** (*str*) – The first light curve file
- **lcfile2** (*str*) – The second light curve file
- **fftlen** (*float*) – The length of the chunks over which FFTs will be calculated, in seconds

Other Parameters

- **save_dyn** (*bool*) – If True, save the dynamical power spectrum
- **bintime** (*float*) – The bin time. If different from that of the light curve, a rebinning is performed
- **pdsrebin** (*int*) – Rebin the PDS of this factor.
- **normalization** (*str*) – 'Leahy' or 'rms'. Default 'Leahy'
- **back_ctrate** (*float*) – The non-source count rate
- **noclobber** (*bool*) – If True, do not overwrite existing files
- **outname** (*str*) – Output file name for the cpds. Default: cpds.[nc|p]

maltpynt.mp_fspec.**mp_calc_fspec**(*files*, *fftlen*, *calc_pds=True*, *calc_cpds=True*, *calc_cospectrum=True*, *calc_lags=True*, *save_dyn=False*, *bintime=1*, *pdsrebin=1*, *outroot=None*, *normalization=u'Leahy'*, *nproc=1*, *back_ctrate=0.0*, *noclobber=False*)

Calculate the frequency spectra: the PDS, the cospectrum, ...

Parameters

- **files** (*list of str*) – List of input file names
- **fftlen** (*float*) – length of chunks to perform the FFT on.

Other Parameters

- **save_dyn** (*bool*) – If True, save the dynamical power spectrum
- **bintime** (*float*) – The bin time. If different from that of the light curve, a rebinning is performed
- **pdsrebin** (*int*) – Rebin the PDS of this factor.
- **normalization** (*str*) – 'Leahy' [1] or 'rms' [2] [3]. Default 'Leahy'.
- **back_ctrate** (*float*) – The non-source count rate
- **noclobber** (*bool*) – If True, do not overwrite existing files
- **outroot** (*str*) – Output file name root
- **nproc** (*int*) – Number of processors to use to parallelize the processing of multiple files

[1] Leahy et al. 1983, ApJ, 266, 160.
[2] Belloni & Hasinger 1990, A&A, 230, 103
[3] Miyamoto et al. 1991, ApJ, 383, 784

References

maltpynt.mp_fspec.**mp_calc_pds**(*lcfile*, *fftlen*, *save_dyn=False*, *bintime=1*, *pdsrebin=1*, *normalization=u'Leahy'*, *back_ctrate=0.0*, *noclobber=False*, *outname=None*)

Calculate the PDS from an input light curve file.

Parameters

- **lcfile** (*str*) – The light curve file
- **fftlen** (*float*) – The length of the chunks over which FFTs will be calculated, in seconds

Other Parameters

- **save_dyn** (*bool*) – If True, save the dynamical power spectrum
- **bintime** (*float*) – The bin time. If different from that of the light curve, a rebinning is performed
- **pdsrebin** (*int*) – Rebin the PDS of this factor.
- **normalization** (*str*) – 'Leahy' or 'rms'
- **back_ctrate** (*float*) – The non-source count rate
- **noclobber** (*bool*) – If True, do not overwrite existing files
- **outname** (*str*) – If speficied, output file name. If not specified or None, the new file will have the same root as the input light curve and the '_pds' suffix

maltpynt.mp_fspec.**mp_decide_spectrum_intervals**(*gtis*, *fftlen*)

Decide the start times of PDSs.

Start each FFT/PDS/cospectrum from the start of a GTI, and stop before the next gap. Only use for events! This will give problems with binned light curves.

Parameters

- **gtis** (*[[gti0_0, gti0_1], [gti1_0, gti1_1], ...]*) –
- **fftlen** (*float*) – Length of the chunks

maltpynt.mp_fspec.**mp_decide_spectrum_lc_intervals**(*gtis*, *fftlen*, *time*)

Similar to mp_decide_spectrum_intervals, but dedicated to light curves.

In this case, it is necessary to specify the time array containing the times of the light curve bins. Returns start and stop bins of the intervals to use for the PDS

Parameters

- **gtis** (*[[gti0_0, gti0_1], [gti1_0, gti1_1], ...]*) –
- **fftlen** (*float*) – Length of the chunks
- **time** (*array-like*) – Times of light curve bins

maltpynt.mp_fspec.**mp_fft**(*lc*, *bintime*)

A wrapper for the fft function. Just numpy for now.

Parameters

- **lc** (*array-like*) –
- **bintime** (*float*) –

Returns

- **freq** (*array-like*)
- **ft** (*array-like*) – the Fourier transform.

maltpynt.mp_fspec.**mp_leahy_cpds**(*lc1*, *lc2*, *bintime*, *return_freq=True*, *return_pdss=False*)
Calculate the cross power density spectrum.

Calculates the Cross Power Density Spectrum, normalized similarly to the PDS in Leahy+1983, ApJ 266, 160., given the lightcurve and its bin time. Assumes no gaps are present! Beware!

Parameters

- **lc1** (*array-like*) – The first light curve
- **lc2** (*array-like*) – The light curve
- **bintime** (*array-like*) – The bin time of the light curve

Other Parameters **return_freq** (*bool, default True*) – Return the frequencies corresponding to the PDS bins?

maltpynt.mp_fspec.**mp_leahy_pds**(*lc*, *bintime*, *return_freq=True*)
Calculate the power density spectrum.

Calculates the Power Density Spectrum a la Leahy+1983, ApJ 266, 160, given the lightcurve and its bin time. Assumes no gaps are present! Beware!

Parameters

- **lc** (*array-like*) – the light curve
- **bintime** (*array-like*) – the bin time of the light curve

Other Parameters **return_freq** (*bool, default True*) – Return the frequencies corresponding to the PDS bins?

maltpynt.mp_fspec.**mp_read_fspec**(*fname*)
Read the frequency spectrum from a file. :param fname: The input file name

Returns

- **ftype** (*str*) – File type
- **freq** (*array-like*) – Frequency array
- **fspec** (*array-like*) – Frequency spectrum array
- **efspec** (*array-like*) – Errors on spectral bins
- **nchunks** (*int*) – Number of spectra that have been summed to obtain fspec
- **rebin** (*array-like or int*) – Rebin factor in each bin. Might be irregular in case of geometrical binning

maltpynt.mp_fspec.**mp_rms_normalize_pds**(*pds*, *pds_err*, *source_ctrate*, *back_ctrate=None*)
Normalize a Leahy PDS with RMS normalization ([4], [5]).

Parameters

- **pds** (*array-like*) – The Leahy-normalized PDS
- **pds_err** (*array-like*) – The uncertainties on the PDS values
- **source_ctrate** (*float*) – The source count rate
- **back_ctrate** (*float, optional*) – The background count rate

[4] Belloni & Hasinger 1990, A&A, 230, 103
[5] Miyamoto+1991, ApJ, 383, 784

Returns

- **pds** (*array-like*) – the RMS-normalized PDS
- **pds_err** (*array-like*) – the uncertainties on the PDS values

References

maltpynt.mp_fspec.**mp_welch_cpds**(*time*, *lc1*, *lc2*, *bintime*, *fftlen*, *gti=None*, *return_ctrate=False*, *return_all=False*)

Calculate the CPDS, averaged over equal chunks of data.

Calculates the Cross Power Density Spectrum normalized like PDS, given the lightcurve and its bin time, over equal chunks of length fftlen, and returns the average of all PDSs, or the sum PDS and the number of chunks

Parameters

- **time** (*array-like*) – Central times of light curve bins
- **lc1** (*array-like*) – Light curve 1
- **lc2** (*array-like*) – Light curve 2
- **bintime** (*float*) – Bin time of the light curve
- **fftlen** (*float*) – Length of each FFT
- **gti** (*[[g0_0, g0_1], [g1_0, g1_1], ...]*) – Good time intervals. Defaults to [[time[0] - bintime/2, time[-1] + bintime/2]]

Returns

- **return_str** (*object, optional*) – An Object containing all return values below, plus the dynamical PDS. This is returned if return_all is True
- **freq** (*array-like*) – array of frequencies corresponding to PDS bins
- **pds** (*array-like*) – the values of the PDS
- **pds_err** (*array-like*) – the values of the PDS
- **n_chunks** (*int*) – the number of summed PDSs (if normalize is False)
- **ctrate** (*float*) – the average count rate in the two lcs

Other Parameters

- **return_ctrate** (*bool*) – if True, return also the count rate
- **return_all** (*bool*) – if True, return everything, including the dynamical PDS

maltpynt.mp_fspec.**mp_welch_pds**(*time*, *lc*, *bintime*, *fftlen*, *gti=None*, *return_ctrate=False*, *return_all=False*)

Calculate the PDS, averaged over equal chunks of data.

Calculates the Power Density Spectrum ‘a la Leahy (1983), given the lightcurve and its bin time, over equal chunks of length fftlen, and returns the average of all PDSs, or the sum PDS and the number of chunks

Parameters

- **time** (*array-like*) – Central times of light curve bins
- **lc** (*array-like*) – Light curve
- **bintime** (*float*) – Bin time of the light curve
- **fftlen** (*float*) – Length of each FFT

- **gti** (*[[g0_0, g0_1], [g1_0, g1_1], ...]*) – Good time intervals. Defaults to [[time[0] - bintime/2, time[-1] + bintime/2]]

Returns

- **return_str** (*object, optional*) – An Object containing all return values below, plus the dynamical PDS. This is returned if return_all is True
- **freq** (*array-like*) – array of frequencies corresponding to PDS bins
- **pds** (*array-like*) – the values of the PDS
- **pds_err** (*array-like*) – the values of the PDS
- **n_chunks** (*int*) – the number of summed PDSs (if normalize is False)
- **ctrate** (*float*) – the average count rate in the two lcs

Other Parameters

- **return_ctrate** (*bool*) – if True, return also the count rate
- **return_all** (*bool*) – if True, return everything, including the dynamical PDS

maltpynt.mp_io module

Functions to perform input/output operations.

maltpynt.mp_io.**is_string**(*s*)
Portable function to answer this question.

maltpynt.mp_io.**mp_get_file_extension**(*fname*)
Get the file extension.

maltpynt.mp_io.**mp_get_file_format**(*fname*)
Decide the file format of the file.

maltpynt.mp_io.**mp_get_file_type**(*fname*, *specify_reb=True*)
Return the file type and its contents.

Only works for maltpynt-format pickle or netcdf files.

maltpynt.mp_io.**mp_load_data**(*fname*)
Load generic data in maltpynt format.

maltpynt.mp_io.**mp_load_events**(*fname*)
Load events from a file.

maltpynt.mp_io.**mp_load_lcurve**(*fname*)
Load light curve from a file.

maltpynt.mp_io.**mp_load_pds**(*fname*)
Load PDS from a file.

maltpynt.mp_io.**mp_print_fits_info**(*fits_file*, *hdu=1*)
Print general info about an observation.

maltpynt.mp_io.**mp_read_from_netcdf**(*fname*)
Read from a netCDF4 file.

maltpynt.mp_io.**mp_save_as_netcdf**(*vars*, *varnames*, *formats*, *fname*)
Save variables in a NetCDF4 file.

maltpynt.mp_io.**mp_save_data**(*struct*, *fname*, *ftype=u'data'*)
Save generic data in maltpynt format.

maltpynt.mp_io.**mp_save_events**(*eventStruct*, *fname*)
Save events in a file.

maltpynt.mp_io.**mp_save_lcurve**(*lcurveStruct*, *fname*)
Save light curve in a file.

maltpynt.mp_io.**mp_save_pds**(*pdsStruct*, *fname*)
Save PDS in a file.

maltpynt.mp_io.**save_as_ascii**(*cols*, *filename=u'out.txt'*, *colnames=None*, *append=False*)
Save arrays as TXT file with respective errors.

maltpynt.mp_io.**save_as_qdp**(*arrays*, *errors=None*, *filename=u'out.qdp'*)
Save arrays in a QDP file.

Saves an array of variables, and possibly their errors, to a QDP file.

Parameters

- **arrays** (*[array1, array2]*) – List of variables. All variables must be arrays and of the same length.
- **errors** (*[array1, array2]*) – List of errors. The order has to be the same of arrays; the value can be: - None if no error is assigned - an array of same length of variable for symmetric errors - an array of len-2 lists for non-symmetric errors (e.g. [[errm1, errp1], [errm2, errp2], [errm3, errp3], ...])

maltpynt.mp_lags module

Functions to calculate lags.

maltpynt.mp_lags.**mp_calc_lags**(*freqs*, *cpds*, *pds1*, *pds2*, *n_chunks*, *rebin*)
Calculate time lags.

Parameters

- **freqs** (*array-like*) – The frequency array
- **cpds** (*array-like*) – The cross power spectrum
- **pds1** (*array-like*) – The PDS of the first channel
- **pds2** (*array-like*) – The PDS of the second channel
- **n_chunks** (*int or array-like*) – The number of PDSs averaged
- **rebin** (*int or array-like*) – The number of bins averaged to obtain each bin

Returns

- **lags** (*array-like*) – The lags spectrum at frequencies corresponding to freqs
- **lagse** (*array-like*) – The errors on *lags*

maltpynt.mp_lags.**mp_lags_from_spectra**(*cpdsfile*, *pds1file*, *pds2file*, *outroot=u'lag'*, *noclobber=False*)
Calculate time lags.

Parameters

- **cpdsfile** (*str*) – The MP-format file containing the CPDS
- **pds1file** (*str*) – The MP-format file containing the first PDS used for the CPDS
- **pds1file** – The MP-format file containing the second PDS

Returns

- **freq** (*array-like*) – Central frequencies of spectral bins
- **df** (*array-like*) – Width of each spectral bin
- **lags** (*array-like*) – Time lags
- **elags** (*array-like*) – Error on the time lags

Other Parameters

- **outroot** (*str*) – Root of the output filename
- **noclobber** (*bool*) – If True, do not overwrite existing files

maltpynt.mp_lcurve module

Light curve-related functions.

maltpynt.mp_lcurve.**mp_filter_lc_gtis**(*time*, *lc*, *gti*, *safe_interval=None*, *delete=False*, *min_length=0*, *return_borders=False*)

Filter a light curve for GTIs.

Parameters

- **time** (*array-like*) – The time bins of the light curve
- **lc** (*array-like*) – The light curve
- **gti** (*[[gti0_0, gti0_1], [gti1_0, gti1_1], ...]*) – Good Time Intervals

Returns

- **time** (*array-like*) – The time bins of the light curve
- **lc** (*array-like*) – The output light curve
- **newgtis** (*[[gti0_0, gti0_1], [gti1_0, gti1_1], ...]*) – The output Good Time Intervals
- **borders** (*[[i0_0, i0_1], [i1_0, i1_1], ...], optional*) – The indexes of the light curve corresponding to the borders of the GTIs. Returned if return_borders is set to True

Other Parameters

- **safe_interval** (*float or [float, float]*) – Seconds to filter out at the start and end of each GTI. If single float, these safe windows are equal, otherwise the two numbers refer to the start and end of the GTI respectively
- **delete** (*bool*) – If delete is True, the intervals outside of GTIs are filtered out from the light curve. Otherwise, they are set to zero.
- **min_length** (*float*) – Minimum length of GTI. GTIs below this length will be removed.
- **return_borders** (*bool*) – If True, return also the indexes of the light curve corresponding to the borders of the GTIs

maltpynt.mp_lcurve.**mp_join_lightcurves**(*lcfilelist*, *outfile=u'out_lc.p'*)

Join light curves from different files.

Light curves from different instruments are put in different channels.

Parameters

- **lcfilelist** –
- **outfile** –

See also:

mp_scrunch_lightcurves() Create a single light curve from input light curves.

maltpynt.mp_lcurve.**mp_lcurve**(*event_list*, *bin_time*, *start_time=None*, *stop_time=None*, *centertime=True*)

From a list of event times, estract a lightcurve.

Parameters

- **event_list** (*array-like*) – Times of arrival of events
- **bin_time** (*float*) – Binning time of the light curve

Returns

- **time** (*array-like*) – The time bins of the light curve
- **lc** (*array-like*) – The light curve

Other Parameters

- **start_time** (*float*) – Initial time of the light curve
- **stop_time** (*float*) – Stop time of the light curve
- **centertime** (*bool*) – If False, time is the start of the bin. Otherwise, the center

maltpynt.mp_lcurve.**mp_lcurve_from_events**(*f*, *safe_interval=0*, *pi_interval=None*, *e_interval=None*, *min_length=0*, *gti_split=False*, *ignore_gtis=False*, *bintime=1.0*, *outdir=None*, *outfile=None*, *noclobber=False*)

Bin an event list in a light curve.

Parameters

- **f** (*str*) – Input event file name
- **bintime** (*float*) – The bin time of the output light curve

Returns **outfiles** – List of output light curves

Return type list

Other Parameters

- **safe_interval** (*float or [float, float]*) – Seconds to filter out at the start and end of each GTI. If single float, these safe windows are equal, otherwise the two numbers refer to the start and end of the GTI respectively
- **pi_interval** (*[int, int]*) – PI channel interval to select. Default None, meaning that all PI channels are used
- **e_interval** (*[float, float]*) – Energy interval to select (only works if event list is calibrated with *mp_calibrate*). Default None
- **min_length** (*float*) – GTIs below this length will be filtered out
- **gti_split** (*bool*) – If True, create one light curve for each good time interval
- **ignore_gtis** (*bool*) – Ignore good time intervals, and get a single light curve that includes possible gaps
- **outdir** (*str*) – Output directory
- **outfile** (*str*) – Output file

- **noclobber** (*bool*) – If True, do not overwrite existing files

maltpynt.mp_lcurve.**mp_lcurve_from_fits**(*fits_file*, *gtistring=u'GTI'*, *timecolumn=u'TIME'*, *ratecolumn=None*, *ratehdu=1*, *fracexp_limit=0.9*, *outfile=None*, *noclobber=False*)

Load a lightcurve from a fits file and save it in MaLTPyNT format.

Note: FITS light curve handling is still under testing. Absolute times might be incorrect depending on the light curve format.

Parameters **fits_file** (*str*) – File name of the input light curve in FITS format

Returns **outfile** – Returned as a list with a single element for consistency with *mp_lcurve_from_events*

Return type [str]

Other Parameters

- **gtistring** (*str*) – Name of the GTI extension in the FITS file
- **timecolumn** (*str*) – Name of the column containing times in the FITS file
- **ratecolumn** (*str*) – Name of the column containing rates in the FITS file
- **ratehdu** (*str or int*) – Name or index of the FITS extension containing the light curve
- **fracexp_limit** (*float*) – Minimum exposure fraction allowed
- **outfile** (*str*) – Output file name
- **noclobber** (*bool*) – If True, do not overwrite existing files

maltpynt.mp_lcurve.**mp_lcurve_from_txt**(*txt_file*, *outfile=None*, *noclobber=False*)

Load a lightcurve from a text file.

Parameters **txt_file** (*str*) – File name of the input light curve in text format. Assumes two columns: time, counts. Times are seconds from MJDREF 55197.00076601852 (NuSTAR).

Returns **outfile** – Returned as a list with a single element for consistency with *mp_lcurve_from_events*

Return type [str]

Other Parameters

- **outfile** (*str*) – Output file name
- **noclobber** (*bool*) – If True, do not overwrite existing files

maltpynt.mp_lcurve.**mp_scrunch_lightcurves**(*lcfilelist*, *outfile=u'out_scrlc.p'*, *save_joint=False*)

Create a single light curve from input light curves.

Light curves are appended when they cover different times, and summed when they fall in the same time range. This is done regardless of the channel or the instrument.

Parameters **lcfilelist** (*list of str*) – The list of light curve files to scrunch

Returns

- **time** (*array-like*) – The time array
- *lc* – The new light curve
- **gti** (*[[gti0_0, gti0_1], [gti1_0, gti1_1], ...]*) – Good Time Intervals

Other Parameters

- **outfile** (*str*) – The output file name
- **save_joint** (*bool*) – If True, save the per-channel joint light curves

See also:

`mp_join_lightcurves()` Join light curves from different files

maltpynt.mp_plot module

Quicklook plots.

maltpynt.mp_plot.**baseline_fun**(*x*, *a*)
A constant function.

maltpynt.mp_plot.**mp_plot_cospectrum**(*fnames*, *figname=None*)
Plot the cospectra from a list of CPDSs, or a single one.

maltpynt.mp_plot.**mp_plot_lc**(*lcfiles*, *figname=None*)
Plot a list of light curve files, or a single one.

maltpynt.mp_plot.**mp_plot_pds**(*fnames*, *figname=None*)
Plot a list of PDSs, or a single one.

maltpynt.mp_read_events module

Read and save event lists from FITS files.

maltpynt.mp_read_events.**mp_load_events_and_gtis**(*fits_file*, *return_limits=False*, *additional_columns=None*, *gtistring=None*, *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*)

maltpynt.mp_read_events.**mp_load_gtis**(*fits_file*, *gtistring=None*)
Load GTI from HDU EVENTS of file fits_file.

maltpynt.mp_read_events.**mp_treat_event_file**(*filename*, *noclobber=False*, *gti_split=False*, *min_length=4*)
Read data from an event file, with no external GTI information.

Parameters **filename** (*str*) –

Other Parameters

- **noclobber** (*bool*) – if a file is present, do not overwrite it
- **gti_split** (*bool*) – split the file in multiple chunks, containing one GTI each
- **min_length** (*float*) – minimum length of GTIs accepted (only if gti_split is True)

maltpynt.mp_rebin module

Functions to rebin light curves and frequency spectra.

maltpynt.mp_rebin.**mp_const_rebin**(*x*, *y*, *factor*, *yerr=None*, *normalize=True*)
Rebin any pair of variables.

Might be time and counts, or freq and pds. Also possible to rebin the error on y.

maltpynt.mp_rebin.**mp_geom_bin**(*freq*, *pds*, *bin_factor=None*, *pds_err=None*, *npds=None*, *return_nbins=False*)
Given a PDS, bin it geometrically.

Freely taken from an algorithm in isisscripts.sl

maltpynt.mp_rebin.**mp_rebin_file**(*filename*, *rebin*)
Rebin the contents of a file, be it a light curve or a spectrum.

maltpynt.mp_save_as_xspec module

Functions to save data in a Xspec-readable format.

maltpynt.mp_save_as_xspec.**mp_save_as_xspec**(*fname*, *direct_save=False*)
Save frequency spectra in a format readable to FTOOLS and Xspec.

Parameters

- **fname** (*str*) – Input MaLTPyNT frequency spectrum file name
- **direct_save** (*bool*) – If True, also call *flx2xsp* to produce the output .pha and .rsp files. If False (default), flx2xsp has to be called from the user

Notes

Uses method described here: https://asd.gsfc.nasa.gov/XSPECwiki/fitting_timing_power_spectra_in_XSPEC

maltpynt.mp_sum_fspec module

Function to sum frequency spectra.

maltpynt.mp_sum_fspec.**sum_fspec**(*files*, *outname=None*)
Take a bunch of (C)PDSs and sums them.

Module contents

MaLTPyNT - Matteo's libraries and tools in Python for NuSTAR timing

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

m

B

C

G

I

M

S