maltpynt Documentation

Release 2.0a3

Matteo Bachetti

January 19, 2016

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.

Part I

What’s new

In preparation for the 2.0 release, the API has received some visible changes. Names do not have the `mp_` prefix anymore, as they were very redundant; the structure of the code base is now based on the AstroPy structure; tests have been moved and the documentation improved.

`MPexposure` is a new livetime correction script on sub-second timescales for NuSTAR. It will be able to replace `nulccorr`, and get results on shorter bin times, in observations done with a specific observing mode, where the observer has explicitly requested to telemeter all events (including rejected) and the user has run `nupipeline` with the `CLEANCOLS = NO` option. This tool is under testing.

`MPfake` is a new script to create fake observation files in FITS format, for testing. New functions to create fake data will be added to `maltpynt.fake`.

Part II

Preliminary notes

CHAPTER 1

MaLTPyNT vs FTOOLS (and together with FTOOLS)

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.

1.2 Clarification about dead time treatment

MaLTPyNT **does not supersede** nulccorr (yet). 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 upcoming release MaLTPyNT 2.0, `MPexposure` tries to push the livetime correction to timescales below 1 s, allowing livetime-corrected timing analysis above 1 Hz. The feature is under testing

CHAPTER 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.

CHAPTER 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.

Part III

Getting started

CHAPTER 4

Installation Instructions

4.1 Prerequisites

You'll need a recent python 2.7+ 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. An additional dependency that is now used only sparsely (if installed) but will become important in future versions is Numba.

4.2 Quick Installation(stable releases)

Run

```
$ pip install maltpynt
```

and that's it!

4.3 Installing the Development version

4.3.1 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).

4.3.2 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
```

CHAPTER 5

Tutorials

5.1 Quick-look analysis

5.1.1 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.

5.1.2 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

5.1.3 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.

5.1.4 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.

5.1.5 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 `MPscrunchlc`:

```
$ 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!

5.1.6 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
```

5.1.7 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
```

5.1.8 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 `plot.py` (Use the source, Luke!).

5.1.9 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
```

5.1.10 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? ;))

5.2 Data simulation

To simulate datasets, `MaLTPyNT` includes the `MPfake` script. It can simulate event lists with a fixed count rate or from an input light curve. Also, it is able to apply a dead time filter to the simulated event lists.

5.2.1 Basic operations

To simulate a short observation (1025 s) at a given count rate (e.g., 150 ct/s), it is sufficient to call `MPfake -c`

```
$ MPfake -c 150
$ ls
events.evt
```

To simulate an event list from an input light curve, use the `-l` (or `--lc`) option. The light curve can be in FITS or MaLTPyNT native format (or one can use MPlcurve for the conversion from text format):

```
$ MPfake -l lightcurve.fits
```

To apply dead time to the generated events, use the `--deadtime` option. deadtime can be supplied as a single number, meaning a constant dead time

```
$ MPfake -l lightcurve.fits --deadtime 2.5e-3
```

or as two numbers (`mean`, `sigma`), meaning a Gaussian distribution of dead times with the specified mean and sigma.

More advanced options are available using the functions in `maltpynt.fake`.

Part IV

Command line interface

CHAPTER 6

Command line interface

6.1 MP2xspec

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

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

positional arguments:
  files                 List of files

optional arguments:
  -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
```

6.2 MPcalibrate

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

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

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
  --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
```

6.3 MPcreategti

```
usage: MPcreategti [-h] [-f FILTER] [-c] [--overwrite] [-a APPLY_GTI]
                   [-l MINIMUM_LENGTH]
                   [--safe-interval SAFE_INTERVAL SAFE_INTERVAL]
                   [--loglevel LOGLEVEL] [--debug]
                   files [files ...]

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

positional arguments:
  files                 List of files

optional arguments:
  -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)
  --overwrite           Overwrite original file (Default: False)
  -a APPLY_GTI, --apply-gti APPLY_GTI
                        Apply a GTI from this file to input files
  -l MINIMUM_LENGTH, --minimum-length MINIMUM_LENGTH
                        Minimum length of GTIs (below this length, they will
                        be discarded)
  --safe-interval 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
```

6.4 MPdumpdyn

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

Dump dynamical (cross) power spectra

positional arguments:
  files       List of files in any valid MaLTPyNT format for PDS or CPDS

optional arguments:
  -h, --help  show this help message and exit
  --noplot    plot results
```

6.5 MPexposure

```
usage: MPexposure [-h] [-o OUTROOT] [--loglevel LOGLEVEL] [--debug] [--plot]
                  lcfile uffile

Create exposure light curve based on unfiltered event files.
```

```
positional arguments:
  lcfile                Light curve file (MaltPyNT format)
  uffile                Unfiltered event file (FITS)

optional arguments:
  -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)
  --debug               use DEBUG logging level
  --plot                Plot on window
```

6.6 MPfake

```
usage: MPfake [-h] [-e EVENT_LIST] [-l LC] [-c CTRATE] [-o OUTNAME]
              [-i INSTRUMENT] [--tstart TSTART] [--tstop TSTOP]
              [--mjdref MJDREF] [--deadtime DEADTIME [DEADTIME ...]]
              [--loglevel LOGLEVEL] [--debug]

Create an event file in FITS format from an event list, or simulating it. If
input event list is not specified, generates the events randomly

optional arguments:
  -h, --help            show this help message and exit
  -e EVENT_LIST, --event-list EVENT_LIST
                        File containint event list
  -l LC, --lc LC        File containing light curve
  -c CTRATE, --ctrate CTRATE
                        Count rate for simulated events
  -o OUTNAME, --outname OUTNAME
                        Output file name
  -i INSTRUMENT, --instrument INSTRUMENT
                        Instrument name
  --tstart TSTART       Start time of the observation (s from MJDREF)
  --tstop TSTOP         End time of the observation (s from MJDREF)
  --mjdref MJDREF       Reference MJD
  --deadtime DEADTIME [DEADTIME ...]
                        Dead time magnitude. Can be specified as a single
                        number, or two. In this last case, the second value is
                        used as sigma of the dead time distribution
  --loglevel LOGLEVEL   use given logging level (one between INFO, WARNING,
                        ERROR, CRITICAL, DEBUG; default:WARNING)
  --debug               use DEBUG logging level
```

6.7 MPfspec

```
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 ...]

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

```
input ligthcurves

positional arguments:
  files                 List of light curve files

optional arguments:
  -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
```

6.8 MPlags

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

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

positional arguments:
  files                 Three files: the cross spectrum and the two power
                        spectra

optional arguments:
  -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
```

6.9 MPlcurve

```
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] [-o OUTFILE] [--loglevel LOGLEVEL]
                [--nproc NPROC] [--debug] [--noclobber] [--fits-input]
                [--txt-input]
                files [files ...]

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

positional arguments:
  files                 List of files

optional arguments:
  -h, --help            show this help message and exit
  -b BINTIME, --bintime BINTIME
                        Bin time; if negative, negative power of 2
  --safe-interval SAFE_INTERVAL SAFE_INTERVAL
                        Interval at start and stop of GTIs used for filtering
  --pi-interval PI_INTERVAL PI_INTERVAL
                        PI interval used for filtering
  -e E_INTERVAL E_INTERVAL, --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
  -o OUTFILE, --outfile OUTFILE
                        Output file name
  --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
```

6.10 MPplot

```
usage: MPplot [-h] [--noplot] [--figname FIGNAME] [--xlog] [--ylog] [--xlin]
              [--ylin] [--fromstart] [--axes AXES AXES]
              files [files ...]

Plot the content of MaLTPyNT light curves and frequency spectra

positional arguments:
  files             List of files
```

```
optional arguments:
  -h, --help         show this help message and exit
  --noplot           Only create images, do not plot
  --figname FIGNAME  Figure name
  --xlog             Use logarithmic X axis
  --ylog             Use logarithmic Y axis
  --xlin             Use linear X axis
  --ylin             Use linear Y axis
  --fromstart        Times are measured from the start of the observation
                     (only relevant for light curves)
  --axes AXES AXES   Plot two variables contained in the file
```

6.11 MPreadevents

```
usage: MPreadevents [-h] [--loglevel LOGLEVEL] [--nproc NPROC] [--noclobber]
                    [-g] [--min-length MIN_LENGTH] [--gti-string GTI_STRING]
                    [--debug]
                    files [files ...]

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

positional arguments:
  files                 List of files

optional arguments:
  -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
  --min-length MIN_LENGTH
                        Minimum length of GTIs to consider
  --gti-string GTI_STRING
                        GTI string
  --debug               use DEBUG logging level
```

6.12 MPreadfile

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

Print the content of MaLTPyNT files

positional arguments:
  files       List of files

optional arguments:
  -h, --help  show this help message and exit
```

6.13 MPrebin

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

Rebin light curves and frequency spectra.

positional arguments:
  files                 List of light curve files

optional arguments:
  -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
```

6.14 MPscrunchlc

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

Sum lightcurves from different instruments or energy ranges

positional arguments:
  files                 List of files

optional arguments:
  -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
```

6.15 MPsumfspec

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

Sum (C)PDSs contained in different files

positional arguments:
  files                 List of light curve files

optional arguments:
  -h, --help            show this help message and exit
  -o OUTNAME, --outname OUTNAME
                        Output file name for summed (C)PDS. Default:
                        tot_(c)pds.nc
```

Part V

API documentation

CHAPTER 7

MaLTPyNT API

7.1 maltpynt package

7.1.1 Submodules

7.1.2 maltpynt.base module

A miscellaneous collection of basic functions.

maltpynt.base.**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 : float

The mean count rate

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.base.**check_gtis**(*gti*)
Check if GTIs are well-behaved. No start>end, no overlaps.

Raises
AssertionError

If GTIs are not well-behaved.

maltpynt.base.**common_name**(*str1*, *str2*, *default='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

maltpynt.base.**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

maltpynt.base.**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.

maltpynt.base.**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.base.**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*

maltpynt.base.**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*

maltpynt.base.**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.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 `check_gtis`.

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

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

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

Parameters
filename : str

maltpynt.base.**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.base.**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.base.**r_det**(*td*, *r_in*)
Calculate detected countrate given dead time and incident countrate.

maltpynt.base.**r_in**(*td*, *r_0*)
Calculate incident countrate given dead time and detected countrate.

maltpynt.base.**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.base.**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

7.1.3 maltpynt.calibrate module

Calibrate event lists by looking in rmf files.

maltpynt.calibrate.**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 default_nustar_rmf() is used.

maltpynt.calibrate.**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.calibrate.**main**(*args=None*)
Main function called by the `MPcalibrate` command line script.

maltpynt.calibrate.**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 default_nustar_rmf() is used.

maltpynt.calibrate.**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 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

7.1.4 maltpynt.create_gti module

Functions to create and apply GTIs.

maltpynt.create_gti.**apply_gti**(*fname*, *gti*, *outname=None*, *minimum_length=0*)
Apply a GTI list to the data contained in a file.

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

maltpynt.create_gti.**create_gti**(*fname, filter_expr, safe_interval=[0, 0], outfile=None, minimum_length=0*)
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 : [[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.

outfile : str

The output file name. If None, use a default root + '_gti' combination

maltpynt.create_gti.**filter_gti_by_length**(*gti*, *minimum_length*)
Filter a list of GTIs: keep those longer than `minimum_length`.

maltpynt.create_gti.**main**(*args=None*)
Main function called by the `MPcreategti` command line script.

7.1.5 maltpynt.exposure module

Calculate the exposure correction for light curves.

Only works for data taken in specific data modes of NuSTAR, where all events are telemetered.

maltpynt.exposure.**correct_lightcurve**(*lc_file*, *uf_file*, *outname=None*, *expo_limit=1e-07*)
Apply exposure correction to light curve.

Parameters
lc_file : str

The light curve file, in MaLTPyNT format

uf_file : str

The unfiltered event file, in FITS format

Returns
outdata : str

Output data structure

Other Parameters
outname : str

Output file name

maltpynt.exposure.**get_exposure_from_uf**(*time*, *uf_file*, *dt=None*, *gti=None*)
Get livetime from unfiltered event file.

Parameters
time : array-like

The time bins of the light curve

uf_file : str

Unfiltered event file (the one in the event_cl directory with the _uf suffix)

Returns
expo : array-like

Exposure (livetime) values corresponding to time bins

Other Parameters
dt : float

If time array is not sampled uniformly, dt can be specified here.

maltpynt.exposure.**get_livetime_per_bin**(*times*, *events*, *priors*, *dt=None*, *gti=None*)
Get the livetime in a series of time intervals.

Parameters
times : array-like

The array of times to look at

events : array-like

A list of events, producing dead time

priors : array-like

The livetime before each event (as in the PRIOR column of unfiltered NuSTAR event files)

Returns

livetime_array : array-like

An array of the same length as times, containing the live time values

Other Parameters

dt : float or array-like

The width of the time bins of the time array. Can be a single float or an array of the same length as times

gti : [[g0_0, g0_1], [g1_0, g1_1], ...]

Good time intervals. Defaults to [[time[0] - dt[0]/2, time[-1] + dt[-1]/2]]

`maltpynt.exposure.`**`main`**(*args=None*)

Main function called by the `MPexposure` command line script.

7.1.6 maltpynt.fake module

Functions to simulate data and produce a fake event file.

`maltpynt.fake.`**`fake_events_from_lc`**(*times*, *lc*, *use_spline=False*, *bin_time=None*)

Create events from a light curve.

Parameters

times : array-like

the center time of each light curve bin

lc : array-like

light curve, in units of counts/bin

Returns

event_list : array-like

Simulated arrival times

`maltpynt.fake.`**`filter_for_deadtime`**(*ev_list*, *deadtime*, *bkg_ev_list=None*, *dt_sigma=None*, *paralyzable=False*, *additional_data=None*, *return_all=False*)

Filter an event list for a given dead time.

Parameters

ev_list : array-like

The event list

deadtime: float

The (mean, if not constant) value of deadtime

Returns

new_ev_list : array-like

The filtered event list

additional_output : dict

Object with all the following attributes. Only returned if `return_all` is True

uf_events : array-like

Unfiltered event list (events + background)

is_event : array-like

Boolean values; True if event, False if background

deadtime : array-like

Dead time values

bkg : array-like

The filtered background event list

mask : array-like, optional

The mask that filters the input event list and produces the output event list.

Other Parameters

bkg_ev_list : array-like

A background event list that affects dead time

dt_sigma : float

The standard deviation of a non-constant dead time around deadtime.

return_all : bool

If True, return the mask that filters the input event list to obtain the output event list.

maltpynt.fake.**generate_fake_fits_observation**(*event_list=None*, *filename=None*, *pi=None*, *instr='FPMA'*, *gti=None*, *tstart=None*, *tstop=None*, *mjdref=55197.00076601852*, *livetime=None*, *additional_columns={}*)

Generate fake NuSTAR data.

Takes an event list (as a list of floats) All inputs are None by default, and can be set during the call.

Parameters

event_list : list-like

List of event arrival times, in seconds from mjdref. If left None, 1000 random events will be generated, for a total length of 1025 s or the difference between tstop and tstart.

filename : str

Output file name

Returns

hdulist : FITS hdu list

FITS hdu list of the output file

Other Parameters

mjdref : float

Reference MJD. Default is 55197.00076601852 (NuSTAR)

pi : list-like

The PI channel of each event

tstart : float

Start of the observation (s from mjdref)

tstop : float

End of the observation (s from mjdref)

instr : str

Name of the instrument. Default is 'FPMA'

livetime : float

Total livetime. Default is tstop - tstart

maltpynt.fake.**jit**(*fun*)
Dummy decorator in case jit cannot be imported.

maltpynt.fake.**main**(*args=None*)
Main function called by the MPfake command line script.

7.1.7 maltpynt.fspec module

Functions to calculate frequency spectra.

maltpynt.fspec.**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.fspec.**calc_fspec**(*files*, *fftlen*, *do_calc_pds=True*, *do_calc_cpds=True*, *do_calc_cospectrum=True*, *do_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' [3] or 'rms' [4] [5]. 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

References

[3] Leahy et al. 1983, ApJ, 266, 160.

[4] Belloni & Hasinger 1990, A&A, 230, 103

[5] Miyamoto et al. 1991, ApJ, 383, 784

maltpynt.fspec.**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.fspec.**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

Returns

spectrum_start_times : array-like

List of starting times to use in the spectral calculations.

maltpynt.fspec.**decide_spectrum_lc_intervals**(*gtis*, *fftlen*, *time*)

Similar to 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.fspec.**dumpdyn**(*fname*, *plot=False*)
Dump a dynamical frequency spectrum in text files.

Parameters
fname : str

The file name

Other Parameters
plot : bool

if True, plot the spectrum

maltpynt.fspec.**dumpdyn_main**(*args=None*)
Main function called by the MPdumpdyn command line script.

maltpynt.fspec.**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.fspec.**leahy_cpds**(*lc1*, *lc2*, *bintime*)
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

Returns
freqs : array-like

Frequencies corresponding to PDS

cpds : array-like

The cross power density spectrum

cpdse : array-like

The error on the cross power density spectrum

pds1 : array-like

The power density spectrum of the first light curve

pds2 : array-like

The power density spectrum of the second light curve

maltpynt.fspec.**leahy_pds**(*lc*, *bintime*)
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

Returns
freqs : array-like

Frequencies corresponding to PDS

pds : array-like

The power density spectrum

maltpynt.fspec.**main**(*args=None*)
Main function called by the MPfspec command line script.

maltpynt.fspec.**read_fspec**(*fname*)
Read the frequency spectrum from a file.

Parameters
fname : str

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.fspec.**rms_normalize_pds**(*pds*, *pds_err*, *source_ctrate*, *back_ctrate=None*)
Normalize a Leahy PDS with RMS normalization (*[R1]*, *[R2]*).

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

Returns

pds : array-like

the RMS-normalized PDS

pds_err : array-like

the uncertainties on the PDS values

References

[R1], *[R2]*

maltpynt.fspec.**welch_cpds**(*time*, *lc1*, *lc2*, *bintime*, *fftlen*, *gti=None*, *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

An Object containing all return values below

f : array-like

array of frequencies corresponding to PDS bins

cpds : array-like

the values of the PDS

ecpds : array-like

the values of the PDS

ncpds : int

the number of summed PDSs (if normalize is False)

ctrate : float

the average count rate in the two lcs

dyncpds : array-like, optional

dynecpds : array-like, optional

dynctrate : array-like, optional

times : array-like, optional

Other Parameters

return_all : bool

if True, return everything, including the dynamical PDS

maltpynt.fspec.**welch_pds**(*time*, *lc*, *bintime*, *fftlen*, *gti=None*, *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 values below.

f : array-like

array of frequencies corresponding to PDS bins

pds : array-like

the values of the PDS

epds : array-like

the values of the PDS

npds : int

the number of summed PDSs (if normalize is False)

ctrate : float

the average count rate in the two lcs

dynpds : array-like, optional

dynepds : array-like, optional

dynctrate : array-like, optional

times : array-like, optional

Other Parameters

return_all : bool

if True, return everything, including the dynamical PDS

7.1.8 maltpynt.io module

Functions to perform input/output operations.

`maltpynt.io.`**`get_file_extension`**(*fname*)
Get the file extension.

`maltpynt.io.`**`get_file_format`**(*fname*)
Decide the file format of the file.

`maltpynt.io.`**`get_file_type`**(*fname*, *specify_reb=True*)
Return the file type and its contents.

Only works for maltpynt-format pickle or netcdf files.

`maltpynt.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 header structure, or a dictionary

keyword : str

The key to read in the header

Returns

value : long double

The value of the key

`maltpynt.io.`**`load_data`**(*fname*)
Load generic data in maltpynt format.

`maltpynt.io.`**`load_events`**(*fname*)
Load events from a file.

maltpynt.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

maltpynt.io.**load_gtis**(*fits_file*, *gtistring=None*)

Load GTI from HDU EVENTS of file fits_file.

maltpynt.io.**load_lcurve**(*fname*)

Load light curve from a file.

maltpynt.io.**load_pds**(*fname*)

Load PDS from a file.

maltpynt.io.**main**(*args=None*)

Main function called by the `MPreadfile` command line script.

maltpynt.io.**print_fits_info**(*fits_file*, *hdu=1*)

Print general info about an observation.

maltpynt.io.**read_from_netcdf**(*fname*)

Read from a netCDF4 file.

maltpynt.io.**save_as_ascii**(*cols*, *filename=u'out.txt'*, *colnames=None*, *append=False*)

Save arrays as TXT file with respective errors.

maltpynt.io.**save_as_netcdf**(*vars*, *varnames*, *formats*, *fname*)

Save variables in a NetCDF4 file.

maltpynt.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.io.`**`save_data`**(*struct*, *fname*, *ftype=u'data'*)
Save generic data in maltpynt format.

`maltpynt.io.`**`save_events`**(*eventStruct*, *fname*)
Save events in a file.

`maltpynt.io.`**`save_lcurve`**(*lcurveStruct*, *fname*)
Save light curve in a file.

`maltpynt.io.`**`save_pds`**(*pdsStruct*, *fname*)
Save PDS in a file.

`maltpynt.io.`**`sort_files`**(*files*)
Sort a list of MaLTPyNT files, looking at `Tstart` in each.

7.1.9 maltpynt.lags module

Functions to calculate lags.

`maltpynt.lags.`**`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.lags.`**`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 : str

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.lags.`**`main`**(*args=None*)
Main function called by the `MPlags` command line script.

7.1.10 maltpynt.lcurve module

Light curve-related functions.

`maltpynt.lcurve.`**`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.lcurve.**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:

`scrunch_lightcurves`

Create a single light curve from input light curves.

maltpynt.lcurve.**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.lcurve.**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

List of output light curves

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 `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.lcurve.**lcurve_from_fits**(*fits_file*, *gtistring=u'GTI'*, *timecolumn=u'TIME'*, *ratecolumn=None*, *ratehdu=1*, *fracexp_limit=0.9*, *outfile=None*, *noclobber=False*, *outdir=None*)

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 : [str]

Returned as a list with a single element for consistency with `lcurve_from_events`

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.lcurve.**lcurve_from_txt**(*txt_file*, *outfile=None*, *noclobber=False*, *outdir=None*)

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 : [str]

Returned as a list with a single element for consistency with `lcurve_from_events`

Other Parameters

outfile : str

Output file name

noclobber : bool

If True, do not overwrite existing files

maltpynt.lcurve.**main**(*args=None*)

Main function called by the `MPlcurve` command line script.

maltpynt.lcurve.**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:

`join_lightcurves`

Join light curves from different files

maltpynt.lcurve.**scrunch_main**(*args=None*)

Main function called by the `MPscrunchlc` command line script.

7.1.11 maltpynt.plot module

Quicklook plots.

maltpynt.plot.**main**(*args=None*)

Main function called by the `MPplot` command line script.

maltpynt.plot.**plot_cospectrum**(*fnames*, *figname=None*, *xlog=None*, *ylog=None*)

Plot the cospectra from a list of CPDSs, or a single one.

maltpynt.plot.**plot_generic**(*fnames*, *vars*, *errs=None*, *figname=None*, *xlog=None*, *ylog=None*)
Generic plotting function.

maltpynt.plot.**plot_lc**(*lcfiles*, *figname=None*, *fromstart=False*, *xlog=None*, *ylog=None*)
Plot a list of light curve files, or a single one.

maltpynt.plot.**plot_pds**(*fnames*, *figname=None*, *xlog=None*, *ylog=None*)
Plot a list of PDSs, or a single one.

7.1.12 maltpynt.read_events module

Read and save event lists from FITS files.

maltpynt.read_events.**main**(*args=None*)
Main function called by the `MPreadevents` command line script.

maltpynt.read_events.**treat_event_file**(*filename*, *noclobber=False*, *gti_split=False*, *min_length=4*, *gtistring=None*)
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

gtistring: str

comma-separated set of GTI strings to consider

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)

7.1.13 maltpynt.rebin module

Functions to rebin light curves and frequency spectra.

maltpynt.rebin.**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.

Parameters
x : array-like

y : array-like

factor : int

Rebin factor

yerr : array-like, optional

Uncertainties of y values (it is assumed that the y are normally distributed)

Returns

new_x : array-like

The rebinned x array

new_y : array-like

The rebinned y array

new_err : array-like

The rebinned yerr array

Other Parameters

normalize : bool

maltpynt.rebin.**geom_bin**(*freq*, *pds*, *bin_factor=None*, *pds_err=None*, *npds=None*)

Given a PDS, bin it geometrically.

Parameters

freq : array-like

pds : array-like

bin_factor : float > 1

pds_err : array-like

Returns

retval : object

An object containing all the following attributes

flo : array-like

Lower boundaries of the new frequency bins

fhi : array-like

Upper boundaries of the new frequency bins

pds : array-like

The rebinned PDS

epds : array-like

The uncertainties on the rebinned PDS points (be careful. Check with simulations if it works in your case)

nbins : array-like, optional

The new number of bins averaged in each PDS point.

Other Parameters

npds : int

The number of PDSs averaged to obtain the input PDS

Notes

Some parts of the code are copied from an algorithm in isisscripts.sl

maltpynt.rebin.**main**(*args=None*)

Main function called by the `MPrebin` command line script.

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

7.1.14 maltpynt.save_as_xspec module

Functions to save data in a Xspec-readable format.

maltpynt.save_as_xspec.**main**(*args=None*)
Main function called by the `MP2xspec` command line script.

maltpynt.save_as_xspec.**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

7.1.15 maltpynt.sum_fspec module

Function to sum frequency spectra.

maltpynt.sum_fspec.**main**(*args=None*)
Main function called by the `MPsumfspec` command line script.

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

7.1.16 maltpynt.version module

maltpynt.version.**get_git_devstr**(*sha=False*, *show_warning=True*, *path=None*)
Determines the number of revisions in this repository.

Parameters
sha : bool

If True, the full SHA1 hash will be returned. Otherwise, the total count of commits in the repository will be used as a "revision number".

show_warning : bool

If True, issue a warning if git returns an error code, otherwise errors pass silently.

path : str or None

If a string, specifies the directory to look in to find the git repository. If `None`, the current working directory is used, and must be the root of the git repository. If given a filename it uses the directory containing that file.

Returns

devversion : str

Either a string with the revision number (if sha is False), the SHA1 hash of the current commit (if sha is True), or an empty string if git version info could not be identified.

maltpynt.version.**update_git_devstr**(*version*, *path=None*)

Updates the git revision string if and only if the path is being imported directly from a git working copy. This ensures that the revision number in the version string is accurate.

7.1.17 Module contents

This is proposed as an Astropy affiliated package.

Part VI

Indices and tables

- genindex
- modindex
- search

[R1] Belloni & Hasinger 1990, A&A, 230, 103

[R2] Miyamoto+1991, ApJ, 383, 784

m

M

O

P

R

S

T

U

W