ixpeobssim Documentation

Release 33.0.0

The ixpeobssim team

May 29, 2026

INTRODUCTION

`ixpeobssim` is a simulation and analysis framework specifically developed for the Imaging X-ray Polarimetry Explorer (IXPE). Given a source model and the response functions of the telescopes, it is designed to produce realistic simulated observations, in the form of event lists in FITS format, containing a strict superset of the information included in the publicly released IXPE data products.

The core simulation capabilities are complemented by a suite of post-processing applications which support the spatial, spectral, and temporal models needed for analysis of typical polarized X-ray sources, allowing for the implementation of complex, polarization-aware analysis pipelines, and facilitating the inter-operation with the standard visualization and analysis tools traditionally in use by the X-ray community.

Important

`ixpeobssim` is made available and maintained on a best effort basis under the GNU General Public License v3, in the hope that it will be useful, but it is not officially maintained by either NASA or ASI.

The IXPE page on the HEASARC should be considered the ultimate source of information for everything concerning the observatory and the analysis of the science data it provides. Should there be any discrepancy between the information included in the `ixpeobssim` code and documentation and that provided by the HEASARC, the latter should be assumed to be correct. (And we kindly ask users to report any such instance on our issue tracker, so that we can fix it.)

The `ixpeobssim` development takes place on a git repository hosted on on github. The *Installation*, *Overview* and *Quick start* pages should be enough to get you up and running. If that is not the case, or should you encounter any issue at all, we encourage you to use our issue tracker to file for bug reports and feature requests.

For convenience, a static version of the latest documentation in pdf format is available at this link.

Latest changes

ixpeobssim (33.0.0) - Fri, 29 May 2026 06:29:25 +0200

- Merging pull request https://github.com/lucabaldini/ixpeobssim/pull/747
- Adding new response files for calendar years 2025 and 2026, that should match precisely what is available on the HEASARC CALDB for the same time intervals (modulo the fact that ixpeobssim uses the arf files, while ixpecalcarf recalculates everything starting from the modulation factor and the quantum efficiency).
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/738

ixpeobssim (32.0.1) - Tue, 14 Apr 2026 12:32:28 +0200

...

See the *Release notes* for a full change log.

If you found `ixpeobssim` useful, feel free to include an acknowledgment in papers and/or presentations—here is the most up-to-date reference.

CHAPTER

ONE

OVERVIEW

The basic simulation and analysis facilities implemented in ixpeobssim are made available through the following Python script:

- *xpobssim*: produce a photon list for a specific observation time, given a *source model* and a set of *instrument response functions*;
- *xpphase*: calculate the phase of a periodic source based on ephemeris and generates a new FITS file appending the phase column.
- *xpophase*: calculate the orbital phase for binary systems;
- *xpeselect*: select sub-samples of photons in a given event file, based on the event energy, direction, time or phase, producing a new event file;
- *xpbin*: allows to bin the data using several different algorithms, producing as output files *binned events lists*;
- *xpbinview*: provide the visualization interface to the binned data given as input files;
- *xppimms*: calculate the MDP for a a specific observation time, given a *source model* and a set of *instrument response functions*;
- *xpmdp*: calculate the MDP for a a specific observation time, assuming a power-law spectrum with adjustable parameters;
- *xpxspec*: perform a spectro-polarimetric fit in XSPEC.

Where applicable, the data formats are consistent with the common display and analysis tools used by the community, e.g., the binned count spectra can be fed into XSPEC, along with the corresponding response functions, for doing standard spectral analysis (note that the response files used are the *same* for the simulation and the analysis tasks.)

All the ixpeobssim simulation and analysis tools are fully configurable via command-line and the corresponding signatures are detailed here. In addition, ixpeobssim provides a pipeline facility that allow to script in Python all the aforementioned functionalities (e.g., for time-resolved polarimetry this would mean: create an observation simulation for the system under study, run *xpeselect* to split the photon list in a series of phase bins, run *xpbin* to create suitable modulation cubes for each of the data subselections and analyze the corresponding binned output files).

CHAPTER

TWO

SHOWCASE

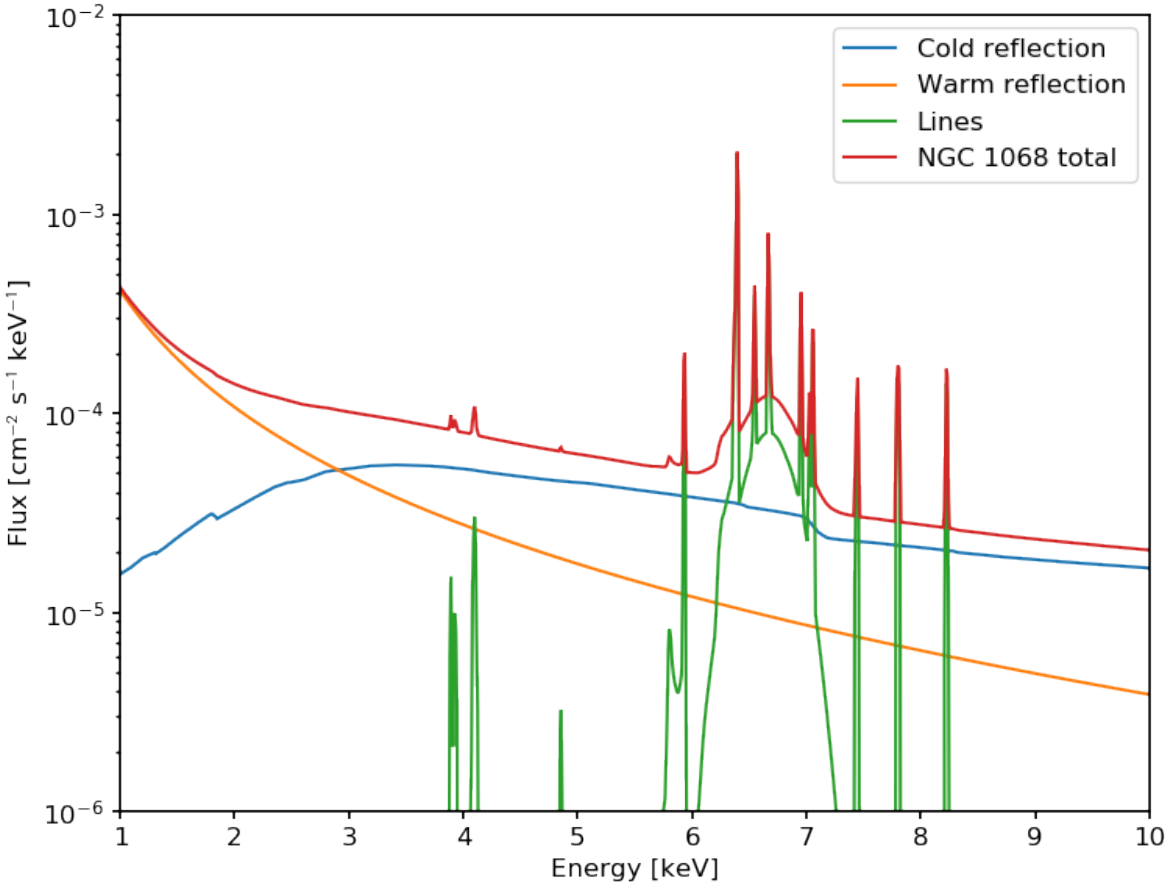
2.1 NGC 1068

- Source file: [github]/ixpeobssim/config/ngc1068.py
- Simulation/analysis pipeline: [github]/ixpeobssim/examples/ngc1068.py

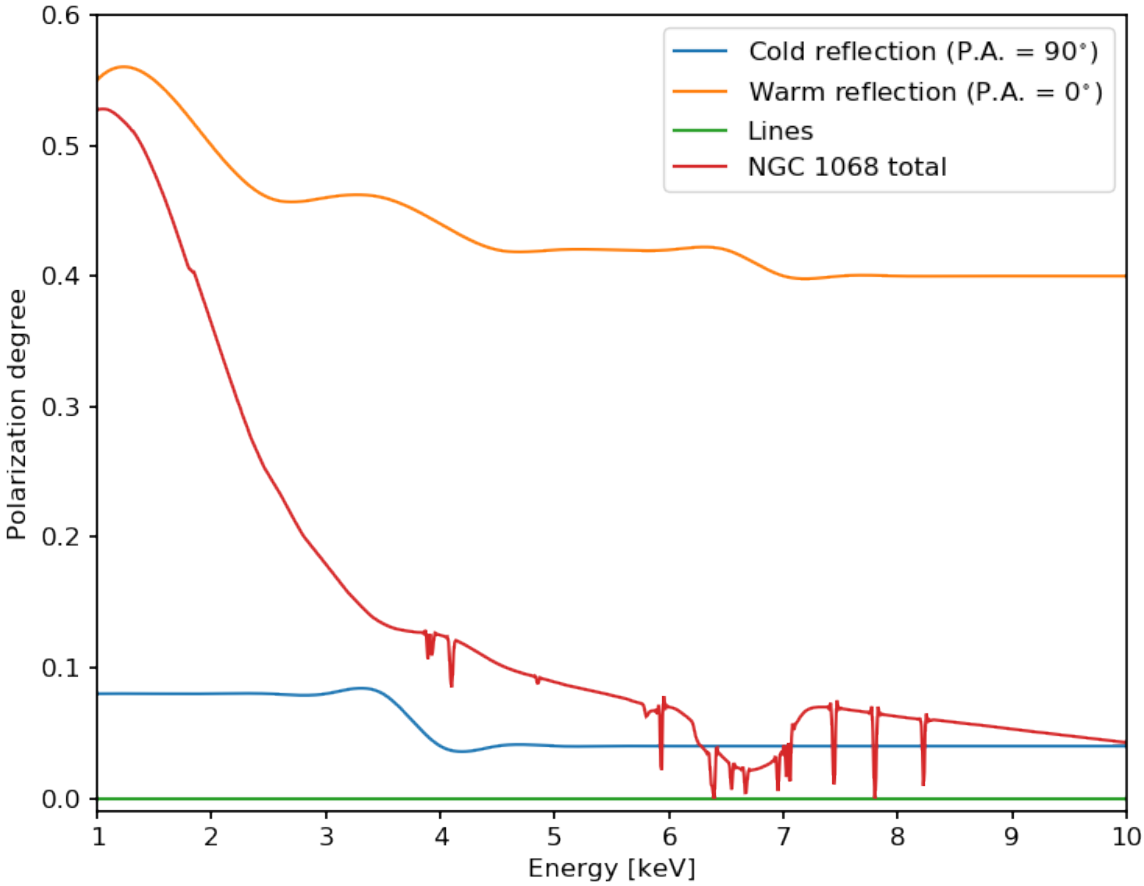
This is an example of a spectro-polarimetric model of a Compton-thick AGN, and a well-known one, where we have detailed information about the source geometry and energy spectrum.

There are three distinct spectral components to the model, each one with its own polarimetric pattern:

- the cold reflection from the torus;
- the warm reflection from the cone;
- the emission lines.



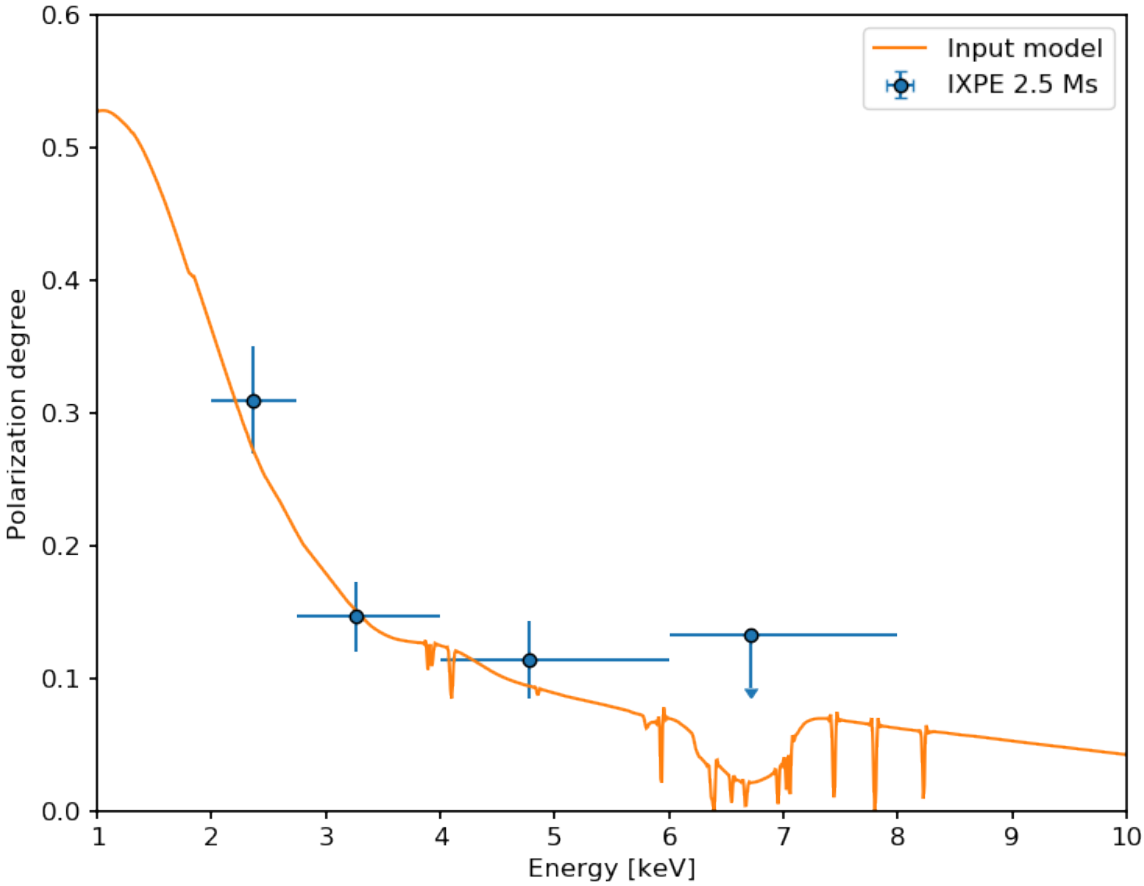
As far as the geometry for the polarimetric modeling is concerned, the ionization cone is assumed to be perpendicular to the plane of the torus, and the inclination angle of the system is 75 degrees.



We note that the position angles for the cold and warm relflection components are orthogonal to each other, so that this source is a good illustration of the harmonic addition in ixpeobssim.

2.1.1 Simulation output

Below is an output of an ixpeobssim simulation, where the measured polarization degree, estimated in a few energy bins, is compared to the input model.



References

[1] Ghisellini, G.; Haardt, F.; Matt, G., "The Contribution of the Obscuring Torus to the X-Ray Spectrum of Seyfert Galaxies - a Test for the Unification Model", Monthly Notices of the Royal Astronomical Society, Vol. 267, NO. 3/APR1, P. 743, 1994

[2] Goosmann, R. W.; Matt, G., "Modeling X-ray polarimetry while flying around the misaligned outflow of NGC 1068", SF2A-2011: Proceedings of the Annual meeting of the French Society of Astronomy and Astrophysics, 2011

#Crab pulsar #———— # #* Source file: [github]/ixpeobssim/config/crab_pulsar.py #* Simulation/analysis pipeline: [github]/ixpeobssim/examples/crab_pulsar.py # #.. automodule:: ixpeobssim.config.crab_pulsar

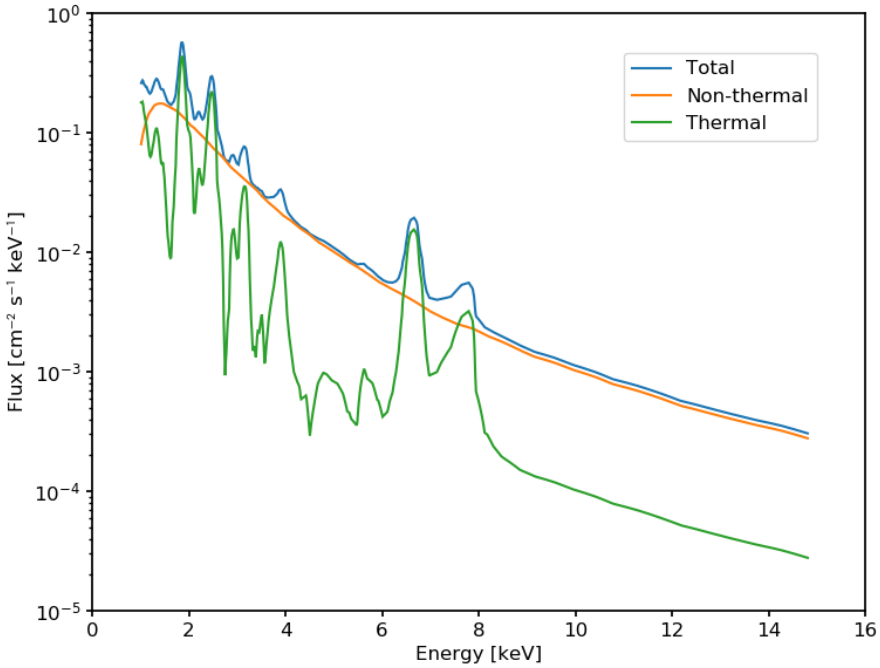
2.2 Cas A

Full source model definition can be found in ixpeobssim.config.toy_casa.py.

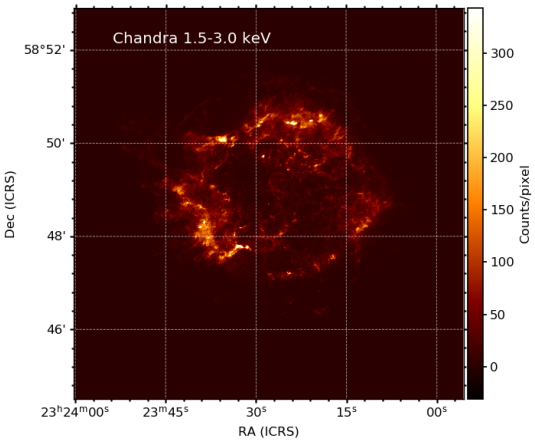
The spectral model is taken from E.A. Helder and J. Vink, "Characterizing the non-thermal emission of Cas A", Astrophys.J. 686 (2008) 1094–1102, http://arxiv.org/abs/0806.3748. The spectrum of Cas A is a complex superposition of thermal and non thermal emission, and for our purposes, we call thermal anything that is making up for the lines and non-thermal all the rest.

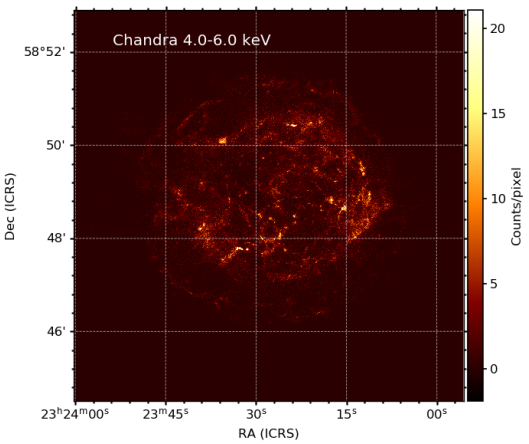
We have two images of Cas A, at low (1.5–3.0 keV) and high (4.0–6.0 keV) and, due to the absence of lines between 4 and 6 keV we're attaching the latter to the non-thermal spectrum and the former to the thermal component.

Here is the photon spectrum we are using for the simulation of Cas A:

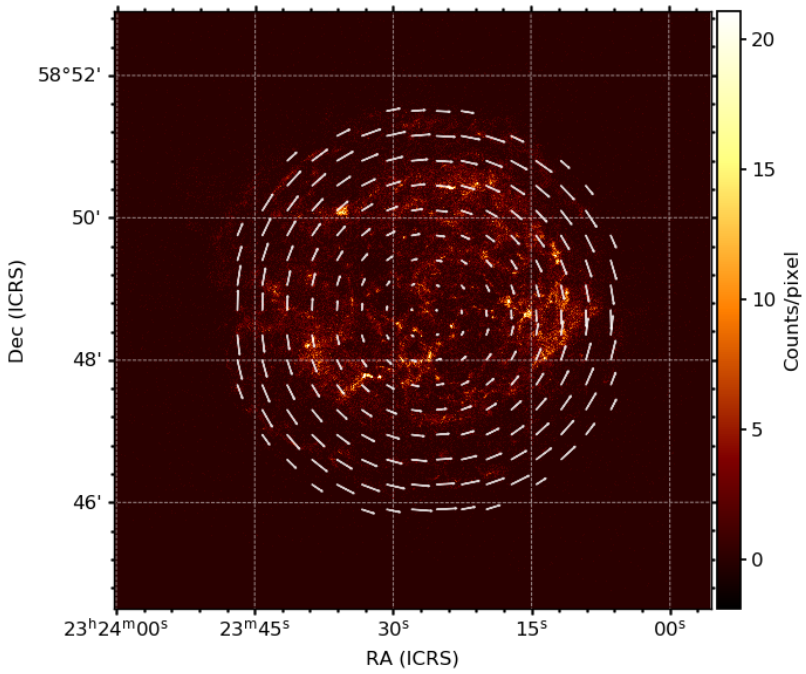


The morphology of the source is energy-dependent in a non trivial way. We start from two Chandra images (in the 1.5–3 keV and 4–6 keV energy ranges, respectively) and associate the former to the thermal spectral component and the latter to the non-thermal one (note the absence of spectral lines between 4 and 6 keV).





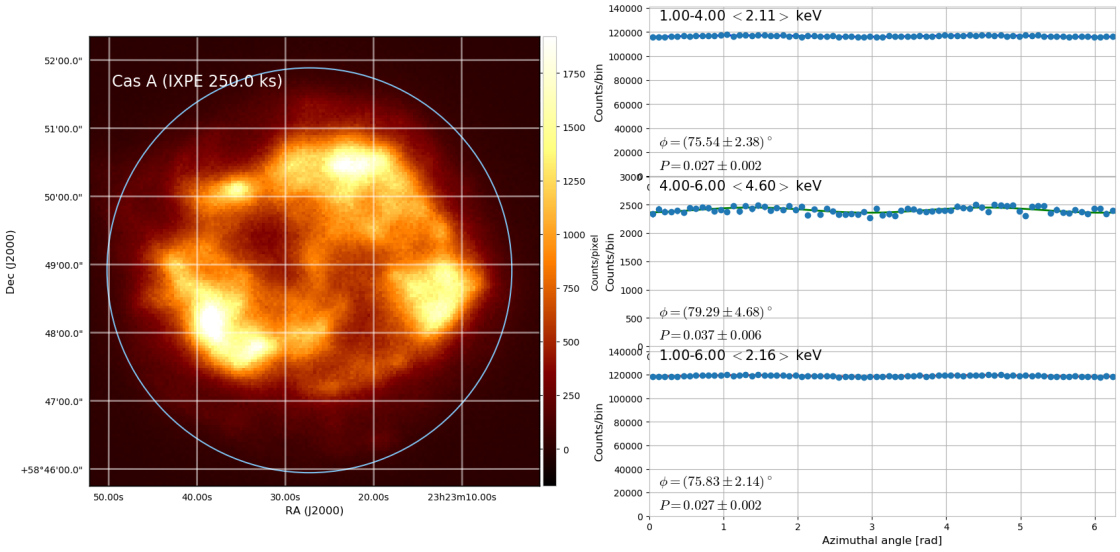
For the polarization, we assume that the thermal component is unpolarized, while for the non-thermal component we use a simple geometrical, radially symmetric model (loosely inspired from radio observations) where the polarization angle is tangential and the polarization degree is zero at the center of the source and increases toward the edges reaching about 50% on the outer rim of the source (see figure below).



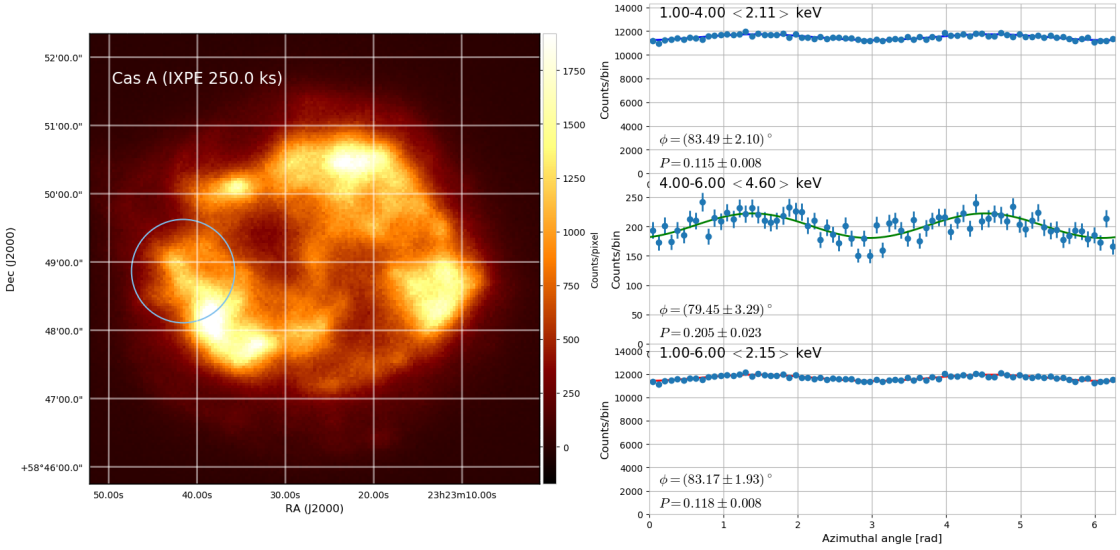
Our total model of the region of interest is therefore the superposition of two indipendent components, with different spectral, morphological and polarimetric properties. Crude as it is, it's a good benchmark for the observation simulator.

Simulation output

Below is a binned count map of a 250 ks simulated IXPE observation of Cas A, based on the model described above. When the entire source is analyzed at once, most of the polarization averages out and even in the high-energy band, where the emission is predominantly non-thermal, the residual polarization degree resulting from the averaging of the different emission regions is of the order of 5%.



On the other hand, spatially- and energy-resolved polarimetry would in this case reveal much of the richness in the original polarization pattern. Below is an example of the azimuthal distributions in the two energy bands for the circular region of interest indicated by the white circle in the left plot. For reference, the corresponding flux integrated in the region is about 3.5% of that of the entire source. The comparison with the previous, spatially averaged distributions is striking.

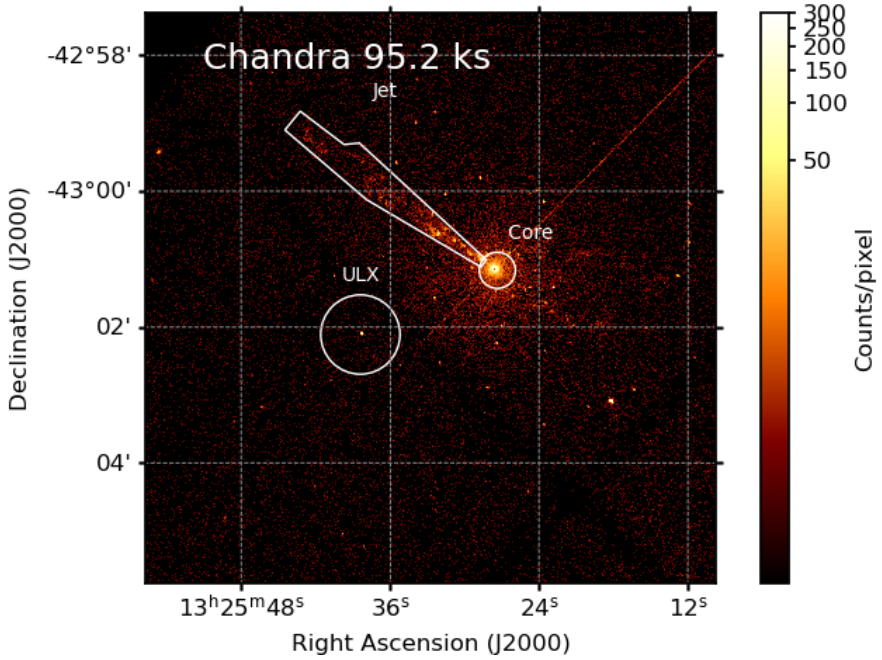


By mapping the entire field of view with suitable regions of interest we can in fact (at least qualitatively) recover the input polarization pattern.

2.3 Cen A

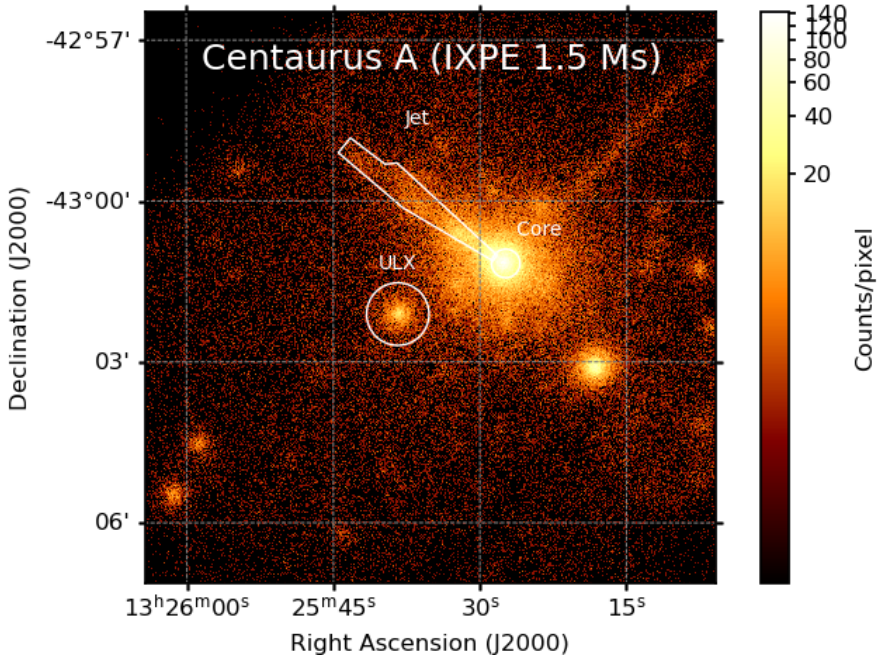
- Source file: [github]/ixpeobssim/config/cena.py
- Simulation/analysis pipeline: [github]/ixpeobssim/examples/cena.py

Here we illustrate an example of how to convert a Chandra observation using the xpobssim tool. The simulation concerns the Centaurus A source, based on a Chandra event file taken from the Chandra database (for reference the observation id is 8489). With the xpobssim tool you can also define regions within the ROI using regions. In this example we defined the jet and core regions and the ULX star as can be seen in the figure below.



Simulation output

The plot below refers to a 1.5 Ms simulation. We selected the same three regions to perform spatially resolved polarimetry, marked in the figure below:



For each of these regions we calculated the MDP value. In this simulation we have also included the instrumental background and when calculating the MDP we take these photons as the background. The results are resumed in following snippet.

```
Jet region:
2.00--8.00 keV: 12487 src counts (99.5%) in 1.5 Ms, MDP 11.13%
Core:
2.00--8.00 keV: 25844 src counts (100.0%) in 1.5 Ms, MDP 6.46%
ULX:
2.00--8.00 keV: 4370 src counts (99.2%) in 1.5 Ms, MDP 19.24%
```

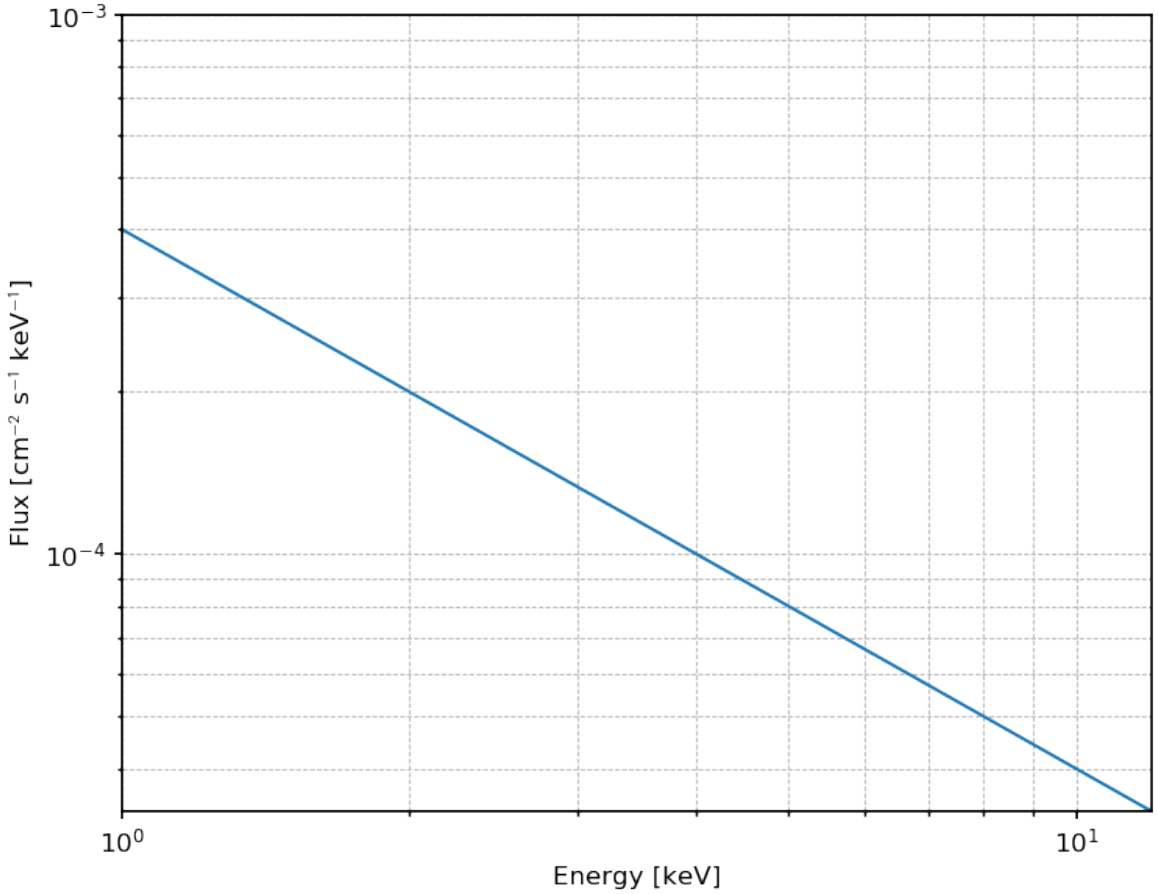
2.4 Instrumental background

The various incarnations of the instrumental background are peculiar among source components in that the simulation happens in detector (as opposed to sky) coordinates, and the resulting event list is not convolved with any of the instrument response function.

The instrumental background is implemented as an instance of the *ixpeobssim.srcmodel.bkg.xInstrumentalBkg* class, or any of its subclasses. At simulation time, the object effectively produces a uniform, non-polarized source in the GPD reference frame.

- Source file: [github]/ixpeobssim/config/instrumental_bkg.py
- Simulation/analysis pipeline: [github]/ixpeobssim/examples/instrumental_bkg.py

This is the simplest possible realization of the instrumental background, where the energy spectrum is a power law fitted to the data from Bunner et al. (1978ApJ...220..261B). Here the authors provide the non X-ray background rates for their three detectors and we are using values for the Neon-filled detector in Table 3 of the paper.



For completeness: this model component can be imported in any ixpeobssim configuration file by simply adding the following line:

```
from ixpeobssim.config.instrumental_bkg import bkg
```

Then all you need to do is to add the bkg source to your region of interest.

Warning

Care must be taken in interpreting the output of instrumental background simulations. While the original distribution is flat in detector coordinates, when the photons are looked at in sky coordinates the dithering of the observatory, if enabled, will produce a smearing at the edges of the field of view. In addition, since the default photon spectrum is fairly hard, the energy dispersion can cause noticeable deviations from the input power law.

CHAPTER

THREE

INSTALLATION

If you are in a rush, and to make a long story short, as of version 29.2.0 ixpeobbsim is hosted on PyPI and you can install it via pip

```
pip install ixpeobssim --user
```

This should install all the necessary dependences and work out of the box (we recommend using the `--user` option not to pollute everybody else's environment on the host machine, but you're definitely free to opt for a system-wide installation by simply omitting this specific option).

Warning

ixpeobssim includes NumPy as a binary dependence, and it is generally not recommended to install Numpy from pip. Should you encounter problems with the Numpy installation, you might consider to use a science-ready version of Python that comes with NumPy (and the associated Python scientific ecosystem) built-in, like the Anaconda distribution.

That said, most of the remaining of this section is left to the reader for historical reasons, and in the hope it will be useful to debug installation problems, should any arise.

3.1 Prerequisites

The package is based on the Python scripting language and the SciPy Python-based ecosystem. You will need a working Python installation including several different third-party packages, most notably:

- NumPy: the fundamental package for scientific computing with Python.
- SciPy: a Python-based ecosystem of open-source scientific software.
- matplotlib: a Python 2D plotting library.
- Astropy: a Python astronomy package (including tools to manipulate FITS files).
- regions: an Astropy affiliated package for handling region (including ds9 region files).
- skyfield: a package computing positions for stars, planets and satellites.
- PyXspec: the Python binding for XSPEC (optional).

Loosely speaking you should be able to open the Python terminal and execute the following `import` statements with no errors.

```
>>> import numpy
>>> import scipy
>>> import matplotlib
>>> import astropy
>>> import regions
>>> import skyfield
```

If any of the required packages fails to import, take a deep breath and fix the issue before you move on.

3.1.1 Dependencies in depth

Warning

ixpeobbsim as a project started in late 2015 when requiring users to install Python 3 was more a political decision that anything else. Over the years, it has evolved along with the Python ecosystem and the Python language itself.

Although we have not deliberately taken any action to break backward compatibility with older versions of the dependencies (and in fact, you might very well find that ixpeobbsim is still working with Python 2), you are strongly encouraged to make sure that your setup matches the indications in this section, as new features will not be tested against older environments.

We recommend installing Python 3.6 or later. In case you missed it, the Python 2x series reached the end of life on January 1, 2020, which means that no new bug reports, fixes, or changes will be made to Python 2, and Python 2 is no longer supported. If you are still using it, you should definitely consider upgrading to Python 3 for your projects.

As far as the dependencies are concerned, our requirement specification reads:

```
numpy>=1.17
scipy>=1.4
matplotlib>=3.0
astropy>=4.0
regions>=0.4
skyfield>=1.23
```

You can easily find out which version of any specific package you are using through the interactive Python interpreter, e.g.:

```
[lbaldini@nblbaldini ixpeobssim]$ python
Python 3.8.3 (default, May 15 2020, 00:00:00)
[GCC 10.1.1 20200507 (Red Hat 10.1.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> numpy.__version__
'1.18.4'
>>>
```

In addition, there is a unit test in the test folder (`test/test_environment.py`) that you can run to diagnose your environment.

3.2 Installing Python through Anaconda

> **Tip**
>
> Making a long story short: we encourage people to use Python 3 and install Python and the associated ecosystem through Anaconda. If you know what you are doing you can find your way through the framework through alternative ways (e.g., the Python installation provided by your GNU/Linux distribution) feel free to do so, but we do recommend Anaconda as a decent, platform-independent Python experience.

You can download the installer for your os from the download page (make sure you pick the Python 3 branch) and follow the installation instructions.

In a nutshell, Anaconda will create on your hard drive a self-contained directory structure with the Python executable and associated goodies, and all you have to do to use it is to prepend the folder containing the Python executable itself to your `PATH` environmental variable. The installer is pretty simple in that it comes in the form of a single bash script (under GNU/Linux and Mac) or a self-extracting installer (under Windows) that you execute and does everything for you—just follow the instructions.

It is your responsibility to have the proper environment set up for Anaconda Python. As mentioned in the previous paragraph, this simply boils down to have the `PATH` environmental variable pointing to the folder containing the Python executable, which is

- the `bin` folder in the Anaconda installation folder under GNU/Linux and Mac;
- the top-level Anaconda installation folder under Windows.

Once you have correctly done that, you should be able to launch Python from the shell or the command prompt and see something in response along these lines:

```
[lbaldini@nblbaldini doc]$ python
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 18:10:19)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

3.3 Installing the Python dependencies

The default Anaconda installer comes with most of the Python scientific ecosystem that you need (i.e., numpy, scipy, matplotlib and astropy among many other packages).

For the missing bits, Anaconda comes with the `pip` utility that allows you to install them in a matter of seconds (with the exception of the Python bindings for Xspec, see the following section). If you're starting from a fresh installation you will typically need to do

```
pip install regions --user
pip install skyfield --user
```

and you should be all set.

> **Warning**
>
> If you have an error when trying and install regions, you might want to try and do

```
pip install wheel --user
```

(There are more extensive resources on the web about this very issue, including this)

Under Windows, depending on your installation, you might come across the infamous
`error: Unable to find vcvarsall.bat`

while installing regions or skyfield. This is essentially because pip is trying to compile part of the library, and you can find all the gory details of the issue here. In a nutshell, assuming that you are using Python 3.5 or later, you will need to install the Visual C++ Build Tools 2015.

3.4 Installing the Xspec Python bindings

This is an interesting corner of the pre-requisite installation, as PyXspec is used in some of the ixpeobssim analysis-related modules, and it is not readily available through Anaconda.

Note

Getting PyXspec up and running might prove less than trivial depending on your OS and the details of your installation. It is important to point out that you can get away through the vast majority of the ixpeobssim functionalities without PyXspec, and if you have Xspec installed you might use that netively after all.

If, on the other end, you are interested in making full use of the ixpeobbsim analysis pipelines, detailed instructions about how get the Xspec Python bindings up and running follow.

At any time you can check whether you have PyXspec up and running on your ixpeobssim installation by doing something along the lines of

```
>>> from ixpeobssim.utils.environment import PYXSPEC_INSTALLED
>>> if PYXSPEC_INSTALLED:
>>>     import ixpeobssim.evt.xspec_ as xspec_
```

Please keep in mind to protect the PyXspec-related sections of your code with a similar guard in order not to break everything for the users who do not have PyXspec installed.

The PyXspec Python bindings for Xspec are fully integrated into the general HEASOFT build procedure and they will be built and installed automatically with the rest of XSPEC/HEASOFT. Note that the source code distribution of XSPEC is required for using PyXspec. What you want to do is to go to the HEASOFT download page and:

- select the source code distribution software type (STEP 1) for your target architecture;
- select the desired packages (XANADU/Xspec at the very minimum) at STEP 2;
- click the submit button to get the installation tarball tailored to your choices.

At this point you will have an archive, say `heasoft-<version>src.tar.gz`, that you should decompress into your target installation folder. There are detailed installation instructions for all the major platforms, including

- Red Hat-based GNU/Linux
- Debian-based GNU/Linux
- Mac OS X
- Windows with cygwin

You should look carefully at the list of prerequisites for your target OS before you start compiling the code.

3.5 Compiling PyXspec under GNU/Linux

Compiling is pretty easy, the only real thing to pay attention to being the fact that by default Xspec will try to create the bindings for Python 2 by default, which is not what you want if you have been following the instructions upstream. In order to use Python 3 the correct sequence of events is to configure the build process

```
cd BUILD_DIR/
./configure
```

and, before you start the compilation, tweak the Xspec configuration file `heasoft-<version>/Xspec/BUILD_DIR/hmakerc` and explicitly set the `PYTHON_INC` and `PYTHON_LIB` variables along the lines of

```
PYTHON_INC="-I/path/to/your/anaconda_py3/include/python3.6m"
PYTHON_LIB="-L/path/to/your/anaconda_py3/lib -lpython3.6m"
```

At this point you are ready to trigger the actual build process, i.e.,

```
make
make install
```

Refer to the PyXspec installation page for all the gory details and the troubleshooting information. Once everything is compiled and installed you will have to source the proper setup file, e.g.

```
export HEADAS=/path/to/your/installed/heasoft-<version>/<platform>
. $HEADAS/headas-init.sh
```

See the section about *XSPEC support* for a more in-depth discussion about the interactions between ixpeobssim and XSPEC.

3.6 Installing ixpeobssim

As mentioned at the beginning, the easiest way to install ixpeobssim is via pip. If you plan on actively contributing to the software development (as opposed to just using it) you will need to clone the github repository, as explained in the *Code development* page.

3.6.1 Installing ixpeobssim as a user

Installing ixpeobbsim as a user should be as simple as doing

```
pip install ixpeobssim --user
```

Loosely speaking, if at this point you can open a Python prompt and do

```
>>> import ixpeobssim
```

without getting back an error message like this

```
>>> import ixpeobssim
    Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
    ImportError: No module named ixpeobssim
>>>
```

you should be all set. You should be able to call all the executables from their entry points (without the .py extension) as well, e.g.

```
xpobssim --help
```

In case, for any reason, you do need to uninstall the package, just type

```
pip uninstall ixpeobssim
```

If you installed PyXspec and you want to use it, make sure also to define the HEADAS path to HEASOFT package and run the corresponding setup script, e.g.

```
export HEADAS=/path/to/your/installed/heasoft-<version>/<platform>
. $HEADAS/headas-init.sh
```

Warning

When you install `ixpeobbsim` in user mode you want to make sure you don't have any leftover from previous installation, as that is bound to lead to cryptic errors that would be hard to debug. Make sure you uninstall any old version and you don't leave any .pyc file behind.

Likewise, if you ever decide to switch from user to developer mode (see next section), make sure you clean up any previous installation first.

3.6.2 Up and running in development mode

If you are actively developing code, having it installed (possibly within system folders) is unlikely to be what you want. Once you have cloned the github repository all you really need to do in this case is to make sure that the root folder of the repository is included in the `PYTHONPATH` environmental variable.

ixpeobssim comes with its own minimal shell ([github]/setup.sh, for GNU/Linux or Mac) and batch ([github]/setup.bat, for Windows) setup scripts that you're welcome to take advantage of—you should be up and running by just doing

```
source setup.sh
```

under GNU/Linux or Mac, and

```
setup.bat
```

under Windows. (Mind this will also add the path to the folder with the ixpeobssim executable scripts to the `PATH` evironmental variable.)

Warning

If you're operating in development mode you won't have the entry points for the ixpeobssim applications installed, e.g., you won't be able to call the executables without the .py extensions.

Most of the documentation is written from the user standpoint, so keep this in mind: whenever you read, e.g.,

```
xpobssim --help
```

if you're working in developer mode you will actually need to type

```
xpobssim.py --help
```

3.6.3 Changing the default output directory

By default, ixpeobssim will create an `ixpeobssimdata` folder in your `HOME` directory and stuff all the output files (e.g., FITS event lists) in there—unless you specify the path to the output files from the command line.

If you want to change the default behavior and select your favourite directory for storing all the output files, you just need to define the `IXPEOBSSIM_DATA` environmental variable pointing to the directory itself.

```
export IXPEOBSSIM_DATA=/path/to/the/new/output/folder
```

CHAPTER

FOUR

QUICK START

The main purpose of this simulation package is to simulate an observation of a given source model, based on suitable detector response functions.

4.1 xppimms

As a first test case we can evaluate the MDP for a single point source using the *xppimms* tool. Its capabilities and options are described in the *Application reference* section of the documentation.

In order to reproduce the results reported here for a standard source (power law spectrum with index 2 and normalization 10 at 1 keV, no absorption), in 100 ks and in the 2-8 keV energy range, we run the command

```
xppimms --duration 10000
```

Which prints out the following MDP table:

```
...
2.00--2.83 keV:      494.5 counts, effective mu = 0.209, MDP = 92.42%
2.83--4.00 keV:      236.8 counts, effective mu = 0.333, MDP = 83.67%
4.00--5.66 keV:       77.8 counts, effective mu = 0.407, MDP = 119.53%
5.66--8.00 keV:       18.2 counts, effective mu = 0.477, MDP = 211.20%
2.00--8.00 keV:      827.3 counts, effective mu = 0.269, MDP = 55.47%
```

This program calculates the MDP by direct numerical integration of the power-law input spectrum. As a consequence, part of the richness of the detector response (most notably, the energy dispersion and the effective area vignetting) is not captured here. For use cases beyond simple stationary point sources, the use of *xpobssim* and *xpbin* in PCUBE mode are recommended, as this approach offers the maximum flexibility (see next section for an example).

4.2 xpobssim

The main Monte Carlo simulation application is *xpobssim*. For a quick reference on this tool see *Application reference* section. Assuming that the current working directory is the ixpeobssim root folder, the command

```
xpobssim --configfile ixpeobssim/config/toy_point_source.py --duration 100000
```

Will produce three event (FITS) files (one for each Detector Unit) with a 100 ks simulation of a stationary source with a power-law spectrum (with an index of 2 and normalization of 10) with energy- and time-independent polarization degree (0.5) and angle (30°), correctly folded with all the instrument response functions: effective area, modulation factor, energy dispersion and point-spread function.

In order to bin the event files in energy (from 2 keV to 8 keV in one single bin) we run the command

```
xpbin $IXPEOBSSIM_DATA$/toy_point_source_du?.fits --algorithm PCUBE --emin 2. --emax 8. -
↪-ebins 1
```

that will produce three new FITS files (called modulation cubes), containing the MDP value (at 99%) and the histogram of the azimuthal distribution of photoelectrons emission for every bin.

The binned output files can be easily visualized using the *xpbinview* tool:

```
xpbinview $IXPEOBSSIM_DATA$/toy_point_source_du?_pcube.fits
```

We are already fully equipped for a basic spectral analysis with XSPEC. The first step is to bin again the event files by running the *xpbin* tool with the PHA1 algorithm.

```
xpbin $IXPEOBSSIM_DATA$/toy_point_source_du?.fits --algorithm PHA1
```

We can feed the binned files (along with the corresponding .arf and .rmf response functions) into XSPEC and recover the input parameters of our source.

```
xpxspec $IXPEOBSSIM_DATA$/toy_point_source_du?_pha1.fits
```

CHAPTER

FIVE

DATA FORMAT

Below is an (automatically-generated) dump of the `ixpeobssim` output data format.

Note

While we have tried since the very beginning of the code development to keep the data format of the output event list synchronized with that of the actual files that we were planning to distribute to the community, `ixpeobssim` does not in fact produce plain level-2 files.

5.1 IXPE Level-2 data format

IXPE filtered level-2 files are fairly minimal, in that contain two extensions:

- `EVENTS`: contains the event data;
- `GTI`: contains the good time intervals for the observation;

The `EVENTS` extension comes in the form of a binary table with 10 columns:

- `TRG_ID` (1J): the trigger identifier. The use of trigger ID, instead of event ID, is to emphasize that the DAQ can discard triggers based on the ROI size.
- `TIME` (1D): sum of `SEC` and `MICROSEC` [s].
- `STATUS` (16X): 16-bits of processing status/error flags.
- `STATUS2` (16X): 16-bits of processing status/error flags.
- `PI` (1J): pixel-equalization and gain-corrected event signal.
- `W_MOM` (1E): statistical weight of this event (from moments analysis).
- `X` (E): calculated position, projected onto the J2000 tangent plane axis parallel to celestial equator using the preliminary aspect correction [pixel].
- `Y` (E): calculated position, projected onto the J2000 tangent plane axis parallel to celestial equator using the preliminary aspect correction [pixel].
- `Q` (D): value of Stokes parameter q in J2000 tangent plane axis.
- `U` (D): value of Stokes parameter u in J2000 tangent plane axis.

Note the primary header does not include the WCS information to map `X` and `Y` in the sky. Note a few differences remain between 'EVENTS' columns format, in data files and in ixpeobssim simulated files.

5.2 ixpeobssim event lists

Event lists consists of a primary header and a number of distinct binary tables. The two main extensions (EVENTS and GTI) mimic the expected content of real celestial observations, although the former contains many more columns, partly for historical reasons, and partly as a mean to provide useful debug information specific to simulated data.

> **Note**
>
> The main interface to event lists, *ixpeobssim.evt.event.xEventFile* is designed with the level-2 files in mind, and we strive to use as much as possible the information in there, in such a way that the analysis tools can interoperate transparently with both real and simulated data.

Additional extensions, specific to simulated data, include:

- MONTE_CARLO: contains the ground truth (e.g., true energy and sky direction) for all the events, which is useful for development and debugging purposes (it goes without saying, this will not be available for flight data);
- ROITABLE: contains the mapping from the values in the SRC_ID column in the MONTE_CARLO extension and the name of the corresponding source in region of interest, which is useful to map the events into the corresponding model components;
- TIMELINE: contains the timeline of the observation in the form of a series of homogeneous epochs with a given SAA and Earth occultation flags (the table is optional and its generation is controlled by the `--timelinedata` command-line switch in *xpobssim*);
- SC_DATA: contains the basic spacecraft data (e.g., position and pointing) on a regular time grid (the table is optional and its generation is controlled by the `--scdata` command-line switch in *xpobssim*, while the step of the grid is controlled by the `--scdatainterval` switch);
- OCTI: contains the on-orbit calibration time interval for a given DU (the table is only generated if the `--onorbitcalib` *xpobssim* command-line switch is set);
- CHRG_MAP: contains the charging map at the end of the observation, in a form that can be used as an input for subsequent simulations (the table is only generated if the `--charging` *xpobssim* command-line switch is set);

While the content on the TIMELINE and SC_DATA contain partially overlapping information (the SAA and Earth occultation flags) they are fundamentally different and serve two different purpose: the former is the most succinct summary of the different states a given observation is traversing, while the latter is a mere sampling of a few interesting quantities on a regularly spaced time grid.

We also emphasize that the GTI and OCTI do not necessarily fill the entirety of the time intervals that can be in principle allocated for science data taking and calibration, the exact behavior of the observatory being controlled by the `--gtiminduration`, `--gtistartpad`, `--gtistoppad`, `--onorbitcaldemult`, `--onorbitcalminduration`, `--onorbitcalstartpad`, `--onorbitcalstoppad`, and `--onorbitcalrate` flags of *xpobssim*.

> **Note**
>
> ixpeobssim provide a small application, *xpobsview* to take a quick-look to the timeline of an observation.

5.3 Primary extension

5.3.1 Header keywords

- `OBS_ID`: *Observation ID*
- `CONTNUMB`: *Contact number*
- `OBS_MODE`: *Observation mode*
- `SRC_CONF`: *Source configuration*
- `ORIGIN`: *Organization responsible for the data*
- `CALDBVER`: *CALDB version*
- `CLOCKCOR`: *Whether the time has been corrected*
- `TELESCOP`: *Telescope name*
- `INSTRUME`: *Instrument name*
- `DETNAM`: *Name of the logical detector unit*
- `DET_ID`: *Name of the physical detector unit*
- `TSTART`: *Observation start time in MET*
- `TSTOP`: *Observation end time in MET*
- `DATE-OBS`: *Observation start datetime*
- `DATE-END`: *Observation end datetime*
- `TELAPSE`: *TSTOP-TSTART*
- `TIMESYS`: *Time system*
- `TIMEUNIT`: *Time units*
- `TIMEREF`: *Time reference*
- `MJDREFI`: *MJD ref day 01 Jan 2017 00:00:00 UTC*
- `MJDREFF`: *Frac part of MJD ref (32.184secs+37leapsecs)*
- `TIMEZERO`: *Zero time*
- `ONTIME`: *On source time*
- `LIVETIME`: *On source time corrected for dead time*
- `DEADC`: *Dead time correction*
- `DEADAPP`: *Has DEADC been applied to data*
- `RA_OBJ`: *[deg] R.A. Object*
- `DEC_OBJ`: *[deg] Dec Object*
- `RA_PNT`: *[deg] RA pointing*
- `DEC_PNT`: *[deg] Dec pointing*
- `OBJECT`: *Name of observed object*
- `FILE_LVL`: *File level*
- `LV2_VER`: *Version of the LV2 data format*

5.4 EVENTS extension

5.4.1 Header keywords

- TELESCOP: *Telescope name*
- INSTRUME: *Instrument name*
- DETNAM: *Name of the logical detector unit*
- DET_ID: *Name of the physical detector unit*
- TSTART: *Observation start time in MET*
- TSTOP: *Observation end time in MET*
- DATE-OBS: *Observation start datetime*
- DATE-END: *Observation end datetime*
- TELAPSE: *TSTOP-TSTART*
- TIMESYS: *Time system*
- TIMEUNIT: *Time units*
- TIMEREF: *Time reference*
- MJDREFI: *MJD ref day 01 Jan 2017 00:00:00 UTC*
- MJDREFF: *Frac part of MJD ref (32.184secs+37leapsecs)*
- TIMEZERO: *Zero time*
- ONTIME: *On source time*
- LIVETIME: *On source time corrected for dead time*
- DEADC: *Dead time correction*
- DEADAPP: *Has DEADC been applied to data*
- RA_OBJ: *[deg] R.A. Object*
- DEC_OBJ: *[deg] Dec Object*
- RA_PNT: *[deg] RA pointing*
- DEC_PNT: *[deg] Dec pointing*
- OBJECT: *Name of observed object*
- EQUINOX: *equinox of celestial coord system*
- RADECSYS: *celestial coord system*
- FILE_LVL: *File level*
- LV2_VER: *Version of the LV2 data format*

5.4.2 Columns

- TRG_ID (J): *Trigger identifier* [None]
- SEC (J): *Integral part of event time (MET)* [s]
- MICROSEC (J): *Fractional part of event time (MET)* [us]
- TIME (D): *Event time in seconds (MET)* [s]
- LIVETIME (J): *Live time since the previous event in microseconds* [us]
- PHA (J): *Event pulse height* [None]
- PI (E): *Event pulse invariant* [None]
- ENERGY (E): *Event energy in keV* [keV]
- NUM_CLU (I): *Number of clusters in the event* []
- DETX (E): *Reconstructed absorption point X (GPD frame)* [mm]
- DETY (E): *Reconstructed absorption point Y (GPD frame)* [mm]
- RA (E): *Event right ascension* [deg]
- DEC (E): *Event declination* [deg]
- X (E): *Event X position (SKY frame)* [pixel]
- Y (E): *Event Y position (SKY frame)* [pixel]
- DETPHI (E): *Photolectron emission angle (GPD frame)* [rad]
- PHI (E): *Photoelectron emission angle (SKY frame)* [rad]
- Q (E): *Corrected event q Stokes parameter* [None]
- U (E): *Corrected event u Stokes parameter* [None]
- W_MOM (E): *Event weight* [None]

5.5 GTI extension

5.5.1 Header keywords

- TELESCOP: *Telescope name*
- INSTRUME: *Instrument name*
- DETNAM: *Name of the logical detector unit*
- DET_ID: *Name of the physical detector unit*
- TSTART: *Observation start time in MET*
- TSTOP: *Observation end time in MET*
- DATE-OBS: *Observation start datetime*
- DATE-END: *Observation end datetime*
- TELAPSE: *TSTOP-TSTART*
- TIMESYS: *Time system*

- TIMEUNIT: *Time units*
- TIMEREF: *Time reference*
- MJDREFI: *MJD ref day 01 Jan 2017 00:00:00 UTC*
- MJDREFF: *Frac part of MJD ref (32.184secs+37leapsecs)*
- TIMEZERO: *Zero time*
- ONTIME: *On source time*
- LIVETIME: *On source time corrected for dead time*
- DEADC: *Dead time correction*
- DEADAPP: *Has DEADC been applied to data*
- RA_OBJ: *[deg] R.A. Object*
- DEC_OBJ: *[deg] Dec Object*
- RA_PNT: *[deg] RA pointing*
- DEC_PNT: *[deg] Dec pointing*
- OBJECT: *Name of observed object*

5.5.2 Columns

- START (D): *GTI start time* [s]
- STOP (D): *GTI stop time* [s]

5.6 MONTE_CARLO extension

5.6.1 Header keywords

- IRFNAME: *name of the IRFs used for the simulation*
- TELESCOP: *Telescope name*
- INSTRUME: *Instrument name*
- DETNAM: *Name of the logical detector unit*
- DET_ID: *Name of the physical detector unit*
- TSTART: *Observation start time in MET*
- TSTOP: *Observation end time in MET*
- DATE-OBS: *Observation start datetime*
- DATE-END: *Observation end datetime*
- TELAPSE: *TSTOP-TSTART*
- TIMESYS: *Time system*
- TIMEUNIT: *Time units*
- TIMEREF: *Time reference*
- MJDREFI: *MJD ref day 01 Jan 2017 00:00:00 UTC*

- MJDREFF: *Frac part of MJD ref (32.184secs+37leapsecs)*
- TIMEZERO: *Zero time*
- ONTIME: *On source time*
- LIVETIME: *On source time corrected for dead time*
- DEADC: *Dead time correction*
- DEADAPP: *Has DEADC been applied to data*
- RA_OBJ: *[deg] R.A. Object*
- DEC_OBJ: *[deg] Dec Object*
- RA_PNT: *[deg] RA pointing*
- DEC_PNT: *[deg] Dec pointing*
- OBJECT: *Name of observed object*

5.6.2 Columns

- SRC_ID (I): *Monte Carlo source identifier* [None]
- MC_ENERGY (E): *Monte Carlo event energy* [keV]
- MC_PHA (J): *Monte Carlo pulse height* [None]
- MC_PI (E): *Monte Carlo pulse invariant* [None]
- MC_RA (E): *Monte Carlo right ascension* [degrees]
- MC_DEC (E): *Monte Carlo declination* [degrees]
- MC_X (I): *Monte Carlo event X position (SKY frame)* [degrees]
- MC_Y (I): *Monte Carlo event Y position (SKY frame)* [degrees]
- MC_GAIN (E): *Relative GEM gain used for the event* [None]

5.7 ROITABLE extension

5.7.1 Header keywords

- ROIRA: *right ascension of the ROI center*
- ROIDEC: *declination of the ROI center*
- EQUINOX: *equinox for RA and DEC*
- TELESCOP: *Telescope name*
- INSTRUME: *Instrument name*
- DETNAM: *Name of the logical detector unit*
- DET_ID: *Name of the physical detector unit*
- TSTART: *Observation start time in MET*
- TSTOP: *Observation end time in MET*
- DATE-OBS: *Observation start datetime*

- DATE-END: *Observation end datetime*
- TELAPSE: *TSTOP-TSTART*
- TIMESYS: *Time system*
- TIMEUNIT: *Time units*
- TIMEREF: *Time reference*
- MJDREFI: *MJD ref day 01 Jan 2017 00:00:00 UTC*
- MJDREFF: *Frac part of MJD ref (32.184secs+37leapsecs)*
- TIMEZERO: *Zero time*
- ONTIME: *On source time*
- LIVETIME: *On source time corrected for dead time*
- DEADC: *Dead time correction*
- DEADAPP: *Has DEADC been applied to data*
- RA_OBJ: *[deg] R.A. Object*
- DEC_OBJ: *[deg] Dec Object*
- RA_PNT: *[deg] RA pointing*
- DEC_PNT: *[deg] Dec pointing*
- OBJECT: *Name of observed object*

5.7.2 Columns

- SRCID (I): *source identifier* [None]
- SRCNAME (A20): *source name* [None]

5.8 TIMELINE extension

5.8.1 Header keywords

- TELESCOP: *Telescope name*
- INSTRUME: *Instrument name*
- DETNAM: *Name of the logical detector unit*
- DET_ID: *Name of the physical detector unit*
- TSTART: *Observation start time in MET*
- TSTOP: *Observation end time in MET*
- DATE-OBS: *Observation start datetime*
- DATE-END: *Observation end datetime*
- TELAPSE: *TSTOP-TSTART*
- TIMESYS: *Time system*
- TIMEUNIT: *Time units*

- TIMEREF: *Time reference*
- MJDREFI: *MJD ref day 01 Jan 2017 00:00:00 UTC*
- MJDREFF: *Frac part of MJD ref (32.184secs+37leapsecs)*
- TIMEZERO: *Zero time*
- ONTIME: *On source time*
- LIVETIME: *On source time corrected for dead time*
- DEADC: *Dead time correction*
- DEADAPP: *Has DEADC been applied to data*
- RA_OBJ: *[deg] R.A. Object*
- DEC_OBJ: *[deg] Dec Object*
- RA_PNT: *[deg] RA pointing*
- DEC_PNT: *[deg] Dec pointing*
- OBJECT: *Name of observed object*

5.8.2 Columns

- START (D): *Epoch start time* [s]
- STOP (D): *Epoch stop time* [s]
- IN_SAA (I): *SAA flag* [None]
- TARGET_OCCULT (I): *Earth occultation flag* [None]

5.9 SC_DATA extension

5.9.1 Header keywords

- ROLL: *spacecraft roll angle*
- TELESCOP: *Telescope name*
- INSTRUME: *Instrument name*
- DETNAM: *Name of the logical detector unit*
- DET_ID: *Name of the physical detector unit*
- TSTART: *Observation start time in MET*
- TSTOP: *Observation end time in MET*
- DATE-OBS: *Observation start datetime*
- DATE-END: *Observation end datetime*
- TELAPSE: *TSTOP-TSTART*
- TIMESYS: *Time system*
- TIMEUNIT: *Time units*
- TIMEREF: *Time reference*

- MJDREFI: *MJD ref day 01 Jan 2017 00:00:00 UTC*
- MJDREFF: *Frac part of MJD ref (32.184secs+37leapsecs)*
- TIMEZERO: *Zero time*
- ONTIME: *On source time*
- LIVETIME: *On source time corrected for dead time*
- DEADC: *Dead time correction*
- DEADAPP: *Has DEADC been applied to data*
- RA_OBJ: *[deg] R.A. Object*
- DEC_OBJ: *[deg] Dec Object*
- RA_PNT: *[deg] RA pointing*
- DEC_PNT: *[deg] Dec pointing*
- OBJECT: *Name of observed object*

5.9.2 Columns

- MET (D): *Mission elapsed time* [s]
- RA_PNT (E): *Pointing RA* [degrees]
- DEC_PNT (E): *Pointing DEC* [degrees]
- LAT_GEO (E): *Spacecraft latitude* [degrees]
- LON_GEO (E): *Spacecraft longitude* [degrees]
- ALT_GEO (E): *Spacecraft elevation* [km]
- SUN_ANGLE (E): *Angle between the Sun and the target* [degrees]
- IN_SAA (I): *SAA flag* [None]
- TARGET_OCCULT (I): *Earth occultation flag* [None]

5.10 OCTI extension

5.10.1 Header keywords

- CALRUNS: *Number of on-orbit calibration runs*
- CALTIME: *Total time for on-orbit calibration in s*
- TELESCOP: *Telescope name*
- INSTRUME: *Instrument name*
- DETNAM: *Name of the logical detector unit*
- DET_ID: *Name of the physical detector unit*
- TSTART: *Observation start time in MET*
- TSTOP: *Observation end time in MET*
- DATE-OBS: *Observation start datetime*

- DATE-END: *Observation end datetime*
- TELAPSE: *TSTOP-TSTART*
- TIMESYS: *Time system*
- TIMEUNIT: *Time units*
- TIMEREF: *Time reference*
- MJDREFI: *MJD ref day 01 Jan 2017 00:00:00 UTC*
- MJDREFF: *Frac part of MJD ref (32.184secs+37leapsecs)*
- TIMEZERO: *Zero time*
- ONTIME: *On source time*
- LIVETIME: *On source time corrected for dead time*
- DEADC: *Dead time correction*
- DEADAPP: *Has DEADC been applied to data*
- RA_OBJ: *[deg] R.A. Object*
- DEC_OBJ: *[deg] Dec Object*
- RA_PNT: *[deg] RA pointing*
- DEC_PNT: *[deg] Dec pointing*
- OBJECT: *Name of observed object*

5.10.2 Columns

- START (D): *OCTI start time* [s]
- STOP (D): *OCTI stop time* [s]

5.11 CHRG_MAP extension

5.11.1 Header keywords

- VERSION: *Extension version number*
- CVSD0001: *Date when this file should first be used*
- CVST0001: *Time of day when this file should first be used*
- NUM_BINS: *Number of bins per side of the map*

5.11.2 Columns

- `BINX` (I): *Index for the x coordinate* [None]
- `BINY` (I): *Index for the y coordinate* [None]
- `SLOW` (D): *Parameters for the charging slow component* [None]
- `FAST` (D): *Parameters for the charging fast component* [None]

5.12 Removing Monte Carlo information

Warning

Added in version 16.6.0.

Support for stripping the ground truth from the photon lists simulated with `ixpeobssim` was added in version 16.6.0 to support the first IXPE data challenge. This new feature required a series of modification for the inter-operability with the analysis tools shipped with `ixpeobssim`, and should be provisionally considered experimental—please report any issue you should encounter with Monte-Carlo-less data sets!

While the Monte Carlo information in the simulated photon lists is extremely useful for developing analysis tools, having the ability to generate files without any information that would not be ordinary available for real data can be useful as well, e.g., to test the inter-operability of any given software tool.

5.13 xpstripmc

xpstripmc is a simple script processing a series of simulating photon lists and removing the ground truth information—more specifically:

- the `MONTE_CARLO` extension;
- the `ROITABLE` extension.

Although this would seem like an operation that should be essentially transparent to the end user, one should realize that the these two extensions do contain valid information in the spirit of streamlining the simulation and analysis process. More specifically, the `MONTE_CARLO` extension header contains the `IRFNAME` keyword, encapsulating the version of the response functions used for the simulation; in normal condition this is automatically picked up by all the tools downstream to ensure that the simulation and the analysis are performed self-consistently—which is usually a sensible thing to do (in real life the IRF version will be supplied by the user).

All the usual analysis tools have been properly updated to support photon lists with no Monte Carlo information, provided that the user supplies the extra information that is needed. (You should get a sensible error message if that is not the case.)

5.14 Using xpbin

When using *xpbin* with stripped-down data files, and depending on the particular binning algorithm being used, you should make sure you provide the relevant information, where appropriate, through the proper command line options. More precisely:

- for all the `PHA1` related binned files (i.e., un-normalized and normalized Stokes spectra), you should provide the name of the response functions to be used (via the `--irfname` command-line switch) so that this is correctly picked up by `XSPEC` downstream;
- for the `CMAP`, `MDPMAP`, `MDPMAPCUBE`, `PCUBE`, `PMAP`, and `PMAPCUBE`, algorithms, you should also provide the name of the response functions to be used (via the `--irfname` command-line switch) so that the appropriate effective area and modulation factor can be correctly loaded.

It goes without saying: with no Monte Carlo information available you will not be able to run with the `--mc` switch.

5.15 Using xpselect

xpeselect should work exactly as before. (The previous remark about the `--mc` switch still holds.)

CHAPTER

SIX

TUTORIAL

This section is dedicated to a few tutorials on how to use ixpeobssim.

6.1 Defining a point source

The heart of the code lies in how we define the source that we wish to simulate and ultimately study. In ixpeobssim we do that by writing a configuration file in python. Several example sources can be found in the `config` directory, here we will describe how to simulate a single point source with a 50 % constant polarization degree, constant polarization angle of 30 degrees and a power-law spectrum with index 2 and normalization of 10. Below is a snapshot of a configuration file defining this single point source:

```
import numpy

from ixpeobssim.config import file_path_to_model_name
from ixpeobssim.srcmodel.roi import xPointSource, xROIModel
from ixpeobssim.srcmodel.spectrum import power_law
from ixpeobssim.srcmodel.polarization import constant

# Define the source properties and create the actual source.
ra, dec = 45., 45.
PL_NORM = 10.
PL_INDEX = 2.
#Constant polarization degree and angle
pol_deg = constant(0.5)
pol_ang = constant(numpy.radians(30.))
#Create the spectrum with a power-law shape with index 2 and normalization 10
spec = power_law(PL_NORM, PL_INDEX)
#Create the point source using the xPointSource class
src = xPointSource('Point source', ra, dec, spec, pol_deg, pol_ang)

# Create the complete region of interest.
ROI_MODEL = xROIModel(ra, dec)
ROI_MODEL.add_source(src)
```

We create our point source by passing our parameters (ra, dec, spectrum, ect) to the *ixpeobssim.srcmodel.roi.xPointSource* class. The first value passed to the xPointSource class is the name of the source, in this case we simply called it 'Point source'. You can also specify the column density (value used to calculate the Galactic absorption) and the redshift of your source by passing it to the class. When running a simulation in ixpeobssim we define our region of interest (ROI) and populate it with the sources that we wish to simulate. This is what is seen in the above configuration file. We define the point source and then define our ROI_MODEL by calling the *ixpeobssim.srcmodel.roi.xROIModel* class. The ra and dec values that we pass to this class define the center of the ROI. We

then proceed to add the source we just defined to the ROI. Note that we can add any number of sources to our ROI and each source can have their own positions (different ra and dec values than those used to define the center of the ROI).

This is all you need to create a single point source in ixpeobssim. Once you have created your configuration file you can run a simulation of the duration you wish as described in the *Quick start* section under the xpobssim section of the documentation.

6.2 Defining an extended source

Now that we know how to define a simple point source we can move on to a more complicated example of how to build an extended source. There are two ways to go about doing that in ixpeobssim both of which rely on observation fits files:

- create the extended source using the *ixpeobssim.srcmodel.roi.xExtendedSource* class
- create the extended source using the *ixpeobssim.srcmodel.roi.xChandraObservation* class

Here we will illustrate how to use both cases.

6.2.1 xExtendedSource class

Using the *ixpeobssim.srcmodel.roi.xExtendedSource* class is very similar to what we did in the case of a single point source in the sense that we will need to define the ROI by calling the xROIModel class and passing it the ra and dec values for the center of your region of interest. You will also need to provide the parameters describing the source itself (just as was done for the point source). The major difference lies in the fact that you need to provide an image fits file (for example a Chandra observation) that describes the morphology of the extended source. So for example if we were to use the same parameters used for the single point source but decide to assign them to the morphology of CasA, then you would do the following:

```
import numpy
import os

from ixpeobssim.srcmodel.roi import xExtendedSource, xROIModel
from ixpeobssim.srcmodel.polarization import constant
from ixpeobssim.srcmodel.spectrum import power_law
from ixpeobssim.core.spline import xInterpolatedUnivariateSplineLinear
from ixpeobssim import IXPEOBSSIM_CONFIG

# Define the source properties and create the actual source.
#This is the center of the extended source (CasA)
ra, dec = 350.8664167, 58.8117778

#Parameters to define the power-law spectrum of the source
#with index 2 and normalization 10
PL_NORM = 10.
PL_INDEX = 2.
spec = power_law(PL_NORM, PL_INDEX)

#Constant polarization degree and angle
pol_deg = constant(0.5)
pol_ang = constant(numpy.radians(30.))
```

(continues on next page)

(continued from previous page)

```
#Here is the path to the Chandra observation in the energy range 4-6 keV
#of  CasA.
img_file_path = os.path.join(IXPEOBSSIM_CONFIG, 'fits', 'casa_4p0_6p0_keV.fits')

#Create the extended source calling it CasA using the xExtendedSource class
src = xExtendedSource('CasA', img_file_path, spec, pol_deg, pol_ang)

# Create the complete region of interest by calling the xROIModel class and
#adding CasA to the ROI.
ROI_MODEL = xROIModel(ra, dec)
ROI_MODEL.add_source(src)
```

6.2.2 xChandraObservation class

The *ixpeobssim.srcmodel.roi.xChandraObservation* class allows you to use a Chandra photon list to describe the source that you wish to simulate, conserving the temporal, energetic and spatial information. How this is implemented in ixpeobssim is described in the *Implementation details* section. The only parameters that you need to provide are those specifying the polarization degree and angle and of course the observation fits file. When building a configuration file using a Chandra photon list you need to use the *ixpeobssim.srcmodel.roi.xChandraROIModel* class instead of the *ixpeobssim.srcmodel.roi.xROIModel* class that was used in the previous examples. It is also possible to pass a map describing the polarization degree and angle for the extended source that you want to simulate. To do this you need to use the *ixpeobssim.srcmodel.polarization.xStokesSkyMap* class. Here is an example of how to build a configuration file to simulate CasA using the morphology from a Chandra observation and describing the polarization properties with fit maps:

```
import os

from ixpeobssim.srcmodel.roi import xChandraObservation, xChandraROIModel
from ixpeobssim.srcmodel.polarization import xStokesSkyMap
from ixpeobssim import IXPEOBSSIM_CONFIG

#Path to the Chandra observation file
chandra_evt_file_path = 'path/to/evt/file/casa.fits'

#Here we call the xChandraROIModel class instead of the XROIModel class
#as was done for the previous examples and we pass the path to the
#Chandra observation file. You need to specidy the acis detector of the
#Chandra photon list being used for the simulation.
ROI_MODEL = xChandraROIModel(chandra_evt_file_path, acis='S')

#Here you define the polarization degree and angle you want to assign
#to the source. In this case the polarization map is a simple
#geometrical, radially-symmetric, model.
pol_mapx_path = os.path.join(IXPEOBSSIM_CONFIG, 'fits', 'casa_pmax050_reg000_x.fits')
pol_mapy_path = os.path.join(IXPEOBSSIM_CONFIG, 'fits', 'casa_pmax050_reg000_y.fits')

#We call the xStokesSkyMap class that handles the polarization maps.
#The "rotate" argument is intended to deal with input maps where the
#reference system is rotated by 90 degrees (counter-clockwise) with
#respect to the standard system that we use throughout, i.e., with the
#x-axis horizontal and the angles measured from the x-axis itself
```

(continues on next page)

(continued from previous page)

```
#counter clockwise. In this case the two components of the polarization
#vector are properly swapped, i.e., x -> y and -y -> x.

polarization_map = xStokesSkyMap.load_from_xy(pol_mapx_path, pol_mapy_path, rotate=True)

#We define the functions for the polarization degree and angle
def polarization_angle(E, t, ra, dec):
    return polarization_map.polarization_angle(ra, dec)

def polarization_degree(E, t, ra, dec):
    return polarization_map.polarization_degree(ra, dec)

#Here we create the source called CasA using the XChandraObservation
#class and pass the name of the source, the polarization degree and
#angle you defined above.
casa = xChandraObservation('CasA', polarization_degree,
                           polarization_angle)

#Add the source CasA to the ROI model
ROI_MODEL.add_source(casa)
```

In this example we used a home-made map to define the polarization degree and angle of the source. In ixpeobssim we have the capabilities of accepting maps (in fits format) of the polarization parameters (Q/U, x/y polarization components and polarization degree and angle) through the *ixpeobssim.srcmodel.polarization.xStokesSkyMap* class. In this case we have created a simple model to describe the polarization as a geometrical, radially-symmetric toy-model. Once we have created the polarization maps, the xChandraObservation uses the spatial and spectral information of the source from the Chandra observation and combines it with the polarization model to build the source to simulate.

You can pass an arbitrary number of layers in different energy bands to the *ixpeobssim.srcmodel.polarization.xStokesSkyMap* class, each layer is essentially a Stokes sky map and we interpolate the polarimetric information in three dimensions (Ra, Dec, energy). For an example of how to implement a more complex simulation of an extended source see the ixpeobssim/examples/tycho_3d.py example.

6.3 Defining a periodic source

We have the `ixpeobssim.srcmodel.roi.xPeriodicSource` class that allows us to easily build a point-like periodic source to simulate. The basic ingredients to describe the point source are the same as in the simple point source configuration but this time we need to include also the information on the ephemeris of the pulsar. The ephemeris are handled by the `ixpeobssim.srcmodel.roi.xEphemeris` class. Here is an example of how to write a configuration file describing a pulsar with a power-law photon spectrum with normalization that varies as a function of the pulse phase. To illustrate the flexibility of the code, here we will also make the polarization degree vary as a function of both energy and pulsar phase.

```
import numpy

from ixpeobssim.config import file_path_to_model_name
from ixpeobssim.srcmodel.roi import xPeriodicPointSource, xEphemeris, xROIModel
from ixpeobssim.srcmodel.spectrum import power_law
from ixpeobssim.srcmodel.polarization import constant

ra, dec = 45., 45.
```

(continues on next page)

(continued from previous page)

```
#Method that returns a power-law normalization as a function of phase.
def pl_norm(phase):
   """Photon spectrum: power-law normalization as a function of the pulse
   phase.
   """
   return 1.25 + numpy.cos(4 * numpy.pi * phase)

#Constant power-law index
pl_index = 2.
#Use the function power_law to build the photon spectrum of the pulsar.
spec = power_law(pl_norm, pl_index)

#Method that returns the polarization degree as a function of energy and phase
def pol_deg(E, phase, ra=None, dec=None):
    """Polarization degree as a function of the dynamical variables.

    Since we're dealing with a point source the sky direction (ra, dec) is
    irrelevant and, as they are not used, defaulting the corresponding arguments
    to None allows to call the function passing the energy and phase only.
    """
    norm = numpy.clip(E / 10., 0., 1.)
    return norm * (0.5 + 0.25 * numpy.cos(4 * numpy.pi * (phase - 0.25)))

#Constant polarization angle
pol_ang = constant(numpy.radians(30.))

#Use the xEphemeris class to pass to the xPeriodicPointSource class
ephemeris = xEphemeris(0., 1.)

src = xPeriodicPointSource('Periodic source', ra, dec, spec, pol_deg, pol_ang,
ephemeris)
#Add the source to the ROI
ROI_MODEL = xROIModel(ra, dec, src)
```

The configuration file for this particular example is included as one of the so-called toy_models (toy_periodic_source.py) in the examples folder of the ixpeobssim package.

6.4 Running a simulation with xpobssim

Running a simulation with *xpobssim* is very simple and all you really need to provide in input is the configuration file and the duration of the simulation. Here is the command line that you need to type (assuming that the current working directory is the ixpeobssim root folder) to run a simulation of the periodic source (described above) for a duration of 1000 seconds:

```
xpobssim.py --config ixpeobssim/config/toy_periodic_source.py --duration 1000
```

This will produce three event (FITS) files (one for each Detector Unit) in your $IXPEOBSSIM_DATA folder.

Warning

By default the energy range for the simulation is the same as the one where the response functions are defined (currently between 1 and 12 keV). The emin and emax command-line switches allow you to override the default, should that be needed for whatever reason (e.g., if your spectral model is only defined in a smaller energy range). However this feature should be used with caution, as some padding on both ends of the nominal 2–8 keV IXPE energy band is generally needed due to the energy dispersion of the detector.

6.5 Analysis tools

Once the ixpe observation-simulation has been run, ixpeobssim has a few tools to help in the analysis/manipulation and visualization of the data. These are:

- *xpbin*
- *xpeselect*
- *xpphase*
- *xpophase*

More details on these tools can be found in the *Application reference* section of the documentation. For the sake of this tutorial we will go through a few examples of how to use these tools.

6.5.1 The xpbin tool

All the possible binning algorithms that this tool provides are described in the *Binned data products* and *Application reference* sections of the documentation. Here we will provide a few examples on what can be done with this tool.

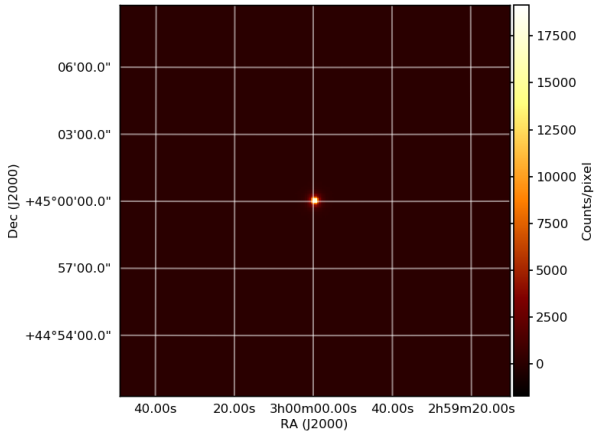
We will go through a few options using the output from the single point source example provided above. Once you have the simulated data of the single point source and you want to create a count map all you need to do is:

```
xpbin.py $IXPEOBSSIM_DATA/single_point_source_du*.fits --algorithm CMAP
```

This will create your count map, and if you want to visualize your count map you can use the *xpbinview* tool. *xpbinview* is capable of visualizing any of the binned outputs of *xpbin* without having to specify which binning algorithm you used.

```
xpbinview.py $IXPEOBSSIM_DATA/single_point_source_du*_cmap.fits
```

And this is what you will see:



In both of the examples illustrated here, we made use of the wild card (*) to pass all three of the event files to the tool. It is also possible to pass a list of the event files, if you prefer.

To produce a polarization cube, all you need to do is run the *xpbin* script specifying the PCUBE option to the algorithm argument.

```
xpbin.py $IXPEOBSSIM_DATA/single_point_source_du*.fits --algorithm PCUBE
```

If you run the PCUBE algorithm the default is to bin the data in 4 energy bins. If you would like to have an PCUBE in one single energy bin, you can specify it via this command line option:

```
xpbin.py $IXPEOBSSIM_DATA/single_point_source_du*.fits --algorithm PCUBE --ebins 1
```

The *xpbin* tool also has the `--ebinning` option that allows you to pass a list containing the bin edges for the energy binning you want for your analysis. There are several other options and binning algorithms that the *xpbin* script can do, for a complete list all you need to do make use of the help option or read the *Application reference* of the documentation.

6.5.2 The xpeselect tool

As the name implies, this tool allows you to select subsets of your simulated data. Say for example that you just simulated the MSH15-52 nebula + pulsar and that you want to analyze the pulsar. To do that all you need to do is to run the *xpeselect* script passing the radius of the region around the pulsar and the RA and DEC of the pulsar.

```
xpselect.py $IXPEOBSSIM_DATA/msh1552_simulated_data_du*.fits --ra RA --dec DEC --rad␣
↪radius_region
```

xpeselect also has the options of selecting on energy, time, phase and source id. This last option is for cases where you have more than one source in your ROI.

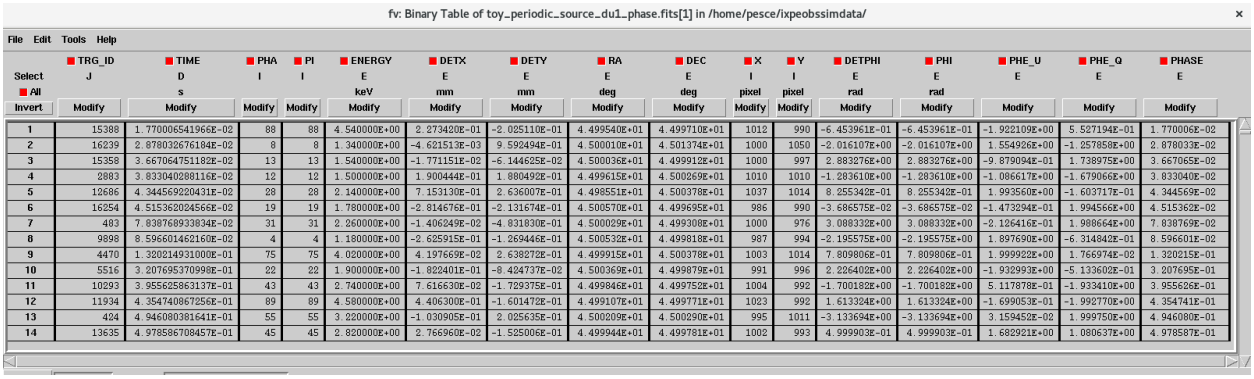
6.5.3 The xpphase tool

This particular tool is needed when you are simulating periodic sources, in particular *xpphase* creates an extra column in the event file for the phase of the periodic source. All this tool needs is the file list (output from the simulation) and the configuration file used to run the simulation.

> **Warning**
>
> The following is obsolete and must be edited.

```
xpphase.py $IXPEOBSSIM_DATA/toy_periodic_source_du*.fits --config ixpeobssim/config/toy_
↪periodic_source.py
```

This will create three new fits files in your `$IXPEOBSSIM_DATA` directory that are the same as the output of the simulation but now with an extra column for the phase of the pulsar. The name of the output fits files will be the same as the input but with an added suffix 'phase'. You can also specify the suffix for the output files by including the `--suffix` option when running *xpophase*. Below is a snapshot of the fits file columns after running the above command.



	TRG_ID	TIME	PHA	PI	ENERGY	DETX	DETY	RA	DEC	X	Y	DETPHI	PHI	PHE_U	PHE_Q	PHASE
Select	J	D	I	I	E	E	E	E	E	I	I	E	E	E	E	E
All		s			keV	mm	mm	deg	deg	pixel	pixel	rad	rad			
Invert	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify
1	15388	1.770006541966E-02	88	88	4.540000E+00	2.273420E-01	-2.025110E-01	4.499540E+01	4.499710E+01	1012	990	-6.453961E-01	-6.453961E-01	-1.922109E+00	5.527194E-01	1.770006E-02
2	16239	2.878032676184E-02	8	8	1.340000E+00	-4.621513E-03	9.592494E-01	4.500010E+01	4.501374E+01	1000	1050	-2.016107E+00	-2.016107E+00	1.554926E+00	-1.257858E+00	2.878033E-02
3	15358	3.667064751182E-02	13	13	1.540000E+00	-1.771151E-02	-6.144625E-02	4.500036E+01	4.499912E+01	1000	997	2.883276E+00	2.883276E+00	-9.879094E-01	1.738975E+00	3.667065E-02
4	2883	3.833040288116E-02	12	12	1.500000E+00	1.900444E-01	1.880492E-01	4.499615E+01	4.500269E+01	1010	1010	-1.283610E+00	-1.283610E+00	-1.086617E+00	-1.679066E+00	3.833040E-02
5	12686	4.344569220431E-02	28	28	2.140000E+00	7.153130E-01	2.636007E-01	4.498551E+01	4.500378E+01	1037	1014	8.255342E-01	8.255342E-01	1.993560E+00	-1.603717E-01	4.344569E-02
6	16254	4.515362024566E-02	19	19	1.780000E+00	-2.814676E-01	-2.131674E-01	4.500570E+01	4.499695E+01	986	990	-3.686575E-02	-3.686575E-02	-1.473294E-01	1.994566E+00	4.515362E-02
7	483	7.838768933834E-02	31	31	2.260000E+00	-1.406249E-02	-4.831830E-01	4.500029E+01	4.499308E+01	1000	976	3.088332E+00	3.088332E+00	-2.126416E-01	1.980664E+00	7.838769E-02
8	9898	8.596601462160E-02	4	4	1.180000E+00	-2.625915E-01	-1.269446E-01	4.500532E+01	4.499818E+01	987	994	-2.195575E+00	-2.195575E+00	1.897690E+00	-6.314842E-01	8.596601E-02
9	4470	1.320214931000E-01	75	75	4.020000E+00	4.197669E-02	2.638272E-01	4.499915E+01	4.500378E+01	1003	1014	7.809806E-01	7.809806E-01	1.999922E+00	1.766974E-02	1.320215E-01
10	5516	3.207695370998E-01	22	22	1.900000E+00	-1.822401E-01	-8.424737E-02	4.500369E+01	4.499879E+01	991	996	2.226402E+00	2.226402E+00	-1.932993E+00	-5.133602E-01	3.207695E-01
11	10293	3.955625863137E-01	43	43	2.740000E+00	7.616630E-02	-1.729375E-01	4.499846E+01	4.499752E+01	1004	992	-1.700182E+00	-1.700182E+00	5.117878E-01	-1.933410E+00	3.955626E-01
12	11934	4.354740867256E-01	89	89	4.580000E+00	4.406300E-01	-1.601472E-01	4.499107E+01	4.499771E+01	1023	992	1.613324E+00	1.613324E+00	-1.699053E-01	-1.992770E+00	4.354741E-01
13	424	4.946080381641E-01	55	55	3.220000E+00	-1.030905E-01	2.025635E-01	4.500209E+01	4.500290E+01	995	1011	-3.133694E+00	-3.133694E+00	3.159452E-02	1.999750E+00	4.946080E-01
14	13635	4.978586708457E-01	45	45	2.820000E+00	2.766960E-02	-1.525006E-01	4.499944E+01	4.499781E+01	1002	993	4.999903E-01	4.999903E-01	1.682921E+00	1.080637E+00	4.978587E-01

6.5.4 The xpophase tool

> **Warning**
>
> This section is searching for a writer :-)

6.6 Analysis pipelines

Each ixpeobssim application is wrapped in the *`ixpeobssim.core.pipeline`* module so that it can be called from within a generic Python script with the exact same arguments that one would pass from command line. Several examples of how to do this are described in the *Analysis pipelines* section of the ixpeobssim documentation. There are also several Python scripts in the example folder of the ixpeobssim package that illustrate how to perform a full analysis chain using the pipeline facility.

CHAPTER

SEVEN

APPLICATION REFERENCE

Here is the synopsis of all the applications in the IXPE observation simulation framework, along with the complete, up-to-date summary of the corresponding command-line switches.

7.1 Simulation facilities

7.1.1 xpobssim

```
...

usage: xpobssim.py [-h] [--outfile OUTFILE] --configfile CONFIGFILE
                   [--irfname IRFNAME] [--duration DURATION]
                   [--gtiminduration GTIMINDURATION]
                   [--gtistartpad GTISTARTPAD] [--gtistoppad GTISTOPPAD]
                   [--emin EMIN] [--emax EMAX] [--startdate DATE]
                   [--objname OBJNAME] [--seed SEED]
                   [--vignetting {True,False}] [--dithering {True,False}]
                   [--ditherampl DITHERAMPL] [--ditherpa DITHERPA]
                   [--ditherpx DITHERPX] [--ditherpy DITHERPY]
                   [--grayfilter {True,False}] [--charging {True,False}]
                   [--chrgnside CHRGNSIDE] [--chrgtstep CHRGTSTEP]
                   [--chrgmaps CHRGMAPS [CHRGMAPS ...]]
                   [--chrgparams CHRGPARAMS [CHRGPARAMS ...]]
                   [--deadtime DEADTIME] [--roll ROLL] [--saa {True,False}]
                   [--occult {True,False}] [--onorbitcalib {True,False}]
                   [--onorbitcaldemult ONORBITCALDEMULT]
                   [--onorbitcalminduration ONORBITCALMINDURATION]
                   [--onorbitcalstartpad ONORBITCALSTARTPAD]
                   [--onorbitcalstoppad ONORBITCALSTOPPAD]
                   [--onorbitcalrate ONORBITCALRATE]
                   [--timelinedata {True,False}] [--scdata {True,False}]
                   [--scdatainterval SCDATAINTERVAL] [--lv1a {True,False}]
                   [--lv1version LV1VERSION] [--overwrite {True,False}]

Run the ixpeobssim fast simulator.

This the main application in the package, and produces a set of event files
(a.k.a. photon lists), given a source model and a set of instrument response
functions, for a given observation time.
```

(continues on next page)

(continued from previous page)

```
Internally the source spectral, temporal, morphological and polarimetric
characteristics are convolved with the instrument response functions to produce
a realistic, simulated IXPE observation.

optional arguments:
  -h, --help            show this help message and exit
  --outfile OUTFILE     path to the output file (default: None)
  --configfile CONFIGFILE
                        path to the input configuration file (default: None)
  --irfname IRFNAME     name of the response functions to be used (default:
                        ixpe:obssim20240101:v13)
  --duration DURATION   duration of the observation in s (default: 1000)
  --gtiminduration GTIMINDURATION
                        minimum GTI duration in s (default: 0.0)
  --gtistartpad GTISTARTPAD
                        time padding at the start of each GTI in s (default:
                        0.0)
  --gtistoppad GTISTOPPAD
                        time padding at the stop of each GTI in s (default:
                        0.0)
  --emin EMIN           minimum energy in keV (default: 1.0)
  --emax EMAX           maximum energy in keV (default: 12.0)
  --startdate DATE      observation start date %Y-%m-%d or
                        %Y-%m-%dT%H:%M:%S.%f (default: 2022-04-21)
  --objname OBJNAME     name of the observed object (default: None)
  --seed SEED           random seed for the simulation (default: None)
  --vignetting {True,False}
                        apply MMA vignetting (default: True)
  --dithering {True,False}
                        apply the dithering pattern (default: True)
  --ditherampl DITHERAMPL
                        the dithering amplitude in arcmin (default: 1.6)
  --ditherpa DITHERPA   the dithering main period in s (default: 907.0)
  --ditherpx DITHERPX   the dithering x period in s (default: 101.0)
  --ditherpy DITHERPY   the dithering y period in s (default: 449.0)
  --grayfilter {True,False}
                        enable the gray filter (default: False)
  --charging {True,False}
                        apply the GEM charging (default: False)
  --chrgnside CHRGNSIDE
                        number of spatial bins for the charging model
                        (default: 200)
  --chrgtstep CHRGTSTEP
                        time step for the charging model [s] (default: 30)
  --chrgmaps CHRGMAPS [CHRGMAPS ...]
                        maps of the initial charging (default: None)
  --chrgparams CHRGPARAMS [CHRGPARAMS ...]
                        path to the files of the charging parameters (default:
                        None)
  --deadtime DEADTIME   average dead time per event in s (default: 0.00108)
  --roll ROLL           telescope roll angle in decimal degrees (default: 0.0)
```

(continues on next page)

(continued from previous page)

```
  --saa {True,False}    consider the SAA for the GTI calculation (default:
                        False)
  --occult {True,False}
                        consider the Earth occultations for the GTI
                        calculation (default: False)
  --onorbitcalib {True,False}
                        activate the onboard calibration sources where
                        appropriate (default: False)
  --onorbitcaldemult ONORBITCALDEMULT
                        demultiplier for onboard calibration, i.e., do a DU
                        every n orbit(s) (default: 1)
  --onorbitcalminduration ONORBITCALMINDURATION
                        minimum duration of the onboard calibration runs
                        (default: 600.0)
  --onorbitcalstartpad ONORBITCALSTARTPAD
                        time padding for starting calibration runs in s
                        (default: 30.0)
  --onorbitcalstoppad ONORBITCALSTOPPAD
                        time padding for stopping calibration runs in s
                        (default: 30.0)
  --onorbitcalrate ONORBITCALRATE
                        average rate in Hz for the onboard Cal C source
                        (default: 100.0)
  --timelinedata {True,False}
                        write the observation timeline into the output file
                        (default: True)
  --scdata {True,False}
                        write the spacecraft data into the output file
                        (default: True)
  --scdatainterval SCDATAINTERVAL
                        time interval for the spacecraft data in s (default:
                        5.0)
  --lv1a {True,False}   create a pseudo-Lv1a file (default: False)
  --lv1version LV1VERSION
                        version number for the pseudo-Lv1a support (default:
                        5)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.1.2 xpphotonlist

```
...

usage: xpphotonlist.py [-h] [--outfile OUTFILE] --configfile CONFIGFILE
                       [--irfname IRFNAME] [--duration DURATION]
                       [--gtiminduration GTIMINDURATION]
                       [--gtistartpad GTISTARTPAD] [--gtistoppad GTISTOPPAD]
                       [--emin EMIN] [--emax EMAX] [--startdate DATE]
                       [--objname OBJNAME] [--seed SEED]
                       [--vignetting {True,False}] [--dithering {True,False}]
```

(continues on next page)

(continued from previous page)

```
                        [--ditherampl DITHERAMPL] [--ditherpa DITHERPA]
                        [--ditherpx DITHERPX] [--ditherpy DITHERPY]
                        [--grayfilter {True,False}] [--roll ROLL]
                        [--saa {True,False}] [--occult {True,False}]
                        [--scdata {True,False}]
                        [--scdatainterval SCDATAINTERVAL]
                        [--overwrite {True,False}]

Create a photon list to be fed into ixpesim.

This is a rough equivalent to xpobssim, except for the fact that the output FITS
file(s) represent a list of photons at the top of the GPD window, as opposed to
a list of photoelectron tracks. This allows the list to be fed into xpsim, where
the photons are propagated through the detector, and the complete information
for all the events that trigger, including the track images, is written to file.

In conjunction with ixpesim, this small application allows for a full end-to-end
workflow where arbitrarily complex source models can be simulated with the
ultimate fidelity.

optional arguments:
  -h, --help            show this help message and exit
  --outfile OUTFILE     path to the output file (default: None)
  --configfile CONFIGFILE
                        path to the input configuration file (default: None)
  --irfname IRFNAME     name of the response functions to be used (default:
                        ixpe:obssim20240101:v13)
  --duration DURATION   duration of the observation in s (default: 1000)
  --gtiminduration GTIMINDURATION
                        minimum GTI duration in s (default: 0.0)
  --gtistartpad GTISTARTPAD
                        time padding at the start of each GTI in s (default:
                        0.0)
  --gtistoppad GTISTOPPAD
                        time padding at the stop of each GTI in s (default:
                        0.0)
  --emin EMIN           minimum energy in keV (default: 1.0)
  --emax EMAX           maximum energy in keV (default: 12.0)
  --startdate DATE      observation start date %Y-%m-%d or
                        %Y-%m-%dT%H:%M:%S.%f (default: 2022-04-21)
  --objname OBJNAME     name of the observed object (default: None)
  --seed SEED           random seed for the simulation (default: None)
  --vignetting {True,False}
                        apply MMA vignetting (default: True)
  --dithering {True,False}
                        apply the dithering pattern (default: True)
  --ditherampl DITHERAMPL
                        the dithering amplitude in arcmin (default: 1.6)
  --ditherpa DITHERPA   the dithering main period in s (default: 907.0)
  --ditherpx DITHERPX   the dithering x period in s (default: 101.0)
  --ditherpy DITHERPY   the dithering y period in s (default: 449.0)
  --grayfilter {True,False}
```

(continues on next page)

(continued from previous page)

```
                        enable the gray filter (default: False)
  --roll ROLL           telescope roll angle in decimal degrees (default: 0.0)
  --saa {True,False}    consider the SAA for the GTI calculation (default:
                        False)
  --occult {True,False}
                        consider the Earth occultations for the GTI
                        calculation (default: False)
  --scdata {True,False}
                        write the spacecraft data into the output file
                        (default: True)
  --scdatainterval SCDATAINTERVAL
                        time interval for the spacecraft data in s (default:
                        5.0)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

Warning

`xpphotonlist` is designed to generate list of photons at the top of the focal plane, to be fed into the full simulation of the focal-plane detectors—which we are not planning to release to the public. This application is, therefore, of limited use for the Community; yet, it can be used as an inspiration for a useful feature that could be adapted to different applications.

7.1.3 xpcalib

```
...

usage: xpcalib.py [-h] [--outfile OUTFILE] --configfile CONFIGFILE
                  [--irfname IRFNAME] [--duration DURATION] [--emin EMIN]
                  [--emax EMAX] [--startdate DATE] [--seed SEED]
                  [--charging {True,False}] [--chrgnside CHRGNSIDE]
                  [--chrgtstep CHRGTSTEP] [--chrgmaps CHRGMAPS [CHRGMAPS ...]]
                  [--chrgparams CHRGPARAMS [CHRGPARAMS ...]]
                  [--deadtime DEADTIME] [--lv1a {True,False}]
                  [--overwrite {True,False}]

Create photon lists from IXPE calibration runs, either on ground or in orbit.

optional arguments:
  -h, --help            show this help message and exit
  --outfile OUTFILE     path to the output file (default: None)
  --configfile CONFIGFILE
                        path to the input configuration file (default: None)
  --irfname IRFNAME     name of the response functions to be used (default:
                        ixpe:obssim20240101:v13)
  --duration DURATION   duration of the observation in s (default: 1000)
  --emin EMIN           minimum energy in keV (default: 1.0)
  --emax EMAX           maximum energy in keV (default: 12.0)
  --startdate DATE      observation start date %Y-%m-%d or
```

(continues on next page)

(continued from previous page)

```
                        %Y-%m-%dT%H:%M:%S.%f (default: 2022-04-21)
  --seed SEED           random seed for the simulation (default: None)
  --charging {True,False}
                        apply the GEM charging (default: False)
  --chrgnside CHRGNSIDE
                        number of spatial bins for the charging model
                        (default: 200)
  --chrgtstep CHRGTSTEP
                        time step for the charging model [s] (default: 30)
  --chrgmaps CHRGMAPS [CHRGMAPS ...]
                        maps of the initial charging (default: None)
  --chrgparams CHRGPARAMS [CHRGPARAMS ...]
                        path to the files of the charging parameters (default:
                        None)
  --deadtime DEADTIME   average dead time per event in s (default: 0.00108)
  --lv1a {True,False}   create a (pseudo) Lv1a file (default: False)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.2 Sensitivity estimation

7.2.1 xpmdp

```
...

usage: xpmdp.py [-h] [--outfile OUTFILE] --configfile CONFIGFILE
                [--srcid SRCID] [--irfname IRFNAME] [--duration DURATION]
                [--deadtime DEADTIME] [--eef EEF] [--phasemin PHASEMIN]
                [--phasemax PHASEMAX] [--emin EMIN] [--emax EMAX]
                [--ebinalg {FILE,LIN,LOG,LIST}] [--ebins EBINS]
                [--ebinfile EBINFILE] [--ebinning EBINNING]
                [--overwrite {True,False}]

Calculate the minimum detectable polarization (MDP) for a given source model.

The program takes the same python configuration modules that can be fed
into xpobbsim and calculates the MDP at the 99% CL by direct numerical
integration of the input spectrum (i.e., there are no event lists involved).
As a consequence, part of the richness of the detector response (most
notably, the energy dispersion and the vignetting) is not captured here.

For use cases beyond simple stationary point sources, the use of xpobbsim and
xpbin mode are recommended, as that approach offers the maximum flexibility.

optional arguments:
  -h, --help            show this help message and exit
  --outfile OUTFILE     path to the output file (default: None)
  --configfile CONFIGFILE
                        path to the input configuration file (default: None)
```

(continues on next page)

(continued from previous page)

```
  --srcid SRCID         the source identifier in the ROITABLE extension
                        (default: 0)
  --irfname IRFNAME     name of the response functions to be used (default:
                        ixpe:obssim20240101:v13)
  --duration DURATION   duration of the observation in s (default: 10000.0)
  --deadtime DEADTIME   average dead time per event in s (default: 0.00108)
  --eef EEF             encircled-energy-fraction factor (default: 1.0)
  --phasemin PHASEMIN   minimum phase for periodic sources (default: 0.0)
  --phasemax PHASEMAX   maximum phase for periodic sources (default: 1.0)
  --emin EMIN           minimum energy in keV (default: 2.0)
  --emax EMAX           maximum energy in keV (default: 8.0)
  --ebinalg {FILE,LIN,LOG,LIST}
                        energy binning specification (default: LOG)
  --ebins EBINS         number of bins for LIN/LOG energy binning (default: 4)
  --ebinfile EBINFILE   path to the energy bin definition file (default: None)
  --ebinning EBINNING   list containing the bin edges (default: None)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.2.2 xppimms

```
...

usage: xppimms.py [-h] [--outfile OUTFILE] [--irfname IRFNAME] [--norm NORM]
                  [--index INDEX] [--nH NH] [--redshift REDSHIFT]
                  [--duration DURATION] [--deadtime DEADTIME] [--eef EEF]
                  [--emin EMIN] [--emax EMAX] [--ebinalg {FILE,LIN,LOG,LIST}]
                  [--ebins EBINS] [--ebinfile EBINFILE] [--ebinning EBINNING]
                  [--overwrite {True,False}]

Calculate the minimum detectable polarization (MDP) for simple stationary
point sources with a power-law spectrum, according to the source parameters
provided through the command-line switches.

(Note the default power-law normalization and index and the default duration
correspond to the setup for the IXPE level-1 polarization requirement).

The program calculates the MDP at the 99% CL by direct numerical integration of
the input spectrum and  response functions (i.e., there are no event lists
involved). As a consequence, part of the richness of the detector response
(most notably, the energy dispersion and the effective area vignetting) is not
captured.

For use cases beyond simple stationary point sources, the use of xpobbsim,
xpselect and xpbin are recommended, as that approach offers the maximum
flexibility.

optional arguments:
  -h, --help            show this help message and exit
  --outfile OUTFILE     path to the output file (default: None)
```

(continues on next page)

(continued from previous page)

```
  --irfname IRFNAME     name of the response functions to be used (default:
                        ixpe:obssim20240101:v13)
  --norm NORM           the power-law normalization for energy spectrum
                        (default: 0.0045023)
  --index INDEX         the power-law index for energy spectrum (default: 2.0)
  --nH NH               the hydrogen column density (cm^{-2}) (default: 0.0)
  --redshift REDSHIFT   the cosmological redshift (default: 0.0)
  --duration DURATION   duration of the observation in s (default: 864000.0)
  --deadtime DEADTIME   average dead time per event in s (default: 0.00108)
  --eef EEF             encircled-energy-fraction factor (default: 1.0)
  --emin EMIN           minimum energy in keV (default: 2.0)
  --emax EMAX           maximum energy in keV (default: 8.0)
  --ebinalg {FILE,LIN,LOG,LIST}
                        energy binning specification (default: LOG)
  --ebins EBINS         number of bins for LIN/LOG energy binning (default: 4)
  --ebinfile EBINFILE   path to the energy bin definition file (default: None)
  --ebinning EBINNING   list containing the bin edges (default: None)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.2.3 xpvisibility

```
...

usage: xpvisibility.py [-h] (--srcname NAME | --srccoords RA DEC)
                       [--startdate DATE] [--stopdate DATE]
                       [--sun {True,False}] [--overwrite {True,False}]

Simple IXPE visibility tool.

optional arguments:
  -h, --help            show this help message and exit
  --srcname NAME        name of the target source
  --srccoords RA DEC    coordinates of the target source
  --startdate DATE      observation start date %Y-%m-%d or
                        %Y-%m-%dT%H:%M:%S.%f (default: 2022-04-21)
  --stopdate DATE       observation stop date %Y-%m-%d or %Y-%m-%dT%H:%M:%S.%f
  --sun {True,False}    plot the Sun constraints (default: True)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.3 Analysis facilities

7.3.1 xpbin

```
...

usage: xpbin.py [-h] --algorithm
                {PHA1,PHA1Q,PHA1U,PHA1QN,PHA1UN,CMAP,MDPMAP,MDPMAPCUBE,PCUBE,PMAP,
↪PMAPCUBE,PP,ARMAP,EFLUX,LC,LTCUBE}
                [--suffix SUFFIX] [--irfname IRFNAME]
                [--grayfilter {True,False}] [--acceptcorr {True,False}]
                [--weights {True,False}] [--weightcol WEIGHTCOL]
                [--tbinalg {FILE,LIN,LOG}] [--tmin TMIN] [--tmax TMAX]
                [--tbins TBINS] [--tbinfile TBINFILE] [--phasebins PHASEBINS]
                [--npix NPIX] [--pixsize PIXSIZE] [--xref XREF] [--yref YREF]
                [--proj {AIT,ZEA,ARC,CAR,GLS,MER,NCP,SIN,STG,TAN}]
                [--emin EMIN] [--emax EMAX]
                [--ebinalg {FILE,LIN,LOG,EQP,LIST}] [--ebins EBINS]
                [--ebinfile EBINFILE] [--ebinning EBINNING]
                [--phibins PHIBINS] [--thetabins THETABINS] [--insun INSUN]
                [--ineclipse INECLIPSE] [--mc {True,False}]
                [--overwrite {True,False}]
                filelist [filelist ...]

Basic event binning interface.

This application allows to bin event files (i.e., photon lists) in a fairly
generic fashion, creating count spectra, light curves, maps and more advances
data products such as those for storing the Stokes parameters.

Supported binning algorithms
* PHA1: count spectrum
   --mc, weights, weightcol, grayfilter, overwrite, suffix
* PHA1Q: Stokes Q spectrum
   --mc, weights, weightcol, grayfilter, overwrite, suffix
* PHA1U: Stokes U spectrum
   --mc, weights, weightcol, grayfilter, overwrite, suffix
* PHA1QN: normalized Stokes Q spectrum (Q/I)
   --mc, weights, weightcol, grayfilter, overwrite, suffix
* PHA1UN: normalized Stokes U spectrum (U/I)
   --mc, weights, weightcol, grayfilter, overwrite, suffix
* CMAP: count map in sky coordinates
   --mc, npix, pixsize, proj, xref, yref, overwrite, suffix
* MDPMAP: MDP map in sky coordinates
   --mc, acceptcorr, weights, weightcol, grayfilter, emin, emax, npix, pixsize, proj,␣
↪xref, yref, overwrite, suffix
* MDPMAPCUBE: MDP map cube in sky coordinates and energy layers
   --mc, acceptcorr, weights, weightcol, grayfilter, npix, pixsize, proj, xref, yref,␣
↪emin, emax, ebins, ebinalg, ebinning, ebinfile, overwrite, suffix
* PCUBE: polarization cube (in energy layers)
   --mc, acceptcorr, weights, weightcol, grayfilter, emin, emax, ebins, ebinalg,␣
↪ebinning, ebinfile, overwrite, suffix
```

(continues on next page)

(continued from previous page)

```
* PMAP: polarization map in sky coordinates
   --mc, acceptcorr, weights, weightcol, grayfilter, emin, emax, npix, pixsize, proj,␣
↪xref, yref, overwrite, suffix
* PMAPCUBE: polarization map cube in sky coordinates and energy layers
   --mc, acceptcorr, weights, weightcol, grayfilter, npix, pixsize, proj, xref, yref,␣
↪emin, emax, ebins, ebinalg, ebinning, ebinfile, overwrite, suffix
* PP: pulse profile
   --phasebins, overwrite, suffix
* ARMAP: rate per unit area map in detector coordinates
   --npix, overwrite, suffix
* EFLUX: energy-flux map in detector coordinates
   --npix, overwrite, suffix
* LC: light curve
   --tmin, tmax, tbins, tbinalg, tbinfile, overwrite, suffix
* LTCUBE: Livetime map cube in sky coordinates and off axis angle bins
   --npix, pixsize, proj, xref, yref, overwrite, suffix

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --algorithm {PHA1,PHA1Q,PHA1U,PHA1QN,PHA1UN,CMAP,MDPMAP,MDPMAPCUBE,PCUBE,PMAP,PMAPCUBE,
↪PP,ARMAP,EFLUX,LC,LTCUBE}
                        the binning algorithm (default: None)
  --suffix SUFFIX       suffix for the output files (default: None)
  --irfname IRFNAME     name of the response functions to be used (default:
                        None)
  --grayfilter {True,False}
                        enable the gray filter (default: False)
  --acceptcorr {True,False}
                        enable/disable the acceptance correction for
                        polarization cubse and maps (default: True)
  --weights {True,False}
                        Use weights for the analysis (default: False)
  --weightcol WEIGHTCOL
                        name of the weight column to be used (default: W_MOM)
  --tbinalg {FILE,LIN,LOG}
                        time binning specification (default: LIN)
  --tmin TMIN           minimum time in s (default: None)
  --tmax TMAX           maximum tims in s (default: None)
  --tbins TBINS         number of bins for LIN/LOG time binning (default: 100)
  --tbinfile TBINFILE   path to the optional time bin definition file
                        (default: None)
  --phasebins PHASEBINS
                        number of bins for phase binning (default: 50)
  --npix NPIX           number of pixels per side in the output image
                        (default: 200)
  --pixsize PIXSIZE     the pixel size of the output image in arcseconds
                        (default: None)
  --xref XREF           the horizontal position of the image center (default:
                        None)
```

(continues on next page)

(continued from previous page)

```
  --yref YREF           the vertical position of the image center (default:
                        None)
  --proj {AIT,ZEA,ARC,CAR,GLS,MER,NCP,SIN,STG,TAN}
                        coordinate projection (default: TAN)
  --emin EMIN           minimum energy in keV (default: 2.0)
  --emax EMAX           maximum energy in keV (default: 8.0)
  --ebinalg {FILE,LIN,LOG,EQP,LIST}
                        energy binning specification (default: LOG)
  --ebins EBINS         number of bins for LIN/LOG energy binning (default: 4)
  --ebinfile EBINFILE   path to the energy bin definition file (default: None)
  --ebinning EBINNING   list containing the bin edges (default: None)
  --phibins PHIBINS     number of bins for LIN/LOG phi binning (default: 75)
  --thetabins THETABINS
                        number of bins for off-axis angle binning (default:
                        17)
  --insun INSUN         INSUN file for background flare subtraction.
                        (mandatory if --ineclipse is set) (default: None)
  --ineclipse INECLIPSE
                        INECLIPSE file for background flare
                        subtraction(mandatory if --insun is set) (default:
                        None)
  --mc {True,False}     use Monte Carlo information (default: False)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.3.2 xpeselect

```
...

usage: xpselect.py [-h] [--suffix SUFFIX] [--tmin TMIN] [--tmax TMAX]
                   [--tinvert {True,False}] [--phasemin PHASEMIN]
                   [--phasemax PHASEMAX] [--phaseinvert {True,False}]
                   [--emin EMIN] [--emax EMAX] [--einvert {True,False}]
                   [--ra RA] [--dec DEC] [--rad RAD] [--innerrad INNERRAD]
                   [--regfile REGFILE] [--reginvert {True,False}]
                   [--mask MASK] [--mcsrcid MCSRCID] [--mc {True,False}]
                   [--ltimeupdate {True,False}] [--ltimealg {LTSUM,LTSCALE}]
                   [--overwrite {True,False}]
                   filelist [filelist ...]

Basic data selection interface. This utility can process IXPE event files
(i.e., photon lists) and apply a generic selection on sky-position, time,
phase, energy, etc., producing smaller event files. In conjunction with
xpbin this is handy to, e.g., energy- or phase-resolved polarization
measurements.

xpselect supports spatial selections from a ds9 region file thorough the
--regfile command-line switch.

positional arguments:
```

(continues on next page)

(continued from previous page)

```
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --suffix SUFFIX       suffix for the output files (default: None)
  --tmin TMIN           minimum time in s (default: None)
  --tmax TMAX           maximum tims in s (default: None)
  --tinvert {True,False}
                        invert the time selection (default: False)
  --phasemin PHASEMIN   minimum phase for periodic sources (default: None)
  --phasemax PHASEMAX   maximum phase for periodic sources (default: None)
  --phaseinvert {True,False}
                        invert the phase selection (default: False)
  --emin EMIN           minimum energy in keV (default: None)
  --emax EMAX           maximum energy in keV (default: None)
  --einvert {True,False}
                        invert the energy selection (default: False)
  --ra RA               RA of acceptance cone in decimal degrees (default:
                        None)
  --dec DEC             Dec of acceptance cone in decimal degrees (default:
                        None)
  --rad RAD             ROI radius in arcminutes (default: None)
  --innerrad INNERRAD   ROI inner radius in arcminutes (for selecting annuli)
                        (default: None)
  --regfile REGFILE     path to a ds9 region file (default: None)
  --reginvert {True,False}
                        invert the ds9 region file selection (default: False)
  --mask MASK           path to .npy file with a saved boolean event mask
                        (default: None)
  --mcsrcid MCSRCID     the Monte Carlo source ID to select (default: [])
  --mc {True,False}     use Monte Carlo information (default: False)
  --ltimeupdate {True,False}
                        update the livetime-related keywords in the output
                        files (default: False)
  --ltimealg {LTSUM,LTSCALE}
                        algorithm to be used to update the livetime (default:
                        LTSCALE)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.3.3 xpsun

```
...
    This is free software, and you are welcome to redistribute it under certain
    conditions. See the LICENSE file for details.

    Visit https://bitbucket.org/ixpesw/ixpeobssim for more information.

usage: xpsun.py [-h] [--l2files L2FILES [L2FILES ...]] [--show] folder
```

(continues on next page)

(continued from previous page)

```
Separate time windows when the detector is illuminated/not illuminated by
the sun.

This application uses housekeeping information to split observation files
into an INSUN and an INECLIPSE section, based on sun visibility. The livetime
of the two parts is updated based on LV1 data.

Required input folder structure
-------------------------------
The application expects a standard IXPE observation folder layout::

    <folder>/
        event_l2/   Level-2 calibrated event files
                    (matched by ``ixpe*_det<DU>_evt2_v??.fits*``)
        event_l1/   Level-1 event files used to recompute LIVETIME
                    (matched by ``ixpe*_det<DU>_evt1_v??.fits*``)
        hk/         Housekeeping files containing spacecraft attitude data
                    (matched by ``ixpe*_all_adc_0110_v??.fits*``)

xpsun supports both .fits and .fits.gz files, but the latter will prevent
memory mapping potentially leading to several tens of gigabites of RAM usage.
Consider uncompressing files for large data sets.

level-2 files can be passed explicitly via ``--l2files`` instead of using
automatic parsing; the level-1 and HK files are always located by scanning
the observation folder. Beware: level 1 files are all those contained in the
event_l1 folder, so if some processing products are present they will all
be included leading to potential issues. A clean folder structure is recommended
when running xpsun. In case of level 2 subselections, use --l2files to
specify the files to process.

GTI creation
------------
The core of the sun/eclipse separation is the ``create_ineclipse_gtis()``
function (in ``ixpeobssim.instrument.sc``), which reads the HK file columns:

* **ADSEC2SUN** -- angular distance of the sun relative to the spacecraft
* **ADSEC2ECL** -- angular distance encoding the Earth-occultation geometry

The INECLIPSE condition is defined as:

    ``(ADSEC2SUN < 0) AND (ADSEC2ECL >= 0)``

while the INSUN condition is its complement:

    ``(ADSEC2ECL < 0) OR (ADSEC2SUN >= 0)``

The boolean mask over the HK TIME column is converted into arrays of GTI
start/stop pairs.

Event filtering
---------------
```

(continues on next page)

(continued from previous page)

```
For each detector unit and each level-2 file, the ``intersect_gti()`` function
(in ``ixpeobssim.evt.event``) performs the following steps:

1. Reads the existing GTI table from the level-2 event file.
2. Intersects it with the INSUN (or INECLIPSE) GTI intervals, producing a
   merged set of good-time intervals.
3. Replaces the GTI extension in the FITS file with the new intervals.
4. Filters the event list, keeping only rows whose TIME falls within the
   new GTIs.
5. Recomputes LIVETIME by summing the LIVETIME column from matching level-1
   files over the new GTI intervals, and updates the ONTIME keyword from the
   total good time.
6. Writes the result to a new file with a ``_insun`` or ``_inecl`` tag
   appended to the original filename.

Output files
------------
For an input file ``ixpe01234567_det1_evt2_v02.fits``, the application
produces:

* ``ixpe01234567_det1_evt2_v02_insun.fits``  -- events in sun-illuminated GTIs
* ``ixpe01234567_det1_evt2_v02_inecl.fits``  -- events in eclipse GTIs

Both files retain the same FITS structure as the original level-2 file,
with updated GTI, LIVETIME and ONTIME values.

positional arguments:
  folder                path to the input observation folder

optional arguments:
  -h, --help            show this help message and exit
  --l2files L2FILES [L2FILES ...]
                        level 2 file list (default: None)
  --show                show INSUN/INECLIPSE bands over the observation light
                        curve (default: False)
```

7.3.4 xpxspec

```
...

usage: xpxspec.py [-h] --model MODEL [--params PARAMS]
                  [--fixpars FIXPARS [FIXPARS ...]] [--statmethod {}]
                  [--niterations NITERATIONS] [--emin EMIN] [--emax EMAX]
                  [--fit {True,False}] [--error {True,False}]
                  [--plot {True,False}] [--figname FIGNAME]
                  [--overwrite {True,False}]
                  filelist [filelist ...]

Minimal interface to perform a spectro-polarimetric fit in XSPEC, via the
Python bindings.
```

(continues on next page)

(continued from previous page)

```
This is primarily designed to fit a spectral and polarimetric model to a series
of I, Q and U Stokes spectra (typically nine input files---three spectra for
each detector unit) or a purely polarimetric fit to a series of normalized
Q/I and U/I spectra (typically six input files).

Although only a small part of the flexibility provided by a real XSPEC session
is currently exposed, this simple application is a sensible mean to get started
with the use of the ixpeobssim polarimetric local models within XSPEC.

Note that the application fully supports purely spectral or polarimetric fits
in a transparent fashion.

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --model MODEL         the spectral model for the fit (default: None)
  --params PARAMS       the initial values of the fit parameters (default:
                        None)
  --fixpars FIXPARS [FIXPARS ...]
                        freeze one or more fit-parameter value(s) (default:
                        [])
  --statmethod {}       the fit statistics to be used (default: chi)
  --niterations NITERATIONS
                        maximum number of fit iterations (default: 25)
  --emin EMIN           minimum energy in keV (default: 2.0)
  --emax EMAX           maximum energy in keV (default: 8.0)
  --fit {True,False}    fit the data (default: True)
  --error {True,False}  calculate confidence intervals for fit parameters
                        (default: True)
  --plot {True,False}   plot the fit results (default: True)
  --figname FIGNAME     base name for the output figures (default: XSPEC fit)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.3.5 xpphase

```
...

usage: xpphase.py [-h] [--suffix SUFFIX] [--parfile PARFILE] [--met0 MET0]
                  [--nu0 NU0] [--nudot0 NUDOT0] [--nuddot NUDDOT]
                  [--phi0 PHI0] [--overwrite {True,False}]
                  filelist [filelist ...]

Utility to build phase column into event file.

The program generates a phase array using pulsar time and ephemeris
and builds a new event FITS file with the new PHASE column.
```

(continues on next page)

(continued from previous page)

```
positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --suffix SUFFIX       suffix for the output files (default: phase)
  --parfile PARFILE     path to the input parameter file (default: None)
  --met0 MET0           reference MET of the ephemeris in s (default: None)
  --nu0 NU0             frequency at t0 in Hz (default: None)
  --nudot0 NUDOT0       time first derivative of frequency at t0 in 1/(s^2)
                        (default: None)
  --nuddot NUDDOT       time second derivative of frequency in 1/(s^3)
                        (default: None)
  --phi0 PHI0           phase zero value (default: 0.0)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

Warning

When using `xpphase` from command line, since the derivatives of the frequence are typically (small) negative numbers, it is customary to bump into an odd corner of the Python argparse module, where the "e" character of the exponent specifier, in conjunction with the leading minus sign, tricks Python into thinking that the value for the `nudot0` and/or the `nuddot` command line arguments are actually a separate option. The deal, here, is to use, e.g., the `nudot0=-1.e13` form of the options specification, *with the equal sign*.

See this issue for more details.

7.3.6 xpophase

```
...

usage: xpophase.py [-h] [--suffix SUFFIX] --parfile PARFILE [--ra RA]
                   [--dec DEC] [--phi0 PHI0] [--bary {True,False}]
                   [--overwrite {True,False}]
                   filelist [filelist ...]

Utility to build orbital phase column into event file.

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --suffix SUFFIX       suffix for the output files (default: ophase)
  --parfile PARFILE     path to the input parameter file (default: None)
  --ra RA               source right ascension in deg (default: None)
  --dec DEC             source declination in deg (default: None)
  --phi0 PHI0           phase zero value (default: 0.0)
  --bary {True,False}   apply or not barycentric time correction (default:
```

(continues on next page)

(continued from previous page)

```
                        False)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.3.7 xpstokesalign

```
...

usage: xpstokesalign.py [-h] --mode {RAD,TAN,QU,PDA}
                        [--modelfiles MODELFILES [MODELFILES ...]] [--ra RA]
                        [--dec DEC] [--suffix SUFFIX] [--mc {True,False}]
                        [--overwrite {True,False}]
                        filelist [filelist ...]

Align the reconstructed Stokes parameters to a given polarization model.

This is an application processing a photon list and `rotating` the Stokes parameters
so that, on an event-by-event basis, the zero for the measurement of the
photoelectron direction is aligned to a given input model at the position of the
event.

In the simplest form the alignment can be either radial or tangential, which is
achieved by selecting the RAD or TAN modes, respectively, and optionally passing
the right ascension and the declination of the center for the rotation via the
--ra and --dec command-line switches.

Additionally, the user can feed into tha application pair of FITS images
(in either the Q/U, X/Y components of the polarization vector or polarization
degree/angle) in the same exact fashion of the machinery used for simulating
complex polarization patterns for extended sources in ixpeobssim. This is
achieved via a combination of the QU, XY of PDA modes, along with the proper
model files passed to the --modelfiles command-line switch.

Supported alignment algorithms:
- RAD: radial
- TAN: tangential
- QU : generic polarization model (from FITS maps of U and Q)
- XY : generic polarization model (from FITS maps of polarization components)
- PDA: generic polarization angle model (from a single FITS map)

Known limitations:
- the applications is currently not supporting Stokes sky-cubes, i.e., models
  where the polarization angle is energy-dependent (although this can be
  implemented if there is need for);
- when running in PDA mode, although in principle the map of the polarization
  angle would be enough to operate the rotation, for techincal reasons having
  to do with the internals of the code, we still need to pass the maps of both
  the polarization degree and angle (and, again, this is a nuisance that we
  can overcome if there is really need for).
```

(continues on next page)

(continued from previous page)

```
positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --mode {RAD,TAN,QU,PDA}
                        the alignment mode
  --modelfiles MODELFILES [MODELFILES ...]
                        path to the input model file(s) (default: None)
  --ra RA               RA of the model center in decimal degrees (default:
                        None)
  --dec DEC             Dec of the model center in decimal degrees (default:
                        None)
  --suffix SUFFIX       suffix for the output files (default: None)
  --mc {True,False}     use Monte Carlo information (default: False)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.3.8 xpstokesrandom

```
...

usage: xpstokesrandom.py [-h] [--seed SEED] [--suffix SUFFIX]
                         [--overwrite {True,False}]
                         filelist [filelist ...]

Process a level-2 file and replace the Q and U column with random values sampled
according to a fully unpolarized simulation.

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --seed SEED           random seed for the simulation (default: 1)
  --suffix SUFFIX       suffix for the output files (default: stokesrandom)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.3.9 xpstokesshuffle

```
...

usage: xpstokesshuffle.py [-h] [--seed SEED] [--suffix SUFFIX]
                          [--overwrite {True,False}]
                          filelist [filelist ...]

Process a level-2 file and shuffle the value of the Q and U columns across the
events.
```

(continues on next page)

(continued from previous page)

```
positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --seed SEED           random seed for the simulation (default: 1)
  --suffix SUFFIX       suffix for the output files (default: stokesshuffle)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.3.10 xpstokessmear

```
...

usage: xpstokessmear.py [-h] [--innersigma INNERSIGMA]
                        [--outersigma OUTERSIGMA] [--innerradius INNERRADIUS]
                        [--nside NSIDE] [--suffix SUFFIX] [--seed SEED]
                        [--interactive {True,False}]
                        [--overwrite {True,False}]
                        filelist [filelist ...]

Smear the event-by-event Stokes parameters with a gaussian probability density
function with zero average.

This is supposed to be a cheap shortcut to test the effect of the correction for
the spurious modulation in realistic analysis without having to simulate the
effect to start with---we assume that the actual correction gets the average
right, and we are just introducing the statistical fluctuations of the
correction, here, in such a way that the output Stokes parameters are not
properly normalized, as will happen for real level-2 files.

The smearing is done over a fixed grid of pixels, with sigma values different in
the center of the detector and on the borders, and in a single energy band.
While this is defintely not capturing the entire richness of the phenomenology
of the spurious modulation, it is a sensible zero-order approximation to test
possible biases of the correction on actual science analysis.

For reference, the spurious modulation maps in the IXPE CALDB are mapped onto
300 x 300 bins on the detector surface, and across 6 energy bands. The typical
errors on Q and U are 0.025 in the inner 50 pixel (or 2.5 mm) radius circle,
and 0.10 in the outer portion of the detector.

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --innersigma INNERSIGMA
                        the standard deviation for the Q and U gaussian
```

(continues on next page)

(continued from previous page)

```
                        smearing in the center (default: 0.025)
  --outersigma OUTERSIGMA
                        the standard deviation for the Q and U gaussian
                        smearing on the border (default: 0.1)
  --innerradius INNERRADIUS
                        the inner radius in mm, defining the region of the
                        high-statistics calibration (default: 2.5)
  --nside NSIDE         the size of the binning grid in DETX and DETY
                        (default: 300)
  --suffix SUFFIX       suffix for the output files (default: stokessmear)
  --seed SEED           random seed for the simulation (default: None)
  --interactive {True,False}
                        plot some diagnostic plot (default: False)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.3.11 xpexposure

```
...

usage: xpexposure.py [-h] --cmapfiles CMAPFILES [CMAPFILES ...]
                     [--irfname IRFNAME] [--irftype {arf,mrf,both}]
                     [--expunits {None,cm2s,s}] [--overwrite {True,False}]
                     filelist [filelist ...]

Read a series of binned livetime cubes and perform the PSF convolution,
exposure calculation and correction for the effective area and/or the
modulation response functions.

The modified response files can be used, e.g., for spectro-polarimetric fitting
within XSPEC. (Note ixpeobssim offers the xpancrkey.py script to set the
ANCRFILE header keywords in the event files.)

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --cmapfiles CMAPFILES [CMAPFILES ...]
                        path(s) to the CMAP file(s) (default: None)
  --irfname IRFNAME     name of the response functions to be used (default:
                        ixpe:obssim20240101:v13)
  --irftype {arf,mrf,both}
                        type of irf to correct and save (default: both)
  --expunits {None,cm2s,s}
                        flag to save the exposure cube using the required
                        units (default: None)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.4 Visualization facilities

7.4.1 xpbinview

```
...

usage: xpbinview.py [-h] [--outfile OUTFILE] [--overwrite {True,False}]
                    [--stretch STRETCH] [--vmin VMIN] [--vmax VMAX]
                    [--zlabel ZLABEL] [--dpi DPI] [--mjd {True,False}]
                    filelist [filelist ...]

Generic display interface to ixpeobssim binned data products.

This application allows to display the content of xpbin output files in all
the possible flavors. It is equipped to recognize automagically the binning
algotithm used to create the FITS file and use the proper facilty for the
display, i.e., you should be able to pass any binned product coming out of
xpbin and xpbinview should do the right thing to display that file to you.

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --outfile OUTFILE     path to the output file (default: None)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
  --stretch STRETCH     The stretch function for image color normalization
                        {'sqrt', 'asinh', 'log', 'power', 'linear'} (default:
                        linear)
  --vmin VMIN           minimum data range for colormaps (default: None)
  --vmax VMAX           maximum data range for colormaps (default: None)
  --zlabel ZLABEL       the color map laber for count maps (default: None)
  --dpi DPI             resolution of the output image in dot per inches
                        (default: 100)
  --mjd {True,False}    Show light curves in MJD time (as opposed to MET)
                        (default: False)
```

7.4.2 xpradialprofile

```
...

usage: xpradialprofile.py [-h] [--ra RA] [--dec DEC] [--rmin RMIN]
                          [--rmax RMAX] [--rcore RCORE] [--rbins RBINS]
                          [--autocenter {True,False}] [--psf {True,False}]
                          [--residuals {True,False}]
                          [--interactive {True,False}] [--irfname IRFNAME]
                          filelist [filelist ...]

Create a radial profile plot, in sky coordinates, for a given observation.
```

(continues on next page)

(continued from previous page)

```
This program takes a level-2 file as an input and creates a radial histogram of
the event counts, weighted with 1. / r to correct for the solid angle. This is
a useful diagnostics to gauge the background level in a given observation.

In order to provide more context to the plots, the PSF radial profile for the
relevant DU is overlayied, and the estimated fraction of source events outside
a circle of radius r is annotated over the PSF line for a discrete set of values
of r.

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --ra RA               RA of the center of the radial plot in decimal degrees
                        (default: None)
  --dec DEC             Dec of the center of the radial plot in decimal
                        degrees (default: None)
  --rmin RMIN           minimum radial distance in arcminutes (default: 0.0)
  --rmax RMAX           maximum radial distance in arcminutes (default: 10.0)
  --rcore RCORE         core radius for the psf fit in arcminutes (default:
                        0.5)
  --rbins RBINS         number of bins for the radial profile (default: 200)
  --autocenter {True,False}
                        recenter the image based on the count map (default:
                        True)
  --psf {True,False}    overimpose the PSF radial profile (default: True)
  --residuals {True,False}
                        plot the estimated residual background (default:
                        False)
  --interactive {True,False}
                        plot some diagnostic plot (default: False)
  --irfname IRFNAME     name of the response functions to be used (default:
                        ixpe:obssim20240101:v13)
```

7.4.3 xpevtdisplay

```
...

usage: IXPE single event display.

This application provides visualization support for track images contained in
Level-1 IXPE files.

       [-h] [--evtlist EVTLIST] [--targetname TARGETNAME] [--emin EMIN]
       [--emax EMAX] [--seed SEED] [--clustering {True,False}]
       [--numclusters NUMCLUSTERS] [--clumindensity CLUMINDENSITY]
       [--cluminsize CLUMINSIZE] [--resample RESAMPLE]
       [--absorption {True,False}] [--barycenter {True,False}]
```

(continues on next page)

(continued from previous page)

```
       [--direction {True,False}] [--pixpha {True,False}]
       [--indices {True,False}] [--cmap CMAP] [--cmapoffset CMAPOFFSET]
       [--minaxside MINAXSIDE] [--autostop AUTOSTOP] [--batch {True,False}]
       [--autosave {True,False}] [--outfolder OUTFOLDER]
       [--imgformat IMGFORMAT] [--imgdpi IMGDPI] [--timestamp TIMESTAMP]
       file

positional arguments:
  file                  path to the input file

optional arguments:
  -h, --help            show this help message and exit
  --evtlist EVTLIST     path to the auxiliary (Level-2 file) event list
                        (default: None)
  --targetname TARGETNAME
                        name of the celestial target (default: N/A)
  --emin EMIN           minimum energy in keV (default: 2.0)
  --emax EMAX           maximum energy in keV (default: 8.0)
  --seed SEED           random seed for the simulation (default: 1)
  --clustering {True,False}
                        run the DBscan clustering on the events (default:
                        True)
  --numclusters NUMCLUSTERS
                        the number of clusters to be displayed for each event
                        (default: 2)
  --clumindensity CLUMINDENSITY
                        the minimum density point for the DBscan clustering
                        (default: 5)
  --cluminsize CLUMINSIZE
                        the minimum cluster size for the DBscan clustering
                        (default: 6)
  --resample RESAMPLE   the power-law index for resampling events in energy
                        (default: None)
  --absorption {True,False}
                        draw the reconstructed absorption_point (default:
                        True)
  --barycenter {True,False}
                        draw the reconstructed barycenter (default: True)
  --direction {True,False}
                        draw the reconstructed track direction (default: True)
  --pixpha {True,False}
                        indicate the pixel PHA values (default: False)
  --indices {True,False}
                        draw the row and column indices of the readout matrix
                        (default: False)
  --cmap CMAP           the color map for the pixel values (default: Reds)
  --cmapoffset CMAPOFFSET
                        the PHA offset for the color map (default: 10)
  --minaxside MINAXSIDE
                        the axis side for the event display (default: 2.0)
  --autostop AUTOSTOP   stop automatically after a given number of events
                        (default: None)
```

(continues on next page)

(continued from previous page)

```
  --batch {True,False}  run in batch mode (default: False)
  --autosave {True,False}
                        save the event displays automatically (default: False)
  --outfolder OUTFOLDER
                        path to the output folder (default:
                        /home/docs/ixpeobssimdata)
  --imgformat IMGFORMAT
                        the image format for the output files when autosave is
                        True (default: png)
  --imgdpi IMGDPI       resolution of the output image in dot per inches
                        (default: 250)
  --timestamp TIMESTAMP
                        timestamp of a single specific event to be displayed
                        (default: None)
```

7.4.4 xpobsdisplay

```
...

usage: IXPE composite carousel to display observations.

This application extends xpevtdisplay to include the most relevant cumulative
distributions for a given observation.

       [-h] [--evtlist EVTLIST] [--targetname TARGETNAME] [--emin EMIN]
       [--emax EMAX] [--seed SEED] [--clustering {True,False}]
       [--numclusters NUMCLUSTERS] [--clumindensity CLUMINDENSITY]
       [--cluminsize CLUMINSIZE] [--resample RESAMPLE]
       [--absorption {True,False}] [--barycenter {True,False}]
       [--direction {True,False}] [--pixpha {True,False}]
       [--indices {True,False}] [--cmap CMAP] [--cmapoffset CMAPOFFSET]
       [--minaxside MINAXSIDE] [--autostop AUTOSTOP] [--batch {True,False}]
       [--autosave {True,False}] [--outfolder OUTFOLDER]
       [--imgformat IMGFORMAT] [--imgdpi IMGDPI] [--irfname IRFNAME]
       [--pdmax PDMAX] [--pdstep PDSTEP] [--xref XREF] [--yref YREF]
       [--npix NPIX] [--pixsize PIXSIZE] [--subtitle SUBTITLE]
       file

positional arguments:
  file                  path to the input file

optional arguments:
  -h, --help            show this help message and exit
  --evtlist EVTLIST     path to the auxiliary (Level-2 file) event list
                        (default: None)
  --targetname TARGETNAME
                        name of the celestial target (default: N/A)
  --emin EMIN           minimum energy in keV (default: 2.0)
  --emax EMAX           maximum energy in keV (default: 8.0)
  --seed SEED           random seed for the simulation (default: 1)
```

(continues on next page)

(continued from previous page)

```
--clustering {True,False}
                      run the DBscan clustering on the events (default:
                      True)
--numclusters NUMCLUSTERS
                      the number of clusters to be displayed for each event
                      (default: 2)
--clumindensity CLUMINDENSITY
                      the minimum density point for the DBscan clustering
                      (default: 5)
--cluminsize CLUMINSIZE
                      the minimum cluster size for the DBscan clustering
                      (default: 6)
--resample RESAMPLE   the power-law index for resampling events in energy
                      (default: None)
--absorption {True,False}
                      draw the reconstructed absorption_point (default:
                      True)
--barycenter {True,False}
                      draw the reconstructed barycenter (default: True)
--direction {True,False}
                      draw the reconstructed track direction (default: True)
--pixpha {True,False}
                      indicate the pixel PHA values (default: False)
--indices {True,False}
                      draw the row and column indices of the readout matrix
                      (default: False)
--cmap CMAP           the color map for the pixel values (default: Reds)
--cmapoffset CMAPOFFSET
                      the PHA offset for the color map (default: 10)
--minaxside MINAXSIDE
                      the axis side for the event display (default: 2.0)
--autostop AUTOSTOP   stop automatically after a given number of events
                      (default: None)
--batch {True,False}  run in batch mode (default: False)
--autosave {True,False}
                      save the event displays automatically (default: False)
--outfolder OUTFOLDER
                      path to the output folder (default:
                      /home/docs/ixpeobssimdata)
--imgformat IMGFORMAT
                      the image format for the output files when autosave is
                      True (default: png)
--imgdpi IMGDPI       resolution of the output image in dot per inches
                      (default: 250)
--irfname IRFNAME     name of the response functions to be used (default:
                      ixpe:obssim20240101:v13)
--pdmax PDMAX         maximum polarization degree for the Stokes plot
                      (default: 0.2)
--pdstep PDSTEP       polarization degree step for the Stokes plot grid
                      (default: 0.05)
--xref XREF           the horizontal position of the image center (default:
                      None)
```

(continues on next page)

(continued from previous page)

```
  --yref YREF           the vertical position of the image center (default:
                        None)
  --npix NPIX           number of pixels per side for the count map in sky
                        coordinates (default: 200)
  --pixsize PIXSIZE     pixel size in arcseconds for the count map in sky
                        coordinates (default: None)
  --subtitle SUBTITLE   subtitle for the animation (default: None)
```

7.4.5 xpirfview

```
...

usage: xpirfview.py [-h] [--save] [--outfolder OUTFOLDER] file

Generic display application for viewing response files.

This application allows to display the content of IXPE effective area,
point-spread-function, energy dispersion and modulation factor response files.

positional arguments:
  file                  path to the input file

optional arguments:
  -h, --help            show this help message and exit
  --save                save the output products (default: False)
  --outfolder OUTFOLDER
                        path to the output folder (default:
                        /home/docs/ixpeobssimdata)
```

7.4.6 xpobsview

```
...

usage: xpobsview.py [-h] filelist [filelist ...]

Observation timeline and spacecraft data quick look.

This application retrieves the TIMELINE, GTI and OCTI data from a series of
event lists and plot them as strip charts.

positional arguments:
  filelist    path(s) to the input file(s)

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

7.5 Miscellanea

7.5.1 xpsonify

```
...
```

7.5.2 xpbkgtemplate

```
...

usage: xpbkgtemplate.py [-h] [--ssmooth SSMOOTH] [--outfile OUTFILE]
                        filelist [filelist ...]

Create a background template model starting from a series of PHA1 background files.

The PHA1 files should be prepared from a dark field with a arbitrary shapes that
have been cut from one or more files and have a BACKSCAL keyword defined.

This is suitable both for residual and total background, depending on the user
needs.

The count spectrum, normalized by the backscal, the fiducial area of the
detector and by the bin width, is parametrized with a non interpolated
spline and written to file to be used later with an appropriate config file.

The output file is written on a regular energy grid as a simple text file
with two columns---energy and background rate.

positional arguments:
  filelist           path(s) to the input file(s)

optional arguments:
  -h, --help         show this help message and exit
  --ssmooth SSMOOTH  The smoothing coefficient ("s" argument in the scipy
                     documentation) used for the non interpolating spline.
                     Note this is very important, as it controls the level at
                     which the spline is capturing the fluctuations of the
                     input data points. (s=0 is effectively an interpolating
                     spline, but the actual value depends on the scale of the
                     input data and it is not trivial to establish a priori.)
                     (default: 0.0005)
  --outfile OUTFILE  path to the output file (default: /home/docs/checkouts/re
                     adthedocs.org/user_builds/ixpeobssim/checkouts/latest/ixp
                     eobssim/srcmodel/ascii/instrumental_bkg_template.txt)
```

7.5.3 xpevtstat

```
...

usage: xpevtstat.py [-h] [--emin EMIN] [--emax EMAX]
                    [--ebinalg {FILE,LIN,LOG,LIST}] [--ebins EBINS]
                    [--ebinfile EBINFILE] [--ebinning EBINNING]
                    [--mc {True,False}]
                    filelist [filelist ...]

Read a series of photon lists and print some basic statistics of the various
model components in the form of a simple text table.

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --emin EMIN           minimum energy in keV (default: 2.0)
  --emax EMAX           maximum energy in keV (default: 8.0)
  --ebinalg {FILE,LIN,LOG,LIST}
                        energy binning specification (default: LOG)
  --ebins EBINS         number of bins for LIN/LOG energy binning (default: 3)
  --ebinfile EBINFILE   path to the energy bin definition file (default: None)
  --ebinning EBINNING   list containing the bin edges (default: None)
  --mc {True,False}     use Monte Carlo information (default: False)
```

7.5.4 xpgrppha

```
...

usage: xpgrppha.py [-h] [--suffix SUFFIX] --comm COMM [--chatter CHATTER]
                   [--overwrite {True,False}]
                   filelist [filelist ...]

Small Python wrapper around the HEASARC GRPPHA utility, see
https://heasarc.gsfc.nasa.gov/ftools/caldb/help/grppha.txt

In a nutshell, you should be able to pass any command that you would pass to
grppha interactively via the --comm command-line switch, e.g.

> xpgrppha.py --comm "group min 100" pha1.fits

will (loosely) map to:

> grppha infile="pha1.fits" outfile="pha1_grppha.fits" comm="group min 100 & write & exit
↪" chatter=5 clobber=yes

(Note that "& write & exit" are automatically added at the end.)

While this is supposed to be as faithful as possible to the original, underlying
```

(continues on next page)

(continued from previous page)

```
application, there are some notable differences you should keep in mind:

* you can provide an arbitrary number of input files as command-line
  arguments, and the wrapper will happily iterate over them;
* sticking to the ixpeobssim convensions, the name of the output file is
  programmatically built from the input, and you can (at least partially)
  control that via the --suffix command-line switch;
* the wrapper, as all the other ixpeobssim applications, offers a --overwrite
  command-line option that is similar in spirit to the grppha clobber
  option (for technical reasons grppha is always run with the clobber option
  set to "yes" and the check on the output file is done independently).

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --suffix SUFFIX       suffix for the output files (default: grppha)
  --comm COMM           the GRPPHA command string (default: None)
  --chatter CHATTER     value of the GRPPHA chatter commad-line switch
                        (default: 5)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.5.5 xpsrccoords

```
...
```

7.5.6 xpancrkey

```
...

usage: xpancrkey.py [-h] [--arffiles ARFFILES [ARFFILES ...]]
                    [--suffix SUFFIX] [--overwrite {True,False}]
                    filelist [filelist ...]

Update the ANCRFILE keyword in the primary header of the target binned file.

This is germane to what the fparkey FTOOL
https://heasarc.gsfc.nasa.gov/lheasoft/ftools/fhelp/fparkey.html
is doing, except for the much more limited scope. On the plus side, this small
app will perform some basic check on the value being written in the header in
order to avoid common pitfalls.

positional arguments:
  filelist              path(s) to the input file(s)

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

(continues on next page)

(continued from previous page)

```
  --arffiles ARFFILES [ARFFILES ...]
                        path(s) to the new .arf files to use. (default: [])
  --suffix SUFFIX       suffix for the output files (default: ancrkey)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.5.7 xpchrgmap

```
...

usage: xpchrgmap.py [-h] [--outfolder OUTFOLDER] [--overwrite {True,False}]
                    filelist [filelist ...]

Extract charging maps from an observation file.

This application is meant to open an observation file, read the CHRG_MAP
extension and copy it in a different FITS file, adding the proper header to
the PRIMARY extension so that the format is the same as the actual charging map
files.

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --outfolder OUTFOLDER
                        path to the output folder (default:
                        /home/docs/ixpeobssimdata)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.5.8 xppicorr

```
...

usage: xppicorr.py [-h] [--offset OFFSET] [--slope SLOPE]
                   [--corrfile CORRFILE] [--deterministic {True,False}]
                   [--seed SEED] [--suffix SUFFIX] [--overwrite {True,False}]
                   filelist [filelist ...]

Correct the PI column in a level-2 file, given either a constant scale factor
and/or offset, or a proper FITS file with a time-dependent correction.

In order to avoid steps in the output corrected files, a floating-point smearing
factor in the [-0.5, 0.5] interval is added to the original PI before the
correction. When run in deterministic mode this factor is determined by the
event time (more precisely, using the fractional part of the timestamps), otherwise
we just throw a random number with a uniform distribution.
```

(continues on next page)

(continued from previous page)

```
Note this is mainly intended for debugging purposes---you are changing the
actual data, so you'd better know what you are doing :-)

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --offset OFFSET       global offset for the PI column (default: 0.0)
  --slope SLOPE         global scale factor for the PI column (default: 1.0)
  --corrfile CORRFILE   path to the FITS file for a time-dependent correction
                        (default: None)
  --deterministic {True,False}
                        run in deterministic mode (default: True)
  --seed SEED           random seed for the simulation (default: None)
  --suffix SUFFIX       suffix for the output files (default: picorr)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.5.9 xpsimfmt

```
...

usage: xpsimfmt.py [-h] [--suffix SUFFIX] [--weightname WEIGHTNAME]
                   [--auxversion AUXVERSION] [--mc {True,False}]
                   [--detphiname DETPHINAME] [--stripmode {FULL,L2}]
                   [--overwrite {True,False}]
                   filelist [filelist ...]

Format reconstructed event lists generated with ixpesim to make them
interoperable with ixpeobssim (e.g., for spectro-polarimetric fits in XSPEC).
Phrased in a slightly different way, this is tranforming the rough equivalent
of Level-1 files produced by ixpesim into the rough equivalent of Level-2 files.

This is adding all the necessary keywords to the relevant headers, as well as
a few columns in the EVENTS extensions that are necessary for the
spectro-polarimetric analysis in XSPEC, most notably:
- PI
- Q
- U
- RA, DEC
- X, Y
- W_MOM

Additionally, the script allows to strip off useless columns, producing smaller
output files that easier to exchange in several different context. This
functionality is controlled by the --stripmode command-line switch and, more
specifically:

* --stripmode FULL preserves the entire content of the ixpesim output files;
```

(continues on next page)

(continued from previous page)

```
* --stripmode L2 produces a smaller file that is genermane to Level 2 files and
  can be effectively used to test a spectro-polarimetric analysis with the
  photon-list workflow;

Note this requires the input files to be processed with a recent enough gpdsw
version (13.10.0 or later) in order for the conversion to be supported.

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --suffix SUFFIX       suffix for the output files (default: simfmt)
  --weightname WEIGHTNAME
                        name of the weights to be used (default: alpha075)
  --auxversion AUXVERSION
                        target version for the auxiliary files (default: 3)
  --mc {True,False}     use Monte Carlo information (default: False)
  --detphiname DETPHINAME
                        The column name for the azimuthal angle (default:
                        DETPHI2)
  --stripmode {FULL,L2}
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

7.5.10 xpsimspec

```
...

usage: xpsimspec.py [-h] [--emin EMIN] [--emax EMAX] [--ebins EBINS]
                    [--index INDEX] [--nH NH]

Write a text file to be fed as an input custom spectrum to ixpesim to
simulate a realistic astrophysical spectrum.

At the moment the spectrum is limited to a power law with an adjustable index,
and the H column density can be controlled via command line. The effect of the
mirror effective area and the transparency of the UV filter are taken into
account.

The output is a simple text file with two columns---the energy in MeV and the
flux in arbitrary units.

optional arguments:
  -h, --help     show this help message and exit
  --emin EMIN    minimum energy in keV (default: 1.0)
  --emax EMAX    maximum energy in keV (default: 12.0)
  --ebins EBINS  number of energy bins (default: 250)
  --index INDEX  the power-law index for energy spectrum (default: 2.0)
  --nH NH        the hydrogen column density (cm^{-2}) (default: 0.0)
```

7.5.11 xpstripmc

```
...

usage: xpstripmc.py [-h] [--suffix SUFFIX] [--overwrite {True,False}]
                    filelist [filelist ...]

Remove all the Monte Carlo information from ixpeobssim photon lists.

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --suffix SUFFIX       suffix for the output files (default: nomc)
  --overwrite {True,False}
                        overwrite existing output files (default: True)
```

CHAPTER

EIGHT

CREATING SOURCE MODELS

> **Warning**
>
> While this is possibly one of the most important pieces of documentation, it is still work in progress. Do not hesitate to complain if something is not clear incomplete, or simply wrong.

Source models are specified in ixpeobssim through Python configuration files—effectively simple Python modules. As we shall see in a second, this is the key of the flexibility in terms of source specification provided by the simulation framework. All the source model configuration files shipped with ixpeobssim are included in the [github]/ixpeobssim/config folder, and that is a good starting point to get up and running with creating your own observation-simulation setup.

In a nutshell, the one thing that a source configuration file *must* contain is an instance of the *ixpeobssim.srcmodel.roi.xROIModel* class called `ROI_MODEL` (mind that the name and case of the variable are important, as when you run a simulation the relevant application will try and import that exact name from the Python configuration module—and fail miserably if that is not possible). In the rest of this section we shall walk through the relevant classes for the definition of a region of interest (ROI) to be simulated.

8.1 Spectra and polarization patterns

Understanding how photon spectra and polarization patterns are specified in ixpeobssim is key to being able to create a functional simulation setup. The two basic rules, here, are:

1. the photon spectrum can be an arbitrary function of energy and time, expressed in units of $\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{keV}^{-1}$;
2. the polarization degree and angle can be arbitrary functions of energy, time, *and* position in the sky.

From the implementation standpoint, the photon spectrum and the polarization degree and angle are simply Python functions with the proper signature:

```
def spec(E, t):
    """Definition of the photon spectrum (cm^-2 s^-1 keV^-1).
    """
    # Function definition here.

def pol_deg(E, t, ra, dec):
    """Definition of the polarization degree (between 0 and 1).
    """
    # Function definition here.
```

(continues on next page)

(continued from previous page)

```
def pol_ang(E, t, ra, dec):
    """Definition of the polarization angle (in radians).
    """
    # Function definition here.
```

These functions are then used internally by the simulation code to generate a proper photon list, and we shall see in the next section how they are passed to the relevant classes. We emphasize that this approach is fairly flexible, as the user can put in the function bodies literally anything: a constant, an analitic formula, an interpolator constructed from a numeric table, etc. *The only important thing is to make sure that at runtime the function can be called with the proper signature* (and, by the way, can operate on numpy arrays—so always make sure that you use the numpy version of the mathematical functions that you need in your function body).

Note

At this point you might wonder why the rules are comparatively restrictive for the photon spectra which, in contrast to the polarization patterns, cannot be dependent on the position on the sky. Well, the reason is that a fully arbitrary source model doesn't really fit with the current implementation of the basic simulation strategy. But before you panic, a couple of things:

1. for extended sources, as we shall see in the next section, the intensity of the emission is determined by the spatial template and therefore it depends on the position—in other words it is really the spectral shape that is bound to be constant across the source extension;
2. there are ways to break an extended source in an arbitrary number of smaller components, each with its own independent spectral shape—which effectively allows to overcome the limitation, albeit in a possibly inefficient way;
3. the Chandra-to-IXPE conversion tool shipped with ixpeobssim provides an alternative observation-simluation strategy fully preserving the correlations between the sky position and the spectrak shape.

Say that you want, e.g., simulate a stationary source with a power-law energy spectrum; all you have to do is something along the lines of the following snippet.

```
def spec(E, t):
    """Definition of the photon spectrum (cm^-2 s^-1 keV^-1).

    Change the power-law normalization and index to the values that suit
    you. Also, note that t is not used, i.e., the source is stationary, but
    you are free to put in the function body whatever complicated expression
    of energy `and` time, e.g., a prefactor and an index that vary with time.
    """
    return 10. * (E**-2.)
```

On a related note, a polarization degree constant in energy, time and position in the sky is readily specified.

```
def pol_deg(E, t, ra, dec):
    """Definition of the polarization degree (between 0 and 1).
    """
    return 0.5
```

Note that in all cases the signature of the function must be the correct one, no matter whether any specific dynamical variable is effectively used in the function body or not.

Tip

Since we don't necessarily want you to reinvent the wheel every time, ixpeobssim provides convenience wrappers to power laws and constants, among other things. The first and last snippets above would be more succintly coded as

```
from ixpeobssim.srcmodel.spectrum import power_law
from ixpeobssim.srcmodel.polarization import constant

spec = power_law(10., 2.)
pol_deg = constant(0.5)
```

And you should take a look at the documentation of the two functions for more details (note that the power-law wrapper support time-varying prefactor and index).

8.1.1 Using XSPEC models

Arbitrary XSPEC models can be fed into xpobssim, providing that PyXSPEC is up and running on your system. (If you are in a hurry, the section about *Toy source models* provides a working example that you can take inspiration from.)

In order to effectively use an underlying XSPEC model, you have to create a *ixpeobssim.srcmodel.spectrum.xXspecModel* instance. This is essentially an univariate, interpolated spline (coming with all the standard spline facilities), with the only exception that it can be called with a phony, second argument (in addition to the energy) so that it can be readily used to define a source component—in this case the second argument is acting as the time and has no effect.

The easiest way to create an XSPEC model is through a generic expression and a list specifying all the model parameters.

```
from ixpeobssim.srcmodel.spectrum import xXspecModel

spec = xXspecModel('powerlaw', [2., 0.1])
```

The *ixpeobssim.srcmodel.spectrum.xXspecModel* constructor mimics the underlying XSPEC bindings and allows passing the parameters in the form of a dictionary indexed by sequential identifiers, which is handy when one wants to only set a subset of them and use the XSPEC defaults for the others:

```
from ixpeobssim.srcmodel.spectrum import xXspecModel

spec = xXspecModel('powerlaw', {1: 2., 2: 0.1})
spec = xXspecModel('powerlaw', {2: 0.1})
```

Alternatively, the model can be read from a text file with a suitable syntax, see *ixpeobssim.srcmodel.spectrum.xXspecModel.from_file()*

```
from ixpeobssim.srcmodel.spectrum import xXspecModel

spec = xXspecModel.from_file('path/to/your/file')
```

In addition, XSPEC models provide a convenient facility to dump the data points to an ASCII file, see *ixpeobssim.srcmodel.spectrum.xXspecModel.save_ascii()*.

8.2 Model components

The concept of a *model component* is central in the definition of an observation simulation setup, and is used throughout ixpeobssim to encapsulate the relevant properties of a few slightly different, but related, things:

- a simple celestial source that cannot be further decomposed, such as that defined in [github]/ixpeobssim/config/toy_point_source.py;
- different physical components of the same source—e.g., the thermal and non thermal emission in Cas A as modeled in [github]/ixpeobssim/config/toy_casa.py;
- different sources in the same ROI—e.g., the Crab pulsar and nebula in [github]/ixpeobssim/config/crab_complex.py.

From a technical standpoint, a model component is basically an instance of the *ixpeobssim.srcmodel.roi.xCelestialModelComponentBase* class (or, to be more precise, an instance of a subclass of the base class) and encapsulates, among other things, the source name, photon spectrum, and polarization degree and angle, the three latter quantities being specified at construction time as explained in the previous section. (Take a look at the signature of the constructors in the module documentation for more details.)

The subclasses of *ixpeobssim.srcmodel.roi.xModelComponentBase* deal for the most part (with the exception of that describing periodic sources) with the source morphology, re-implementing the method for extracting vectors of positions in the sky (ra, dec). For completeness, the actual subclasses that can be instantiated and used in simulations are

Source type	Specialized subclass
Point source	*ixpeobssim.srcmodel.roi.xPointSource*
Periodic source	*ixpeobssim.srcmodel.roi.xPeriodicPointSource*
Binary system	*ixpeobssim.srcmodel.roi.xBinarySource*
Uniform disk	*ixpeobssim.srcmodel.roi.xUniformDisk*
Gaussian disk	*ixpeobssim.srcmodel.roi.xGaussianDisk*
Uniform annulus	*ixpeobssim.srcmodel.roi.xUniformAnnulus*
Extended source	*ixpeobssim.srcmodel.roi.xExtendedSource*
Chandra observation	*ixpeobssim.srcmodel.roi.xChandraObservation*
Instrumental background	*ixpeobssim.srcmodel.bkg.xInstrumentalBkg*
Instrumental background	*ixpeobssim.srcmodel.bkg.xPowerLawInstrumentalBkg*
Extragalactic background	*ixpeobssim.srcmodel.bkg.xExtragalacticBkg*
Galactic background	*ixpeobssim.srcmodel.bkg.xGalacticBkg*

and provide a fairly broad and diverse selection of morphological characteristics that should be sufficient for most of the applications.

Now, the simplest possible model component (which in this case is a fully-fledged source) can be defined with only a few lines of code.

```
import numpy

from ixpeobssim.srcmodel.roi import xPointSource
from ixpeobssim.srcmodel.spectrum import power_law
from ixpeobssim.srcmodel.polarization import constant

ra, dec = 45., 45.
spec = power_law(10., 2.)
pol_deg = constant(0.5)
```

(continues on next page)

(continued from previous page)

```
pol_ang = constant(numpy.radians(65.))
src = xPointSource('Point source', ra, dec, spec, pol_deg, pol_ang)

print(src)
```

And, for completeness, the output of the last `print` function should be something along the lines of the following snippet. We're almost there.

```
...
xPointSource "Point source" (id = None)
    Galactic column density: 0.000e+00 cm^{-2}
    Redshift: 0.000
    Unabsorbed flux @ t = 0: 2.221e-08 erg/cm2/s (1110.55 mcrab)
    Position: RA = 45.0 deg, Dec = 45.0 deg
```

Note

ixpeobssim has a concept of Calibration sources and associated region of interest. The reader is referred to the *toy_flat_field.py* and *fcw_calC.py* configuration files for usage examples.

(Note that calibration configuration files are supposed to be run through *xpcalib* rather than *xpobssim*.)

8.3 Regions of interest

A complete model of a region of interest is basically a collection of an arbitrary number of model components—and it is the basic simulation unit in the context of ixpeobssim.

From an implementation standpoint, it is an instance of the *ixpeobssim.srcmodel.roi.xROIModel* class, and is specified passing the coordinates of its center; components are added after the fact. A small tweak to the last snippet in the previous section

```
import numpy

from ixpeobssim.srcmodel.roi import xPointSource, xROIModel
from ixpeobssim.srcmodel.spectrum import power_law
from ixpeobssim.srcmodel.polarization import constant

# Define the source properties and create the actual source.
ra, dec = 45., 45.
spec = power_law(10., 2.)
pol_deg = constant(0.5)
pol_ang = constant(numpy.radians(65.))
src = xPointSource('Point source', ra, dec, spec, pol_deg, pol_ang)

# Create the complete region of interest.
ROI_MODEL = xROIModel(ra, dec)
ROI_MODEL.add_source(src)

print(ROI_MODEL)
```

make it for a valid ixpeobsim configuration file that can be used to run an actual simulation. The `print` output will look like this.

```
...
ROI centered at (45.0000, 45.0000):
- xPointSource "Point source" (id = 0)
    Galactic column density: 0.000e+00 cm^{-2}
    Redshift: 0.000
    Unabsorbed flux @ t = 0: 2.221e-08 erg/cm2/s (1110.55 mcrab)
    Position: RA = 45.0 deg, Dec = 45.0 deg
```

8.4 Chandra region of interest

The region of interest for a Chandra observation follows the same concept as described above, with the difference that the spectral and morphological information of the source is directly taken from the provided Chandra event list and does not need to be specified. Then the xpobssim tool will take care to convert the Chandra observation into a correponding IXPE event list, opportunely down-sampling and smearing the events with the instrument response functions. This simulation technique has the advantage to preserve the full correlation between the source morphology and the spectrum, especially important for extended sources.

From tecnical standpoint, it is an instance of the *ixpeobssim.srcmodel.roi.xChandraROIModel* class, derived from the *ixpeobssim.srcmodel.roi.xROIModel* base class, and is initialized by passing the Chandra event file and choosing the ACIS detector used for the observation (I or S). Sub-regions of the ROI can be defined (with individual polarization models) as instances of the *ixpeobssim.srcmodel.roi.xChandraObservation* class by specifying the id, polarization degree and angle and the region definition (via a ds9 region file).

```
import os

from ixpeobssim.srcmodel.roi import xChandraObservation, xChandraROIModel
from ixpeobssim.srcmodel.polarization import constant
from ixpeobssim.utils.astro import read_ds9
from ixpeobssim import IXPEOBSSIM_CONFIG

# This is the input event file taken from the Chandra database (obs id 8489)
# http://cda.harvard.edu/pop/mainEntry.do
# ftp://cdaftp.cfa.harvard.edu//pub/science/ao08/cat7/8489/primary/acisf08489N003_evt2.
↪fits.gz
EVENT_FILE_PATH = os.path.join(IXPEOBSSIM_CONFIG, 'fits', 'cena_evt.fits')

# cena_jet+core contains the definition of jet and core in the Chandra map
REG_SOURCE_FILE_PATH = os.path.join(IXPEOBSSIM_CONFIG, 'fits',
                                    'cena_jet+core.reg')

regs = read_ds9(REG_SOURCE_FILE_PATH)

ROI_MODEL = xChandraROIModel(EVENT_FILE_PATH, acis='I')

polarization_degree = constant(0.)
polarization_angle = constant(0.)

# jet region
jet = xChandraObservation('Jet', polarization_degree, polarization_angle, regs[0])
```

(continues on next page)

(continued from previous page)

```
ROI_MODEL.add_source(jet)
# core region
core = xChandraObservation('Core', polarization_degree, polarization_angle, regs[1])
ROI_MODEL.add_source(core)
# the remaining part
core = xChandraObservation('Others', polarization_degree, polarization_angle)
ROI_MODEL.add_source(core)
```

It is also possible to remove sources in the region of interest by passing the exclude option in the instance of the *ixpeobssim.srcmodel.roi.xChandraObservation* class and to add sources using the standard model components described above. These features are particularly useful for treating the pileup phenomenon occurring in Chandra CCD detectors for bright sources, whenever two or more photons are detected as a single event. In these cases, the simplest strategy is to remove the bright source suffering of pileup from the conversion ROI and replace it with a standard model component with an appropriate source model description. The converter will then take care of it, simulating the source in the standard way and merging the photon lists at the end of the process. For reference, here are the additional lines that need to be included to the source model example above to remove a region and add a point source to the chandra region of interest:

```
# core region, exclude the core region by passing the exclude=True
core = xChandraObservation('Core', polarization_degree, polarization_angle,
                           regions[1], exclude=True)
ROI_MODEL.add_source(core)
# the remaining part
core = xChandraObservation('Others', polarization_degree, polarization_angle)
ROI_MODEL.add_source(core)

#Add a point source at a given location in the roi.
RA = 201.43
DEC = -43.00
PL_NORM = 0.0004,
PL_INDEX = 2.
spec = power_law(PL_NORM, PL_INDEX)
src = xPointSource('Point source', RA, DEC, spec, polarization_degree,
                   polarization_angle)
ROI_MODEL.add_source(src)
```

8.5 Source model etiquette

Although the ROI_MODEL top-level object is all ixpeobssim is looking for in a configuration file (and everything will run happily as long as you provide that in a valid form) there is undoubtedly value in trying and keep a consistent style when creating source models. People will get acquainted to the conventions and will understand new models more easily. This section is essentially a list of suggestions on how to go about creating your own models.

We note, in passing, that the section about *Toy source models* is a good read, along with the material in this section.

Give your model file a sensible name (which, in most cases, should be obvious). It is conceivable that we might have multiple, alternative, configuration files for a given source; in this case, use your best judgement to try and convey with the file name the *intent* of any of these alternative models. (Having crab.py, crab2.py, crab_new.py, crab_newer.py stops being funny after a while.) Even more important, try and factor out as much as possible of the code that is shared across multiple models—resist the temptation to cut and paste code around!

Try and document your model in a reasonable fashion. This is typically achieved by putting arbitrary reStructuredText that can be then parsed in sphinx at the beginning of the Python configuration module, right after the license notice and before the `__future__` Python imports.

> **Warning**
>
> For the documentation to be parsed by sphinx, the reStructuredText *has* to be before any actual Python code. Keep this in mind.

When you run a simulation and you want to compare the ixpeobssim output with the input model, you will need to have all the input parameters at hand. Keep this in mind when writing configuration files:

```
spec = power_law(1., 2.)
```

is more succinct than

```
pl_norm = 1.
pl_index = 2.
spec = power_law(pl_norm, pl_index)
```

but the second formulation makes it easier for you to retrieve the input spectral index when you fit your binned `PHA1` spectrum in xspec and want to compare the best-fit parameters with the models.

Take a second to read our *Coding guidelines* and try and stick to them. In the long run people will be grateful to you for doing it.

Create a `display()` method in each configuration file that, when called, will plot anything that might be interesting in you model. (Each source model is different and it is hard to achieve this through a common, completely automated interface.) This method will not partecipate in the observation simulation when the model is parsed by ixpeobssim, but is something that, when protected by a `if __name__ == '__main__'` can be useful as a quick-look interface and for creating figures for the documentation. To this end, we have implemented a small bootstrapping function that you are welcome to use by doing something along the lines of

```
if __name__ == '__main__':
    from ixpeobssim.config import bootstrap_display
    bootstrap_display()
```

This will create a custom option parse for you with a limited set of options (e.g., to save the plots in a specific folder) and will fire up your `display()` method right away. Try at any time

```
python path/to/config/file.py --help
```

to have an up-to date help output.

Being consistent in terms of axis labels and units, conventions for file names and all that is cool. Really. Take a seconds to acquaint yourself with the following small utility functions and classes:

- *ixpeobssim.utils.matplotlib_.save_gcf()*: save the current figure
- *ixpeobssim.utils.matplotlib_.setup_gca()*: setup the current plot
- *ixpeobssim.utils.fmtaxis.fmtaxis*: list of axis labels and units

CHAPTER

NINE

TOY SOURCE MODELS

Toy source models are a great way to get acquainted with the internals of ixpeobssim and learn how to build physically meaningful models. Below is a short gallery of simple models illustrating the basic functionalities of ixpeobssim.

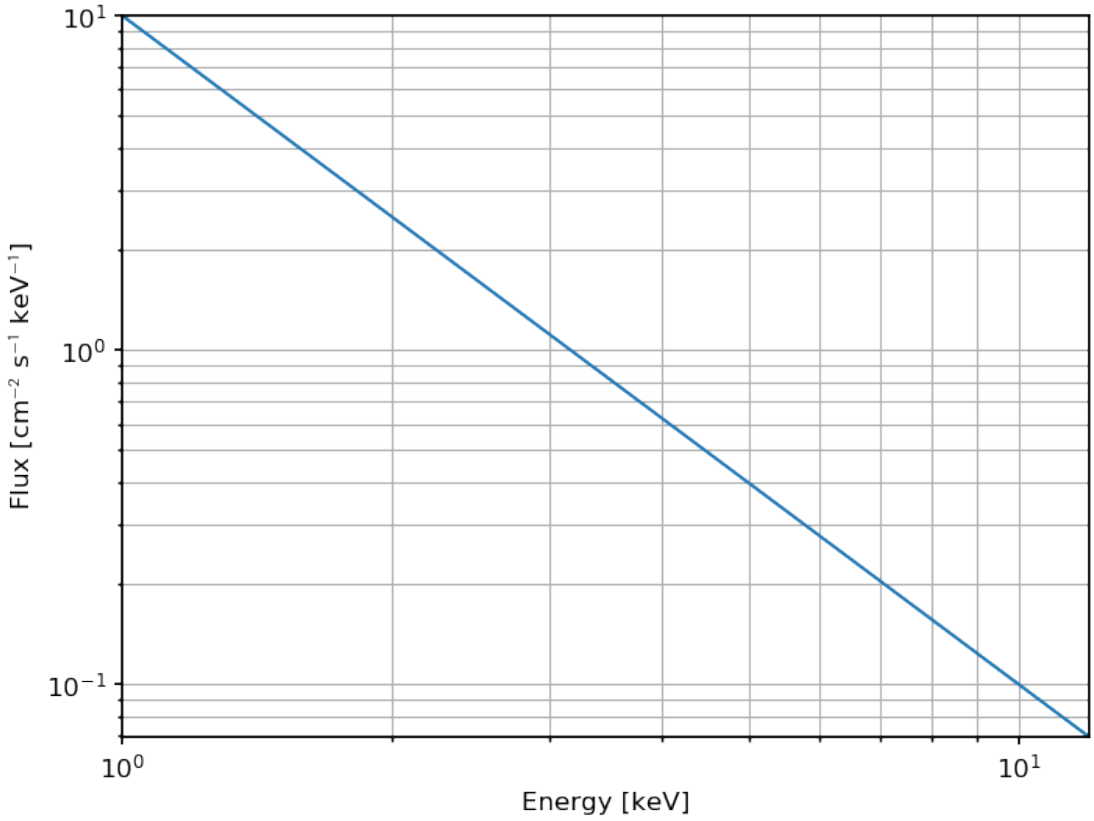
9.1 Single point source

- Source file: [github]/ixpeobssim/config/toy_point_source.py
- Simulation/analysis pipeline: [github]/ixpeobssim/examples/toy_point_source.py

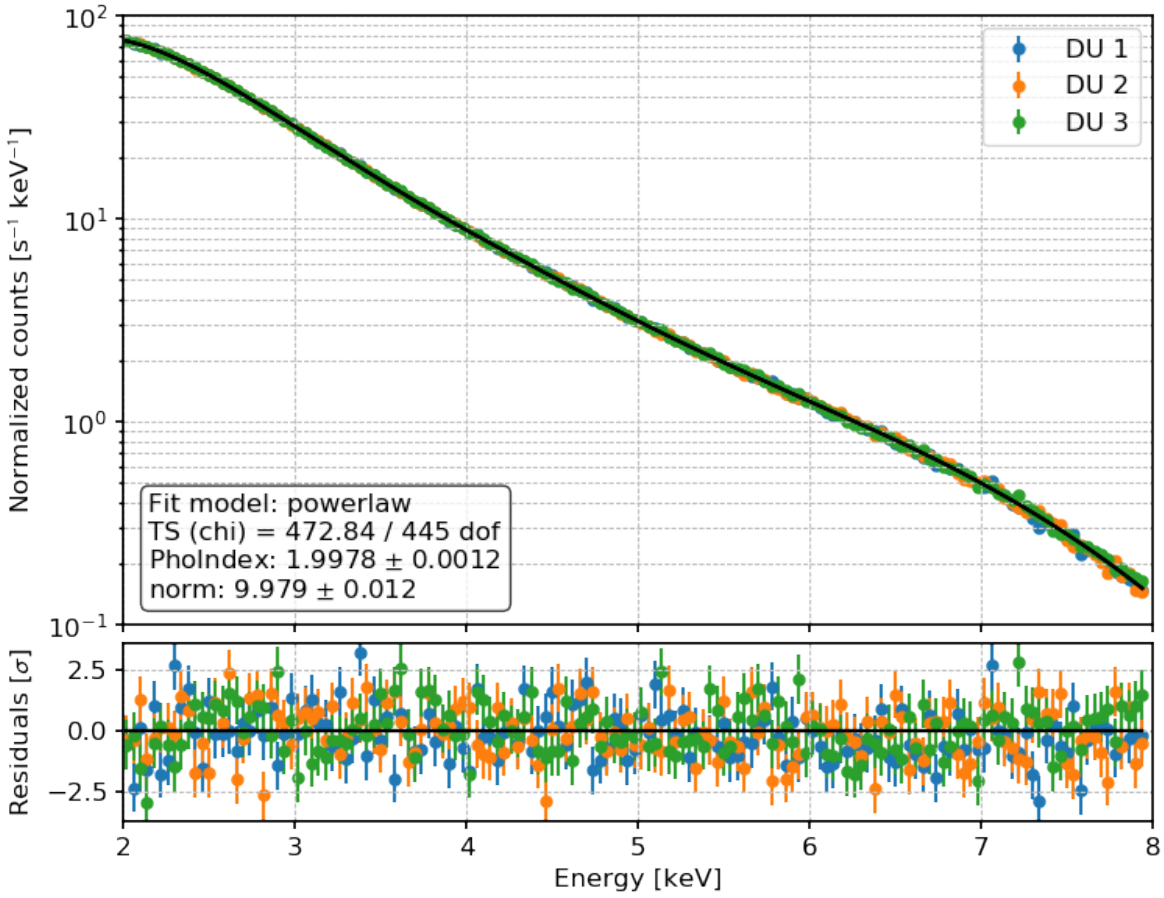
```
import numpy

from ixpeobssim.config import file_path_to_model_name
from ixpeobssim.utils.matplotlib_ import plt, setup_gca
from ixpeobssim.srcmodel.roi import xPointSource, xROIModel
from ixpeobssim.srcmodel.spectrum import power_law
from ixpeobssim.srcmodel.polarization import constant
from ixpeobssim.utils.fmtaxis import fmtaxis


__model__ = file_path_to_model_name(__file__)
ra, dec = 30., 45.
pl_norm = 10.
pl_index = 2.
spec = power_law(pl_norm, pl_index)
pd = 0.1
pa = 30.
pol_deg = constant(pd)
pol_ang = constant(numpy.radians(pa))

src = xPointSource('Point source', ra, dec, spec, pol_deg, pol_ang)

ROI_MODEL = xROIModel(ra, dec, src)
```

This is possibly the simplest interesting model that can be simulated: a point source with a time-independent power-law spectrum and time- and energy-independent polarization degree and angle.

The only meaningful feature of the source that can be plotted is its energy spectrum, which is shown below.



And here is a spectral fit with XSPEC to a set of event lists created with xpobssim and binned with xpbin:



9.2 Two point sources

- Source file: [github]/ixpeobssim/config/toy_mutiple_sources.py
- Simulation/analysis pipeline: [github]/ixpeobssim/examples/toy_mutiple_sources.py

```
import numpy

from ixpeobssim.config import file_path_to_model_name
from ixpeobssim.utils.matplotlib_ import plt, setup_gca
from ixpeobssim.srcmodel.roi import xPointSource, xROIModel
from ixpeobssim.srcmodel.spectrum import power_law
from ixpeobssim.srcmodel.polarization import constant
from ixpeobssim.utils.units_ import arcmin_to_degrees
from ixpeobssim.utils.fmtaxis import fmtaxis


__model__ = file_path_to_model_name(__file__)
ra, dec = 30., 45.
```

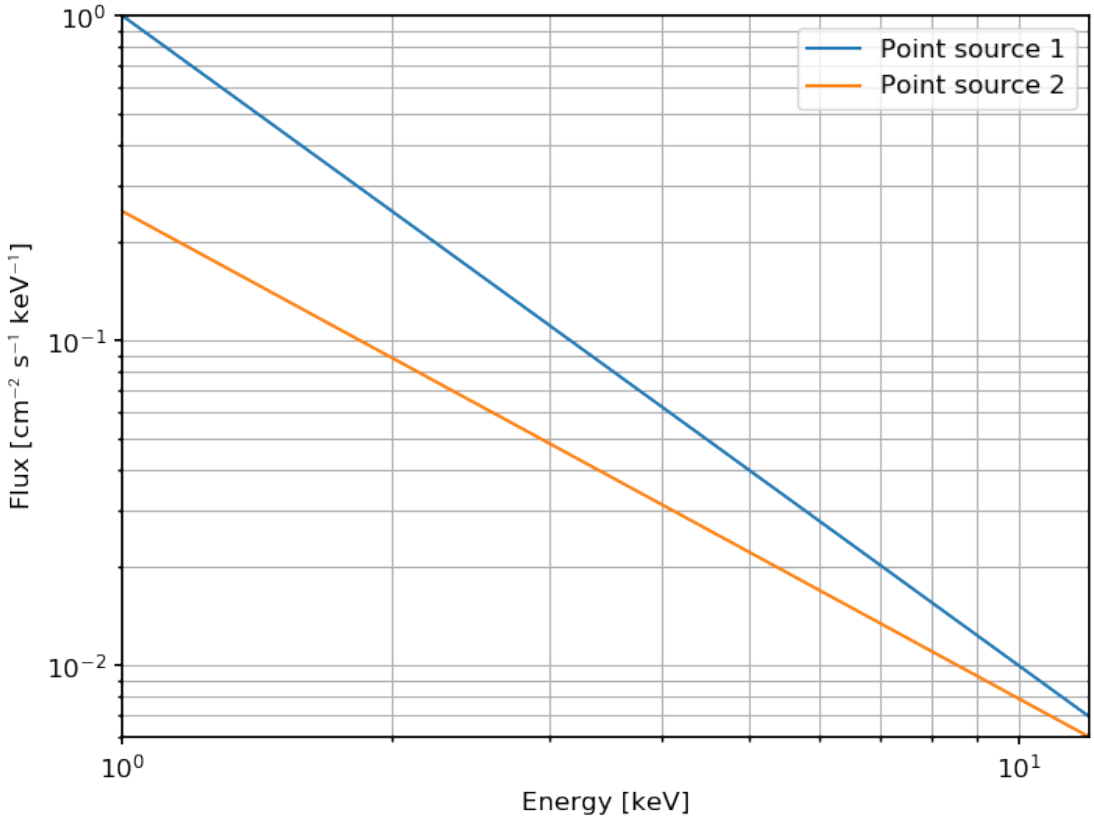
(continues on next page)

(continued from previous page)

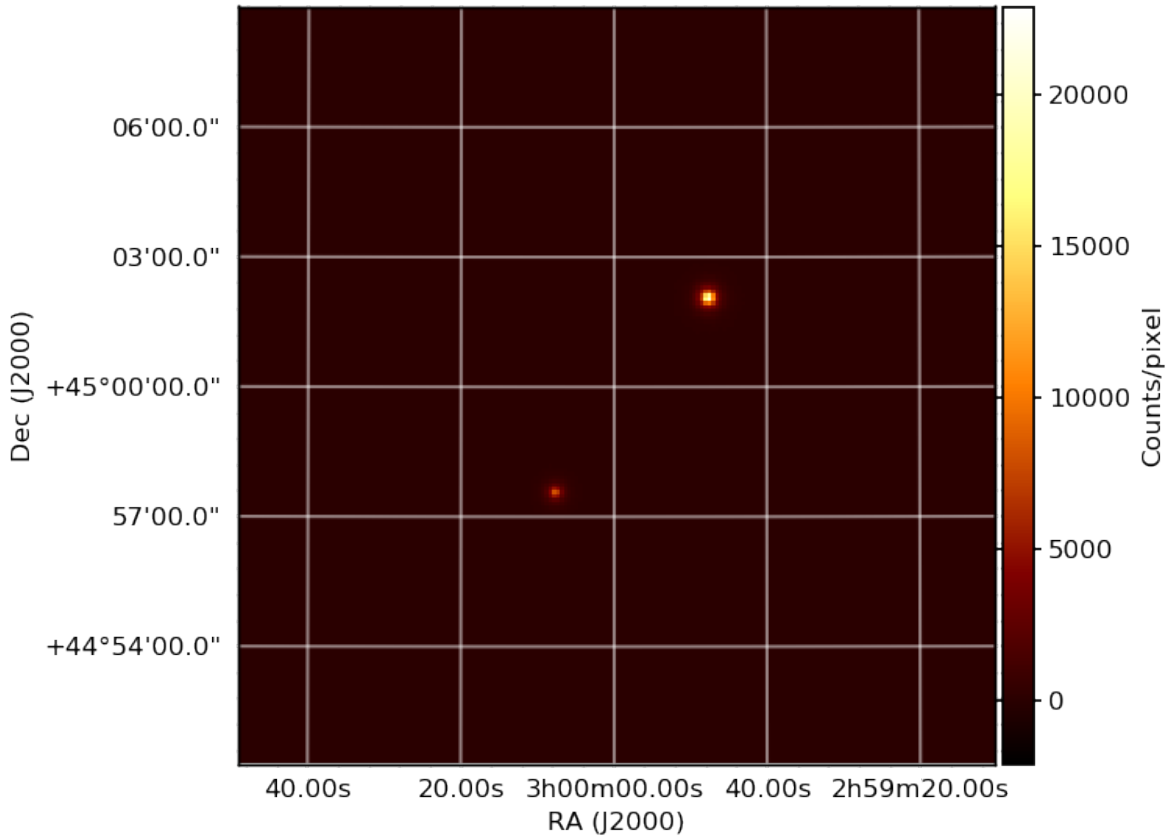
```
ra1, dec1 = ra, dec
pl_norm1 = 1.
pl_index1 = 2.
spec1 = power_law(pl_norm1, pl_index1)
pol_deg1 = constant(0.25)
pol_ang1 = constant(numpy.radians(30.))

ra2, dec2 = ra + arcmin_to_degrees(2.), dec - arcmin_to_degrees(3.)
pl_norm2 = 0.25
pl_index2 = 1.5
spec2 = power_law(pl_norm2, pl_index2)
pol_deg2 = constant(0.)
pol_ang2 = constant(numpy.radians(0.))

src1 = xPointSource('Point source 1', ra1, dec1, spec1, pol_deg1, pol_ang1)
src2 = xPointSource('Point source 2', ra2, dec2, spec2, pol_deg2, pol_ang2)

ROI_MODEL = xROIModel(ra, dec, src1, src2)
```

This is a slight variation over the single point source, where we include in the same region of interest two stationary point sources, with different power-law spectra and polarization patterns, offset by a few arcminutes with respect to the center of the ROI.

For completeness, below are the energy spectra for the two sources.



And here is a count map created from a simulated observation of this ROI model using xpbin in CMAP mode.



Tip

In this case we are using the ability of ixpeobssim to overlay multiple sources in the same ROI to simulate two physically distinct objects, but subclasses of *ixpeobssim.srcmodel.roi.xModelComponentBase* should in fact be thought of as *model components*, and this is the very same technique that we use to simulate different physical components of the same object.

9.3 Uniform disk

- Source file: [github]/ixpeobssim/config/toy_disk.py
- Simulation/analysis pipeline: [github]/ixpeobssim/examples/toy_disk.py

```
import numpy

from ixpeobssim.config import file_path_to_model_name
from ixpeobssim.utils.matplotlib_ import plt, setup_gca
from ixpeobssim.srcmodel.roi import xUniformDisk, xROIModel
from ixpeobssim.srcmodel.spectrum import power_law
from ixpeobssim.srcmodel.polarization import constant
```

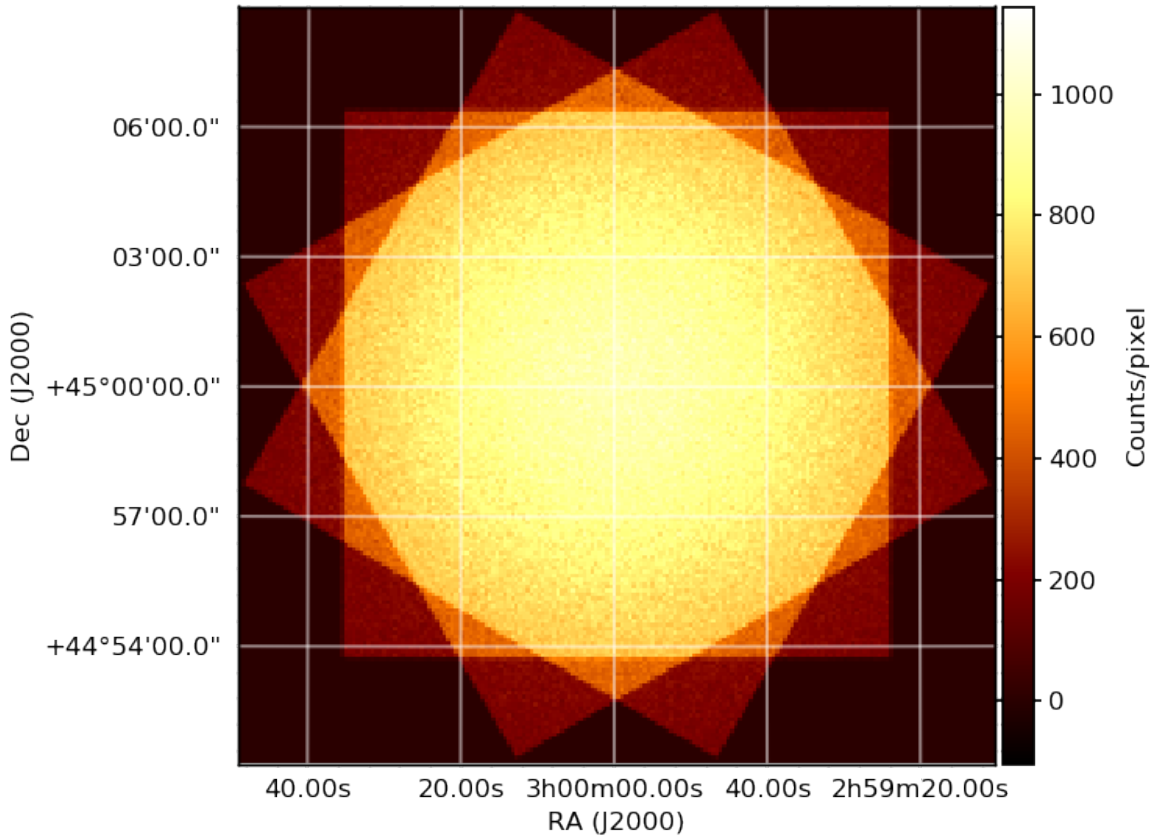
(continues on next page)

(continued from previous page)

```
from ixpeobssim.utils.units_ import arcmin_to_degrees
from ixpeobssim.utils.fmtaxis import fmtaxis


__model__ = file_path_to_model_name(__file__)
ra, dec = 45., 45.
radius = arcmin_to_degrees(10.)
pl_norm = 1.
pl_index = 2.
spec = power_law(pl_norm, pl_index)
pol_deg = constant(0.5)
pol_ang = constant(numpy.radians(65.))

src = xUniformDisk('Uniform disk', ra, dec, radius, spec, pol_deg, pol_ang)

ROI_MODEL = xROIModel(ra, dec, src)
```

This is an extended source with the morphology of a uniform disk and a simple power-law spectrum.

The most interesting feature of this toy model is possibly the fact that the disk is larger than the field of view and therefore, if one runs a long-enough simulation and creates a binned map out of the event file, the vignetting of the optics and the footprint of the three focal-plane detector units become evident.



9.4 Gaussian disk

- Source file: [github]/ixpeobssim/config/toy_gauss_disk.py
- Simulation/analysis pipeline: [github]/ixpeobssim/examples/toy_gauss_disk.py

```
import numpy

from ixpeobssim.config import file_path_to_model_name
from ixpeobssim.utils.matplotlib_ import plt, setup_gca
from ixpeobssim.srcmodel.roi import xGaussianDisk, xROIModel
from ixpeobssim.srcmodel.spectrum import power_law
from ixpeobssim.srcmodel.polarization import constant
from ixpeobssim.utils.units_ import arcmin_to_degrees
from ixpeobssim.utils.fmtaxis import fmtaxis


__model__ = file_path_to_model_name(__file__)
ra, dec = 45., 45.
radius = arcmin_to_degrees(2.)
pl_norm = 1.
```

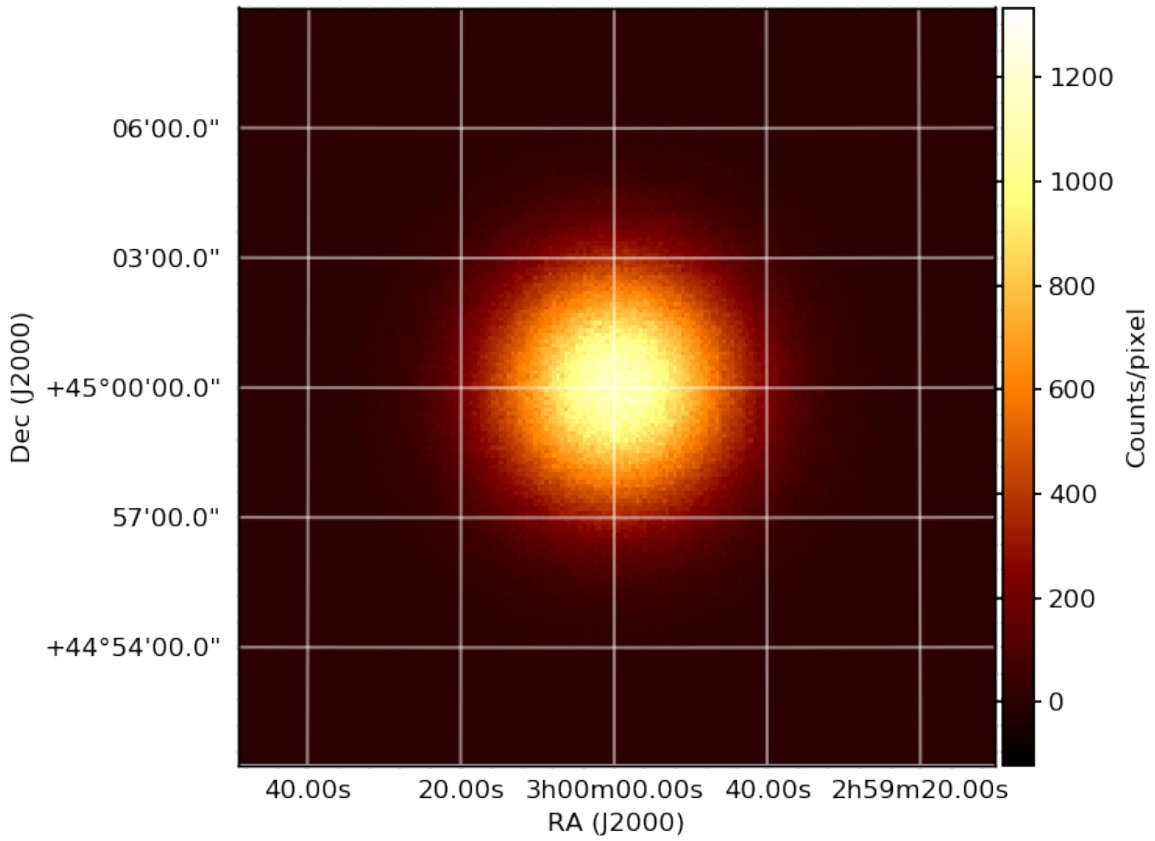
(continues on next page)

(continued from previous page)

```
pl_index = 2.
spec = power_law(pl_norm, pl_index)
pol_deg = constant(0.5)
pol_ang = constant(numpy.radians(65.))

src = xGaussianDisk('Gaussian disk', ra, dec, radius, spec, pol_deg, pol_ang)

ROI_MODEL = xROIModel(ra, dec, src)
```

This is very similar to the uniform disk example, except that the morpholgy of the source is a bivariate gaussian (with a diagonal covariance matrix) and the source does not extend as much across the field of view, so that it is pretty much completely contained.

Below is yet another binned coount map from xpbin in CMAP mode.



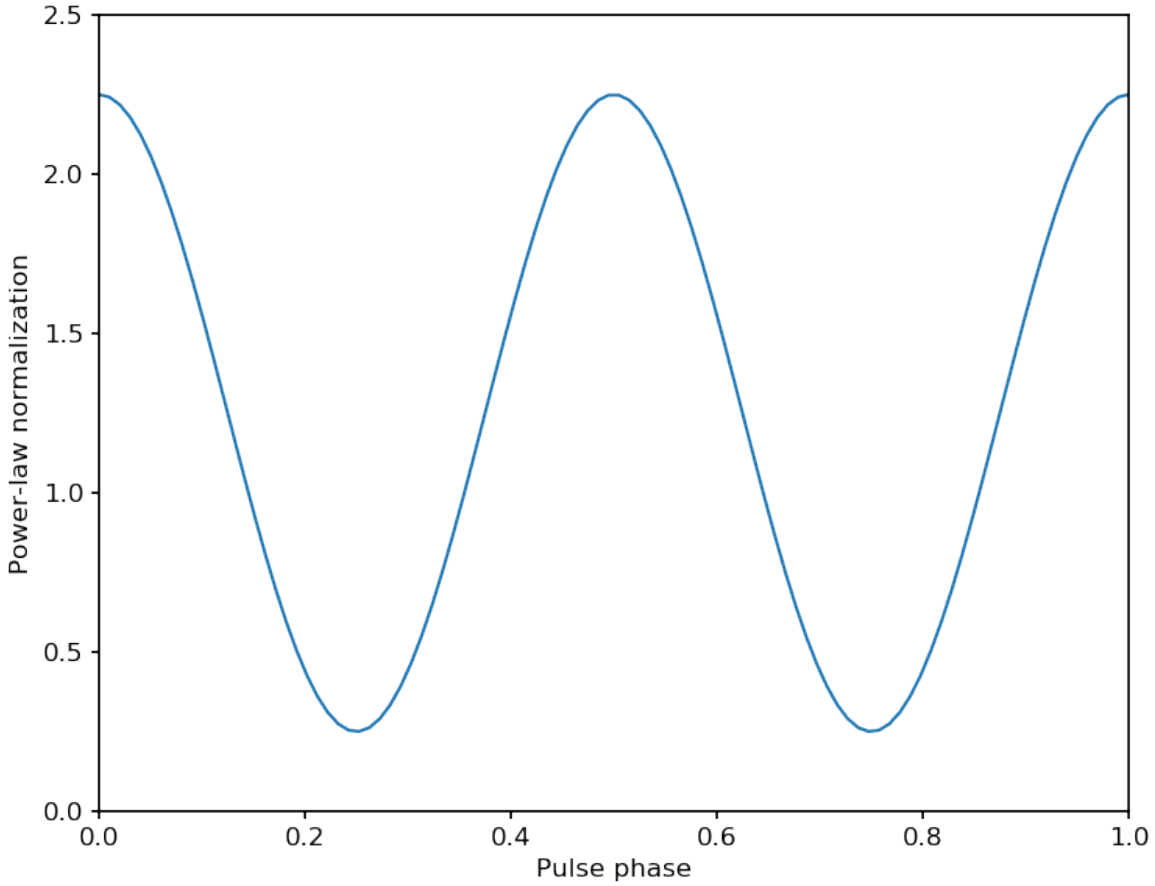
9.5 A periodic source

- Source file: [github]/ixpeobssim/config/toy_periodic_source.py
- Simulation/analysis pipeline: [github]/ixpeobssim/examples/toy_periodic_source.py

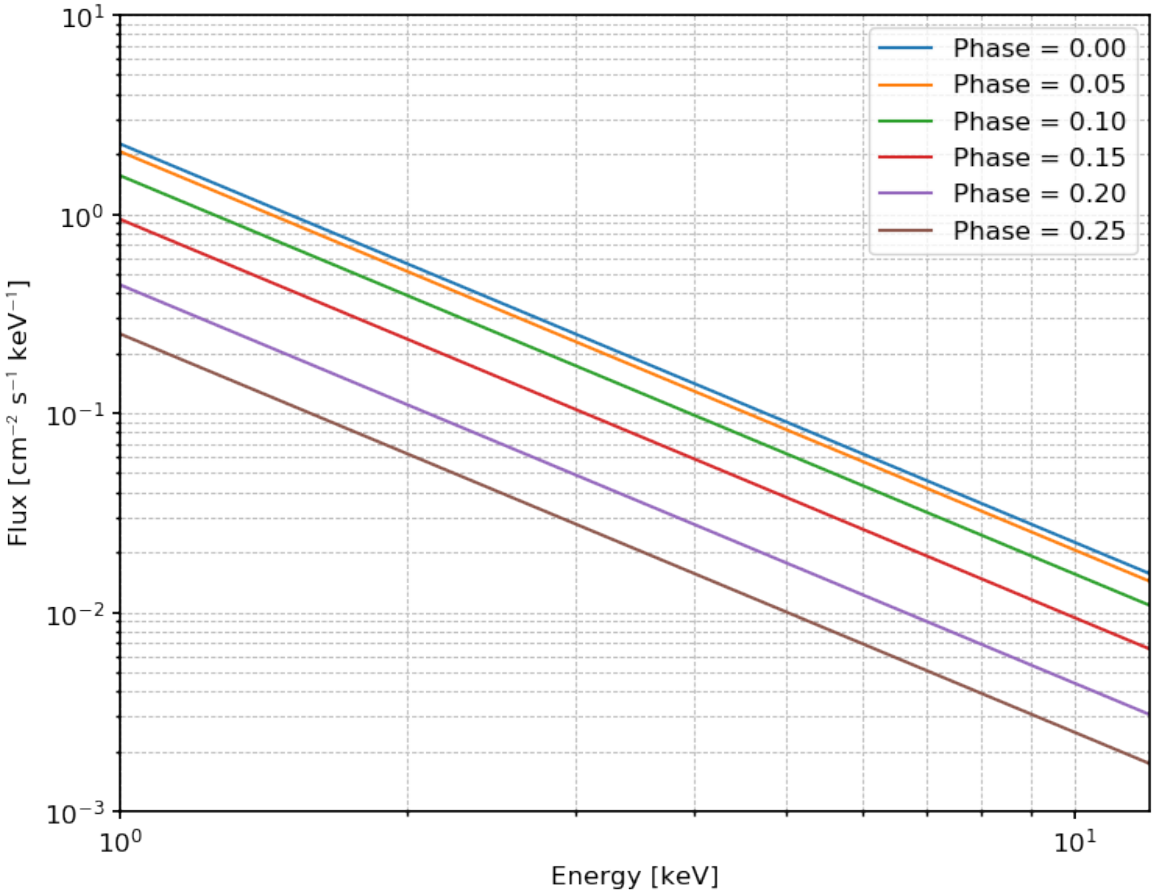
```
import numpy

from ixpeobssim.config import file_path_to_model_name
from ixpeobssim.utils.matplotlib_ import plt, setup_gca
from ixpeobssim.srcmodel.roi import xPeriodicPointSource, xROIModel
from ixpeobssim.srcmodel.ephemeris import xEphemeris
from ixpeobssim.srcmodel.spectrum import power_law
from ixpeobssim.srcmodel.polarization import constant
from ixpeobssim.utils.fmtaxis import fmtaxis
from ixpeobssim.utils.time_ import string_to_met_utc


__model__ = file_path_to_model_name(__file__)
ra, dec = 45., 45.

def pl_norm(phase):
    """Energy spectrum: power-law normalization as a function of the pulse
    phase.
    """
    return 2. * (1.25 + numpy.cos(4 * numpy.pi * phase))

pl_index = 2.
spec = power_law(pl_norm, pl_index)

def pol_deg(E, phase, ra=None, dec=None):
    """Polarization degree as a function of the dynamical variables.

    Since we're dealing with a point source the sky direction (ra, dec) is
    irrelevant and, as they are not used, defaulting the corresponding arguments
    to None allows to call the function passing the energy and phase only.
    """
    norm = numpy.clip(E / 10., 0., 1.)
    return norm * (0.5 + 0.25 * numpy.cos(4 * numpy.pi * (phase - 0.25)))

pol_ang = constant(numpy.radians(65.))

start_date = '2022-04-21'
t0 = string_to_met_utc(start_date, lazy=True)
nu0 = 0.1363729749
nudot0 = 0.543638e-13
ephemeris = xEphemeris(t0, nu0, nudot0)
src = xPeriodicPointSource('Periodic source', ra, dec, spec, pol_deg, pol_ang, ephemeris)

ROI_MODEL = xROIModel(ra, dec, src)
```

This is a fairly complicated toy example, illustrating a number of ixpeobssim features.

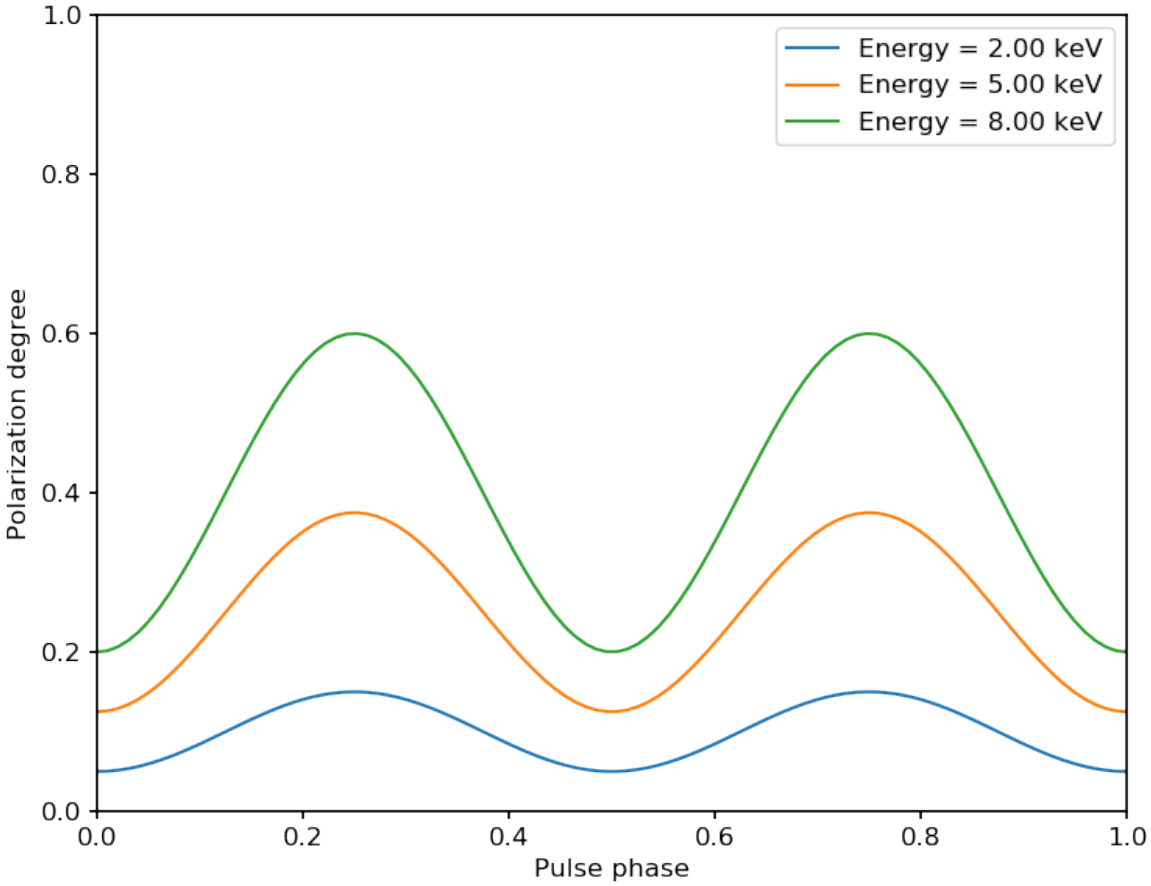
The ROI includes a single point source with a power-law spectrum, but this time around, the source being periodic, the corresponding power-law normalization varies with the pulse phase according to a simple analytical formula. (In order to make the comparison of the simulation output with the input model straightforward the power-law index is constant.) Below is the normalization as a function of the pulse phase



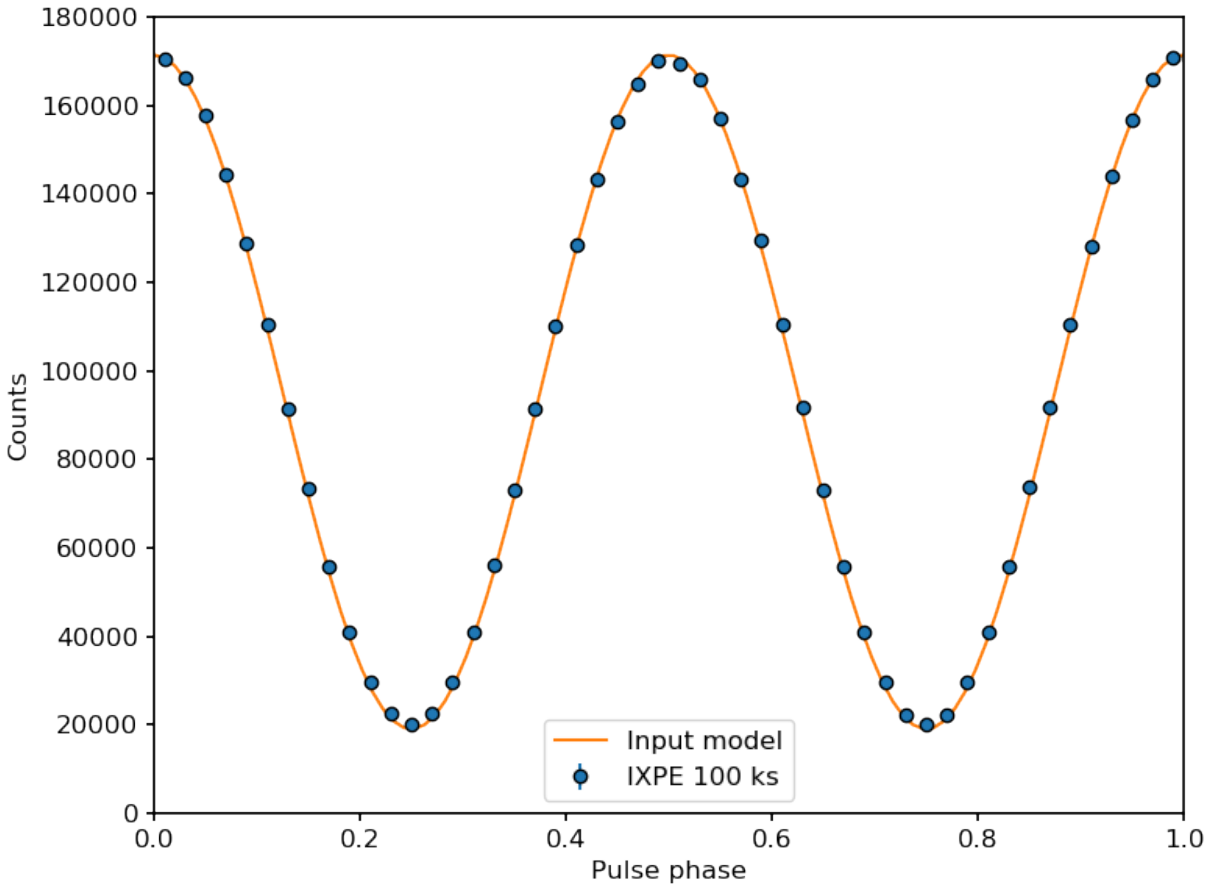
along with the actual differential energy spectrum, plotted at several different pulse phases.



The polarization angle is constant, but for illustration purposes the polarization degree varies with the pulse phase according to yet another simple formula, as shown in the plot below:

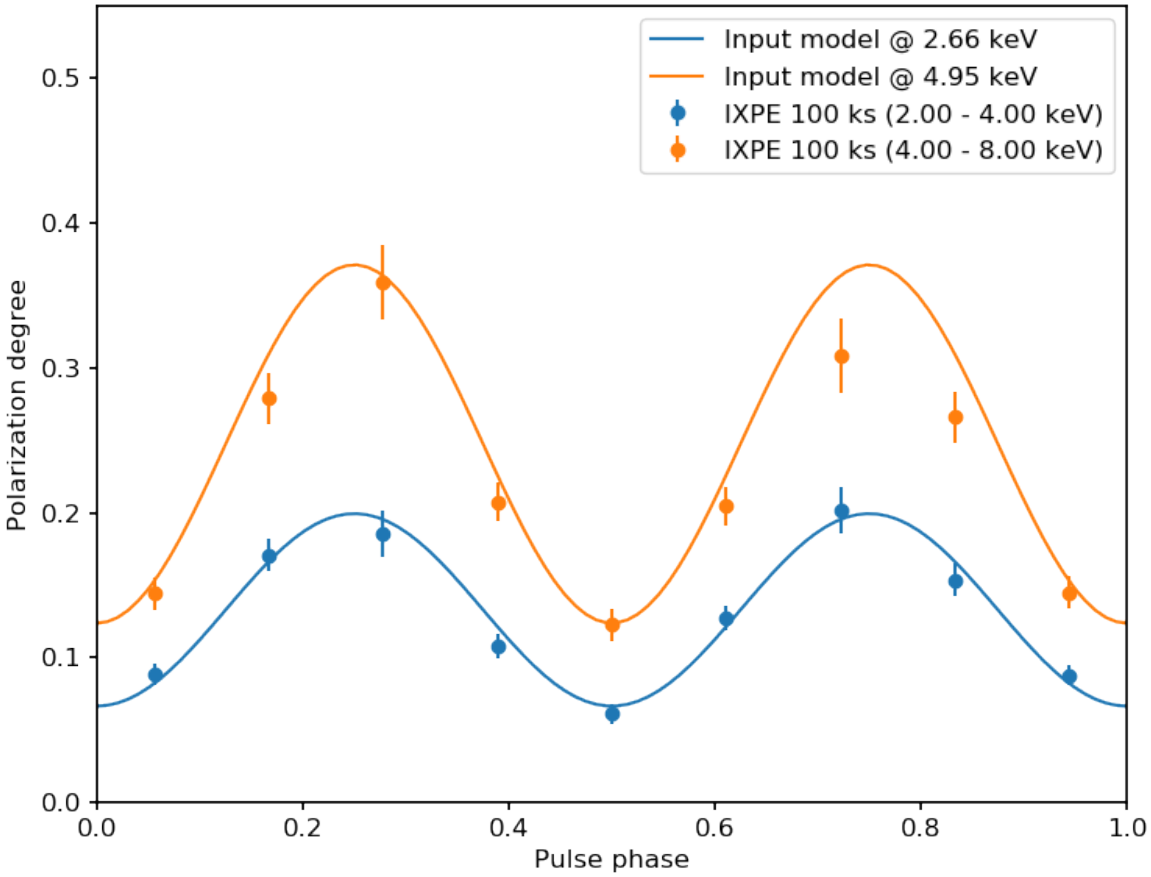


Here is an example of a binned pulse profile from an event list produced by ixpeobssim, compared with the input model.



And below is an example of a slightly more complicated analysis, where we

- fold the original event list with the ephemeris of the input model;
- split the original even list into a number of phase bins;
- estimate the polarization degree into two energy bins (within each phase bin) and compare with the input model.



For completeness, the polarization degree is evaluated by summing the event-by-event Stokes parameters, i.e., without taking into account the energy redstribution—which is probably account for most of the slight bias that can be see in the plot. The input model is evaluated at the average photon energy within each bin.

9.6 A model from XSPEC

- Source file: [github]/ixpeobssim/config/toy_xspec_point_source.py
- Simulation/analysis pipeline: [github]/ixpeobssim/examples/toy_xspec_point_source.py

```
import numpy

from ixpeobssim.config import file_path_to_model_name
from ixpeobssim.utils.matplotlib_ import plt, setup_gca
from ixpeobssim.srcmodel.roi import xPointSource, xROIModel
from ixpeobssim.srcmodel.polarization import constant
from ixpeobssim.srcmodel.spectrum import xXspecModel


__model__ = file_path_to_model_name(__file__)
ra, dec = 45., 45.
```

(continues on next page)

(continued from previous page)

```
# Parameters for a complex XSPEC spectral model phabs*(bbody+powerlaw)
expression = 'phabs * (bbody + powerlaw)'
col_dens = 1.
bb_temp = 1. #in keV
# bb_norm is the luminosity in units of 1e39 erg/s over
# the distance squared in units of 10 kpc
bb_norm = 1e-3 / (0.1**2)
pl_index = 2.
pl_norm = 10.

spec = xXspecModel(expression, [col_dens, bb_temp, bb_norm, pl_index, pl_norm])
pd = 0.5
pa = 30.
pol_deg = constant(pd)
pol_ang = constant(numpy.radians(pa))

src = xPointSource('Point source', ra, dec, spec, pol_deg, pol_ang)

ROI_MODEL = xROIModel(ra, dec, src)
```

CHAPTER

TEN

DATA CHALLENGE 1

The photon lists for the first IXPE Data Challence (DC1) were released on June 1, 2021. (The data repository contains exactly three files per observation—one for each of the three telescopes.)

All the input models have been merged into the main ixpeobssim repository as of version 16.16.0, and this page is acting as the documentation for them.

All the model files live in the usual `ixpeobssim/config` directory.

10.1 Pulsar ephemeris

Based on the consideration that, under most circumstances, the ephemeris for known periodic sources will be available when IXPE observes the target, we provide below those used for the simulation for the two pulsars involved in the data challenge.

10.1.1 PSR J1708-4008

```
* Source name: PSR J1708-4008
* R. A.      : 257.204167
* Dec.       : -40.152778
* Epoch MJD  : 59823.000000 (MET 178761600.000 s)
* nu0        : 0.090798237 Hz
* nudot0     : -1.598360e-13 Hz s^{-1}
* nuddot     : 0.000000e+00 Hz s^{-2}
```

10.1.2 RX B1509-58

```
* Source name: RX B1509-58
* R. A.      : 228.481333
* Dec.       : -59.135778
* Epoch MJD  : 59611.000000 (MET 160444800.000 s)
* nu0        : 6.572649741 Hz
* nudot0     : -6.582043e-11 Hz s^{-1}
* nuddot     : 1.918559e-21 Hz s^{-2}
```

10.2 Multi-wavelength context

For Mrk 421 we do have quasi-simultaneous optical polarization information available for the six IXPE observations, as listed in the following table.

MJD	R [mag]	Pol. Degree [%]	Pol. Angle [deg]
59692.904	12.53 +/- 0.05	2.60 +/- 0.20	39.3 +/- 3.0
59706.556	12.51 +/- 0.05	2.88 +/- 0.20	47.6 +/- 3.0
59720.754	11.97 +/- 0.05	6.01 +/- 0.20	-47.1 +/- 3.0
59728.883	12.44 +/- 0.05	3.14 +/- 0.20	41.0 +/- 3.0
59734.496	12.44 +/- 0.05	2.82 +/- 0.20	42.2 +/- 3.0
59748.431	12.49 +/- 0.05	3.40 +/- 0.20	41.5 +/- 3.0

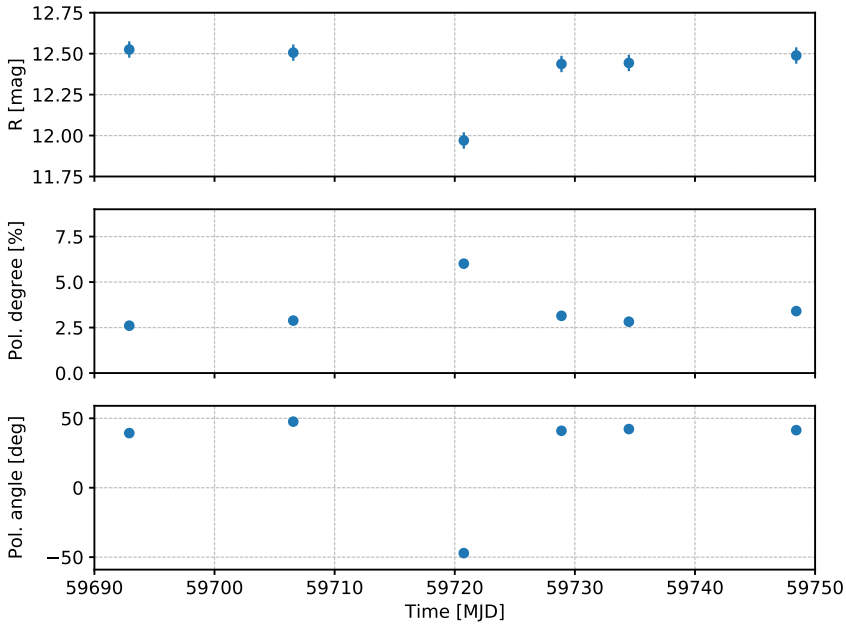


Fig. 1: Quasi-simultaneous optical polarization for Markarian 421.

10.3 Observing Plan

10.3.1 Cygnus X-1

- Source: Cygnus X-1 in the hard state
- Observation pattern: single ~300 ks observation
- Observation start: 2022-04-01T00:00:00.000000

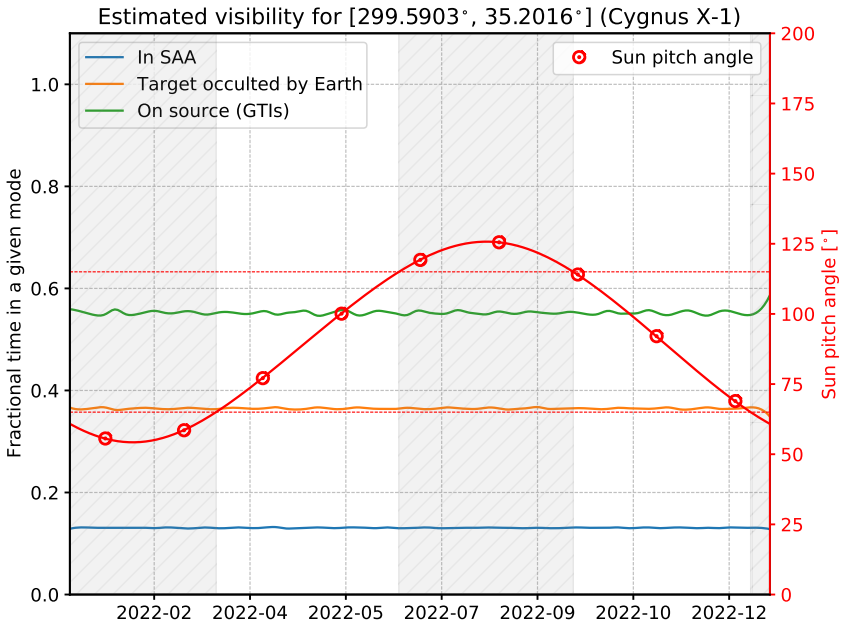


Fig. 2: Visibility for Cygnus X-1 in 2022.

Spectral Model

The spectral model for Cygnus X-1 in the hard state is taken from Parker at al., 2015 (Figure 6 and Table 4). In the IXPE energy range this includes three components:

- a comptonization component (eqpair);
- a reflection component (relxilllp);
- a narrow line at 6.4 keV.

Polarization Model

Giorgio suggested to use a 1–2% polarization degree for the main (eqpair) component, with an arbitrary angle, and a 2% polarization degree, oriented at 90 degrees, for the reflection component. The line should be unpolarized.

Also, he suggested not to include any temporal variability.

For completeness, here is an MDP table for the brightest source component.

```
xPointSource "eqpair" (id = 0)
    Galactic column density: 5.100e+21 cm^{-2}
    Redshift: 0.000
    Unabsorbed flux @ t = 0: 4.832e-09 erg/cm2/s (241.60 mcrab)
    Position: RA = 299.59031591 deg, Dec = 35.20160625 deg
MDP table for 300000.0 s observation time
2.00--2.83 keV:  4407218.0 counts, effective mu = 0.236, MDP = 0.87%
2.83--4.00 keV:  2484315.3 counts, effective mu = 0.380, MDP = 0.72%
4.00--5.66 keV:   891665.1 counts, effective mu = 0.473, MDP = 0.96%
5.66--8.00 keV:   231725.5 counts, effective mu = 0.564, MDP = 1.58%
2.00--8.00 keV:  8014923.9 counts, effective mu = 0.316, MDP = 0.48%
```

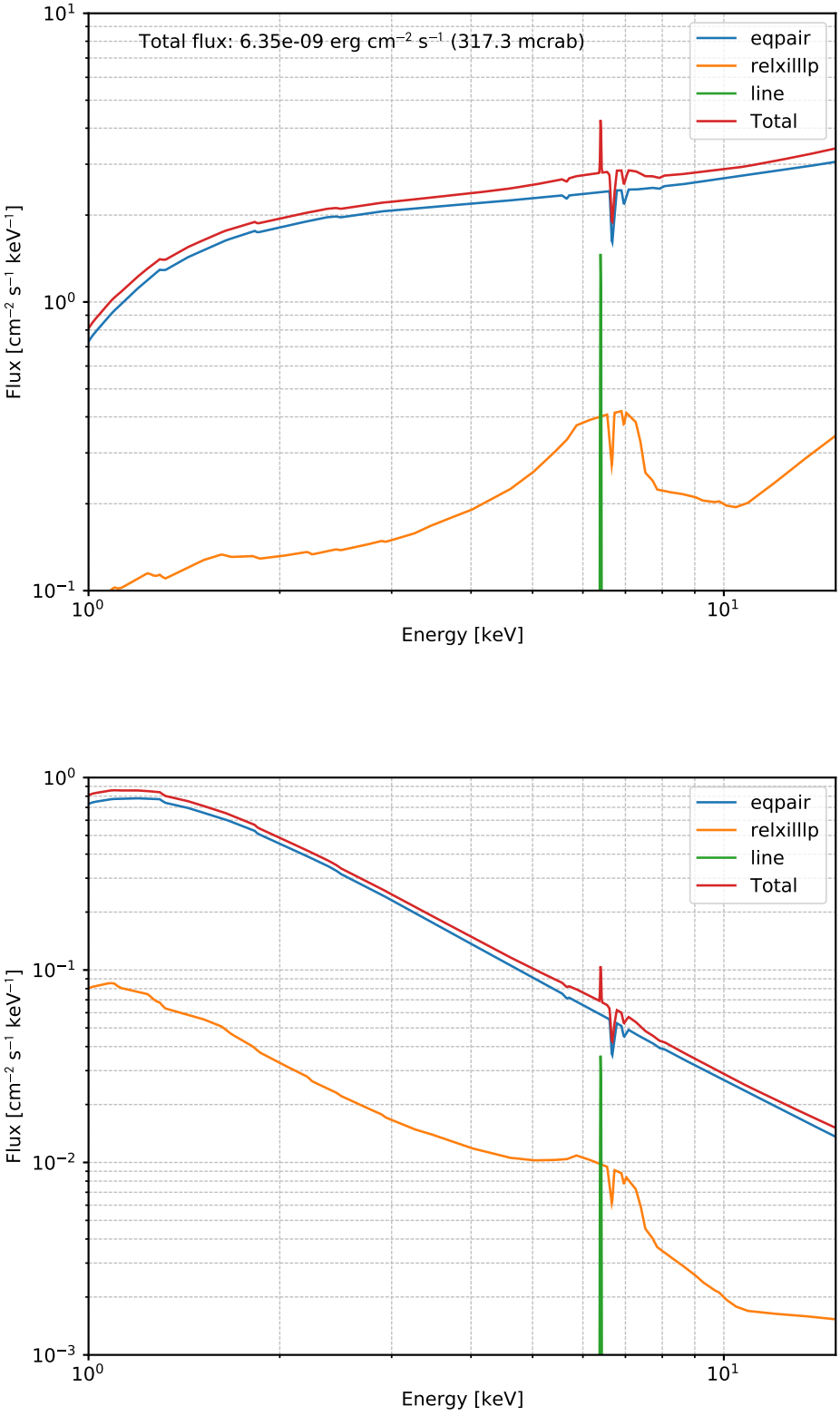


Fig. 3: Input spectrum.

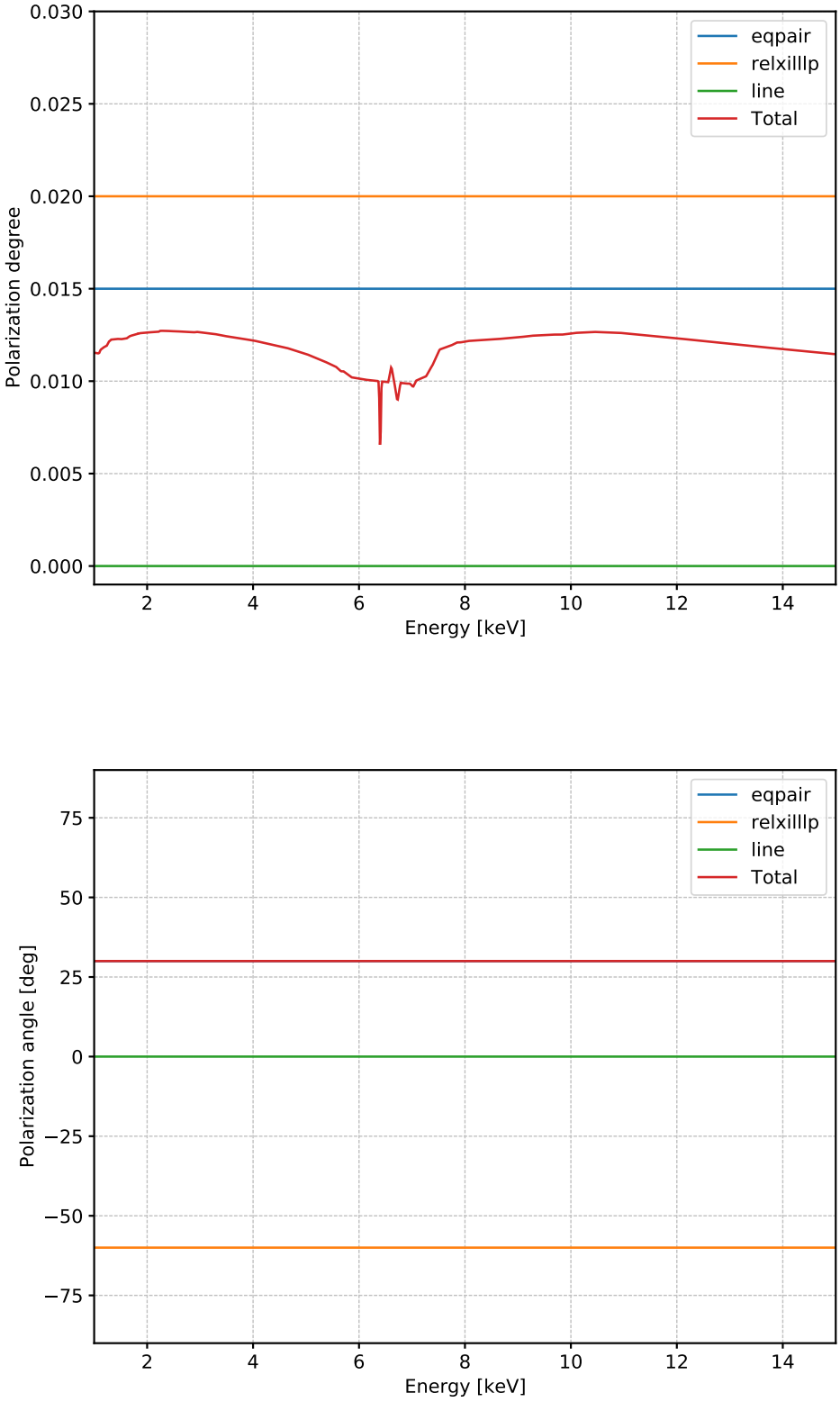


Fig. 4: Polarization model.

10.3.2 Markarian 421

- Source: Markarian 421
- Observation pattern: six ~50 ks observations spaced by ~two weeks
- Observation start:
  - 2022-04-23T00:00:00.000000 (obs_id 1)
  - 2022-05-07T00:00:00.000000 (obs_id 2)
  - 2022-05-21T00:00:00.000000 (obs_id 3)
  - 2022-05-28T00:00:00.000000 (obs_id 4)
  - 2022-06-04T00:00:00.000000 (obs_id 5)
  - 2022-06-19T00:00:00.000000 (obs_id 6)

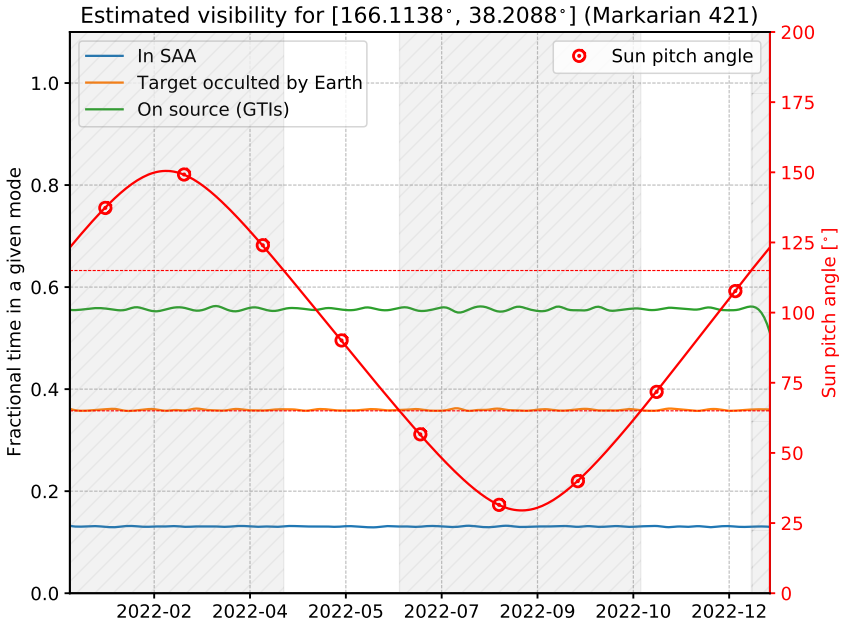


Fig. 5: Visibility for Markarian 421 in 2022.

The input model for Markarian 421 comes with two independent components that we shall refer to as *Faint* and *Bright*, respectively.

Spectral Model

The two spectral components are modeled as simple power laws with a normalization that varies in time and fixed spectral indices.

The *Faint* component features a moderate secular variation of the order of 10% of the average values, randomly generated with an interpolated cubic spline.

The *Bright* component has a negligible baseline, and a prominent peak (shaped like a gamma function) more or less at the center of the observation. Its spectral index is significantly harger than that of the *Faint* component.

Below is the integral flux in the IXPE band for the two components and for their sum.

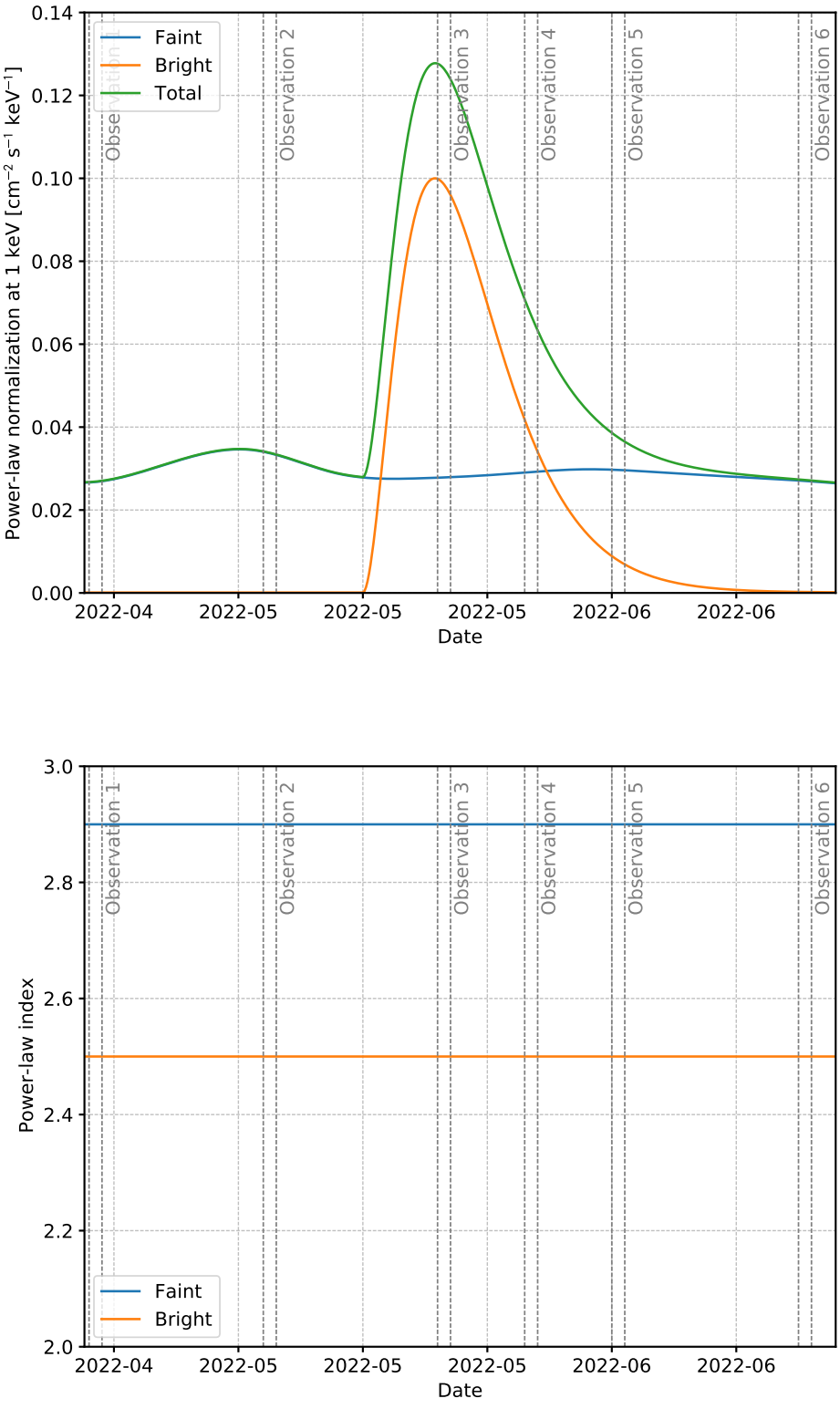


Fig. 6: Power-law normalization and index for the two model components as a function of time, for the target observation period. (The gray vertical lines indicate the six observations.)

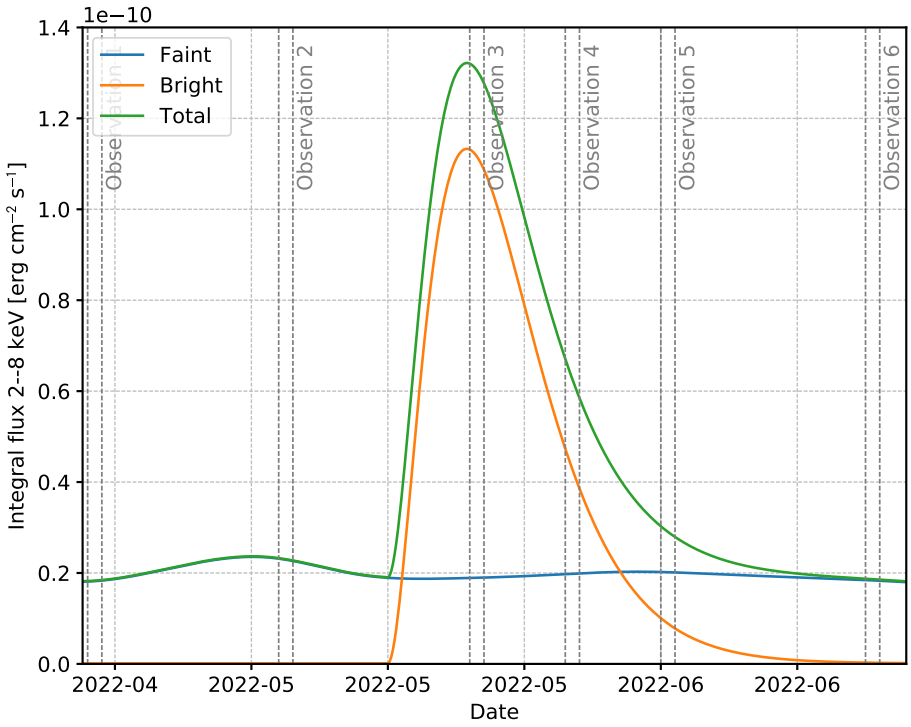


Fig. 7: Integral flux for the two model components as a function of time, for the target observation period. (The gray vertical lines indicate the six observations.)

Polarization Model

Both components feature a polarization degree increasing with energy over the IXPE band—roughly speaking the polarization degree doubles moving from 3 to 10 keV. The polarization vectors for the two components are orthogonal to each other.

Although neither polarization component has an explicit time dependence (both the polarization degree and polarization angle do not depend on time), the fact that they are orthogonal generates interesting effects as the corresponding spectra evolve with time. The specific profile of the *Bright* component and the actual observation intervals have been admittedly cherry picked to maximize the variety of combination we observe.

10.3.3 MSH 15-52

- Source: MSH 15-52
- Observation pattern: single 1.5 Ms observation
- Observation start: 2022-02-01T00:00:00.000000

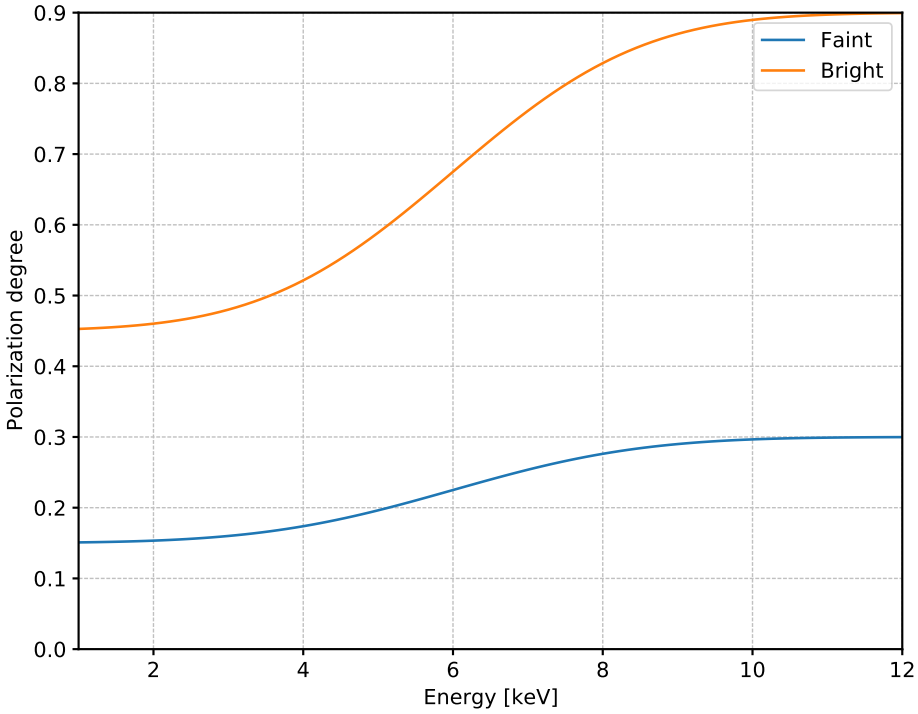
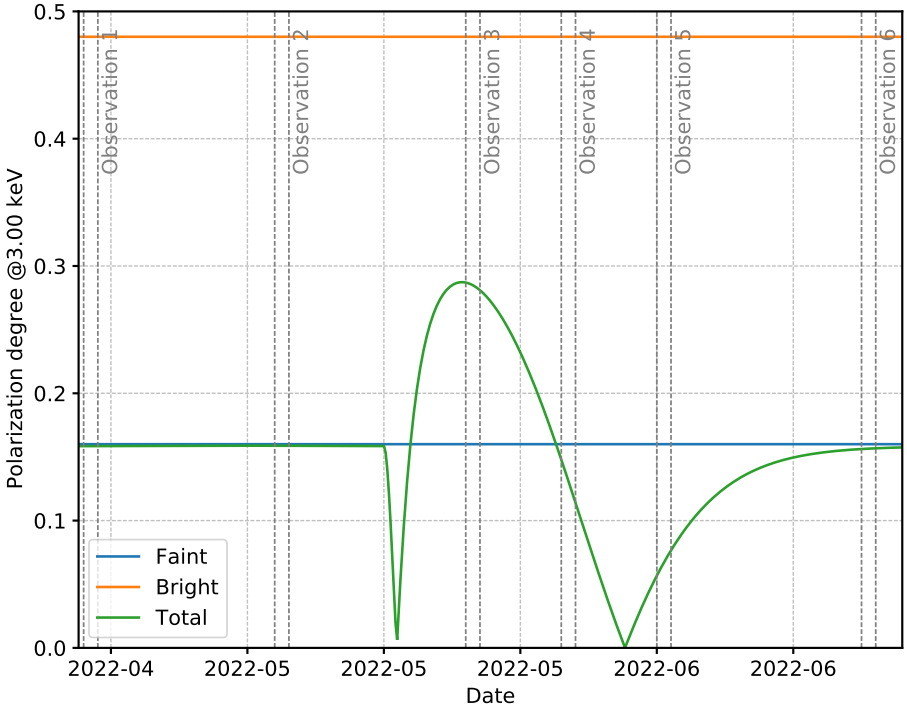


Fig. 8: Polarization degree as a function of the energy for the two coponents.



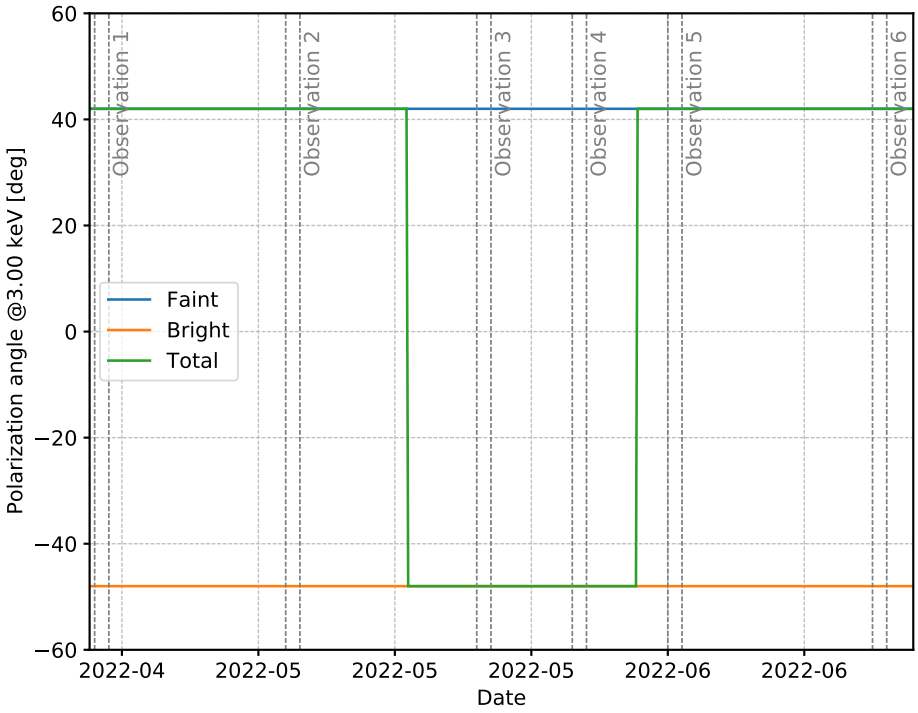


Fig. 9: Polarization degree and angle at a fixed energy (3 keV) as a function of time. (Note this is representative of the polarization at the peak of the sensitivity, but it is only a proxy of the broadband polarization we will measure in real life.)

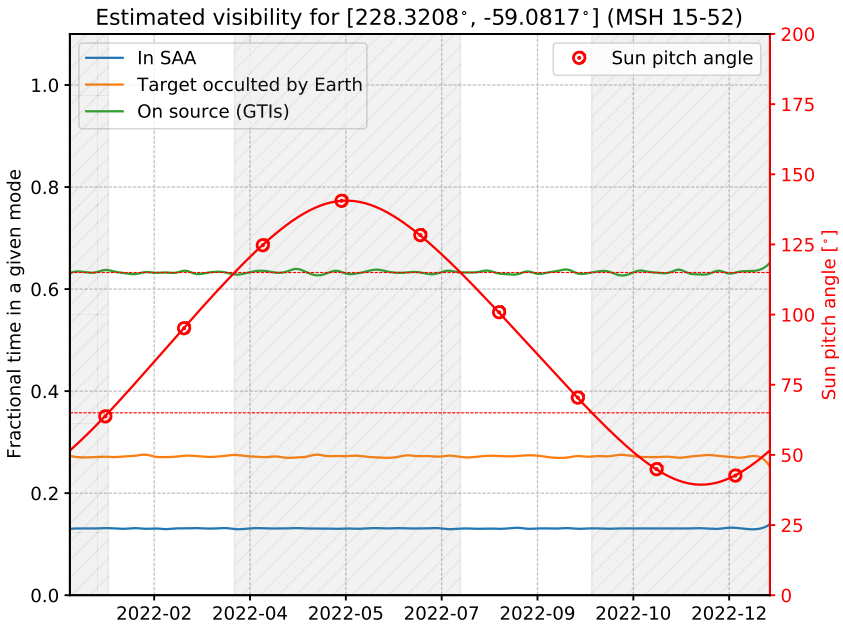
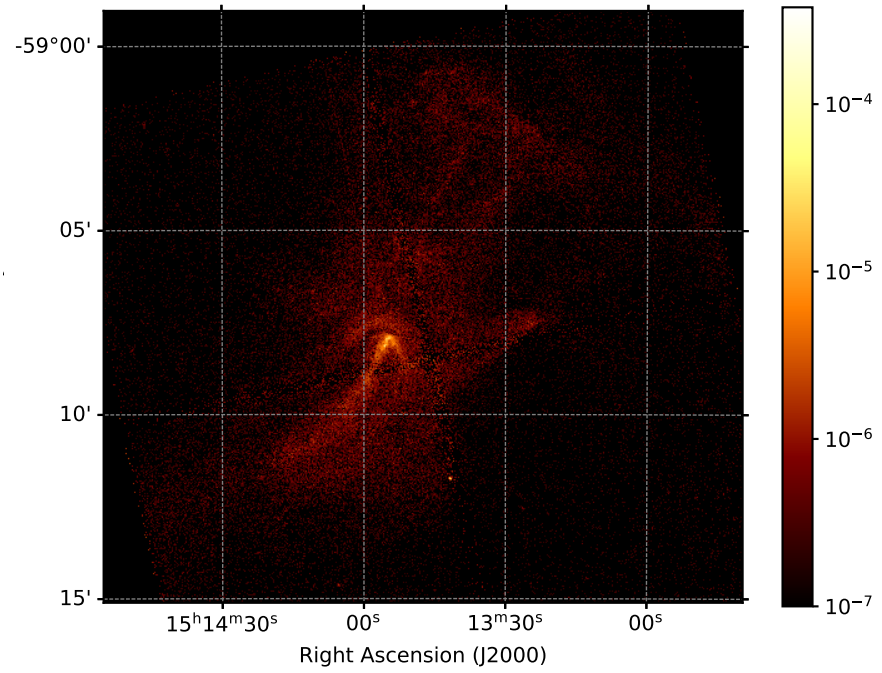


Fig. 10: Visibility for MSH 15-52 in 2022.

Morphological and Spectral Model

The basic morphological and spectral model for the nebula comes from a Chandra event list, that we feed into `ixpeobssim` to be folded with the IXPE response functions.



We cut out a small region at the position of the pulsar to avoid any problem with pile up in Chandra and plug in our own pulsar model.

Polarization Model

The baseline polarization model was created by Niccolo Bucciantini.

The PSR B1509-58 Pulsar

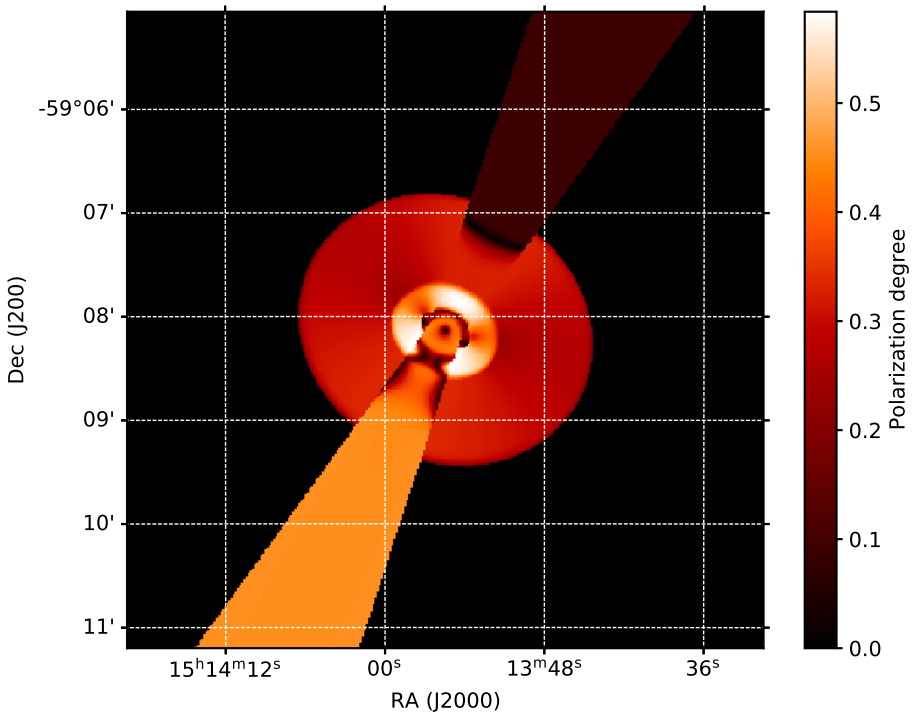
The pulsar is located at R.A. = 228.48133333, Dec. = -59.13577778. The basic input for the modeling comes from the NuSTAR observation Ge Chen at al., 2015, where a full phase-resolved spectral analysis is available.

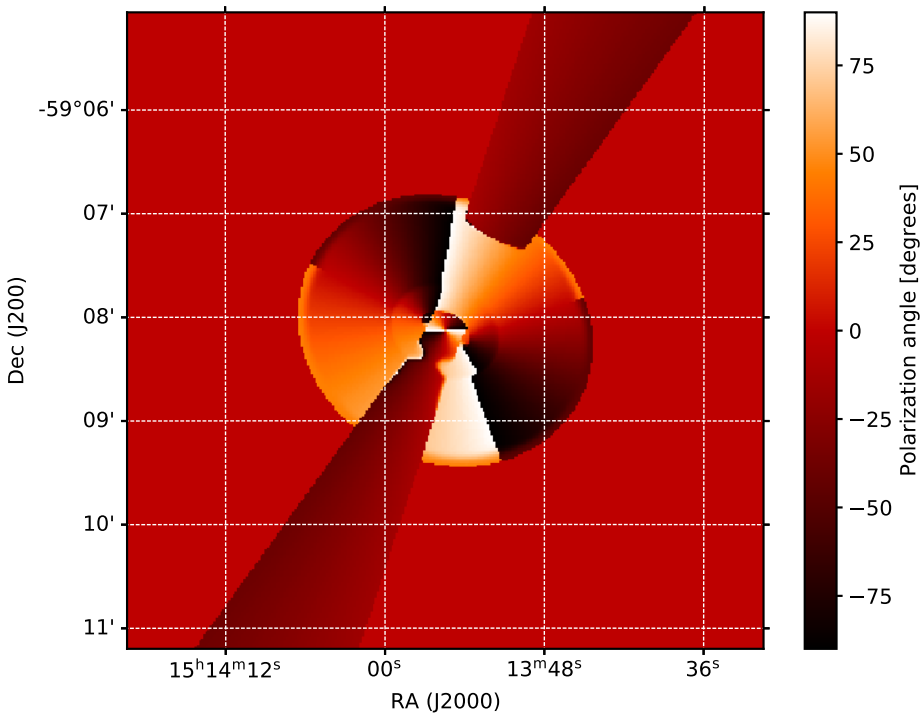
We use a simple power-law parametrization, with a purely phenomenological phase-dependence of the index, as illustrated in the figure.

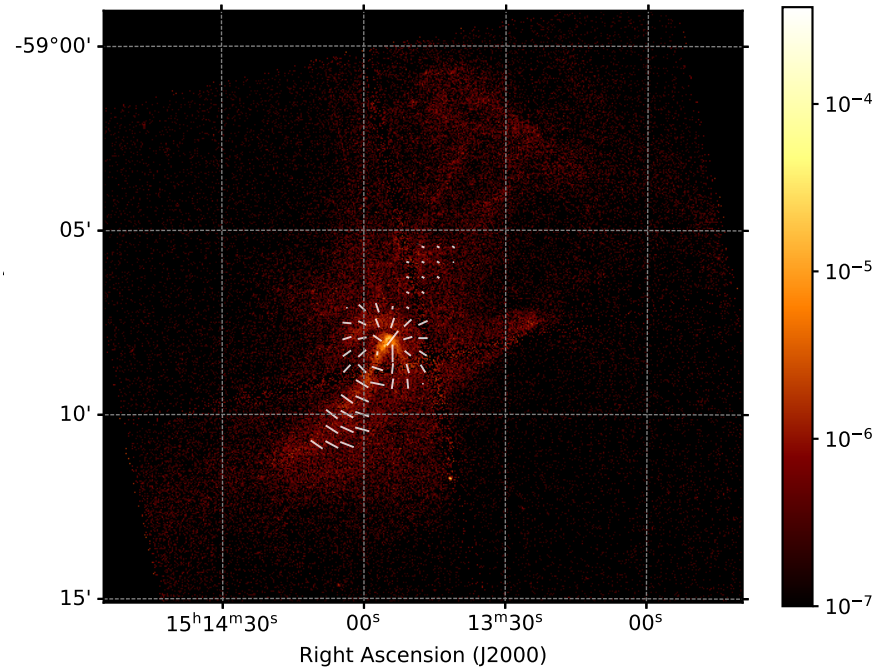
The ephemeris are taken from Livingstone and Kaspi, 2011:

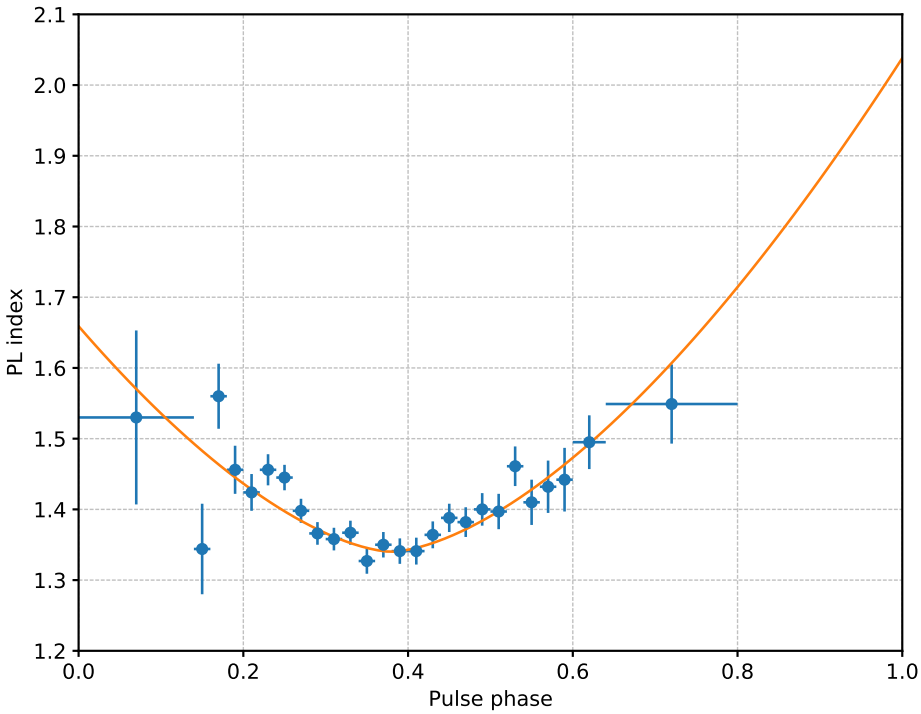
```
Dates (Modified Julian Day)  50148.096-55521.082
Epoch (Modified Julian Day)  52834.589000
nu                           6.611515243850 s-1
nudot0                       -6.694371307e-11 s-2
nuddot                       1.9185594e-21 s-3
```

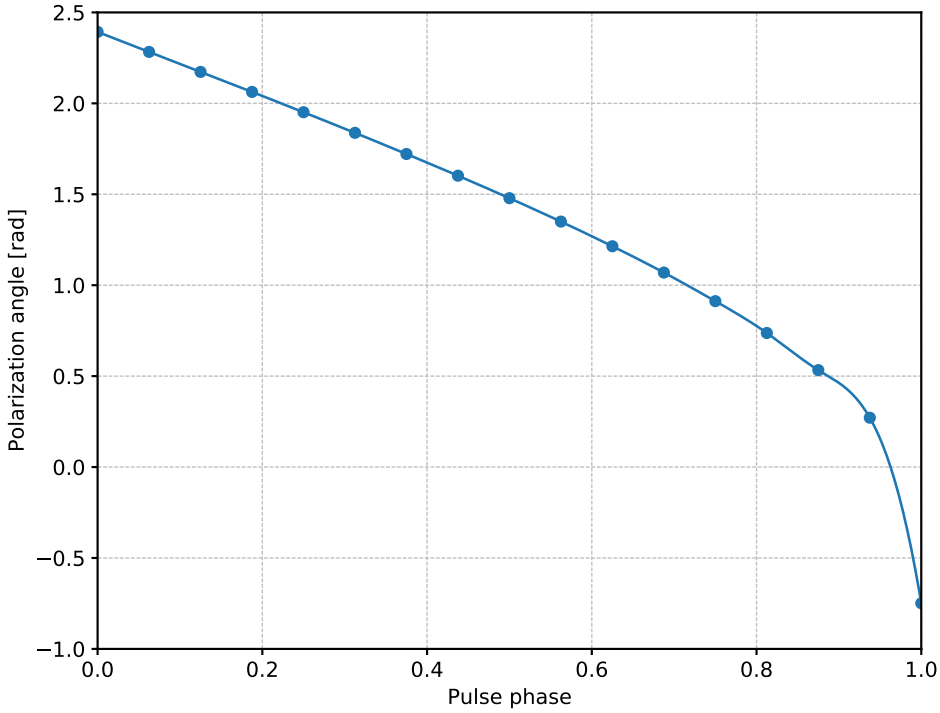
The polarization degree for the pulsar is taken as 35%, independently from the pulse phase, with a polarization degree vs. pulse-phase profile provided by Roger:


-59°06'
07'
08'
09'
10'
11'
Dec (J200)
15h14m12s
00s
13m48s
36s
RA (J2000)
0.5
0.4
0.3
0.2
0.1
0.0
Polarization degree


-59°06'
07'
08'
09'
10'
11'
Dec (J200)
15h14m12s
00s
13m48s
36s
RA (J2000)
75
50
25
0
−25
−50
−75
Polarization angle [degrees]


-59°00'
05'
10'
15'
15h14m30s
00s
13m30s
00s
Right Ascension (J2000)
10−4
10−5
10−6
10−7


PL index
2.1
2.0
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
0.0
0.2
0.4
0.6
0.8
1.0
Pulse phase



10.3.4 RX J1708-4008

- Source: RX J1708-4008
- Observation pattern: single 1 Ms observation
- Observation start: 2022-09-01T00:00:00.000000

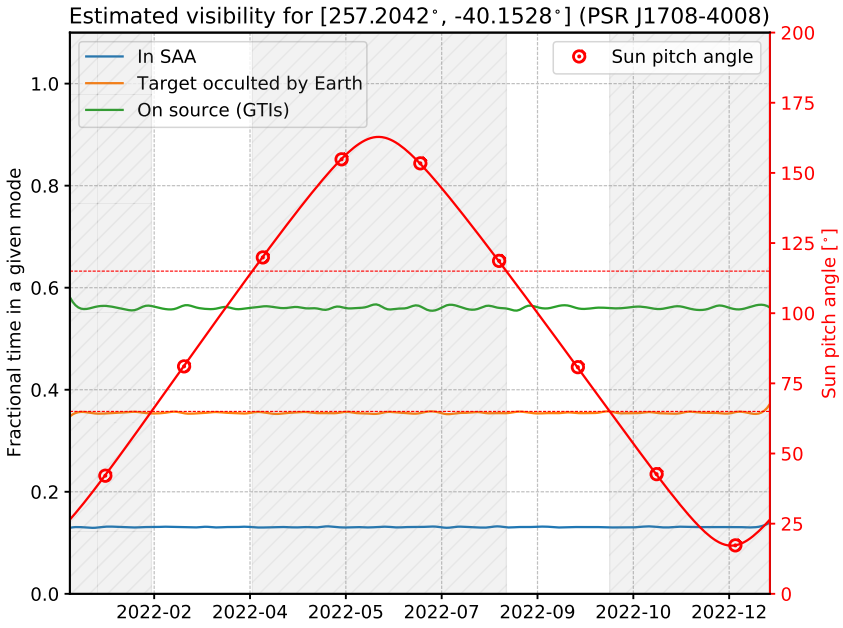


Fig. 11: Visibility for PSR J1708-4008 in 2022.

CHAPTER

ELEVEN

LARGE DATA FILES

When setting up simulation models or analyzing data it is sometimes the case that large data files (say larger than 10 MB) are necessary, XPSEC table models and Chandra event lists to be fed into ixpeobssim being typical examples.

Warning

Please refrain from pushing large data files to the ixpeobssim repository—a code repository is for code, not for data! Even setting aside the fact that data files are essentially impossible to diff (i.e., if and when you push a new version of a file you are effectively pushing the entire thing over and over again, not just the difference with respect to the previous version), if you push 1 GB of FITS files to the repository, you are effectively forcing everybody else to download 1 GB of data at the next pull, and you are increasing the dimensions of the bare repository in a non-reversible way.

As a rule of thumb, try and avoid pushing files larger than a few MB, unless there is a compelling reason for doing so. (Really, once you have pushed a file there's essentially no way to get rid of it in git.)

If you have large auxiliary files that are potentially useful for other collaborators there is a separate repository that we (ab)use specifically for this purpose, so that only the subset of users that care about a specific set of files needs to worry about. You will notice that we don't actually *push* files on that repository—we attach files in the associated download page, instead. You should refer to the README file on the ixpeobssim_auxfile repository for minimal instructions about the usage of that facility.

On the `ixpeobssim` side, as of version 14.0.0, we have a mechanism in place to direct the user to this repository when a functionality that needs one or more auxiliary file(s) is used for the first time. Typically all you have to do is download the necessary file(s) into your `$IXPEOBSSIM_AUXFILES` directory (defaulting to `~/ixpeobssimauxfiles`), and there are facilities to provide instructions to the user to do so. To this end, the *`ixpeobssim.srcmodel.magnetar.xMagnetarModelsT2020`* interface to the magnetar table models described in Taverna et al. 2020 is the original reason for setting up the large-file infrastructure, and a good source of inspiration for the associated usage.

In a nutshell, `ixpeobssim` provides three convenience functions to handle auxiliary (large) files, and you can protect the sensitive part of your configuration or analysis code by a simple call along the lines of

```
from ixpeobsim.utils.logging_ import abort
from ixpeobssim import auxfile_missing

if auxfiles_missing(*file_list):
    abort()
```

This should interrupt the execution of the program and print on the terminal enough information for the user to be able to download the right file(s) in the right location.

CHAPTER

TWELVE

RESPONSE FUNCTIONS

The instrument response functions (IRFs) are a critical part of ixpeobssim, and they are used (in identical form) both in the event simulation and in the analysis of the data products from the simulation itself.

All the response functions are stored in FITS files in the OGIP format defined in CAL/GEN/92-002 (and modifications CAL/GEN/92-002a) and are intended to be fully compatible with the spectral analysis tools provided by XSPEC (see this page for more details).

We identify six different types of response functions:

- effective area (.arf), see *ixpeobssim.irf.arf.xEffectiveArea*;
- vignetting, see *ixpeobssim.irf.vign.xVignetting*;
- energy dispersion (.rmf), see *ixpeobssim.irf.rmf.xEnergyDispersion*;
- point-spread function, see *ixpeobssim.irf.psf.xPointSpreadFunction*;
- modulation factor, see *ixpeobssim.irf.modf.xModulationFactor*;
- modulation response function, see *ixpeobssim.irf.mrf.xModulationResponse*.

If you are familiar with basic spectral analysis in XSPEC, the .arf and .rmf files have exactly the meaning that you would expect, and can be in fact used in XSPEC; the .mrf files have the exact same format as the .arf files.

ixpeobssim provides facilities for reading, displaying and using IRFs, as illustrated below.

12.1 IRF summary

This is a short, top level description of the version v13 of the default IXPE response functions used by ixpeobssim. As explained down in this page, these maps one-to-one with the response files shipped with the official IXPE CALDB released to the public with HEASoft version 6.33.

Warning

Added in version 31.0.0: As of version 31.0.0, ixpeobssim comes with a default set of response files (dubbed v13) whose validity time is binned in 6-month interval. The beginning of each interval is encoded in the file name, in a way that should be human-readable.

Contrary to the HEASOFT framework, ixpeobssim has no internal mechanism to automatically pick the right set of response files for the analysis of a given observation, and it is a responsibility of the user to make sure that the start of the observation is included within the validity interval of the files themselves. (The latter is properly encoded with each file in the form of the canonical header keywords.)

All the response functions are defined between 1 and 12 keV in steps of 40 eV, and are detector-unit based (the pulse-invariant channel space is defined between 0 and 15 keV in steps of 40 eV). As such, there are differences between the three modules in terms of effective area, modulation factor and point-spread function. Since the eight iteration of the response functions, all response files come in two distinct flavors—un-weighted and weighted. Accordingly, all the analysis tools support weights.

12.1.1 Effective area

Below is the nominal IXPE on-axis effective area, in both the un-weighted and weighted flavors. The latter is roughly 15% smaller at the peak value but, as we shall see in a second, the increase in the modulation factor drives the overall sensitivity slightly up.

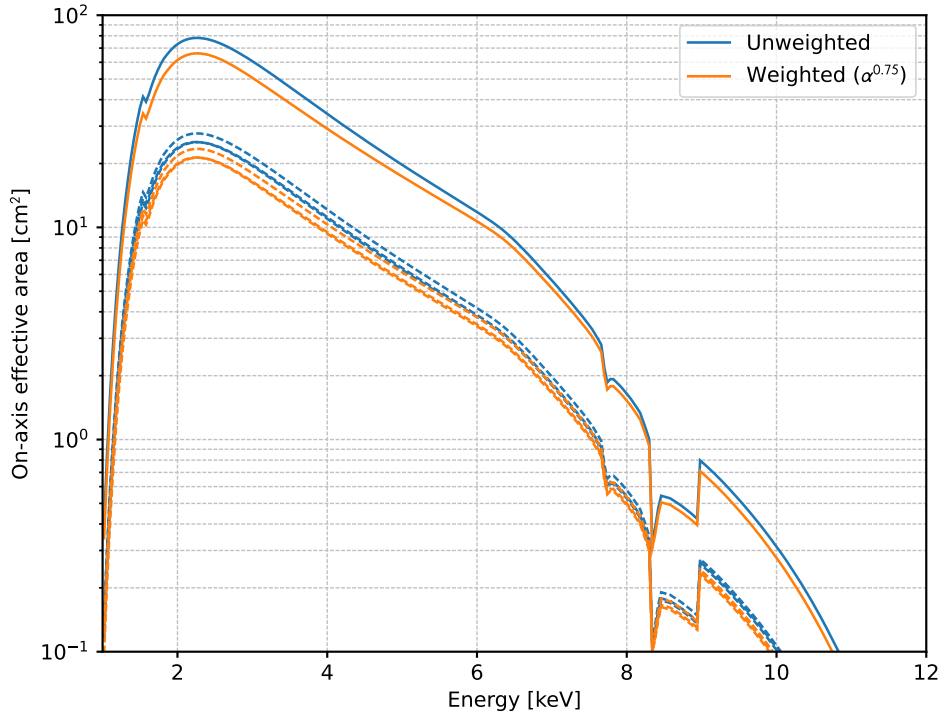


Fig. 1: On-axis effective area as a function of the energy. The solid lines represent the total effective area for the sum of three telescopes, in the un-weighted and weighted version, while the un-labeled dashed lines represent the curve for each of the three single telescopes.

The effective-area curves for the three telescopes are within a few % from each other, the small differences being due to the slightly different mirror effective areas measured during the MMA calibration, as well as the different asymptotic pressure values for the three GPDs at the focal plane.

The effective-area calculation in ixpeobssim includes all the relevant contributions, namely:

- the mirror effective area;
- the transparency of mirror-module-assembly thermal shield;
- the transparency of the detector-unit UV filter;
- the transparency of the GPD window;
- the efficiency of the GPD gaseous active medium;

- the efficiency of the event weighting (in the weighted flavor).

The plots below show the principal ingredients that go into the calculation.

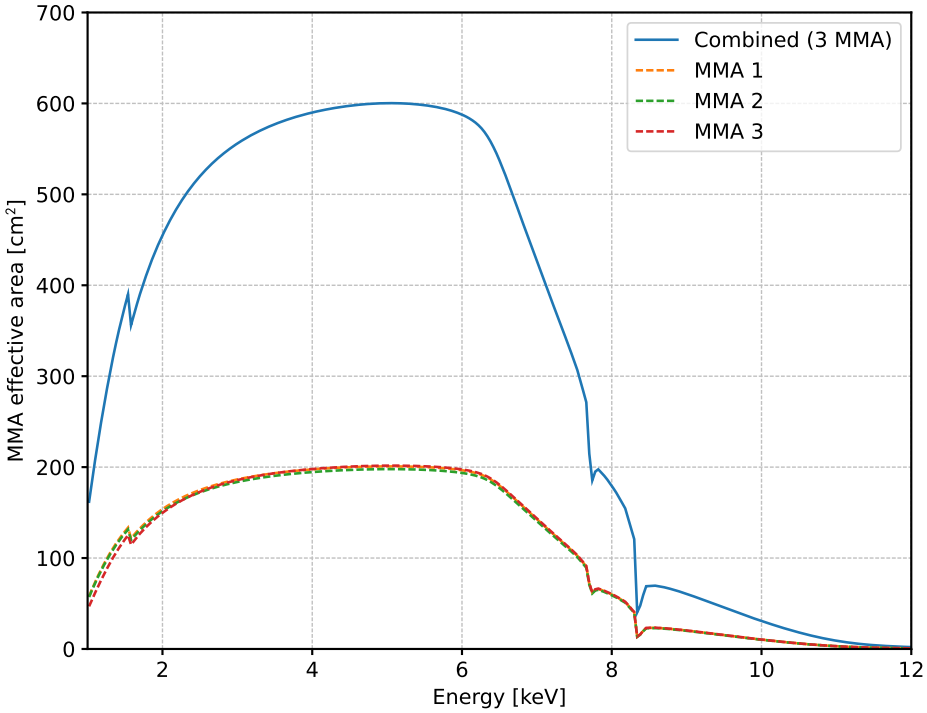


Fig. 2: On-axis effective area as a function of the energy for three Mirror-Module Assemblies (MMA) and for a single module.

The vignetting function shown below comes from a preliminary study by MSFC based upon ray-trace simulations for a perfect mirror module assembly, and is relevant for the simulation of extended sources.

The vignetting, along with the relative orientation of the three IXPE detector units, defines the relative exposure across the field of view of the instrument, as shown in the following two plots. (Note that above 6 keV the drop of the effective area at the edge of the field of view is relatively more important.)

Of course, unless you specifically decide to disable this functionality, ixpeobssim handles all of this behind the scene, so you don't have to worry about it—but keep it in mind when you do back-of-the-envelope calculations.

12.1.2 Energy dispersion

The energy dispersion (a.k.a. the response matrix) comes from a series of line Monte Carlo simulations performed with the IXPE GPD Geant 4 simulation framework. Below is a color representation of the energy dispersion as a function of the energy, which is essentially the content of the binary table in the `MATRIX` extension of the rmf file.

For illustration purposes, here are the corresponding one-dimensional pdfs at a few fixed true energies (i.e., these are just vertical slices of the color plot above).

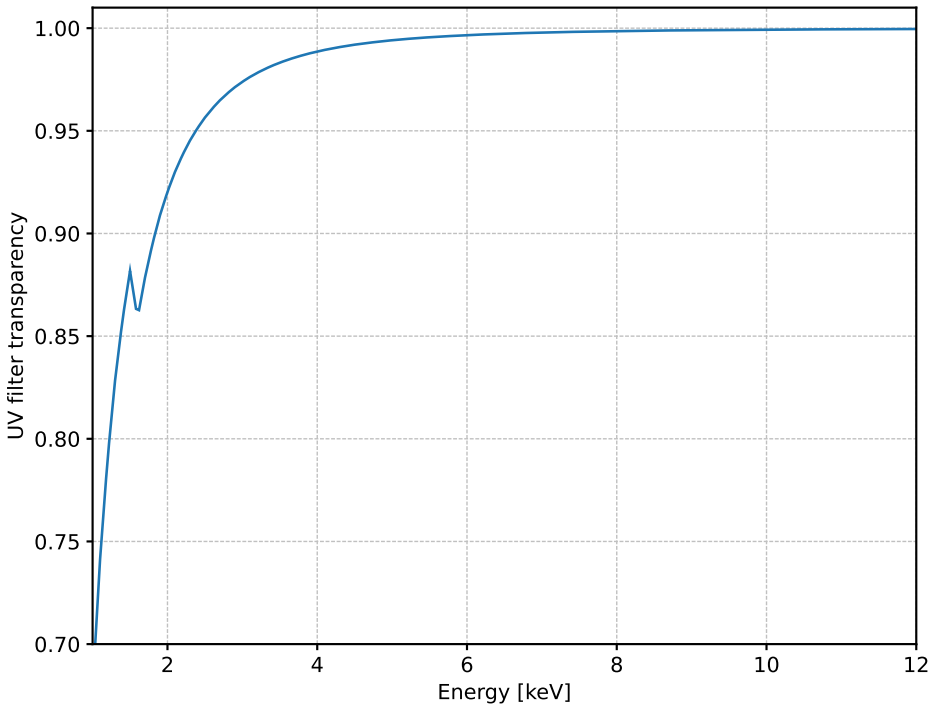


Fig. 3: Transparency of the UV filter as a function of the photon energy.

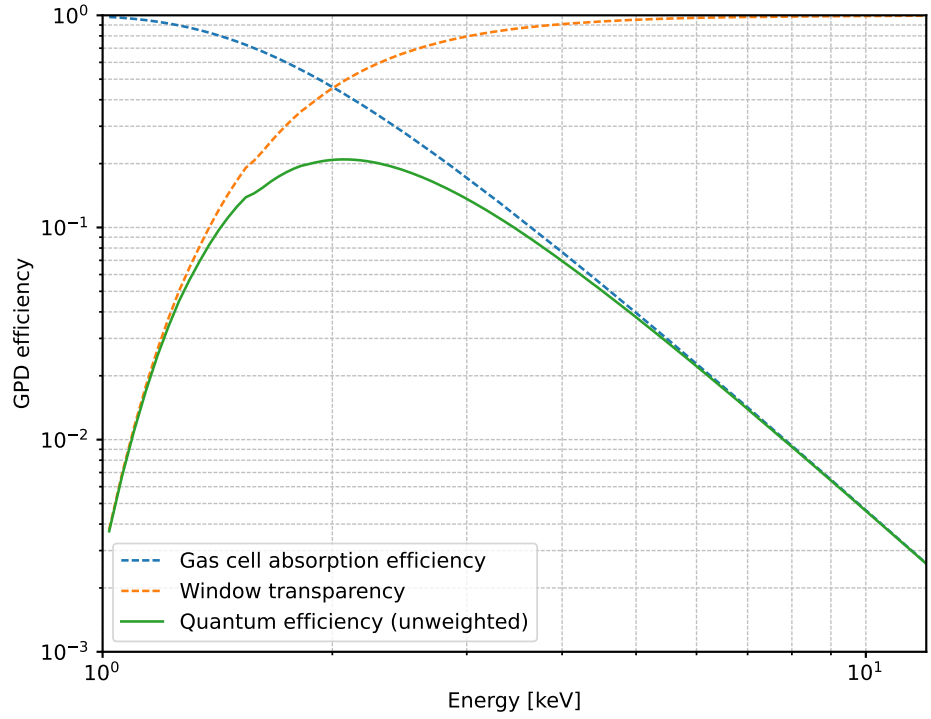


Fig. 4: Quantum efficiency of the GPD as a function of the energy, broken up in its two main components—the Be window transparency, and the gas cell absorbing efficiency.

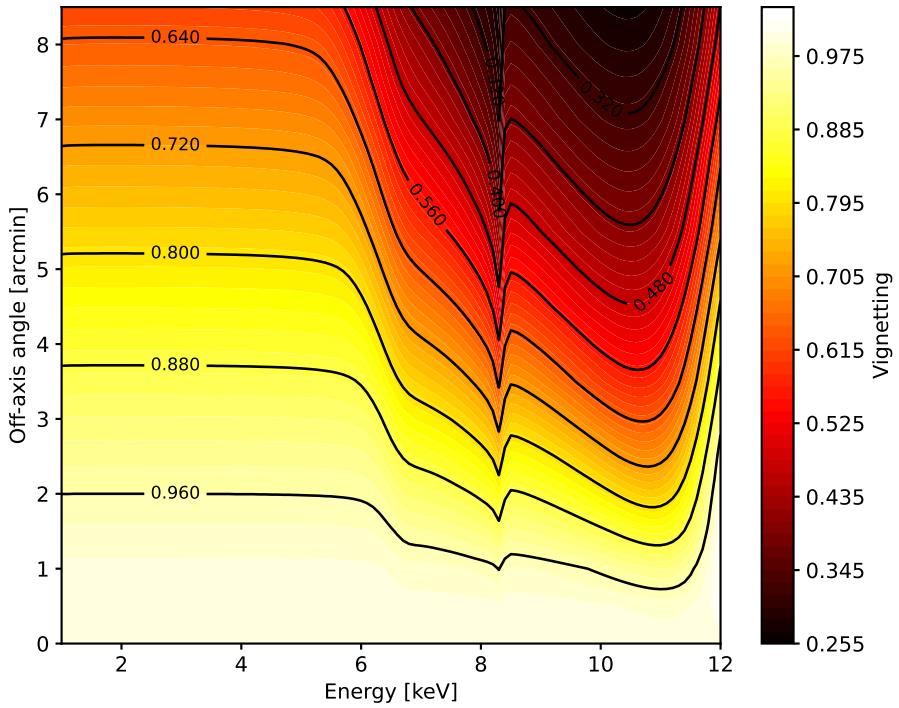


Fig. 5: Preliminary estimation of the vignetting of the optics as a function of energy and off-axis angle.

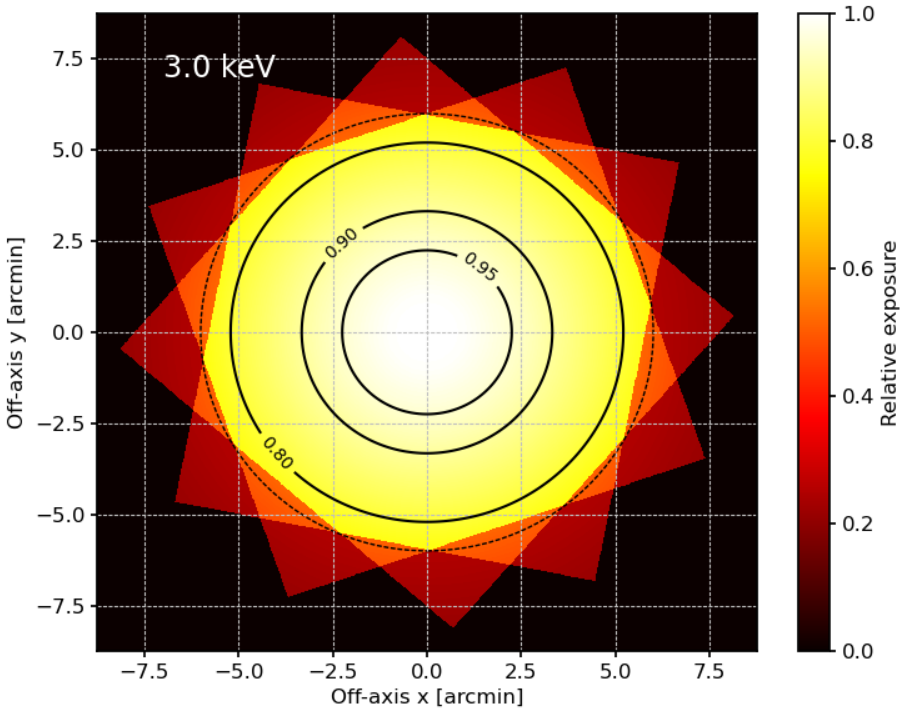


Fig. 6: Relative exposure at 3 keV across the field of view for the set of three telescopes clocked in the IXPE configuration.

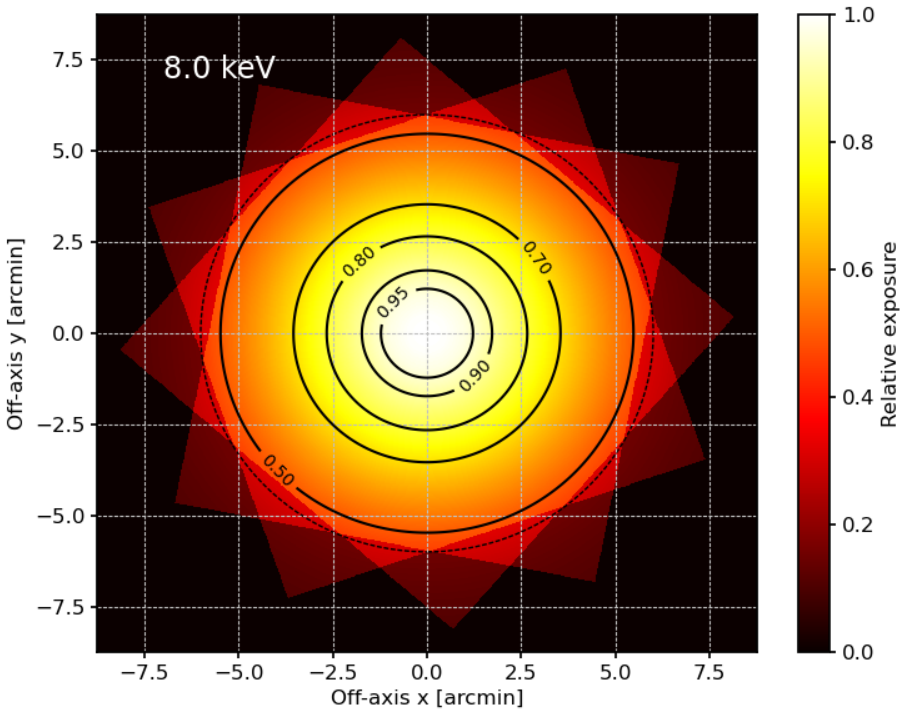


Fig. 7: Relative exposure at 8 keV across the field of view for the set of three telescopes clocked in the IXPE configuration.

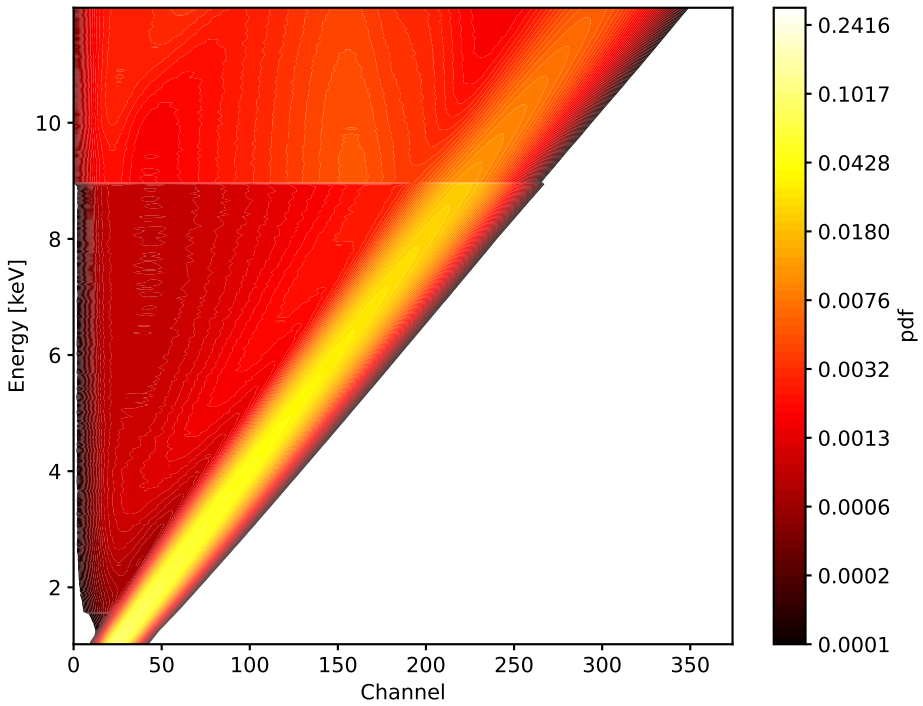


Fig. 8: Representation of the GPD response matrix.

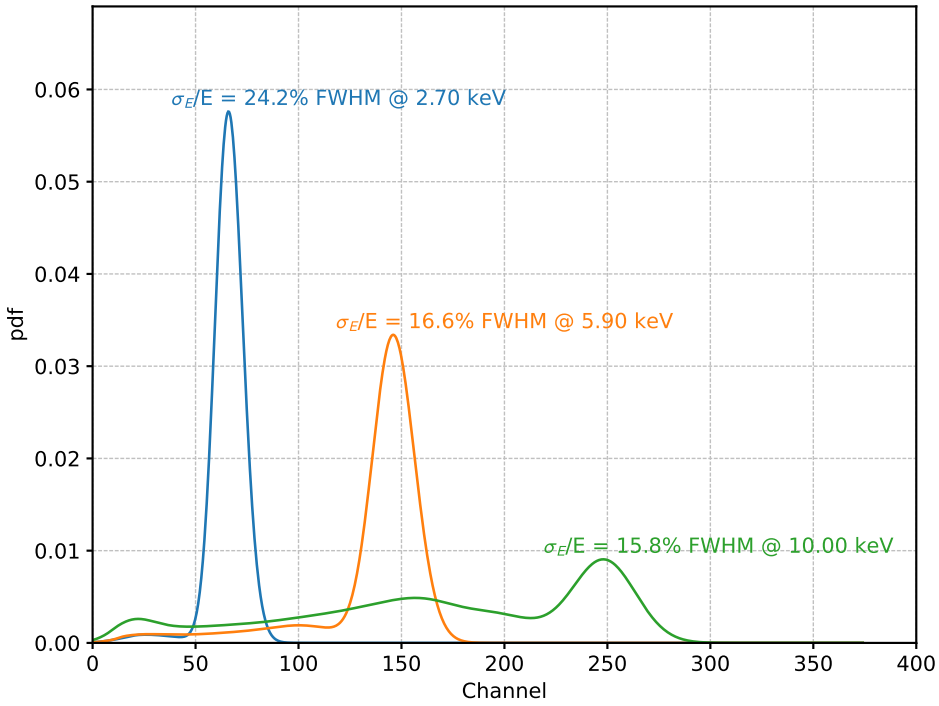


Fig. 9: Energy dispersion (one-dimensional probability density function) at a set of discrete energies. The FWHM energy resolution is indicated for completeness.

12.1.3 Point-spread function

The PSF model is derived from one of the early point-source observations, as described in issue #158. Starting with version 6 of the instrument response function each DU comes with a different PSF scaling factor to account for the differences measured during the mirror calibration. As shown in issue #387, MMA 1 has a significantly better PSF (less than 20 arcsec HPD) than MMAs 2 and 3 (running at more than 25 arcsec HPD).

For completeness, the current set of response functions do not include the mirror aberration, which is nonetheless much smaller than the PSF half-power diameter across the entire field of view and is therefore, to first order, negligible.

12.1.4 Modulation factor

Our parametrization of the modulation factor comes from a series of line-type Monte Carlo simulations, informed by the ground calibrations of the three detector units.

Note

The noticeable edge around 9 keV is due to the K-edge of the copper, above which the extraction of photoelectrons from X-rays absorbed in the GEM becomes significantly more likely. This causes an increase of effective area, accompanied by a dilution of the modulation. While we provide a tabulation of all the IRFs in the standard grid between 1 and 12 keV, significant more work is needed to validate the response of the detector above the Cu K-edge, and simulations outside the 2–8 keV standard range should be interpreted with caution.

Below is an alternative representation of the overall IXPE spectro-polarimetric response, combining the elements de-

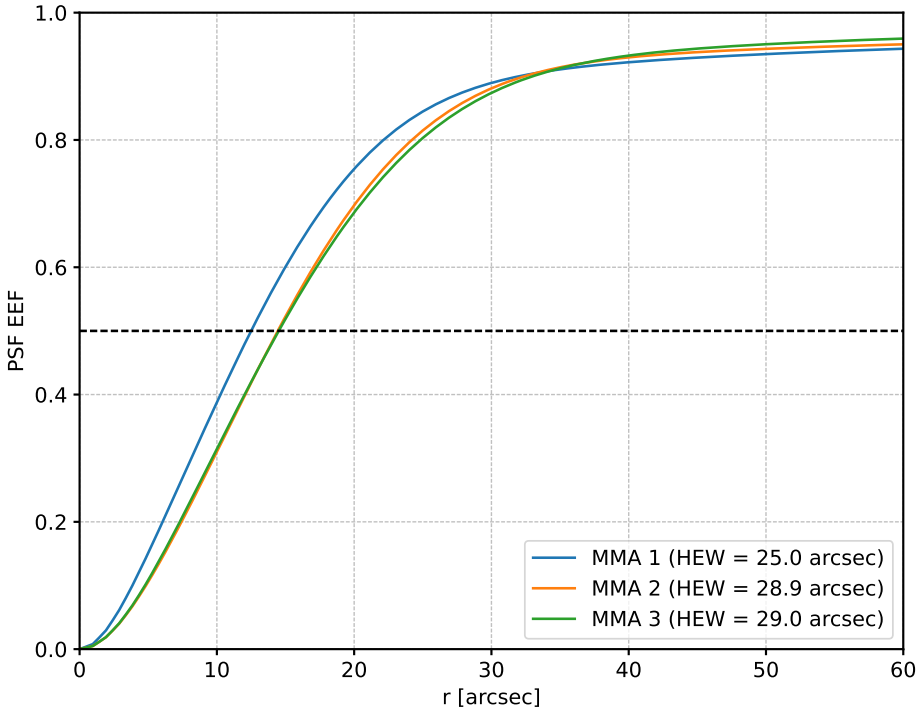


Fig. 10: Encircled energy fraction (EEF) for the PSF of the three IXPE telescopes.

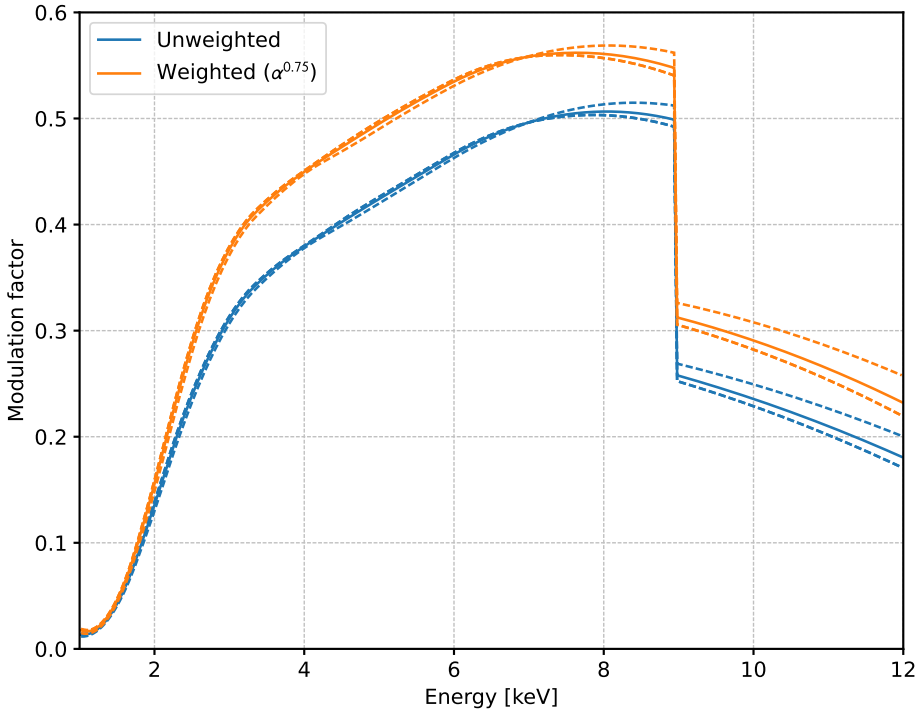


Fig. 11: Modulation factor as a function of the photon energy for the IXPE detectors. The solid line represents the average for the three GPD, in the un-weighted and weighted version, while the un-labeled dashed lines (admittedly, barely visible) represent the curve for each of the three detectors.

scribed above.

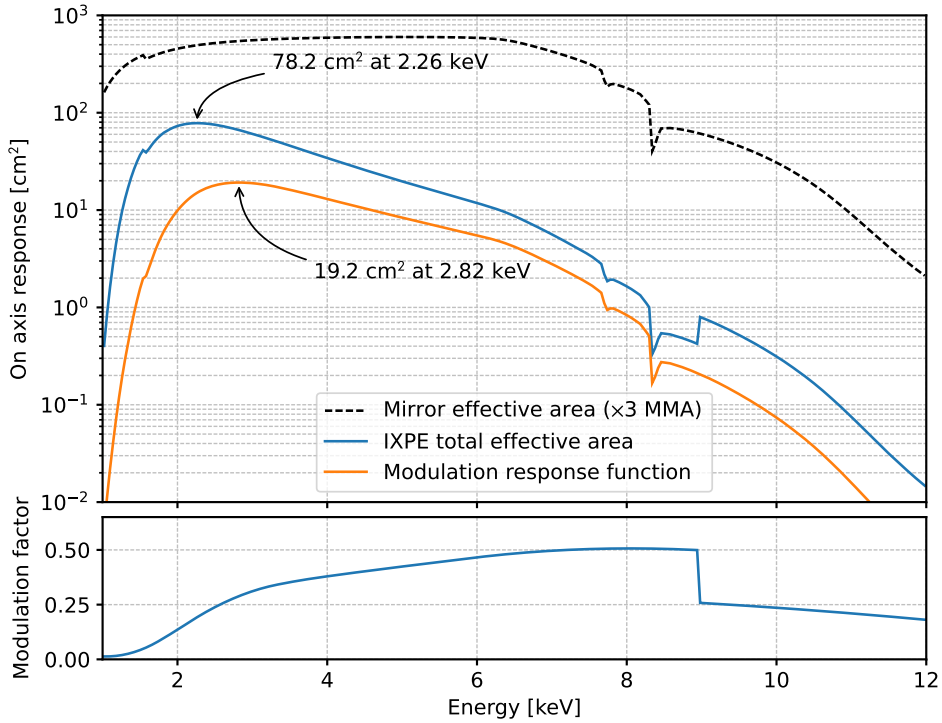


Fig. 12: Overall IXPE spectro-polarimetric response (all the elements are described in the first part of this section).

12.2 Minimum detectable polarization

The effective area curve (for the sum of three mirror modules) and the modulation factor are enough for a crude estimation of the minimum detectable polarization for a point source, and for reference we produce the basic performance plot below for each iteration of the response functions using *xppimms* for definite sets of spectral indices and rescaling for the source flux and the observing time.

12.3 Reading and visualizing IRFs

In a nutshell, the recommended way to load the default set of response functions (whatever that means at any point in time) is

```
from ixpeobssim.irf import load_irf_set

# Load all the default response functions.
irf_set = load_irf_set(du_id=1)

# Access the actual response functions.
aeff = irf_set.aeff
vign = aeff.vignetting
edisp = irf_set.edisp
```

(continues on next page)

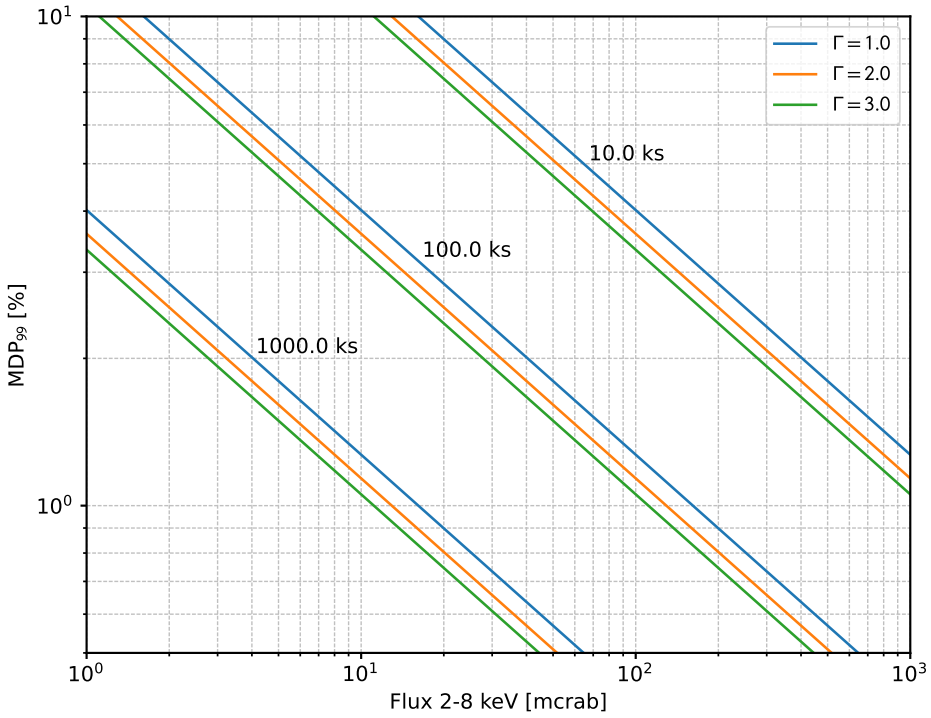


Fig. 13: IXPE Minimum Detectable Polarization (MDP) as a function of the source flux for several different exposure times and spectral indices.

(continued from previous page)

```
psf = irf_set.psf
modf = irf.modf

# Print the effective area and modulation factor at 5 keV.
print(aeff(5.))
print(modf(5.))
```

The reader is referred to the documentation and the source code of the relevant classes for a full description of the interfaces that ixpeobssim provides.

For completeness, ixpeobssim makes available *xpirfview.py* as a single visualization interface to all the response file. Just type

```
xpirfview.py path/to/the/response/file.fits
```

and you should get back some sensible visualization of the thing.

12.4 Pseudo-CALDB

For convenience, at this point in time, `ixpeobssim` is effectively implementing its own, self-contained CALDB—that we sometimes refer to as the `ixpeobssim` *pseudo-CALDB*. The plans for interfacing `ixpeobssim` with the actual IXPE CALDB are not yet defined as there are definitely peculiarities on both sides (simulation and real data) that make having a drop-in replacement structure less than trivial.

The actual FITS files with the response data live (provisionally) in the ixpeobssim/caldb folder and the basic logic determining the naming and the file location is defined here.

Note

Added in version 21.0.0.

Starting from version 21.0.0 the structure of the pseudo-CALDB has been drastically changed to match as closely as possible that of the actual CALDB submitted to HEASARC.

The `ixpeobssim` internal rules for the IRF-name designation have also been modified changing the delimiter, in order to have a better match with the CALDB file names and provide support for weights in a more straightforward fashion.

`ixpeobssim` will, at least provisionally, maintain a separate version numbering with respect to the official CALDB.

The convention we use to name response file is `[base][unit][calibtype][intent][ver]`, where:

- [base] is the base name for the set of response functions, e.g., *ixpe_*;
- [unit] indicates the telescope unit (*du1*, *du2*, *du3*).
- [calibtype] provides an identifier for the calibration data (e.g., *vign* or *psf*), with the exception of the *arf*, *rmf* and *mrf* files, where the data type is indicated by the file extension;
- [intent] is the intent of a particular set of response functions, e.g., *_obssim_*;
- [version] is the CALDB version number for any given file.

Additionally, each coherent set of response functions is identified within ixpeobssim by a name, in the form `[base]:[intent]:[ver]`. ixpeobssim is able to parse a string formed according to this rule and resolve all the relevant paths to the actual response files.

Warning

The pseudo-CALDB contains a number of response files that are not shipped with the real CALDB, including pre-launch estimates that we retain for bookkeeping purposes, but should not be used to analyze flight data.

In a nutshell: all the set of response files named as `*_legacy_*` should be considered of solely historical interest and should never be used in conjunction with flight data samples.

12.4.1 Response file versioning

This is a short description of the main features of different sets of response files that are useful for simulation and science analysis:

- `ixpe:obssim*:v13`: compared to the previous iteration (`ixpe:obssim:v12`), this *collection* of response files includes an updated parametrization of the MMA effective area at high energy (that should improve the spectral residual and the spectral fitting in general), as well as a better estimate of the pressure inside the gas pixel detectors.
- `ixpe:obssim:v12`: compared to the previous iteration (`ixpe:obssim:v11`), this version includes an updated parametrization of the effective area for MMA 3, accounting for its thicker thermal shield (the effect is of a few % at 2 keV, and negligible above 4 keV, but has been shown to improve the spectral residuals at low energy for bright sources), as well as a new set of weighted response files with the SIMPLE prescription, that are necessary for a proper weighted model-independent analysis.
- `ixpe:obssim:v11`: identical to `ixpe:obssim:v10`, except that the PSF parametrization (used on the simulation side of things) has been improved to match the on-orbit radial dependence measured with point sources; all the other response files are unchenged.
- `ixpe:obssim:v10`: this is the first iteration of the response files matching the structure of the actual CALDB, and the first that can be used with real data.

12.4.2 Mapping to the real CALDB

Although the pseudo-CALDB and the real-CALDB are fundamentally different in some key aspects, most of the relevant files (e.g., those containing the effective area, the response matrix and the modulation response function) have a definite, one-to-one correspondence between the two databases—meaning that they are *identical*, modulo a few header keywords.

- `ixpe:obssim*:v13` maps to equivalent files in the CALDB with the same starting date.
- `ixpe:obssim:v12` maps to `20170101_02`
- `ixpe:obssim:v11` maps to `20170101_01`
- `ixpe:obssim:v10` maps to `20170101_01`

The structure of the `ixpeobssim` pseudo-CALDB is designed to match as closely as possible that of the real IXPE CALDB.

```
ixpe
|----gpd
    |----bcf
         |----chrgparams
    |---- cpf
         |----arf
         |----modfact
         |----mrf
         |----rmf
|----mma
    |----bcf
        |----psf
        |----vign
```

The naming conventions for the FITS files have been aligned to the IXPE CALDB starting from version 10, and the file names for the previous iterations have been changed accordingly.

There are two subtle but noticeable differences between the IXPE CALDB and the `ixpeobssim` pseudo-CALDB, namely:

- the pseudo-CALDB has no concept of validity date, and that is reflected both in the file names and in the corresponding header keywords;
- the versioning scheme has a different meaning in the two contexts, and version numbers are physically assigned by different people at different times (more specifically, `ixpeobssim` uses a unique, sequential identifier that is tied to the basic ingredients going into the response functions, while the actual IXPE CALDB can have the same identifiers for files with a different validity epoch).

The latter difference is also reflected in the format string for the version identifier in the file name, which is `'%02d'` for the IXPE CALDB and `'v%03d'` in the pseudo-CALDB.

If you are really careful you will also notice that, for weighted response files, the position of the weight identifier in the file name *for the modulation factor* is different in the pseudo-CALDB, compared with the real one, i.e., the mapping is in this case

```
ixpe/gpd/cpf/modfact/ixpe_d1_obssim_alpha075_mfact_v010.fits
ixpe/gpd/cpf/modfact/ixpe_d1_20170101_mfact_alpha075_01.fits
```

This is just an historical accident that, at this point, is not worth correcting. (And you probably will never need the modulation factor, anyway.)

12.5 Historical notes

> **Warning**
>
> The links provided below are ultimately pointing to IXPE private repositories to which only collaboration members have access.

The release process and the differences with respect to the previous iterations are summarized on our issue tracker at:

- ixpeirfgen issue #21 and issue #22 (release of version 13);
- ixpeirfgen issue #10 (release of version 12);
- ixpeobssim issue #580 (release of version 11);
- ixpeobssim issue #496 (release of version 10);
- ixpeobssim issue #460 (release of version 9);
- ixpeobssim issue #402 (release of version 7);
- ixpeobssim issue #333 (release of version 6);
- ixpeobssim issue #344 (release of version 5);
- ixpeobssim issue #294 (release of version 4);
- ixpeobssim issue #258 (release of version 3);
- ixpeobssim issue #161 (release of version 2 and differences with respect to version 1).

Version 8 of the response files is the first supporting XSPEC spectro-polarimetric analysis with weights. Version 9 is fairly similar, with a refined parametrization of the MMA effective area. Version 10 features a few new header keywords, and is the one on which the first version of the CALDB submitted to HEASARC is based.

Version 7 of the response files features the first non-diagonal response matrix.

Version 6 of the response function is the first iteration taking full advantage of the flight DU calibration and the telescope end-to-end calibration. This is also the last iteration of the response function using the now infamous 80% cut—the next ones will hopefully support ensemble-weighted analyses.

Starting with version 6 of the instrument response function the PI runs from 0 to 374 (included), corresponding to a physical-space binning spanning the 0–15 keV energy range in steps of 40 eV. (In previous iterations the PI spanned the very same energy interval used to define the response functions, i.e., 1–12 keV.)

All the non-standard versions of the response files (e.g., without the standard cuts or with the MMA alone) have been dropped altogether starting from version 4.

In iterations of the response functions up to v3, ixpeobssim used to ship combined versions of the effective area and modulation factor, that were useful for back-of-the envelope sensitivity calculations. From version 4 onward this is no more the case, and all the relevant applications have been modified to make the appropriate loop over the three detector units where the combined response functions were used before.

CHAPTER

THIRTEEN

ORBIT, SAA AND GTIS

ixpeobssim includes a simplified representation of the IXPE orbit. This allows to calculate the SAA epochs, the target occultation and, ultimately, the good time intervals, in a realistic fashion. While many of these capabilities are similar to those done by the Science Operations Center (SOC) for mission planning, we caution that these capabilities in no way supersede or replace those done by the SOC. They are primarily designed to help account for observational realities when simulating event lists.

13.1 Baseline orbit

IXPE was launched on December 9, 2021, in an orbit characterized by:

- altitude: 601.1 km
- inclination: 0.23 decimal degrees
- eccentricity: 0.0012

For the purpose of `ixpeobssim` simulation we dynamically generate a TLE matching the official TLE retrieved right after launch

```
IXPE
1 49954U 21121A   21351.00640149  .00001120  00000-0  35770-4 0  9994
2 49954   0.2300 281.7657 0011347 134.4260 303.9164 14.90740926  1166
```

While we are aware that this TLE will no be valid for the lifetime of the IXPE mission, this approximation was tentatively deemed sufficient for the purpose of simulation observations. (In general we don't aim at getting the actual GTI for a real observation, just realistic ones.) For completeness, we use the skyfield Python package to deal with all the aspects discussed in this section, and the user is referred to the package documentation for more background information.

Note

The nominal mean motion of 14.9 revolutions per day (at an altitude of 600 km) corresponds to a period of 5800 s. Along with the duration of the SAA epochs (810 s on average), this is one of the two fundamental timescales for IXPE observations.

(For completeness, due to the rotation of the Earth, the average time distance between two consecutive SAA entries is 6210 s.)

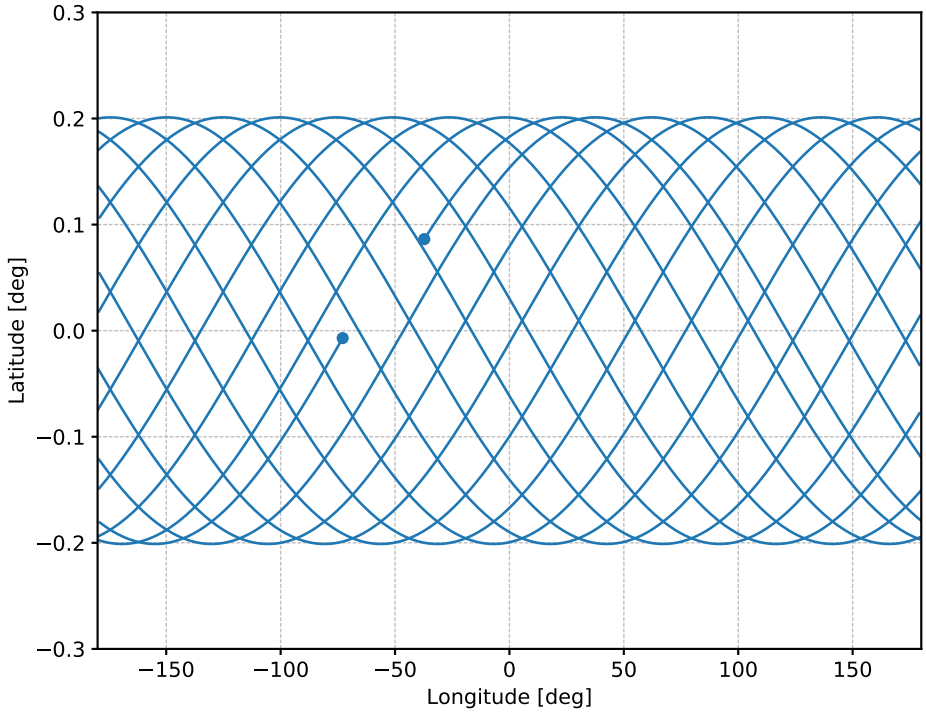


Fig. 1: Footprint of the IXPE orbit over the Earth surface for one day of observations.

13.2 The South Atlantic Anomaly

We parametrize the South Atlantic Anomaly (SAA) as a 12-vertex convex polygon in Earth coordinates. The initial definition of the polygon is based on the heritage of other observatories (e.g., Fermi LAT and GBM, AGILE) and may be refined after launch, but it is more than adequate for pre-launch simulations.

Note

During the SAA passages in nominal science operations we will lower the high voltage for the focal-plane detectors and we shall not acquire science data. It follows that the SAA is one of the fundamental ingredients for determining the good time intervals for the observation.

IXPE being in nearly-equatorial orbits, the duration of the SAA epochs is approximately the same at each passage, the overall variations being of the order of a few %.

Note

With an average SAA epoch duration of 813 s every 6213 s, the average fraction of time spent by IXPE in the SAA is about 13%.

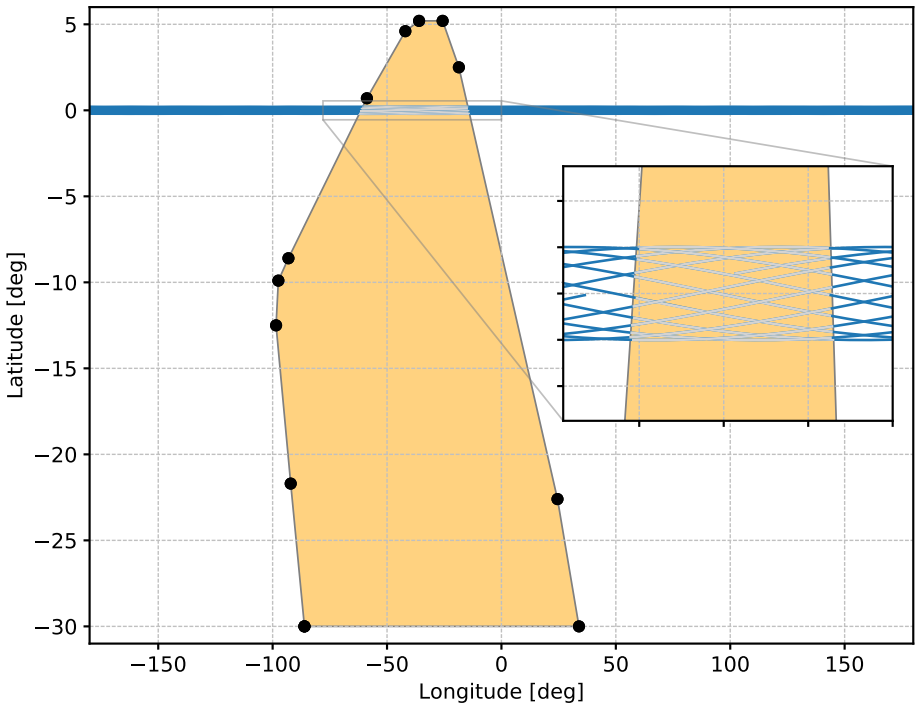


Fig. 2: Footprint of the IXPE orbit over the Earth surface for one day of observation, including the SAA polygon. The shaded parts of the satellite track corresponds to the period of times spent inside the SAA, where the observatory does not take data.

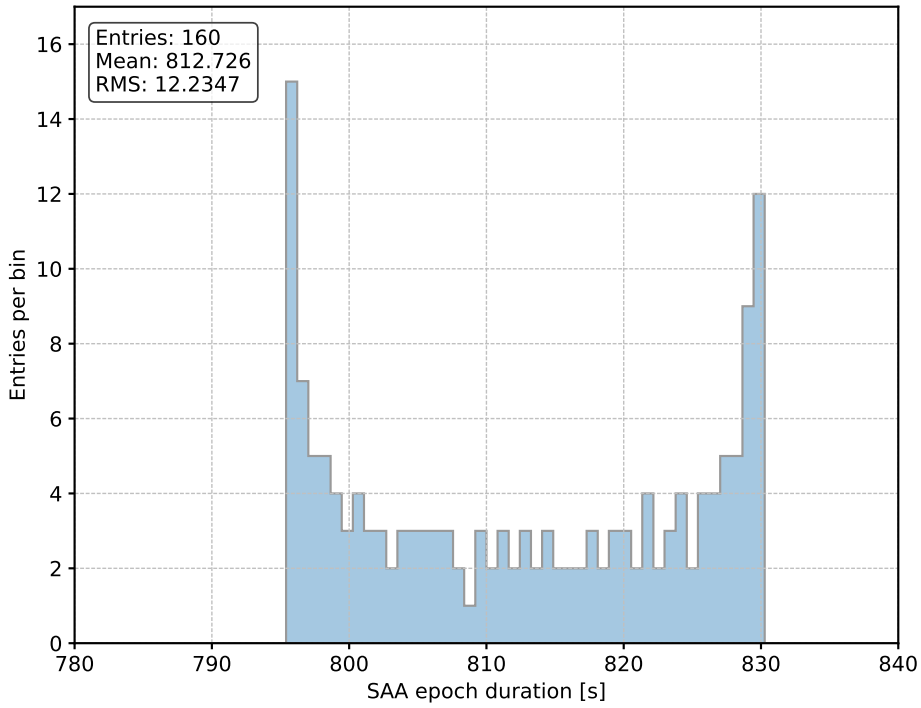


Fig. 3: Distribution of the duration of the SAA epochs over 161 consecutive passages.

13.3 Earth Occultation

A simple consideration of the IXPE orbit geometry (a 600 km altitude orbit above a planet with a radius of 6300 km) makes it clear that a large fraction of the sky will be obstructed by the Earth. Just as the SOC is using in the planning stages of the mission, we assume that a particular sky position is occulted by the Earth whenever the minimum altitude of the line of sight is less than 200 km.

A baseline expectation and rule of thumb for observers is that any particular target with declinations in the range of -50 to 50 degrees will be occulted by the Earth for approximately 40% of an orbit. Sources in the declination range of 50-70 degrees experience less time in occultation, and is effectively negligible for sources beyond +/- 70 degrees declination.

For a given target, the fraction of the orbit where the target is occulted by the Earth can be safely treated as a constant value on both short time scales (days or weeks) as well as longer time scales (months or years). No position on the sky should have particular "window of visibilty" where it is observable for unusually long periods of time.

We finally caution that this calculation in particular does not account for some of the expected higher-order effects that will impact the GTI fraction. As an example, the oblateness of the Earth (which will affect the visibility of a target in particular directions) is not modeled here. Tests against the SOC's tools show that these corrections may affect the total amount of time on-source at about the 1 percent level. Using the target 3C273 as one test case, we found that the SOC predicts this source is visible for 60.0275 percent of the orbit while the calculations included here predict that 3C273 is visible 60.934 percent.

13.4 Sun Pitch Angle

Another observational constraint implemented in ixpeobssim is the times of year where a particular target source right ascension and declination is properly separated from the Sun. We currently set the default separations (hereafter the Sun pitch angle) to reside between 65 and 115 degrees. These restrictions will not affect the GTI's directly (as the Sun pitch angle does not vary significantly over the course of a day or week long exposure) but may help motivate a more sensible choice of start date or help with scheduling a larger number of targets. As stated above, these calculations should only be treated as approximations for the observation planning tools utilized and/or provided by the SOC.

13.5 Good time intervals

The wall-clock time of the observation in xpobssim is controlled by four command-line parameters:

- `startdate`: the start date of the observation (default: "2022-04-21"), assumed to be in UTC. This is a string in the `%Y-%m-%d` (if you don't care about the precise hour of the day) or `%Y-%m-%dT%H:%M:%S.%f` (if you do) format;
- `duration`: the physical duration of the observation in s;
- `saa`: flag instructing xpobssim to consider (i.e., exclude) the SAA passages when calculating the good time intervals (default: False);
- `occult`: flag instructing xpobssim to consider (i.e., exclude) the time interval when the target is occulted by the Earth when calculating the good time intervals (default: False)

Since version 8.6.0 xpobssim is equipped to properly take into account the SAA and the Earth occultation in the calculation of the good time intervals, although it does not do that by default (i.e., you have to enable the two corresponding command-line switches for that to happen). Note that, if you run with the `saa` and `occult` flags enabled, your effective on time will be roughly 55% of the physical duration (this is what will happen in real life).

Warning

In a future ixpeobssim version the `saa` and `occult` flags will be enabled by default. In the meantime, any test of the new functionality is welcome.

13.6 Source visibility

When dealing with sensitivity studies, most of the times one is concerned with the total source ontime (i.e., the sum of all good time intervals) rather that the wall-clock duration of the observation, given that the latter has to be accommodated in the context of the overall observing plan. The combined impact of the SAA and Earth occultation will reduce the good time intervals to 53–87% of the total duration of the observation, with the particular value depending almost exclusively on the declination of the source position. Running simulations with all the physical effects (including the SAA passages and Earth occultations) enabled is instructive for estimating the amount of net exposure for a given observation.

In order to help the users planning an observation, and with all the caveats above, as of version 8.6.0 ixpeobssim provides a rough visibility tool (`xpvisibility.py`) summarizing the relevant information, as illustrated in the figure below.

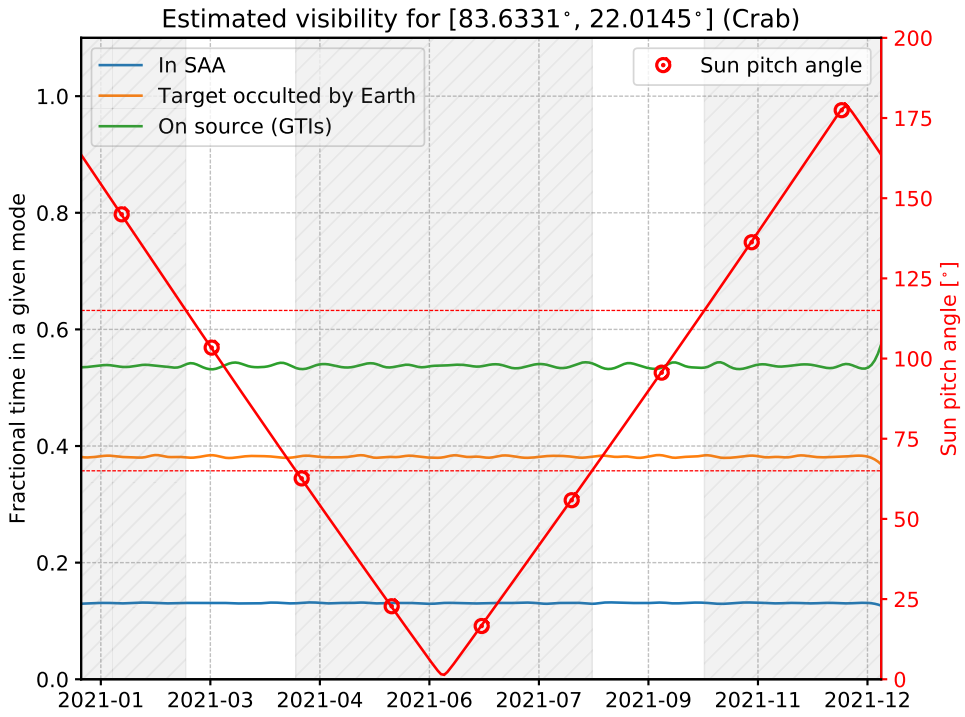


Fig. 4: Output of the `xpvisibility.py` tool, run on the Crab nebula for a time span of a year starting on January 1, 2021.

For reference, the output of

```
xpvisibility.py --srcname Crab --startdate 2021-01-01
```

includes some basic statistics about the duration of the observation, the total sum of good-time internals and the periods when the source is observable due to time constraints.

```
>>> Target coordinates: R. A. 83.633083, Dec. 22.014500
>>> Start MET for the observation: 126230400.000 s
>>> Stop MET for the observation: 157788000.000 s
>>> Wall-clock duration of the observation: 31557.600 ks
>>> Total GTI: 16959.307 ks (53.7 %)
>>> Average visibility efficiency: 61.827%
>>> Viewing periods (due to Sun constraints):
>>> [01] 2021-02-17T19:53:11.046608--2021-04-09T00:41:20.692787
>>> [02] 2021-08-22T06:13:02.833588--2021-10-12T12:26:04.468925
```

This can be compared with the output of the HEASARC Viewing Tool

```
Viewing Results
Input equatorial coordinates:
Crab, resolved by SIMBAD (local cache) to
[ 83.6331°, 22.0145° ], equinox J2000.0

IXPE (planning)

This mission is in the planning stage.

*** VIEWING Version 3.4      run on 2020 May 13 ***
for the period 2020 May 13 to 2022 May 14

With IXPE (Sun angle range =  65-115):
Observable between 2020 Aug 22           and 2020 Oct 12
Observable between 2021 Feb 17           and 2021 Apr 09
Observable between 2021 Aug 22           and 2021 Oct 12
Observable between 2022 Feb 18           and 2022 Apr 09
```

13.7 Dithering

In nominal data-taking configuration the IXPE observatory is dithered around the pointing direction, the main reason for that being averaging out the spurious modulation and making correcting for that practically possible.

The dithering pattern has the basic form

$$\delta x = A\cos(\omega_a t)\cos(\omega_x t)$$
$$\delta y = A\sin(\omega_a t)\sin(\omega_y t)$$

with a default amplitude of 1.6 arcsec and the three periods corresponding to the angular pulsations in a, x and y being 907 s, 101 s and 449 s, respectively.

When convolved with the PSF of the instrument, the dithering pattern provides a image on the focal plane that is approximately uniform (within a factor of 2) for a point source.

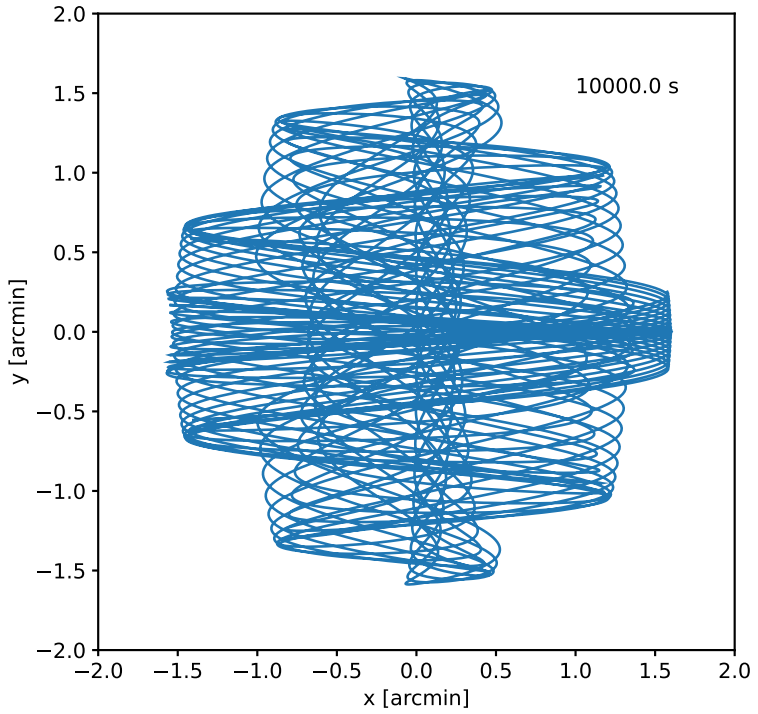


Fig. 5: Representation of the default IXPE dithering path for a 10 ks observation.

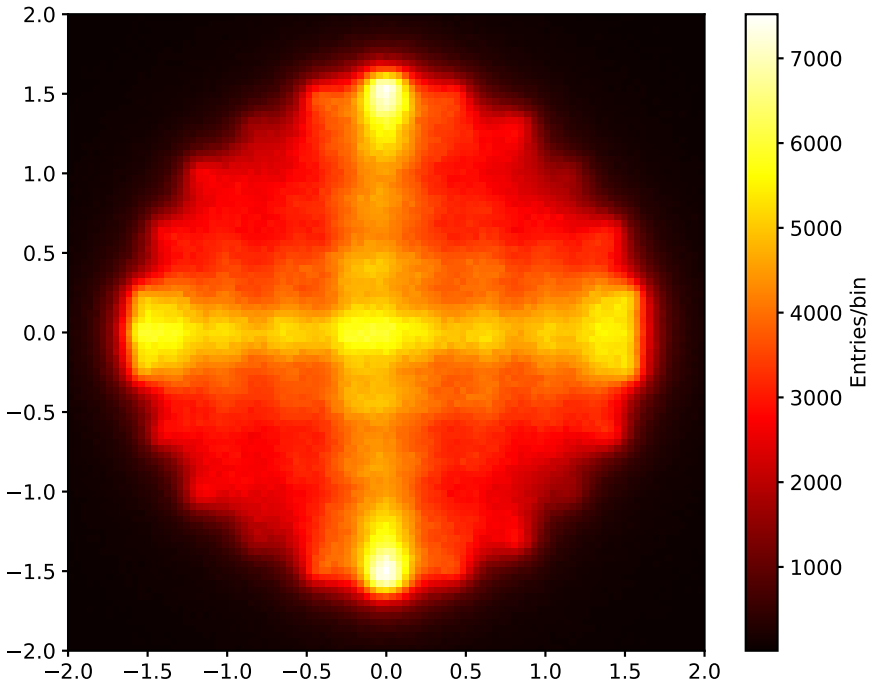


Fig. 6: Representation of the dithered image of a point source on the IXPE focal plane.

13.8 Pointing history

The pointing history is written in a dedicated extension of the IXPE photon lists called `SC_DATA`, with the exact structure described in the section about *Data format*.

The pointing history is written at regular time intervals, and the time step can be set from command line. Tests show that a 10 s interval provides a maximum error of 0.15 arcsec in the pointing direction in the sky, when the latter is reconstructed via an interpolated spline from the `SC_DATA` table. A 5 s time interval provides a sub-arcsec error in the reconstructed pointing direction.

CHAPTER

FOURTEEN

BACKGROUNDS

Although for relatively bright point sources background is typically not an issue for IXPE, it might become important for dimmer point sources and/or extended sources, especially at high energy.

We distinguish three different types of backgrounds: galactic, extra-galactic, and instrumental—as it turns out, the latter is typically the most important. ixpeobssim provides ways to simulate background components and handle backgrounds in high-level analysis. In the remaining of this section we shall briefly cover the simulation part.

Note

All the background components treated by our software are assumed to be completely unpolarized. Some instrumental background components may be polarized, but over the years no way has been found to predict its polarization angle and degree so the only way to asses its polarization is to measure it directly from the data, and then subtract it from the total signal.

See also

The *ixpeobssim.srcmodel.bkg* module contains all the background-related modeling facilities.

14.1 Galactic background

This is, admittedly, the part that is less extensively covered in ixpeobssim. We do have a *ixpeobssim.srcmodel.bkg.xGalacticBkg* class that is designed to interface to the the HEASARC background tool and, particularly, use the ROSAT count rate in the 1–2 keV energy band (R7). The model is overly simple (uniform in sky coordinates over the field of view and with a fixed spectral index), with the overall intensity being the only parameter that can be changed by the user.

Note

Tuning the galactic X-ray background to a specific observation is still tricky in the current ixpeobssim setup. Any help in making this more user-friendly (and, possibly, developing an iterface to the e-ROSITA data) is more than welcome.

14.2 Extra-galactic background

The extra-galactic background, although not really the relevant contribution in any practical situation, is relatively straightforward to handle. We provide a *ixpeobssim.srcmodel.bkg.xExtragalacticBkg*, with the basic parametrization taken from Gruber et al., 1999.

14.3 Instrumental background

This is most often the primary source of background, its main characteristics being that the instrumental background is modeled in instrument coordinates and not convolved with the instrument response functions—not with the effective area, nor with the vignetting, and typically, not with the energy dispersion.

In terms of simulation facilities, the two most important data structures are

- *ixpeobssim.srcmodel.bkg.xPowerLawInstrumentalBkg*: a background component with a power-law energy spectrum with adjustable normalization and spectral index;
- *ixpeobssim.srcmodel.bkg.xTemplateInstrumentalBkg*: a background component with an energy spectrum derived from an actual observation—typically by using a dim point source and removing the source counts at the center of the field of view, and then properly rescaling the the relevant area.
- The current implementation of the X-ray intrumental background is based on the average value of counts in the full IXPE band obtained from years of data taken in occultation (e.g.: when the source is blocked by the Earth). In addition an in-eclipse (e.g..: when the sun is not shining on the detector) filter and a particle cut (Di Marco, et al. AJ 165, 143 (2023)) are applied.
- To our knowledge, this background component is stable over time and shows no significant polarization, although a mild trend of counting rate decrease over multiple yearsmay be expected. It is always a good idea to check that the static background component does not overshoot the data.

By deafault the instrumental background is generated uniformly in instrument coordinates, although we do support a generic linear radial dependence, as briefly summarized in the next section.

14.3.1 Radial dependence

IXPE celestial observations show that the distribution of the instrumental background in detector coordinates is not necessarily rigorously flat. In this case, we provide the means to model itwith a linear function of the radial distance from the center.

The radial slope alpha represents the fractional half-excursion of the variation across the size h of the fiducial rectangle. For alpha = 0 the detector position are distributed uniformly over the fiducial rectangle. For alpha = 2 the radial dependence is maximal, and the density of events is zero at the center of the detector.

The values of alpha found experimentally vary from observation to observation (at a level which is consistent with expectations based on the typical variations of the radiation environment in low-Earth orbit), and are typically lower than 0.2.

14.3.2 Flaring background

In addition to the static background, IXPE observations are affected by flaring in time coincidence with solar flares when the sun is shining on the detector. The time scale of this effect is of the order of minutes to a few hours and can be characterized by the difference between the spectrum obtained when the sun is shining on the detector and the spectrum obtained when the sun is not shining on the detector. There are currently two ways of getting rid of the flaring background

- The flares can then be estimated in the full detector area and subtracted from the data. This relies on the assumption that the flaring activity is unuiform across the detector, and recovers a relatively high statistics of the flare by integrating over it. It can then be subtracted from any spatial selection (or map binning) so that both the spectrum and the polarization properties of the flare are properly accounted for. This is handled by *xpbin* and *xpsun*.
- *xpsun* splits the observation dataset in two parts, the "insun", where the sun is shining on the detector, and the "ineclipse", where the sun is not shining on the detector. The two datasets can be used to create a "deflared" dataset by subtraction
- *xpbin* launched with the level 2 files with the appropriate spatial (not time!) cuts as positional argument, the full-detector insun and ineclipse with the –insun and –ineclipse options will provide a _deflared.fits file which has been already flare-subtracted.
- The flares can be removed with an ad-hoc cut in the time intervals, removing the time intervals in which the count rate increases over some arbitrary threshold. A standard way of doing this is the so-called "sigma-clipping" method with a 3 sigma threshold, or multiple pass sigma clipping. Unfortunately, the solar activity has been proven to be often sub-threshold and carries a small number of counts that sometimes are however persistently strongly polarized (Bucciantini et al. A&A 699, A33 (2025)). The software does not currently support this method.

14.3.3 Creating a template

xpbkgtemplate provides a tool to generate an instrumental background starting from an input pha1 file. The tool is designed specifically to work with dark fields extracted from ixpe observation, and as such it requires the pha1 file to have a backscal keyword defined in its header to account for the size of the extraction region.

The rate of the background is then extracted from each file that is provided, multiplied by its livetime and normalized with respect to the total detector area. Everything is then averaged and rescaled back to physical units to create an average spectrum which is smoothed with a spline and saved as an ascii file that can be used as an agument to generate an `ixpeobssim.srcmodel.bkg.xTemplateInstrumentalBkg` object into a config file to be used by *xpobssim*. Note that while this is designed to create the average spectrum from a single observation, the algorithm will run just fine on multiple observation and treat each file in the same way.

The relevant piece of code for the generation of the templateresides in `ixpeobssim.bkg.instr` in which the function create_background_template is defined and invoked by **|xpbkgtemplate|**.

CHAPTER

FIFTEEN

FILTERING EVENT LISTS

`ixpeobssim` provides facilities to filter event lists in different flavors, selecting in time, phase, energy and position in the sky. Or work-horse application in this respect is *xpeselect*.

```
...

usage: xpselect.py [-h] [--suffix SUFFIX] [--tmin TMIN] [--tmax TMAX]
                   [--tinvert {True,False}] [--phasemin PHASEMIN]
                   [--phasemax PHASEMAX] [--phaseinvert {True,False}]
                   [--emin EMIN] [--emax EMAX] [--einvert {True,False}]
                   [--ra RA] [--dec DEC] [--rad RAD] [--innerrad INNERRAD]
                   [--regfile REGFILE] [--reginvert {True,False}]
                   [--mask MASK] [--mcsrcid MCSRCID] [--mc {True,False}]
                   [--ltimeupdate {True,False}] [--ltimealg {LTSUM,LTSCALE}]
                   [--overwrite {True,False}]
                   filelist [filelist ...]

Basic data selection interface. This utility can process IXPE event files
(i.e., photon lists) and apply a generic selection on sky-position, time,
phase, energy, etc., producing smaller event files. In conjunction with
xpbin this is handy to, e.g., energy- or phase-resolved polarization
measurements.

xpselect supports spatial selections from a ds9 region file thorough the
--regfile command-line switch.

positional arguments:
  filelist              path(s) to the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  --suffix SUFFIX       suffix for the output files (default: None)
  --tmin TMIN           minimum time in s (default: None)
  --tmax TMAX           maximum tims in s (default: None)
  --tinvert {True,False}
                        invert the time selection (default: False)
  --phasemin PHASEMIN   minimum phase for periodic sources (default: None)
  --phasemax PHASEMAX   maximum phase for periodic sources (default: None)
  --phaseinvert {True,False}
                        invert the phase selection (default: False)
  --emin EMIN           minimum energy in keV (default: None)
```

(continues on next page)

(continued from previous page)

```
--emax EMAX           maximum energy in keV (default: None)
--einvert {True,False}
                      invert the energy selection (default: False)
--ra RA               RA of acceptance cone in decimal degrees (default:
                      None)
--dec DEC             Dec of acceptance cone in decimal degrees (default:
                      None)
--rad RAD             ROI radius in arcminutes (default: None)
--innerrad INNERRAD   ROI inner radius in arcminutes (for selecting annuli)
                      (default: None)
--regfile REGFILE     path to a ds9 region file (default: None)
--reginvert {True,False}
                      invert the ds9 region file selection (default: False)
--mask MASK           path to .npy file with a saved boolean event mask
                      (default: None)
--mcsrcid MCSRCID     the Monte Carlo source ID to select (default: [])
--mc {True,False}     use Monte Carlo information (default: False)
--ltimeupdate {True,False}
                      update the livetime-related keywords in the output
                      files (default: False)
--ltimealg {LTSUM,LTSCALE}
                      algorithm to be used to update the livetime (default:
                      LTSCALE)
--overwrite {True,False}
                      overwrite existing output files (default: True)
```

Note

Added in version 22.0.0.

xpeselect was largely refactored in ixpeobssim version 22.0.0, mainly in response to issue #169. If you are using `xpselect` and lagging behind the `ixpeobssim` version, you should definitely consider upgrading.

At the time of writing, the XSELECT FTOOL is being updated to support IXPE, with a new HEASOFT release expected in February, 2022, and it is conceivable that *xpeselect* will be modified in the future to avoid overlap in functionality with the tools officially supported by HEASARC. In a nutshell: we shall try and strip off xpselect anything that will be provided out off the shelf by XSELECT.

The basic idea is that *filtered event lists should be functionally identical to their parents level-2 files* and they should, at least in principle, inter-operate with all the analysis tools in exactly the same fashion. If that is not the case, it is likely a bug :-)

15.1 Selecting in time or phase

Selections in time or phase are achieved by means of the `--tmin/max` and `--phasemin/max` command-line switches. For the sake of making it easier to keep track of the livetime and time-related header keywords when selecting in time and phase, `xpselect` performs some minimal validation on the input values of the command line switches, and will exit with an error message if any of the conditions below is not met:

- `TSTART <= TMIN <= TSTOP`
- `TSTART <= TMAX <= TSTOP`
- `TMAX > TMIN`
- `0. <= PHASEMIN <= 1.`
- `0. <= PHASEMAX <= 1.`

Warning

xpeselect does not currently support concurrent selections on time *and* phase, and there is no specific plan to add this functionality, unless proved to be important in practice. In other words, you can select an arbitrary time *or* phase interval, but not at the same time.

Chaining multiple *xpeselect* calls in time or phase is also discouraged as, depending on the exact setup, this makes keeping track of the livetime extremely difficult. This should not be a limitation for any real analysis.

It goes without saying that, in order to be able to select in phase, you must run the *xpphase* tool first, in order to have the `PHASE` column added to the event list.

15.1.1 Propagating the livetime

Note

Added in version 22.0.0.

As of version *xpeselect* provides optional command-line switches to propagate the relevant time-related header keywords to the filtered event lists. We are fully aware that the treatment is approximate, at the limit of naiveness, and we fully anticipate it will undergo modifications as we gain experience with flight data.

Warning

The new infrastructure for the livetime correction is only suitable to experiment with simulated data, and does not yet inter-operate with filtered level-2 flight data. This is the reason why the correction is disabled by default.

xpeselect tries to do a decent job in keeping track of the livetime when selecting in time or phase. Unfortunately the matter is not completely trivial, and a short discussion is in order, here.

IXPE event lists provide a `LIVETIME` column encapsulating *the livetime from the previous event in microseconds*, or the total time in which the trigger was enabled from the end of the readout of the previous event from the trigger time of the current event. Under normal conditions the sum of the `LIVETIME` column should be equal to the values of the `LIVETIME` and `EXPOSURE` header keywords up to numerical rounding errors.

The default mean for *xpeselect* to re-calculate the livetime, when time or phase selections are applied is to simply sum the LIVETIME column *over the events surviving the selections*. This is what happens when the --ltimealg command-line switch is set to LTSUM, and the thing works as advertised when the selected events are contiguous, e.g., when a simple time selections is applied.

When selecting in phase the thing gets more complicated, as in some circumstances (i.e., when the average time distance between subsequent events is larger than the period of the ephemeris we use for folding) the LIVETIME value for any particular event will in general be referred to an event in a *different phase bin*. In order to cope (approximately) with this situation, *xpeselect* provide an alternative mean of propagating the livetime, that is enabled by using the --ltimealg LTSCALE command-line switch. In a nutshell, we first calculate the average dead time per event for the original event list by simply dividing the difference between the ONTIME and LIVETIME header keywords by the total number of events. (This is typically in the ms ball-park.) This average dead time per event can in turn be used to calculate a total *scaled livetime* within the selection, by simply multiplying it by the number of events within the selection.

Warning

The livetime scaled we just described is only valid in the limit that the average dead time per event in the original photon list is representative of the average dead time per event within the selection. Since the dead time per event depends on the size of the region of interest of the event, which in turns depends on the energy, this assumption might not be accurate in the presence of strong spectral variations as a function of the pulse phase. (In practice, all the dependences are mild, and this should not be an issue in most practical situations.)

In many conditions, and particularly for selections in time, the default LTSUM mechanism will work fine. As a rule of thumb, you can use LTSUM when selecting in time and LTSCALE when selecting in phase, but be advised that there might be cases when one need to examine more closely the situation and opt for an ad-hoc solution.

For completeness, when we select in time, the header keywords in the releavant extensions are update as follows

- TSTART -> max(TSTART, TMIN)
- TSTOP -> min(TSTOP, TMAX)
- ONTIME -> TSTOP - TSTART

and when we select in phase we have instead.

- TSTART -> TSTART (unchanged)
- TSTOP -> TSTOP (unchanged)
- ONTIME -> `ONTIME * selected_phase_fraction

In all cases, the livetime-related columns are updated according to:

- EXPOSURE -> modified livetime (LTSUM or LTSCALE)
- LIVETIME -> same as EXPOSURE
- DEADC -> EXPOSURE / ONTIME

CHAPTER

SIXTEEN

BINNED DATA PRODUCTS

ixpeobssim comes with its own set of facilities for binning event lists in several different flavors. Rather than just for the fun of reinventing the wheel, this was intentionally done with the precise purpose of making easy for people to develop *Analysis pipelines*.

The main interface for binning photon lists is provided by *xpbin*, while the main visualization interface is provided by *xpbinview*.

From an architectural standpoint, each binning flavor comes with its own interface classes for output (i.e., creating binned files from event lists) and input (i.e., reading, visualizing and manipulating binned FITS files). This is documented for developers in the APIs.

Note

All the binned data products implemented in ixpeobssim provide an overloaded __iadd__() dunder that allows to sum multiple files transparently—which is customarily done, e.g., over the three detector units. Additional methods such as subtraction *__isub__()* and multiplication by a scalar *__imul__()* are implemented for polarization data cubes, maps and count spectra, for the purpose of rescaling (e.g.: by the livetime or backscal) and subtracting data (as for background subtraction).

When manipulating binned files programmatically you can call the class constructor directly or, more commonly, use the `from_file_list()` hook, e.g.:

```
from ixpeobssim.binning.polarization import xBinnedCountSpectrum

# Load a count spectrum for a single DU
spec = xBinnedCountSpectrum('du1.fits')

# Load the combined count spectrum for three DUs
spec = xBinnedCountSpectrum.from_file_list(['du1.fits', 'du2.fits', 'du3.fits'])
```

We emphasize that all the polarization analysis takes place in Stokes parameter space, starting from the event-by-event Stokes parameters corrected for the spurious modulation

$$q_i \rightarrow \text{content of the Q column in level-2 files}$$
$$u_i \rightarrow \text{content of the U column in level-2 files}$$

Additionally, the polarization analysis tools are aware of the event weights

$$w_i \rightarrow \text{content of the W\_MOM column in level-2 files}$$

Warning

After the spurious modulation subtraction the event-by-event Stokes parameters are not properly normalized, i.e., can lie outside (and even very much outside) the physical [-2, 2] interval. This is generally not an issue, as the the spurious modulation subtractions provides the right answer *on average* for any reasonably large event sample.

The basic strategy for the spurious modulation subtraction in the SOC pipeline is described in this paper.

See also

Data format

16.1 Stokes spectra

Binned Stokes spectra are the main interface to spectro-polarimetric fitting in XSPEC, and one of the most relevant data structures in ixpeobssim.

Stokes spectra are a simple generalization of the standard pha spectra, and the corresponding FITS files designed to store them are closely modeled on the PHA1 file definition in the OGIP spectral file format (in fact the spectrum corresponding to the I Stokes parameter *is* a standard PHA1 file).

Not surprisingly—at least in the IXPE context, where we neglect circular polarization—Stokes spectra come in three flavors:

- I, mapped to `*pha1.fits` files via the *ixpeobssim.binning.polarization.xEventBinningPHA1* class;
- Q, mapped to `*pha1q.fits` files via the *ixpeobssim.binning.polarization.xEventBinningPHA1Q* class;
- U, mapped to `*pha1u.fits` files via the *ixpeobssim.binning.polarization.xEventBinningPHA1U* class.

Accordingly, binned files in the three flavors are created by invoking *xpbin* with the `PHA1`, `PHA1Q` and `PHA1U` values for the `--algorithm` command-line switch.

The *ixpeobssim.binning.polarization.xBinnedCountSpectrum* is the read interface to all of the three flavors.

16.1.1 Binary table definition

The definition of the FITS binary tables for storing Stokes spectra is the same for the three different incarnations

```
class xBinTableHDUPHA1(xBinTableHDUBase):

    """Binary table for binned PHA1 data.
    """

    NAME = 'SPECTRUM'
    HEADER_KEYWORDS = [
        ('HDUCLASS', 'OGIP'),
        ('HDUCLAS1', 'SPECTRUM'),
        ('HDUCLAS2', 'TOTAL'),
        ('HDUCLAS3', 'RATE'),
```

(continues on next page)

(continued from previous page)

```
        ('CHANTYPE', 'PI'),
        ('HDUVERS' , '1.2.1', 'OGIP version number'),
        ('TLMIN1'  , TLMIN, 'first channel number'),
        ('TLMAX1'  , TLMAX, 'last channel number'),
        ('CORRSCAL', 1., 'scaling for correction file'),
        ('POISSERR', False, 'use statistical errors'),
        ('BACKFILE', ''),
        ('CORRFILE', ''),
        ('SYS_ERR' , 0.),
        ('AREASCAL', 1.),
        ('BACKSCAL', 1.)
        ]
    DATA_SPECS = [
        ('CHANNEL' , 'J'),
        ('RATE'    , 'E', 'counts/s'),
        ('STAT_ERR', 'E', 'counts/s'),
        ]
```

the only difference being the value of the `XFLT0001` keyword of the `SPECTRUM` extension, which needs to be properly set in order to instruct XSPEC to apply the polarization correction—and particularly:

- `Stokes:0` for I spectra;
- `Stokes:1` for Q spectra;
- `Stokes:2` for U spectra.

16.1.2 Technical details

Analytically, Stokes spectra are (possibly weighted) histograms in pulse-invariant space where, given the definition

$$\varepsilon_k = \frac{1}{T}\frac{\sum_{\mathrm{PI}=k} w_i}{\sum_{\mathrm{PI}=k} w_i^2}$$

the content of the k-th bin is given by

$$I_k = \varepsilon_k \sum_{\mathrm{PI}=k} w_i \quad \text{and} \quad \sigma_{I_k} = \varepsilon_k \sqrt{\sum_{\mathrm{PI}=k} w_i^2}$$

$$Q_k = \varepsilon_k \sum_{\mathrm{PI}=k} w_i q_i \quad \text{and} \quad \sigma_{Q_k} = \varepsilon_k \sqrt{\sum_{\mathrm{PI}=k} (w_i q_i)^2}$$

$$U_k = \varepsilon_k \sum_{\mathrm{PI}=k} w_i u_i \quad \text{and} \quad \sigma_{U_k} = \varepsilon_k \sqrt{\sum_{\mathrm{PI}=k} (w_i u_i)^2}$$

It is worth mentioning that for an un-weighted analysis (i.e., when the weights are unitary for all the events) the above expressions reduce to the more familiar ones

$$
\begin{aligned}
\varepsilon_k &\xrightarrow[w_i=1]{} \frac{1}{T} \\
I_k &\xrightarrow[w_i=1]{} \frac{N_k}{T} \quad \text{and} \quad \sigma_{I_k} \xrightarrow[w_i=1]{} \frac{\sqrt{N_k}}{T} \\
Q_k &\xrightarrow[w_i=1]{} \frac{1}{T}\sum_{\text{PI}=k} q_i \quad \text{and} \quad \sigma_{Q_k} \xrightarrow[w_i=1]{} \frac{1}{T}\sqrt{\sum_{\text{PI}=k} q_i^2} \\
U_k &\xrightarrow[w_i=1]{} \frac{1}{T}\sum_{\text{PI}=k} u_i \quad \text{and} \quad \sigma_{U_k} \xrightarrow[w_i=1]{} \frac{1}{T}\sqrt{\sum_{\text{PI}=k} u_i^2}.
\end{aligned}
$$

The reader is referred to the section about the *XSPEC support* for more details about one would use binned Stokes spectra in a spectro-polarimetric fit using XSPEC.

16.1.3 Normalized Stokes spectra

xpbin provides facilities to produce *normalized Stokes spectra* (i.e., spectra where the Q and U channels are divided by the corresponding I channels) with the `PHA1QN` and `PHA1UN` binning algorithms:

- Q/I, mapped to `*pha1qn.fits` files via the *ixpeobssim.binning.polarization.xEventBinningPHA1QN* class;
- U/I, mapped to `*pha1un.fits` files via the *ixpeobssim.binning.polarization.xEventBinningPHA1UN* class.

This can be used in purely polarimetric fits, e.g., in XSPEC, with the spectral part of the problem completely factored out—more about this in the section about *XSPEC support*.

16.2 Polarization analysis

Binned Stokes cubes are not the only binned data product that is relevant to polarization measurement. ixpeobssim provides a series of data structures designed to hold broadband polarization information—possibly in multiple energy layers and sky-coordinate bins.

Warning

One important point to emphasize up-front is that the formalism described in this section neglects *by construction* the effect of the energy dispersion (which is instead properly handled when fitting with XSPEC). The focus, here, is to provide model-independent quantities that are easy to calculate and visualize, rather than guaranteeing full accuracy.

Whether that is actually important or not depends critically on the application. Comparing the relevant data products binned in true and reconstructed energy (you can toggle the `--mc` command-line switch to do just that) is a way to gauge the effect of the energy dispersion in a given, particular setup.

16.2.1 Basic formalism

All the polarization analysis in ixpeobssim is based on *the* paper by Kislat et al. (2015), whose formalism is (partially) implemented *verbatim* (well, almost—you will notice that IXPE adopts a slightly different convention for the event-by-event Stokes parameters, resulting in a factor of 2 popping out here and there) in the `ixpeobssim.evt.kislat2015` Python module.

More specifically, for each event we define the three additive *reconstructed* quantities

$$\tilde{i}_i = \frac{w_i}{A_{\rm eff}(E_i)}$$
$$\tilde{q}_i = \frac{w_i q_i}{A_{\rm eff}(E_i)\mu(E_i)}$$
$$\tilde{u}_i = \frac{w_i u_i}{A_{\rm eff}(E_i)\mu(E_i)}$$

A few remarks on these basic definitions:

- the inverse of the effective area in the definition of the weights acts as an acceptance correction guaranteeing that the relevant quantities are summed (or averaged) over the input source spectrum—as opposed to the measured count spectrum;
- any additional (multiplicative) event-by-event weight can be easily incorporated into the first term (and gets automatically propagated to the other two);
- the effective area and modulation factor are calculated at the measured energy—see the warning above about the effect of the energy dispersion.

(Note the lack of treatment for the energy dispersion is a consequence of our, largely simplistic, implementation of the full formalism described in Kislat et al. (2015), rather than a fundamental limitation of the paper.)

That all said, the measured Stokes parameters over a generic subset S of the events (be that a specific energy range, or a spatial bin in sky coordinates), is obtained by simply summing the event-by-event quantities:

$$I = \sum_{i \in S} \tilde{i}_i$$
$$Q = \sum_{i \in S} \tilde{q}_i$$
$$U = \sum_{i \in S} \tilde{u}_i$$

Warning

This formalism has been thoroughly tested with Monte Carlo simulation, and provides the right answer in the limit of large statistics (i.e., for any sensible polarization analysis). When the statistics is low, though, due to the very nature of the quantities defined above, the recovered Q and U Stokes parameters can lie outside the physical bounds.

Counter-intuitive as it is, this is the same as saying that, with low statistics, you can have an upward fluctuation of the measured modulation and, when you divide by the modulation factor to recover the polarization, there is nothing preventing the latter from exceeding 1.

You should be aware of the limitation of this method when dealing with small samples. The uncertainties on Q and U will tell you right away whether that is the case.

The polarization degree and angle can be recovered with the usual formulae, and Kislat et al. (2015) provides all the facilities to propagate the statistical uncertainties. Note that, in a weighted scheme, the error propagation requires

keeping track of the sum of the weights *squared*

$$W_2 = \sum_{i \in S} w_i^2,$$

and the minimum detectable polarization reads

$$\text{MDP} = \frac{4.29\sqrt{W_2}}{\langle \mu \rangle \, I}.$$

Two related, interesting quantities are the *effective number of events* and the *fractional weight*

$$N_{\text{eff}} = \frac{I^2}{W_2}$$
$$f_w = \frac{N_{\text{eff}}}{N}$$

which reduce to the actual number of counts (and to 1, respectrively), in the un-weighted case. Incidentally, these quantities are used to build the effective area in the weighted case.

Warning

The polarization cubes, maps and map cubes are not equipped (yet) to treat the background, or any estimate of the background, in any specific way.

Note

Added in version 25.0.0.

All the binned data structures have been largely reworked to add the necessary infrastructure to keep track of the errors on the Stokes parameters and the significance of a detection.

16.2.2 Polarization cubes

The simplest possible data structure holding polarization information is a *polarization cube*. A polarization cube is generated by *xpbin* run with the `--algorithm PCUBE` command-line switch, and is stored as a FITS binary table where the relevant metrics are calculated in energy bins.

ixpeobssim.binning.polarization.xBinnedPolarizationCube is the basic read interface to polarization cubes.

The following snippet from the source code should be self-explaining.

```
class xBinTableHDUPCUBE(xBinTableHDUBase):

    """Binary table for binned PCUBE data.
    """

    NAME = 'POLARIZATION'
    HEADER_KEYWORDS = []
    # Be careful: if you change any of these, make sure you update the
    # MDP_COLUMNS and POL_COLUMNS class members below, as this all need to be
    # in synch with the MDP and polarization maps and map cubes.
```

(continues on next page)

(continued from previous page)

```
    DATA_SPECS = [
        ('ENERG_LO', 'E', 'keV', 'low energy bound'),
        ('ENERG_HI', 'E', 'keV', 'high energy bound'),
        ('E_MEAN'  , 'E', 'keV', 'average energy within the bin'),
        ('COUNTS'  , 'J', ''   , 'number of counts'),
        ('MU'      , 'E', ''   , 'effective modulation factor'),
        ('W2'      , 'E', ''   , 'sum of weights squared'),
        ('N_EFF'   , 'E', ''   , 'effective number of events w/o acceptance correction'),
        ('FRAC_W'  , 'E', ''   , 'N_EFF / COUNTS'),
        ('MDP_99'  , 'E', ''   , 'minimum detectable polarization at the 99% CL'),
        ('I'       , 'E', ''   , 'I Stokes parameter'),
        ('I_ERR'   , 'E', ''   , '1-sigma uncertainty on I'),
        ('Q'       , 'E', ''   , 'Q Stokes parameter'),
        ('Q_ERR'   , 'E', ''   , '1-sigma uncertainty on Q'),
        ('U'       , 'E', ''   , 'U Stokes parameter'),
        ('U_ERR'   , 'E', ''   , '1-sigma uncertainty on U'),
        ('QN'      , 'E', ''   , 'normalized Q Stokes parameter Q/I'),
        ('QN_ERR'  , 'E', ''   , '1-sigma uncertainty on Q/I'),
        ('UN'      , 'E', ''   , 'normalized U Stokes parameter Q/I'),
        ('UN_ERR'  , 'E', ''   , '1-sigma uncertainty on U/I'),
        ('QUN_COV' , 'E', ''   , 'covariance between QN and UN'),
        ('PD'      , 'E', ''   , 'measured polarization degree'),
        ('PD_ERR'  , 'E', ''   , '1-sigma uncertainty on the polarization degree'),
        ('PA'      , 'E', 'deg', 'measured polarization angle'),
        ('PA_ERR'  , 'E', 'deg', '1-sigma uncertainty on the polarization angle'),
        ('P_VALUE' , 'E', ''   , 'p-value for the null hypothesis (no polarization)'),
        ('CONFID'  , 'E', ''   , 'confidence of the polarization detection'),
        ('SIGNIF'  , 'E', ''   , 'detection significance in equivalent gaussian sigma')
        ]
    COL_NAMES = [col_name for col_name, *_ in DATA_SPECS]
    # Be careful: these need to be in synch with the DATA_SPECS above.
    # MDP_COL_NAMES defines the extensions of the MDP maps and map cubes.
    # POL_COL_NAMES defines the extensions of the polarization maps and map cubes.
    MDP_COL_NAMES = COL_NAMES[2:10]
    POL_COL_NAMES = COL_NAMES[2:]
```

It is worth mentioning that, while some of the columns (i.e., polarization degree and angle, associated uncertainties and MDP) can be calculated from the others (and, are, in fact, recalculated when polarization cubes are summed over the three detector units) we include this redundant information in the FITS file as a convenience.

16.2.3 Polarization-map cubes

Polarization-map cubes hold the exact same information as polarization cubes, but binned in sky-coordinates. Each column in a polarization cube maps directly into an image extension in the corresponding polarization-map cube file. Each image extension has three dimensions (two for the spatial binning and one for the energy binning), and the `EBOUNDS` binary table holds the physical bounds for the energy layers.

Polarization-map cubes are generated by *xpbin* run with the `--algorithm PMAPCUBE` command-line switch, and the main read-interface is *ixpeobssim.binning.polarization.xBinnedPolarizationMapCube*.

> **Tip**
>
> *xpbin* provides a `PMAP` binning a mode that is simply an alias for a polarization-map cube with a single energy layer—which we refer to as a *polarization map*.

16.2.4 MDP-map cubes

MDP-map cubes are stripped-down versions of polarization cubes where only the necessary information for the calculation of the minimum detectable polarization is retained.

```
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 xPrimaryHDU     50   ()
  1  E_MEAN        1 ImageHDU        24   (1, 1, 1)   float64
  2  COUNTS        1 ImageHDU        24   (1, 1, 1)   float64
  3  MU_MDP        1 ImageHDU        24   (1, 1, 1)   float64
  4  I_MDP         1 ImageHDU        24   (1, 1, 1)   float64
  5  W2_MDP        1 ImageHDU        24   (1, 1, 1)   float64
  6  MDP_99        1 ImageHDU        24   (1, 1, 1)   float64
  7  N_EFF         1 ImageHDU        24   (1, 1, 1)   float64
  8  FRAC_W        1 ImageHDU        24   (1, 1, 1)   float64
  9  EBOUNDS       1 xBinTableHDUEBOUNDS     34   1R x 2C   ['E', 'E']
```

MDP-map cubes are generated by *xpbin* run with the `--algorithm MDPMAPCUBE` command-line switch, and the main read-interface is *ixpeobssim.binning.polarization.xBinnedMDPMapCube*.

> **Tip**
>
> *xpbin* provides a `MDPMAP` binning a mode that is simply an alias for a MDP-map cube with a single energy layer—which we refer to as a *MDP map*.

16.3 Livetime-cubes

xpbin provides livetime map cubes in sky coordinates and off axis angle bins; livetime-cubes are generated by *xpbin* run with the `--algorithm LTCUBE` command-line switch, and the main read-interface is *ixpeobssim.binning.exposure.xBinnedLivetimeCube*);

16.4 Miscellanea

xpbin provides additional facilities not strictly related to polarizations—namely:

- binned count maps in sky coordinates (use the `--algorithm CMAP` command-line switch for generation, read interface is *ixpeobssim.binning.misc.xBinnedMap*);
- binned pulse profiles (use the `--algorithm PP` command-line switch for generation, read interface is *ixpeobssim.binning.misc.xBinnedPulseProfile*);
- binned area-rate maps in detector coordinates (use the `--algorithm ARMAP` command-line switch for generation, read interface is `ixpeobssim.binning.misc.xBinnedAreaRateMap`);

- binned energy-flux maps in detector coordinates (use the `--algorithm EFLUX` command-line switch for generation, read interface is `ixpeobssim.binning.misc.xBinnedAreaEnergyFluxMap`);
- binned light curves (use the `--algorithm LC` command-line switch for generation, read interface is *ixpeobssim.binning.misc.xBinnedLightCurve*).

CHAPTER

SEVENTEEN

XSPEC SUPPORT

Since version 2.4.0 ixpeobssim has a limited support for spectro-polarimetric fitting in XSPEC (thanks to Keith Arnaud). This has been somewhat evolving with time and additional contributions (especially as far as XSPEC models are concenrned) are definitely welcome.

Warning

Added in version 20.0.0.

As of December 13, 2021, a few simple, phenomenological multiplicative models for spectro-polarizaion analysis are available through the repository for XPEC additional models at this github repository.

These are essentially the same models that have been shipped with `ixpeobssim` for a long time, except for the fact that they have been renamed (e.g., `constpol` is now `polconst`), and the parameters have been renamed, too.

Since these are the models that, presumably, will be used by the community, and in the spirit of avoiding confusion, we maintain a local copy of these very same models in `ixpeobssim/xspec`, with the intention for them to be inter-operable in a transparent fashion with the files shipped in the XSPEC repository.

Starting from `ixpeobssim` version 20.0.0 you will have to recompile the local models and use the new names (and parameter names).

17.1 Binned Stokes spectra

xpbin supports the creation of binned Stokes spectra for spectro-polarimetric fits in XSPEC through the `PHA1`, `PHA1Q` and `PHA1U` (or their normalized counterparts `PHA1QN` and `PHA1UN`) algorithms. The first produces standard pha files that can be readily used for simple spectral fits in XSPEC in the usual fashion, and corresponds to the I Stokes parameter. The others produce equivalent binned files for Q and U (or Q/I and U/I), in the exact same format.

XSPEC requires the `XFLT0001` keyword of the `SPECTRUM` extension to be properly set for the polarization correction to be applied. This keyword should have the values

- `Stokes:0` for I spectra;
- `Stokes:1` for Q and Q/I spectra;
- `Stokes:2` for U and U/I spectra,

and *xpbin* conforms to this convention, as explained in more details in the section about *Binned data products*.

17.2 Modulation response files

The provisional ixpeobssim caldb provides modulation response files (i.e., the product of the effective area times the modulation function) in all the relevant flavors, to be used for fitting Q and U spectra. They live in the `ixpeobssim/caldb/bcf/mrf` folder and are essentially identical, in format, to the standard .arf files.

Warning

Added in version 12.0.0.

As of version 12.0.0, ixpeobssim provides experimental support for purely polarimetic fits, using binned spectra of the normalized Stokes parameters Q/I and U/I. In this case the proper response file is the modulation factor (as opposed to the modulation response function).

While FITS representations of the modulation factor has been included in the ixpeobssim pseudo-CALDB since the very beginning, a couple of tweaks were necessary to the corresponding data format in version 12.0.0 in order for XSPEC to recognize the file as a response file. (Be advised you might encounter problems if you try to make a purely-polarimetric mixing ixpeobssim and/or IRF versions.)

To make life easier, *xpbin* will automatically replace the path to the .arf file used to run a simulation with the corresponding .mrf file when producing Q or U binned spectra—and with the corresponding modulation-factor FITS file when producing Q/I and U/I spectra. This, in turn, allows XSPEC to pick up the right response file automatically while performing spectro-polarimetric fits.

17.3 XSPEC local models

You will need suitable local models for doing spectro-polarimetric fits in XSPEC. ixpeobssim is equipped with a few, purely phenomenological models (largely courtesy of Keith Arnaud) that everybody can take inspiration from. They are located in the `ixpeobssim/xspec` folder along with the necessary library files:

- `polconst` has constant polarization degree and angle;
- `pollin` has polarization degree and angle scaling linearly with energy;
- `polquad` has been removed in ixpeobssim 20.0.0;
- `polpow` has polarization degree and angle with a power-law dependence on energy.

Note

You will notice that all the models in the table below came in two different flavors—multiplicative and additive (with an `a` prepended). The former incarnation is meant to be multiplied with an arbitrary spectral model in a full spectro-polarimetric fit, while the latter can be used standalone to fit the normalized Q/I and U/I Stokes parameters.

(In this case ixpeobssim will automatically freeze the model normalization to 1 for convenience when the local models are loaded in memory.)

For completeness, here is the full model file, `ixpeobssim_model.dat`, defining the relevant parameters and bounds.

```
polconst       2   0.          1.e20          C_polconst  mul  0 1
A    " "      1.0    0.       0.       1.0      1.0      0.01
psi  deg     45.0   -90.0    -90.0    90.0     90.0     0.01
```

(continues on next page)

(continued from previous page)

```
pollin         4   0.          1.e20          C_pollin   mul  0 1
A1      " "      1.0    0.      0.      1.0      1.0      0.01
Aslope  " "      0.0   -5.0    -5.0     5.0      5.0      0.01
psi1    deg     45.0   -90.0   -90.0    90.0     90.0     0.01
psislope " "     0.0   -5.0    -5.0     5.0      5.0      0.01

polpow         4   0.          1.e20          C_polpow  mul  0 1
Anorm   " "      1.0    0.      0.      1.0      1.0      0.01
Aindex  " "      0.0   -5.0    -5.0     5.0      5.0      0.01
psinorm deg     45.0   -90.0   -90.0    90.0     90.0     0.01
psiindex " "     0.0   -5.0    -5.0     5.0      5.0      0.01

apolconst       2   0.          1.e20          C_polconst  add  0 1
A    " "      1.0    0.      0.      1.0      1.0      0.01
psi  deg     45.0   -90.0   -90.0    90.0    90.0      0.01

apollin         4   0.          1.e20          C_pollin   add  0 1
A1      " "      1.0    0.      0.      1.0      1.0      0.01
Aslope  " "      0.0   -5.0    -5.0     5.0      5.0      0.01
psi1    deg     45.0   -90.0   -90.0    90.0     90.0     0.01
psislope " "     0.0   -5.0    -5.0     5.0      5.0      0.01

apolpow         4   0.          1.e20          C_polpow  add  0 1
Anorm   " "      1.0    0.      0.      1.0      1.0      0.01
Aindex  " "      0.0   -5.0    -5.0     5.0      5.0      0.01
psinorm deg     45.0   -90.0   -90.0    90.0     90.0     0.01
psiindex " "     0.0   -5.0    -5.0     5.0      5.0      0.01
```

Note that the final 1 at the end of the initial line for each model is very important. (If you don't include this then xspec won't calculate the model separately for each spectrum but will assume that it can get away with a single calculation.)

The actual implementation of the `pollin` model, that can be taken as an inspiration for more complex models, reads:

```
// Multiplicative model for polarization modification
// Assumes a polarization fraction and angle with a linear dependence on energy
// parameters:
//   0       A1: polarization fraction at 1 keV
//   1       Aslope: polarization fraction slope
//   2       psi1: polarization angle at 1 keV (degrees)
//   3       psislope: polarization angle slope

//  A(E) = A1 + (E-1)*Aslope
//  psi(E) = psi1 + (E-1)*psislope

#include "xspec_headers.h"

// calcpol is found in calcpol.cxx
void calcpol(const RealArray& energyArray, int spectrumNumber, const RealArray& A,
             const RealArray& psirad, RealArray& fluxArray);
```

(continues on next page)

(continued from previous page)

```
extern "C"
void pollin(const RealArray& energyArray, const RealArray& params,
            int spectrumNumber, RealArray& fluxArray, RealArray& fluxErrArray,
            const string& initString)
{
  size_t nF = energyArray.size() - 1;
  fluxArray.resize(nF);
  fluxErrArray.resize(0);

  RealArray A(nF);
  RealArray psirad(nF);

  A = params[0] + (energyArray-1.0)*params[1];
  psirad = (M_PI / 180.) * (params[2] + (energyArray-1.0)*params[3]);

  calcpol(energyArray, spectrumNumber, A, psirad, fluxArray);

  return;
}
```

And, for completeness, the `calcpol.cxx` utility, that is shipped with ixpeobssim and performs the actual polarimetric correction when necessary, reads:

```
// This calculates the polarization multiplicative correction for the three Stokes
// parameter spectra for general A(E) and psirad(E). psirad is the polarization
// angle in radians.

// This requires that the input spectral files have an XFLT keyword set
// (likely XFLT0001) to 'Stokes:0', 'Stokes:1', 'Stokes:2' for the I, Q, U cases,
// respectively.

#include "xspec_headers.h"

void calcpol(const RealArray& energyArray, int spectrumNumber, const RealArray& A,
             const RealArray& psirad, RealArray& fluxArray)
{
  // find out which Stokes parameter this spectrum is from
  // integer code is 0 = I, 1 = Q, 2 = U

  string xname("Stokes");
  Real xvalue = FunctionUtility::getXFLT(spectrumNumber, xname);

  int Stokes = (int)xvalue;
  if ( xvalue == BADVAL ) {
    Stokes = 0;
    FunctionUtility::xsWrite("Failed to read Stokes parameter from XFLTnnnn keyword -␣
↪applying no polarization correction.", 5);
  }

  fluxArray = 1.0;
  if ( Stokes == 1 ) {
    fluxArray = A * cos (2.0*psirad);
```

(continues on next page)

(continued from previous page)

```
  } else if ( Stokes == 2 ) {
    fluxArray = A * sin (2.0*psirad);
  }

  return;
}
```

17.3.1 Compiling the local models

In order to build the local model library you can invoke the `initpackage` command from within XSPEC

```
XSPEC12> initpackage ixpeobssim ixpeobssim_model.dat ./
```

or from the shell

```
initpackage ixpeobssim ixpeobssim_model.dat ./
hmake
```

(in both cases we assume you're in the `ixpeobssim/xspec` folder). Alternatively you can use the small `compile.py` utility in the same folder, that does pretty much the same thing:

```
./compile.py
```

> **Warning**
>
> It goes without saying that, in order to compile the local models, you will need an XSPEC distribution compiled and installed from the source files, as the ixpeobssim code will need the proper header files in order to compile. In addition, you are advised against moving files around after the installation, as references to absolute paths are kept under the hood.

Local models can be loaded into your interactive XSPEC session through the usual `lmod` command:

```
XSPEC12> lmod ixpeobssim ./
```

Alternatively, if you are working with the Python bindings, ixpeobssim provides a simple facility for doing the very same thing:

```
import ixpeobssim.evt.xspec_ as xspec_

xspec_.load_local_models()
```

If, for some reason, you need to clean up the `xspec` folder and get rid of all the intermediate and compiled files, just type

```
hmake clean
```

or

```
./compile.py --cleanup
```

Warning

You will have to recompile the local models each time you update to a new ixpeobssim release shipping changes in this area. If you get an error message along the lines of

```
***Error: Xspec was unable to load the model package: ixpeobssim
   Either it could not find the model library file in the directory:
/data/work/ixpe/ixpeobssim/ixpeobssim/xspec
   or the file contains errors.
   (try "load (path)/(lib filename)" for more error info)
```

or

```
 ***XSPEC Error:  No model component named polconst
polconst is not a valid model component name.
```

chances are that you need a good cleanup/recompile power cycle.

Note

If you develop more realistic and complex models for spectro-polarimetric fitting, you are welcome to add them in the proper folder so that all the Collaborators will be able to use them.

Keith is open to include such models in future releases of XSPEC, but he would like us to understand what is useful and what is not before doing that.

17.4 Performing a fit

Assuming that you have properly compiled the local model library, all you need is a set of binned Stokes spectra. You can produce such files by running *xpobssim* with your favorite model and *xpbin* in the `PHA1`, `PHA1Q` and `PHA1Q`.

The simplest thing that you can actually do is to run the analysis pipeline [github]/ixpeobssim/examples/toy_point_source.py. This will create all the necessary files for you. At this point you can fire up XSPEC and do something along the lines of:

```
XSPEC12> lmod ixpeobssim ./

XSPEC12> data ~/ixpeobssimdata/toy_point_source_du1_pha1.fits
XSPEC12> data 2 ~/ixpeobssimdata/toy_point_source_du1_pha1q.fits
XSPEC12> data 3 ~/ixpeobssimdata/toy_point_source_du1_pha1u.fits
XSPEC12> ignore 1-3:0.-2. 8.0-**

XSPEC12> model powerlaw*polconst
XSPEC12> fit
```

(For simplicity, this is for a single detector unit, but in real life you would do a combined fit to the data from all the three units.)

If everything goes as intended you should get out a set of fit parameters similar to those in the following table. (You can compare them with the input model.)

```
========================================================================
Model powerlaw<1>*polconst<2> Source No.: 1   Active/On
```

(continues on next page)

(continued from previous page)

```
Model  Model  Component  Parameter  Unit     Value
 par    comp
   1      1   powerlaw   PhoIndex            2.00161      +/-  1.85709E-03
   2      1   powerlaw   norm                10.0119      +/-  1.96276E-02
   3      2   polconst   A                   0.100509     +/-  2.22653E-03
   4      2   polconst   psi        deg      29.9762      +/-  0.634528
_____________________________________________________________________________
```

17.5 Python support

In addition to native support to XPSEC, ixpeobssim provides support for spectro-polarimetric fit through the *xpxspec* application.

xpxspec is wrapped into the ixpeobssim pipeline facilities, and you can look at the [github]/ixpeobssim/examples/toy_point_source.py analysis pipeline for a usage example—but in a nutshell performing a spectro-polarimetic fit is as simple as:

```
import ixpeobssim.core.pipeline as pipeline

# file_list should contain all the PHA1, PHA1Q and PHA1U files
# (i.e., typically 9 files---three per detector unit).
file_list = ['put', 'all', 'the', 'files', 'here']
fit_results = pipeline.xpxspec(*file_list, model='powerlaw * polconst')
print(fit_results)

>>> Fit model: powerlaw*polconst (chi = 1267.77 / 1337)
        PhoIndex: 2.002e+00 +/- 1.857e-03 (+1.859e-03 / -1.857e-03) FFFFFFFFF
            norm: 1.001e+01 +/- 1.963e-02 (+1.967e-02 / -1.961e-02) FFFFFFFFF
               A: 1.005e-01 +/- 2.227e-03 (+2.227e-03 / -2.226e-03) FFFFFFFFF
             psi: 2.998e+01 +/- 6.345e-01 (+6.347e-01 / -6.347e-01) FFFFFFFFF
```

This will simultaneously fit all the nine input files to the product of a power-law spectral model and a constant (i.e., energy-independent) polarization degree and angle—or 4 parameters in total—as illustrated in the following plots.

Alternatively, one can use the normalized Stokes parameters to factor out the spectral part and do a three-parameter purely-polarimetric fit. This will simultaneously fit the six normalized input files to the *additive flavor* of the constant polarization model returning the best-fit polarization degree and angle. (You should notice that the normalization, in this case, is a phony parameter to make the model additive and is set to 1 prior to the binning of the fit process.) It should be noticed that, in this case, the model is only folded with the modulation factor (as opposed to the modulation response function).

> **Warning**
>
> Beware that, statistically speaking, the treatment of the response matrix in fitting the normalized Stokes parameters is non correct, and therefore this should be considered discouraged and might be removed in a future `ixpeobssim` version.

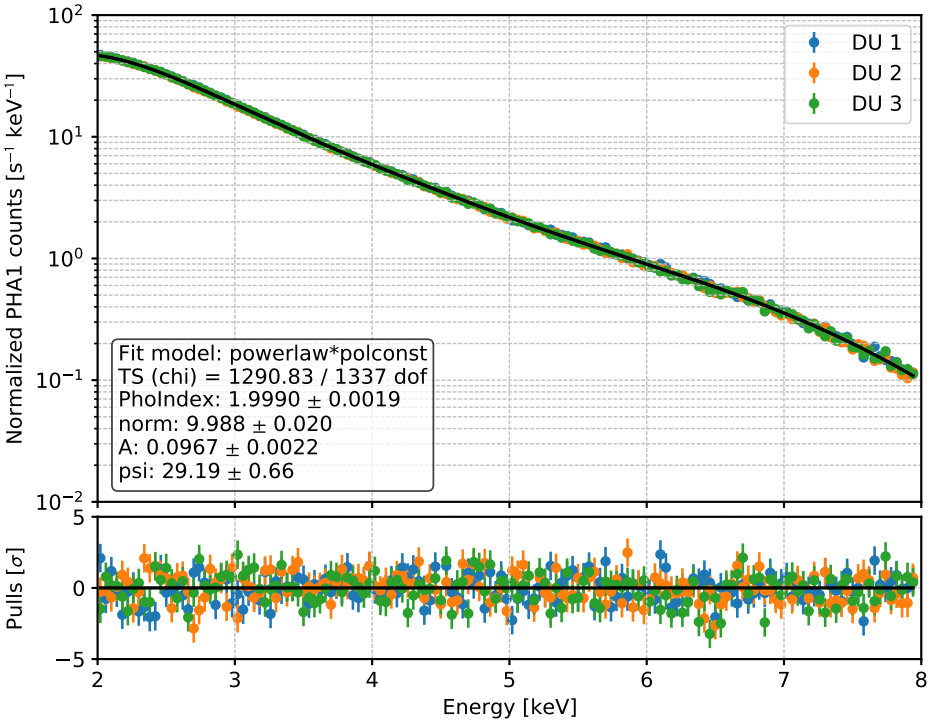


Fig. 1: PHA1 (i.e., Stokes I) counts and best-fit spectral model for a toy example.

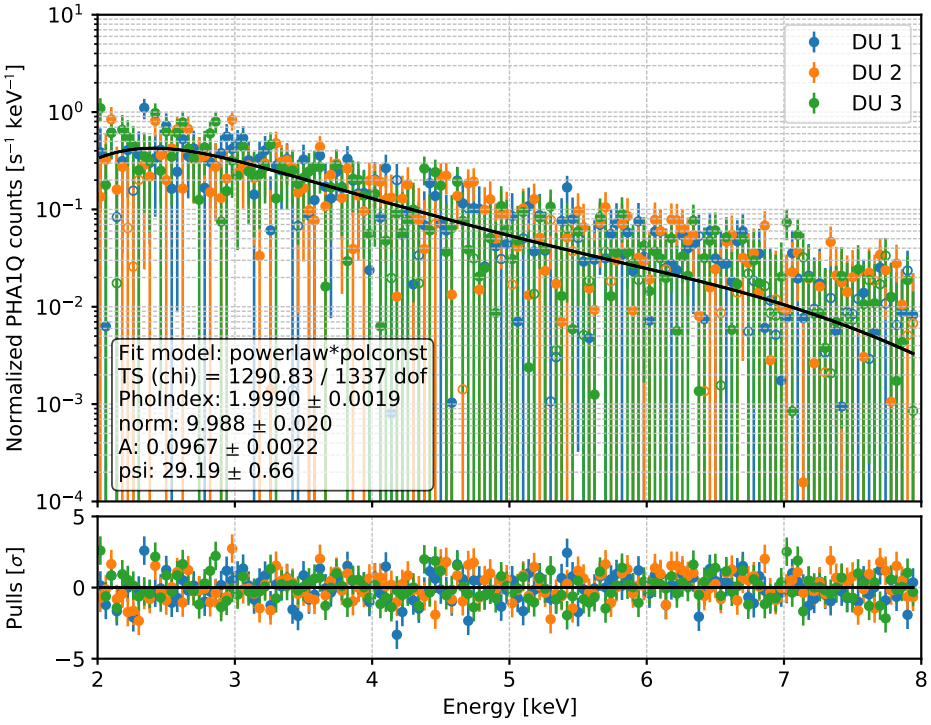


Fig. 2: PHA1Q (i.e., Stokes Q) counts and best-fit spectral model for a toy example.

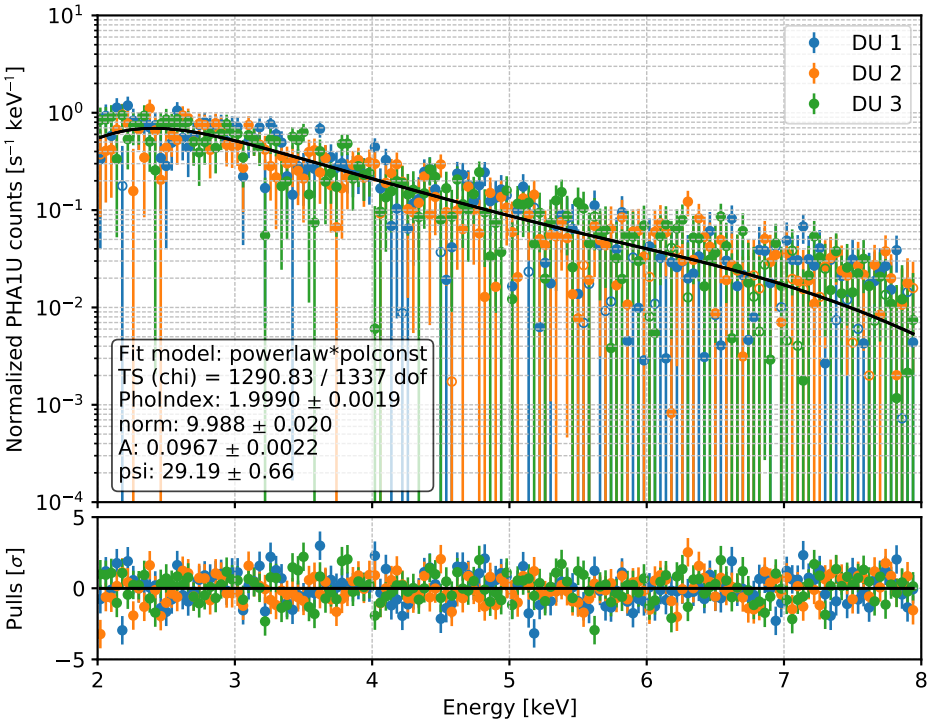


Fig. 3: PHA1U (i.e., Stokes U) counts and best-fit spectral model for a toy example.

CHAPTER

EIGHTEEN

ANALYSIS PIPELINES

One of the ixpeobssim design goals since the very beginning of the code development was to allow the user to develop simulation and analysis pipelines with the minimum possible effort.

The basic technology for developing pipelines is the *ixpeobssim.core.pipeline* module.

18.1 Application wrappers

Each ixpeobssim application, see the *Application reference* page, is wrapped in the *ixpeobssim.core.pipeline* module so that it can be called from within a generic Python script with the exact same arguments that one would pass from command line.

Therefore the shell command

```
xpobssim --configfile config/toy_point_source.py --duration 10000
```

should in principle be *exactly* equivalent to the Python snippet

```
import ixpeobssim.core.pipeline as pipeline

file_list = pipeline.xpobssim(configfile='config/toy_point_source.py', duration=10000)
```

By design all the command-line switches that can be passed when any of the ixpeobssim applications is invoked in the shell can be passed as keyword arguments to the corresponding Python wrapper in *ixpeobssim.core.pipeline*. Easy, isn't it?

18.2 Chaining application calls

ixpeobssim Python wrappers typically return the list of all the files that the function call has created (or, more precisely, a list of strings, each one representing the path to a specific output file). This makes it easy to chain applications one after the other, which is a typical use case:

```
import ixpeobssim.core.pipeline as pipeline

evt_file_list = pipeline.xpobssim(configfile='config/toy_disk.py', duration=10000)
cmap_file_list = pipeline.xpbin(*evt_file_list, algorithm='CMAP')
count_map = xBinnedMap.from_file_list(cmap_file_list)
count_map.plot()
```

While this works as advertised, when dealing with more complex examples it is often the case that one might want to run specific pieces of the pipeline independently from the others. In this case passing around file lists with the mechanism shown in the example above doesn't really make sense, as each step relies on the fact that all the previous ones did run through.

An alternative possibility that the ixpeobssim pipeline framework offers is based on the fact that file paths are typically constructed by adding a suffix to a base name, which in turn coincides with the name of the source model being simluated and anlyzed.

```
import ixpeobssim.core.pipeline as pipeline

pipeline.setup(model='toy_disk')

pipeline.xpobssim(configfile='config/toy_disk.py', duration=10000)

file_list = pipeline.file_list()
pipeline.xpbin(*file_list, algorithm='CMAP')

file_list = pipeline.file_list('cmap')
xBinnedMap.from_file_list(file_list)
count_map.plot()
```

The reader is referred to the [github]/ixpeobssim/examples/toy_periodic_source.py example for a somewhat advanced illustration of the file-list mechanism implemented in the ixpeobssim pipeline.

18.3 Pipeline rc parameters

The ixpeobssim pipeline is implemented as a series of stand-alone methods, and its state is controlled by a look-up dictionary of global parameters that can be effectively used to exchange informations.

Run-commands parameters are set with the `setup()` method and retrieved via the `param()` method

```
import ixpeobssim.core.pipeline as pipeline

pipeline.setup(model=toy_disk)
print(pipeline.param('model'))
```

The `model` rc param plays a peculiar role, in that under normal conditions it can be used as a helper to resolve and create file paths. Once the `model` parameter is specied one can run, e.g., ixpeobssim without specifying the path to the configuration file, assuming you want to use the one in the default folder. The example above can therefore be recasted as

```
import ixpeobssim.core.pipeline as pipeline

pipeline.setup(model='toy_disk')

pipeline.xpobssim(duration=10000)

file_list = pipeline.file_list()
pipeline.xpbin(*file_list, algorithm='CMAP')

file_list = pipeline.file_list('cmap')
```

(continues on next page)

(continued from previous page)

```
xBinnedMap.from_file_list(file_list)
count_map.plot()
```

18.4 Pipelines in action

The *ixpeobssim.core.pipeline* module provides a bootstrap function that should be the preferred way, in practice, to create a simulation and analysis pipeline. A minimal pipeline example will typically look like

```
import ixpeobssim.core.pipeline as pipeline

def run():
    """Do something useful.
    """
    # Put implementation here
    pass

if __name__ == '__main__':
    pipeline.bootstrap_pipeline('toy_model')
```

In this case the boostrap function set the model name for the pipeline, create a custom option parser that allows to control from command line the relevant options, and parse the command-line options.

You can run any of the pipelines in the [github]/ixpeobssim/examples folder with the `--help` option to see what the boostrap function makes available, but in a nutshell this will allow you to

- execute a specific method in your pipeline definition;
- save the plots to file;
- run in batch.

Note that the bootstrap function includes call to the routines showing the plots and saving them—i.e., you should not call the plt.save() or plt.show() methods explicitely.

CHAPTER

NINETEEN

SIMULATION BENCHMARKS

Added in version 11.2.0.

See also

Analysis pipelines

As of version 11.2.0, ixpeobssim provides support for *simulation benchmarks*, or advanced analysis pipelines that are specifically designed to generate a large number of different realizations of the same observation to gauge the statistical properties of a given measurement (e.g., the posterior distribution of the fit parameters).

If you are familiar with the concept of an ixpeobssim analysis pipeline, this is no different, and you can peek at one of the examples, e.g. [github]/ixpeobssim/benchmarks/toy_polconst.py, to take inspiration for your own first benchmark. The rest of this section provides some additional, specific information.

19.1 Generating ensambles

The *ixpeobssim.core.pipeline* module provides a mean to generate a simulation ensamble through the *ixpeobssim.core.pipeline.generate_ensamble()* method. The following minimal snippet, for instance, will generate 1000 different realizations of a 50 ks observation with the `toy_pollin` model.

```
import ixpeobssim.core.pipeline as pipeline

pipeline.bootstrap_pipeline('toy_pollin')
pipeline.generate_ensamble(size=1000., duration=50000.)
```

One (non-obvious) feature of this method is that the processing of the output event lists is embedded in the generation and, by default, the method cleans up after itself by deleting from disk all the (potentially very large) event files, only leaving the processing products (e.g., binned files and/or fit output). This is done because if you are generating a 1000 realization of an observation with many events, you might easily run out of disk space before you realize it—and, after all, once the necessary processing is done you don't need the event files anymore.

By default the pipeline will call *ixpeobssim.core.pipeline.standard_ensamble_processing()* under the hood (which, in turn, will create Stokes spectra and modulation cubes), but you can override this behavior and use your own custom processing function by just doing something along the lines of:

```
import ixpeobssim.core.pipeline as pipeline

def process(file_list):
    """Implement your own processing function here!
```

(continues on next page)

(continued from previous page)

```
    """
    pipeline.xpbin(*file_list, algorithm='PHA1')

pipeline.bootstrap_pipeline('toy_pollin')
pipeline.generate_ensamble(size=1000., duration=50000., processing_function=process)
```

19.2 Post-processing ensambles

Once you have generated and processed all your simulation, you typically want to post-process the binned files, extract the relevant parameters, and store the results for later use (e.g., statistical analysis and/or plotting). As an example, you might want to fit the Stokes spectra with XSPEC with the proper spectro-polarimetric model and store the fit results in a table.

ixpeobssim provides a few facilities for doing just this, including:

- a mechanism for easily retrieving the path to the relevant data product, modeled on that of the simple pipelines;
- a data table class to store and write to FITS the results of the post-processing.

You should look at [github]/ixpeobssim/benchmarks/toy_polconst.py for a real-life example.

19.3 An illustrative use case

The [github]/ixpeobssim/benchmarks/toy_polconst.py benchmark is possibly the easiest toy setup resembling a real observation.

The source model is [github]/ixpeobssim/config/toy_point_source.py is a point source with a power-law spectrum and a constant polarization degree and angle. The benchmark simulates 1000, 50 ks-long observations (mind this is a bright source, and a 50 ks observation time accounts for some 10,000,000 events—so this is a high-statistics benchmark by all metrics).

At the post-processing stage we feed the I, Q and U Stokes spectra into XSPEC and fit them with a `powerlaw * polconst` spectro-polarimetric model. Below are the pulls

$$\frac{\hat{p} - p_{\rm true}}{\sigma_p}$$

for all the four fit parameters—the spectral index and normalization, and the polarization degree and angle. The fact that they are all reasonably distributed as standard normal variables, with zero mean and unit standard deviation, is a sensible sanity check for the entire analysis chain.

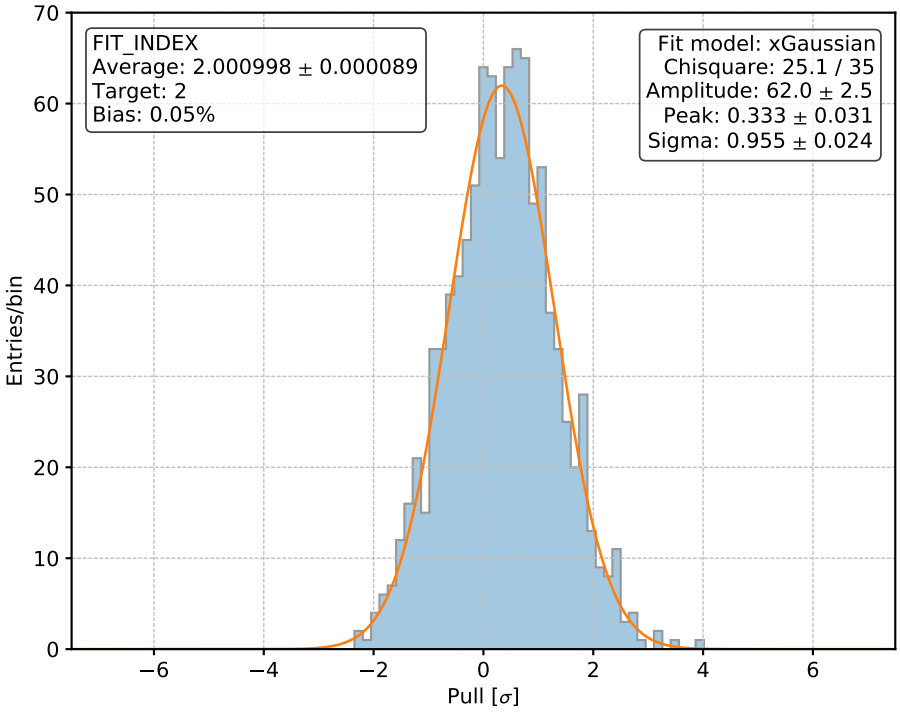


Fig. 1: Posterior distribution of the pulls for the fitted spectral index over 1000 realization of the [github]/ixpeobssim/benchmarks/toy_polconst.py benchmark.

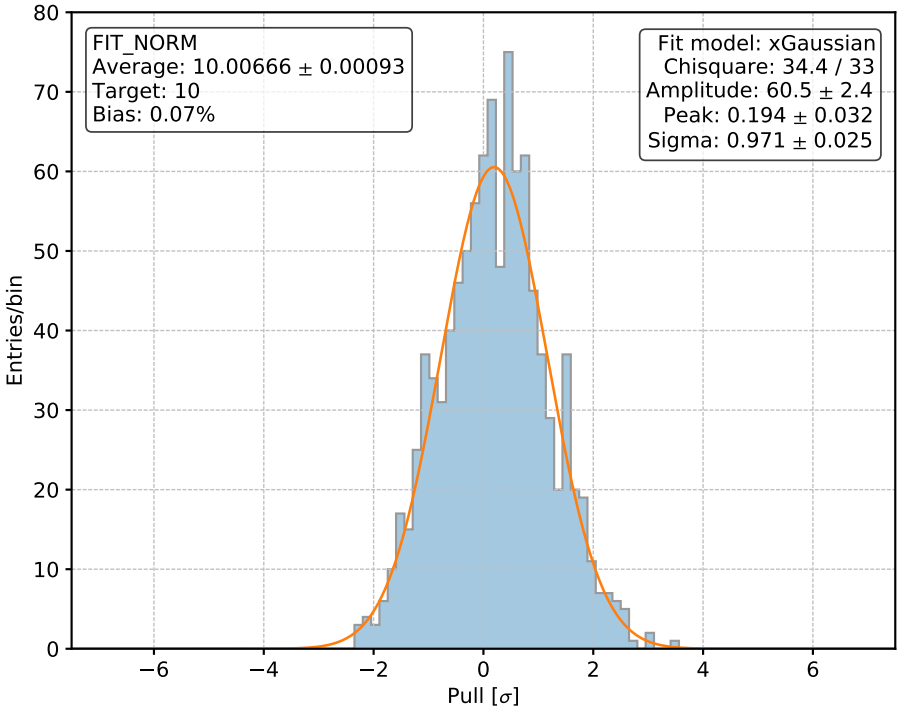


Fig. 2: Posterior distribution of the pulls for the fitted spectral normalization over 1000 realization of the [github]/ixpeobssim/benchmarks/toy_polconst.py benchmark.

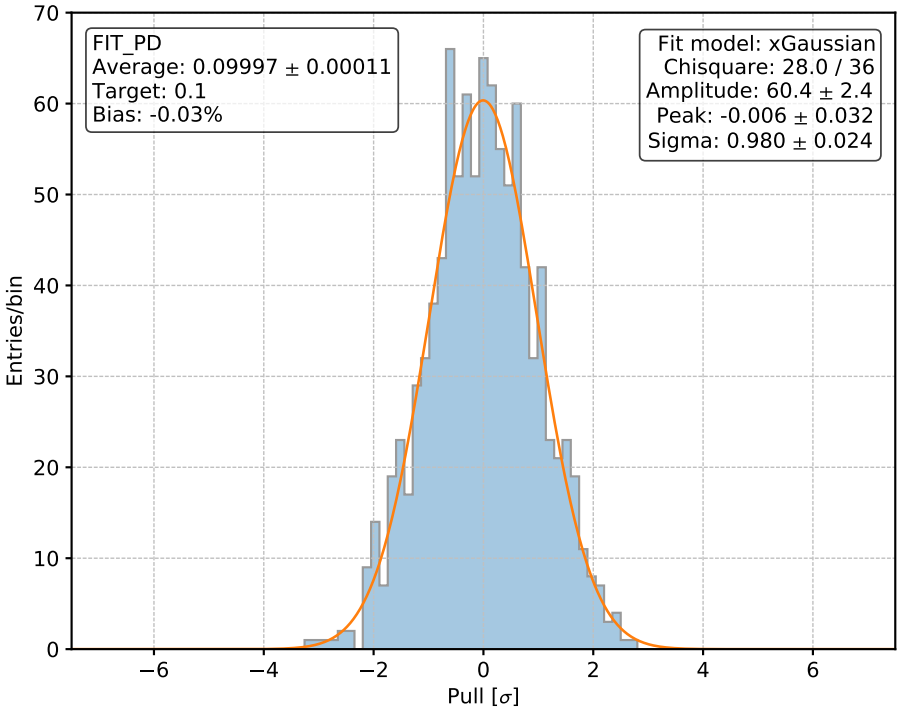


Fig. 3: Posterior distribution of the pulls for the fitted polarization degree over 1000 realization of the [github]/ixpeobssim/benchmarks/toy_polconst.py benchmark.

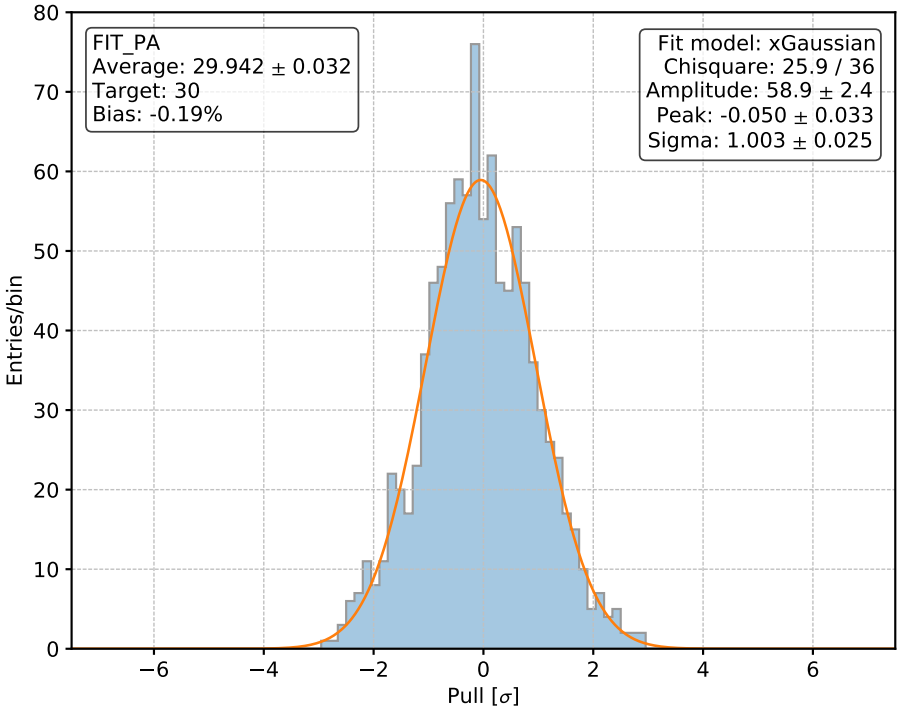


Fig. 4: Posterior distribution of the pulls for the fitted polarization angle over 1000 realization of the [github]/ixpeobssim/benchmarks/toy_polconst.py benchmark.

CHAPTER

TWENTY

BROADBAND POLARIZATION

Estimating the broadband polarization degree and angle, or Stokes parameters, from a (real or simulated) observation is a fairly common task—possibly the most common one. This section is devoted to a succinct discussion of the various means ixpeobssim is offering for the job and the associated merits and pitfalls.

Warning

The section is largely focused on point sources, at the moment. Updates for extended sources are on their way.

(For completeness, by the term *broadband*, in this context, we mean *averaged over an energy range much larger than the intrinsic binning of the underlying response functions*.)

Note

This is a slippery business, and mileage might vary considerably depending on your precise analysis setup, so take the whole discussion in this section *cum grano salis* and make an effort to gauge its validity to the problem at hand.

To make a long story short, going through the effort of a full spectro-polarimetric fit with a proper model is likely to give you the right answer, but there are simpler and faster ways around that might bring you close enough in specific circumstances.

20.1 The ixpeobssim toolbox

ixpeobssim offers several different tools designed to estimate the broadband polarization, given a simulated photon list. We briefly recap the laundry list, here; more details can be found in the chapters about *Binned data products* and *XSPEC support*.

20.1.1 Fitting with XSPEC

Whenever practically possible, one should fit the I, Q and U Stokes spectra with a proper spectro-polarimetric model (e.g., in XSPEC). This is properly taking into account both the effective area and the energy dispersion, and is supposed to provide the ultimate statistical precision—at least for point sources.

Fitting with a *constant* polarimetric multiplicative model (i.e., `polconst` in the ixpeobssim language) will often yield sensible results in terms of the average broadband polarization degree, but it is not guaranteed to be unbiased.

If a suitable spectral model, for any reason, is not available, fitting the normalized Stokes parameters Q/I and U/I with an additive polarimetric model is a possible alternative, although it is not guaranteed to be unbiased, either—the reason being that, strictly speaking, the detector response matrix is not directly applicable to the normalized Stokes spectra.

See also

XSPEC support.

20.1.2 Polarization cubes

Polarization cubes are a legitimate, model-independent, alternative to a fully fledged forward-folding fit. The automatically incorporate the detector acceptance correction, but, by their very nature, cannot handle the energy dispersion. Whether this is an important effect or not depends very much on the setup under study, but the good news is that one can gauge the magnitude of any possible bias by comparing polarization cubes binned in measured and true energy. *Polarization cubes binned in true energy are supposed to provide the right answer*.

(It goes without saying that the true energy is not available in real life, but still this is a perfectly valid thing to do for sensitivity studies and/or debugging purposes.)

See also

Binned data products.

20.1.3 Modulation cubes

In a nutshell: *do not use them*. Modulation cubes perform the relevant sums and/or averages over the count spectrum (as opposed to a proper proxy of the input source spectrum), and do not handle neither the effective area nor the energy dispersion of the detector. Binning in true energy will save you from the latter, but not from the former, which is typically a bigger effect.

Modulation cubes are retained in ixpeobssim, mainly for diagnostic purposes, but their use for science analysis is deprecated.

20.2 A toy case study

In the spirit of substantiate the somewhat generic statements in the previous section, the following plot summarizes the performance of the various methods for retrieving the broadband polarization (between 2 and 8 keV) for a toy setup in which the polarization degree increases linearly with energy and the polarization angle is constant. (For completeness, the spectrum was a simple power law.)

We generated 1000 independent, high-statistics (but unrealistic) realizations of the very same model and, for each single one we applied all the methods described in the previous section.

For completeness, the model source file is at [github]/ixpeobssim/config/toy_pollin.py and the full benchmark code at [github]/ixpeobssim/benchmarks/toy_pollin.py.

A few comments are in order:

- The full XSPEC fit (with a proper model) to the I, Q and U spectra provides the right answer; fitting the normalized Q/I and U/I Stokes parameters to the polarimetric part of the model features a measurable negative bias, as does a fit with a constant polarization model;
- a polarization cube with a single energy bin between 2 and 8 keV features a small, but measurable, negative bias—which is completely recovered when binning in true energy;
- the corresponds single-bin modulation cube is way off in both flavors.

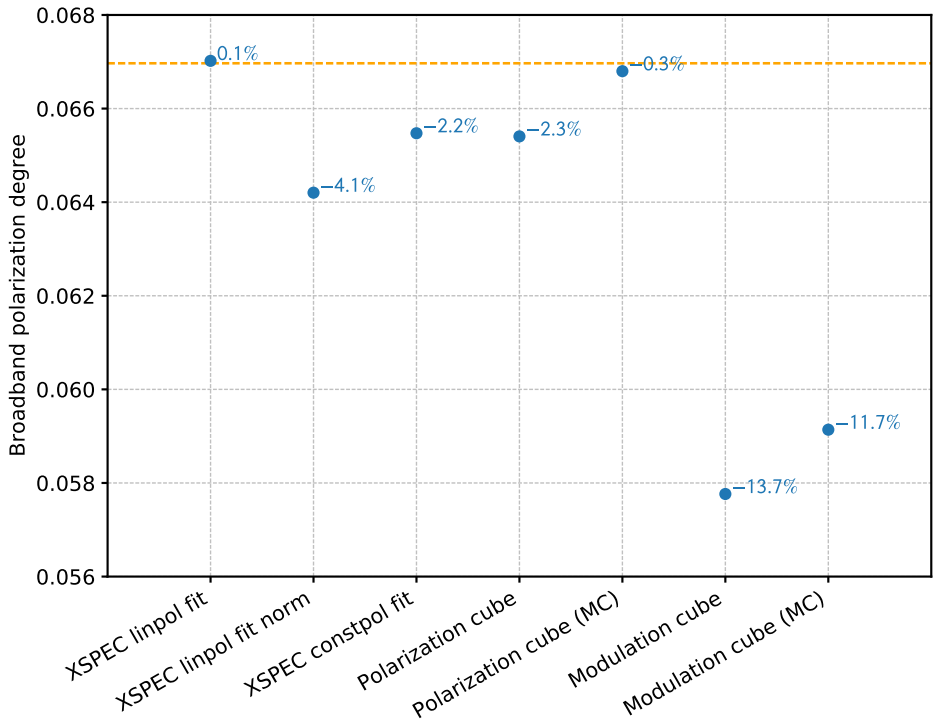


Fig. 1: Average (over 1000 independent realizations of the same model) of the broadband 2–8 keV polarization, estimated by various means. The orange dashed line represents the expectation from the input model.

CHAPTER

TWENTYONE

INTER-OPERABILITY

This section covers the inter-operability between the `ixpeobssim` modules and data products and the standard tools of the X-ray community.

Warning

This section is a stub and needs to be expanded.

21.1 Spectro-polarimetry

21.1.1 XSPEC

XSPEC is the primary benchmark for the `ixpeobssim` tools and data product, and the relevant interfaces are covered in the section about *XSPEC support*.

21.1.2 Sherpa

Sherpa is the CIAO modeling and fitting application. Among other things, it supports the XSPEC spectral models and provides a more modern (and Pythonic) interface to the underlying facilities.

The inter-operation between `ixpeobssim` and Sherpa has been demonstrated at the prototypical level, and the related activity is tracked at issue #291.

21.1.3 3ML

The 3ML Multi-Mission Maximum Likelihood framework provides a common high-level interface and model definition, interfacing under the hood to the official software of the various missions.

3ML provides experimental support to IXPE, which is illustrated, e.g., here.

21.2 Timing

21.2.1 HENDRICS

HENDRICS, the High-Energy Data Reduction Interface from the Command Shell, is sophisticated timing analysis package for X-ray astronomy.

HENDRICS has been tested against `ixpeobssim` simulations, see, e.g., here.

CHAPTER

TWENTYTWO

MAGNETAR MODELS

Roberto and Roberto kindly provided the magnetar models described in Taverna et al. 2020 in a tabular form suitable for fitting with XSPEC. The data files are available on our auxiliary file repository and, as of version 14.0.0, ixpeobssim provides the necessary infrastructure to read and use them.

See also

Large data files.

22.1 Model description

The models come as three separate FITS files for the I, Q, and U Stokes parameters, the basic structure of each one being of the form

```
No.    Name      Ver    Type      Cards   Dimensions   Format
0  PRIMARY       1 PrimaryHDU      15   ()
1  PARAMETERS    1 BinTableHDU     47   6R x 10C   [12A, J, E, E, E, E, E, E, J, PE(13)]
2  ENERGIES      1 BinTableHDU     21   49R x 2C   [D, D]
3  SPECTRA       1 BinTableHDU     21   314496R x 2C   [6E, 49E]
```

(The actual dimensions of the cards might change, but the very fundamental structure of the files is fixed by the OGIP standard.)

The `PARAMETERS` binary table encapsulate the following model parameters:

- *Model*: the underlying Physical model, i.e., `ATMOSPHERE`, `BLACKBODY`, `SOLID_SURFACE_FREE_IONS` and `SOLID_SURFACE_FIXED_IONS`.
- *chi*: angle between the line of sight and the rotation angle of the star.
- *xi*: angle between the magnetic axis ans the rotation angle of the star.
- *delta_phi*: twist angle.
- *beta*: electron velocity in units of c.
- *phase*: the phase bins.

In the initial implementation we have 4 physical models x 13 chi x 8 xi x 12 delta_phi x 7 beta x 9 phase bins, that is, 314496 combinations. Spelling things out explicitly:

```
Model     -> [1, 2, 3, 4]
chi       -> [0.1, 15, 30, 45, 60, 75, 89.9, 105, 120, 135, 150, 165, 179.9]
xi        -> [0.1, 15, 30, 45, 60, 75, 85, 89.9]
delta_phi -> [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4]
beta      -> [0.2, 0.3, 0.34, 0.4, 0.5, 0.6, 0.7]
phase     -> [0.350066, 1.0482, 1.74633, 2.44446, 3.14259,
              3.84072, 4.53886, 5.23699, 5.93512]
```

The `ENERGY` binary extensions encapsulates the energy binning, and in the initial implementations has 49 bins between about 0.5 keV and 10 keV.

Warning

Some of the models can be noisy at high energy, and the fact that tables are limited to less than 10 keV might result in unstable behaviors if you actually try and extrapolate to higher energies. While this should not be an issue for a standard in the nominal 2–8 keV energy band, be aware of this potential issue.

The `SPECTRA` extension contains the spectra for all the possible configurations. It comes with two columns, the first of which contains the 6 model parameters and the second the 49 spectral values. Note the `SPECTRA` extension is filled by looping over the model parameters in reverse order, i.e., from *phase* to *Model*.

22.1.1 Parametrization details

Note

This section contains some technical details about how we parse, store, and manipulate the data in the input FITS tabular model. You can probably skip at a first reading, but if you do run into issues, this might turn out to be useful for debugging.

All the underlying model quantities are binned in 49 bins in energy between 0.5 and 10 keV and 9 bins in phase between 0 and 2 pi, with the corresponding values in the energy grids where the Stokes spectra are sampled representing the centers of the bins. As it turns out, the first and the last energy bins act as underflow/overflow accumulators in the underlying Monte Carlo simulation, and therefore the corresponding spectral and polarimetric quantities are un-phyisical and need to be discarded. This leaves us with I, Q and U tabulated on the following grids.

```
# Energy grid in keV.
[0.549539  0.5841565 0.6209545 0.6600705 0.701651  0.74585   0.7928335
 0.842777  0.8958665 0.9523005 1.0122895 1.076055  1.14384   1.215895
 1.29249   1.37391   1.460455  1.552455  1.65025   1.754205  1.864705
 1.98217   2.10704   2.23977   2.38086   2.530835  2.69026   2.85973
 3.03987   3.231365  3.434925  3.6513    3.88131   4.125805  4.3857
 4.661975  4.95565   5.267825  5.59966   5.9524    6.32737   6.72595
 7.14964   7.60002   8.07877   8.587685  9.12865  ]

# Phase grid divided by 2pi.
[0.05571473  0.1668262   0.27793705  0.38904786 0.50015873 0.61126953
 0.72238195  0.83349282  0.94460368]
```

The first slight complication is that if we create an interpolated bi-variate spline on this grid, since technically we are *extrapolating* at phases 0 and 1, there is no guarantee that the two values will be identical, or even close—i.e., in general

we will have discontinuities when the phase roll over. In order to fix this we add two values to both sides of the energy grid, namely

$$\max(\phi) - 1 \quad \text{and} \quad 0$$

on the left side, and

$$1 \quad \text{and} \quad 1 + \min(\phi).$$

In addition, we impose boundary conditions

$$I(\max(\phi) - 1) = I(\max(\phi)) \quad I(1 + \min(\phi)) = I(\min(\phi)) \quad I(0) = I(1) = \frac{1}{2}\left(I(\min(\phi)) + I(\max(\phi))\right)$$

(and the equivalent for Q and U) such that, by using a spline of order 2 for the interpolation, we are guaranteed that there are no discontinuities when the phase rolls over.

The other issue is that the energy grid is effectively limited to just above 9 keV on the right end, while ixpeobssim, by default, simulates photons between 1 and 12 keV in energy. While this is not a major concern, as photons between 9 and 12 keV are relatively sparse and, even when the energy dispersion is taken into account, they hardly have any effect on any analysis in the 2–8 keV energy range, we need a sensible way to extrapolate the spectra. We accomplish this internally at the stage where the model table is interpolated in the parameter space to create 2-dimensional splines in the phase-energy space to be used for the simulation. More precisely, this is a two-step process where:

- I is fitted with a power law between 6 and 9 keV (adjustable) and the fit model is used to extrapolate at high energy;
- the reduced Stokes parameters Q/I and U/I are fitted with a straight line in the same interval, and the linear model is used for the extrapolation.

22.2 Simulation interface

ixpeobssim provides the *ixpeobssim.srcmodel.magnetar.xMagnetarModelsT2020* class as the basic interface the magnetar table models described above. Instantiating the model table takes one Python line.

```
from ixpeobssim.srcmodel.magnetar import xMagnetarTableModelT2020

# Load the table model. If you need to download any of the large auxiliary
# FITS files, you should be prompted with detailed instructions (and the program
# will exit).
table = xMagnetarTableModelT2020()
```

This will load all the input FITS files, make some integrity check on the underlying binary tables, and cache the necessary arrays for later use. At this point you have the full I, Q and U spectra for the entire range of the model parameters and the of the magnetar rotational phase.

At simulation time, you typically want to calculate the actual spectro-polarimetric model for a given set of input parameters, in a form that can be fed directly into the simulation code. ixpeobssim provides two main interfaces, allowing to find the *nearest* model to a set of input parameters, or to *interpolate* the underlying model table to the aforementioned parameters. Typically you want to do the second, and the following snippet should be fairly self-explanatory.

```
from ixpeobssim.srcmodel.magnetar import xMagnetarModelsT2020, xMagnetarTableModelT2020


# Source name and position.
```

(continues on next page)

(continued from previous page)

```
source_name = 'AXP 1RXS J1708'
ra, dec = 257.2042, -40.1528

# Source period.
nu0 = 0.09089812328

# Magnetar model parameters.
model = xMagnetarModelsT2020.BLACKBODY
chi = 89.9
xi = 60.
delta_phi = 0.5
beta = 0.34

# Phase-averaged integral flux between 2 and 10 keV, in erg/cm2/s.
flux = 3.5e-11

# Load the magnetar model table.
table = xMagnetarTableModelT2020()

# Interpolate the model table to the target parameters.
spec, pol_deg, pol_ang = table.interpolate(model, chi, xi, delta_phi, beta, flux)

# Define the actual ROI model.
ROI_MODEL = xROIModel(ra, dec)
ephem = xEphemeris(0., nu0)
src = xPeriodicPointSource(source_name, ra, dec, spec, pol_deg, pol_ang, ephem)
ROI_MODEL.add_source(src)
```

Note

For completeness, a full model table with the QED effects switched off is available for debugging and illustration purposes. The usage is exactly the same, with the class named `xMagnetarTableModelT2020DedOff`.

CHAPTER

TWENTYTHREE

EXTENDED SOURCES

This page includes some basic information on the facility that `ixpeobssim` provides for the analysis of extended sources.

Warning

This section is a stub, and should be expanded to cover the classes for the simulation and analysis of extended sources.

23.1 Large-scale polarization signatures

`ixpeobssim` facilitates the search for large-scale polarization signatures in extended sources through the *xpstokesalign* application, which takes a photon list as an input and aligns the reconstructed Stokes parameters to a given polarization model, overwriting the relevant columns.

Note

Added in version 20.1.0.

The alignment of the polarization of the input events with a space-dependent model has been recast in Stokes-parameter space, as opposed to the original implementation that operated on the photo-electron azimuthal angles. Accordingly, the associated tool has been renamed from `xpphialign` to `xpstokesalign`.

xpstokesalign processes a photon list *rotating* the Stokes parameters so that, on an event-by-event basis, the zero for the measurement of the polarization is aligned to a given input model at the position of the event

$$q \rightarrow \frac{1}{2}\left(qq_0 + uu_0\right)$$
$$u \rightarrow \frac{1}{2}\left(uq_0 - qu_0\right)$$

In the simplest form the alignment can be either radial or tangential, which is achieved by selecting the RAD or TAN modes, respectively, and optionally passing the right ascension and the declination of the center for the rotation via the –ra and –dec command-line switches.

Additionally, the user can feed into the application pairs of FITS images (in either the Q/U, X/Y components of the polarization vector or polarization degree/angle) in the same exact fashion of the machinery used for simulating complex polarization patterns for extended sources in ixpeobssim. This is achieved via a combination of the QU, XY of PDA modes, along with the proper model files passed to the –modelfiles command-line switch.

The supported alignment algorithms are:

- RAD: radial
- TAN: tangential
- QU : generic polarization model (from FITS maps of U and Q)
- XY : generic polarization model (from FITS maps of polarization components)
- PDA: generic polarization angle model (from a single FITS map)

CHAPTER

TWENTYFOUR

BINARY SYSTEMS

The class *ixpeobssim.srcmodel.roi.xBinarySource* allows to simulate a point-like periodic source in a binary system. The basic ingredients to describe the system are the same as in the simple periodic source configuration but here the input can be in the more generic form of a parameter file including the system ephemeris. The spin and orbital ephemeris are handled by the `ixpeobssim.srcmodel.roi.xOrbitalEphemeris` class.

The `toy_binary` configuration file distributed with `ixpeobssim` describes a pulsar in a binary system with a power-law photon spectrum with normalization that varies as a function of the pulse phase. To illustrate the flexibility of the code, the polarization degree also varies as a function of both energy and pulsar phase.

```
import os
import numpy

from ixpeobssim.config import file_path_to_model_name
from ixpeobssim.utils.matplotlib_ import plt, setup_gca
from ixpeobssim.srcmodel.roi import xROIModel, xBinarySource
from ixpeobssim.srcmodel.spectrum import power_law
from ixpeobssim.srcmodel.polarization import constant
from ixpeobssim.utils.fmtaxis import fmtaxis
from ixpeobssim import IXPEOBSSIM_SRCMODEL
from ixpeobssim.srcmodel.ephemeris import xOrbitalEphemeris

__model__ = file_path_to_model_name(__file__)
ra, dec = 18.14, -35.02

def pl_norm(phase):
    """Energy spectrum: power-law normalization as a function of the pulse
    phase.
    """
    return 1.25 + numpy.cos(4 * numpy.pi * phase)

pl_index = 2.
spec = power_law(pl_norm, pl_index)

def pol_deg(E, phase, ra=None, dec=None):
    """Polarization degree as a function of the dynamical variables.

    Since we're dealing with a point source the sky direction (ra, dec) is
    irrelevant and, as they are not used, defaulting the corresponding arguments
    to None allows to call the function passing the energy and phase only.
    """
```

(continues on next page)

(continued from previous page)

```
    norm = numpy.clip(E / 10., 0., 1.)
    return norm * (0.5 + 0.25 * numpy.cos(4 * numpy.pi * (phase - 0.25)))

pol_ang = constant(numpy.radians(65.))
file_path = os.path.join(IXPEOBSSIM_SRCMODEL, 'parfiles', 'SAXJ1808.4-3658.par')
ephemeris = xOrbitalEphemeris.from_file(file_path)

src = xBinarySource('Periodic source', ra, dec, spec, pol_deg, pol_ang,
                            ephemeris)

ROI_MODEL = xROIModel(ra, dec, src)

def display(emin=1., emax=12.):
    """Display the source model.
    """
    energy = numpy.linspace(emin, emax, 100)
    phase = numpy.linspace(0., 1., 100)

    # Pulse profile: power-law normalization.
    plt.figure('%s PL normalization' % __model__)
    plt.plot(phase, pl_norm(phase))
    setup_gca(ymin=0., ymax=2.5, **fmtaxis.pp_pl_norm)

    # Pulse profile: polarization degree.
    plt.figure('%s polarization degree' % __model__)
    for E in [2., 5., 8.]:
        plt.plot(phase, pol_deg(E, phase), label='Energy = %.2f keV' % E)
    setup_gca(ymin=0., ymax=1., legend=True, **fmtaxis.pp_pol_deg)

    # Energy spectrum at different phase values.
    plt.figure('%s spectrum' % __model__)
    for p in [0.00, 0.05, 0.10, 0.15, 0.20, 0.25]:
        plt.plot(energy, spec(energy, p), label='Phase = %.2f' % p)
    setup_gca(xmin=emin, xmax=emax, logx=True, logy=True, legend=True,
              grids=True, **fmtaxis.spec)


if __name__ == '__main__':
    from ixpeobssim.config import bootstrap_display
    bootstrap_display()
```

CHAPTER

TWENTYFIVE

EVENT DISPLAY

Added in version 29.4.0: ixpeobssim includes a single event display to render the track images shipped with publicly distributed the level-1 files.

As of version 29.4.0 ixpeobssim include a new small application, *xpevtdisplay*, that allows to display the track images shipped with the standard IXPE level-1 publicily distributed through HEASARC.

While the help coming with the program should be largely self-explaining, a few remarks are in order, here. First of all, *xpevtdisplay* operates on level-1 files, which are the only ones containing the track images by default, and whom most user are probably not terribly familiar with. In constrast to the more widely used level-2 files, level-1 files are significantly larger (which means: do not be surprised if firing up the event display on a particular file takes a few seconds), and contain all the event recorded through the observation—no matter whether they correspond to actual GTI, or to periods when the source was occulted by the Earth (and, possibly, with one of the calibration sources in use). As a consequence, just firing up the event display on a level-1 file

```
xpevtdisplay path_to_my_level1_file
```

is not terribly useful, in general: you have no control on which events you are actually displaying—that is, they might very well be from one of the onboard calibration sources, rather than from the celestial source being observed.

For this very reason, *xpevtdisplay* features a `--evtlist` command line flag that allows to feed into the application a level-2 file *in addition* to the level-1 main file (it goes without saying, the two generally need to correspond to the same observation).

```
xpevtdisplay path_to_my_level1_file --evtlist path_to_the_fellow_level2_file
```

When you pass this additional file to the event display, a few things happen behind the scenes, namely:

- the display will loop over the level-2 file, one event at a time, and retrieve all the relevant high-level information (time, energy, sky position and Stokes parameters);
- for each event, a binary search is run over the level-1 file to identify the corresponding raw event data;
- the actual event in the level-1 file gets displayed, along with all the high-level info from the level-2 data.

This is where things get interesting: since you can run *xpeselect* natively over any level-2 file, this provides a convenient mechanism to select small subsamples of event (e.g., in energy or sky position) and get them displayed.

Now, when you start looking at track images, you will probably get bored quite quickly: as the IXPE effective area is sharply peaked around 2.2 keV, most of the events will probably look quite similar to each other. This is fine if you are interested in an unbiased sample of tracks, but it will take you lots of luck to stumble across a high-energy track such as the one shown at the top of the page. For this reason, the `--resample` command-line swicht provides a mechanism to resample in energy the input level-2 file using a power law with the specified index—if you use, e.g., `--resample 3` you should see a large variety of event topologies.

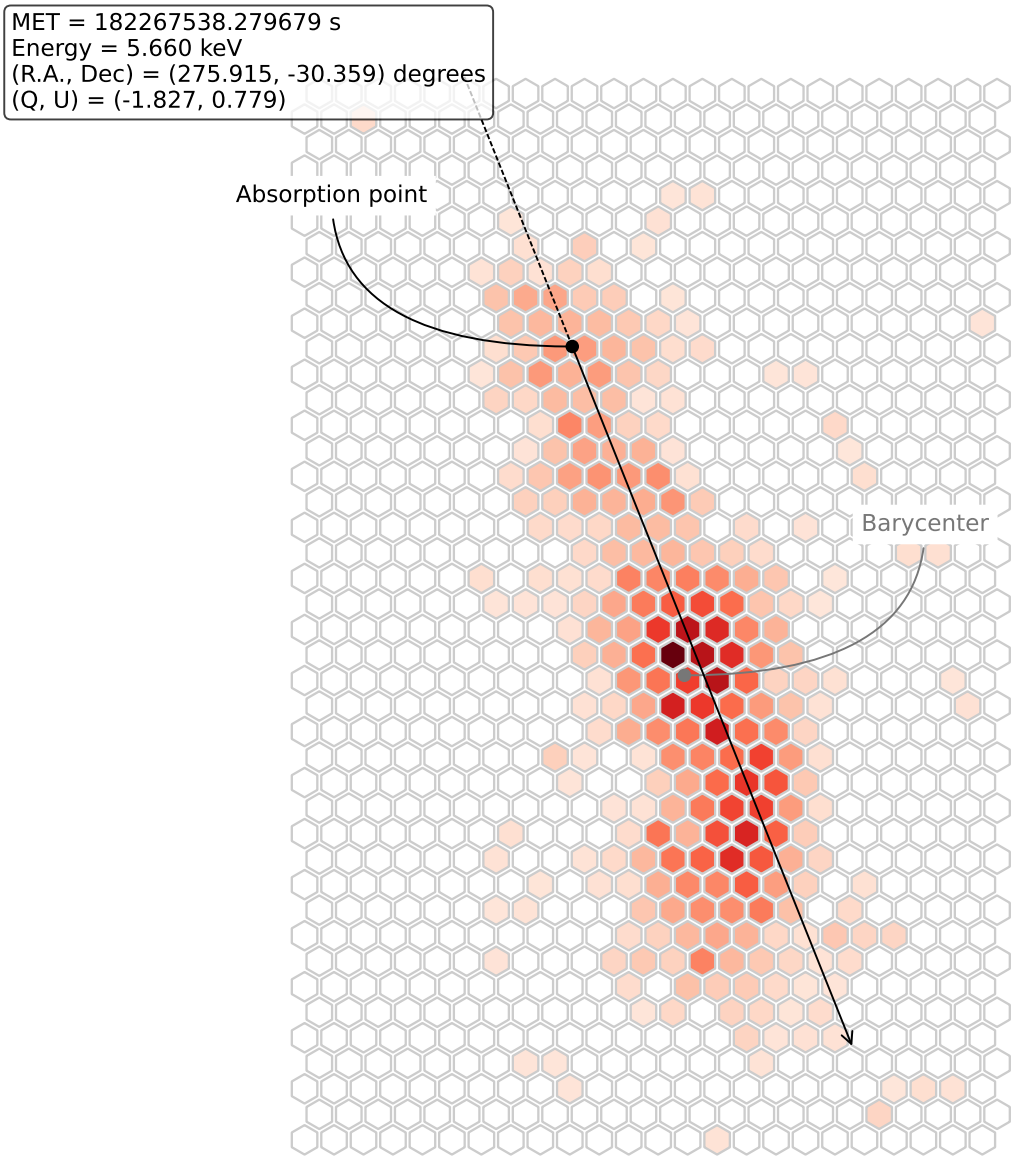


Fig. 1: Sample display of a track image—note that mileage may vary depending on the setting (i.e., command-line switches) used in *xpevtdisplay*.

Warning

Be mindful that, since the event time is the only quantity that we can use to keep in synch level-2 and level-1 data, the event list functionality does not play well with any analysis tool (e.g., the barycorr FTOOL) that change the event time—for the purpose of the event display you want to use event file as distributed, without further processing.

Also, when reading a Level-1 files and passing at the same time an event list in the form of a Level-2 file, you should make sure that the former covers the entire time span of the latter, otherwise the MET bisection will fail. If the Level-2 file covers a longer time span wrt. the Level-1 file, you can simply trim the former with the standard tools.

25.1 Observation animations

Building on top of the single event display, ixpeobssim provides a separate application, *xpobsdisplay*, that allows to create complex animations starting from the standard observation products distributed through HEASARC by jut adding to the track images accumulating histograms of the relevant science quantities: energy, sky position, and Stokes parameters.

The *xpobsdisplay* is fairly similar to that of *xpevtdisplay*. We won't describe the meaning of all the command-line switches here, but the following command

```
xpobsdisplay path_to_my_level1_file --evtlist path_to_the_fellow_level2_file \
    --resample 2 --autostop 200 --targetname "The name of the source" --autosave True \
    --imgdpi 200 --batch True
```

will read in the target files and write to file 200 still track images in batch mode (i.e., no matplotlib canvas will be displayed on the screen).

The user can then combine the still frames into an actual animation by using any of the countless tools available on the market, one possible example being

```
ffmpeg -framerate 1 -i path_to_frames_%04d.png -c:v libx264  -s 1920x1080 -r 30 \
    -pix_fmt yuv420p animation.mp4
```

For completeness, a succinct and yet informative resource about using ffmpeg to create slideshows is here.

CHAPTER

TWENTYSIX

SONIFICATION

Surprisingly enough, `ixpeobssim` comes with built-in sonification facilities. The *xpsonify* application allows to tranform a given event list (in the form of a level-2 FITS file) into a midi and/or wave file.

More precisely, the event list is transformed into a MIDI file, which can in turn be played with any MIDI sequencer, or tranformed into an audio file using a virtual (or real, for what it's worth) synthetizer.

Added in version 19.5.0: First version of the sonification facilities.

Added in version 19.6.0: Minimal support for audio-video animations added.

26.1 Manifesto: listening to the X-ray Universe in stereo

The ideas behind the IXPE-specific sonification facilities are hardly new. It is customary to transform particles in notes by translating the energy into a frequency—for photons this is really straightforward, as there is a perfect analogy between electromagnetic and acoustic waves.

When examined from the standpoint of its spectral response, IXPE is peculiar in that the standard band-pass for scientific analysis (2–8 keV) corresponds to exactly 2 octaves, comparable to the typical vocal range of a singer. The *extended* band in which we define the instrument response (1–12 keV) is in slight excess of 3.5 octaves, which is pretty much the same as the range of an acoustic guitar. Phrased in a slightly different way, the IXPE *dynamics* is comparable with that of typical musical instruments, which makes the conversion between energy and pitch very natural.

When translating X-ray photons into musical notes, pitch is not the only thing that we care about. Notes comes with at least three additional properties that need to be defined:

- velocity: a measure of how vigorously the player initially presses a key on the keyboard, to use a piano analogy—the note is louder when the key is struck more forcefully (and there are other subtle tonal differences, as well);
- duration: the time difference between the key press and release;
- pan: the location of the sound within the stereo field of the instrument (e.g., left, right, center).

The pan is where IXPE really shines. The IXPE polarization-sensitive detectors are unique in that they not only provide a time, energy and direction in the sky for each photon—they also provide a proxy of the photon polarization in the form of the azimuthal angle of emission of the photoelectron. Being an angle in the plane, this can be mapped in a straightforward way into a point in a stereo field, i.e., a position in the left-center-right space. This is something that has essentially no parallel in the vast majority of previously flown X-ray detectors.

In a sense, IXPE allows us to listen to the Universe in stereo for the first time.

26.2 Installation

Working with audio files requires a couple more Python packages on top of the standard `ixpeobssim` dependencies. Since this feature is not of general interest, these packages are not covered in the installations instructions, and are not included in the requirement file. If you try and run the sonification code without the relevant dependencies, you should get sensible error messages pointing you in the right direction, but essentially everything boils down to two things:

- mido, a MIDI manipulation Python library;
- fluidsynth, a cross-platform software synthesizer based on the SoundFont 2 specifications.

> **Note**
>
> If you're mot familiar with music production, you can picture a MIDI file as a LaTeX source file, and fluidsynth like a typesetting engine producing output pdf document ready for visualization. The fonts in this process are called _soundfonts_.

Being `mido` a pure-Python package, installing it should be as easy as

```
pip install --user mido
```

For `fluidsynth` you should refer to the online documentation. If, e.g., you're running a GNU Linux Fedora distribution you are all set:

```
sudo dnf install fluidsynth
```

In addition, you will also need some soundfonts—after all, fluidsynth is the rough equivalent of a typesetting system like LaTeX, and you cant't do much without fonts. If you get an error message sounding like

```
fluidsynth: error: fluid_is_soundfont(): fopen() failed: 'File does not exist.'
...
fluidsynth: error: Failed to load SoundFont "/usr/share/soundfonts/default.sf2"
Rendering audio to file '/home/lbaldini/ixpeobssimdata/toy_point_source_du1.wav'..
fluidsynth: warning: No preset found on channel 0 [bank=0 prog=98]
```

this is almost certainly your problem. You distro might include some soundfont bank out of the box

```
sudo dnf install fluid-soundfont*
```

Alternatively, there's plenty of place out there where you can find General MIDI soundfonts banks—just start from here.

The animation facilities are based on the matplotlib animation module, and you will need ffmpeg, a cross-platform, open-source video recording and editing software. Again, if you're running a GNU Linux Fedora distribution you are all set:

```
sudo dnf install ffmpeg
```

(Otherwise refer to the documentation online.)

26.3 Sonification in details

Warning

This section is a stub.

xpsonify provides a large variety of settings to translate the properties of the X-rays impinging on the gas pixel detector into notes.

The velocity is determined by the fractional amount of energy released in the active volume—the photoelectron track is not always fully collected, and, at any given energy, the amount of energy *collected* by the detector is a sensible analogous to *how hard the piano player presses the keys*.

The note duration can be related to the track size in the detector or, more precisely, to the number of pixels in the region of interest for the readout: the larger the readout window, the longer it takes to the readout electronics for processing, formatting and transferring out the track.

The pan is determined by the photoelectron azimuthal angle, as explained in the beginning of the section.

Note

There are others expressive means that the MIDI protocol offers (such as the *aftertouch*), and could be incorporated into the sonification process in the future, assuming they are not too subtle to be effective.

26.4 API

Most of the relevant API dealing with sonification are include in the *ixpeobssim.evt.animate* and `ixpeobssim.evt.sonification` Python modules.

CHAPTER

TWENTYSEVEN

IXPESIM INTERFACE

Warning

This section deals with a piece of software that is not publicly distributed, and is likely not of general interest—and yet is kept here for the sake of IXPE collaborators, that do have access to full detector simulation.

If you have never heard about it, `ixpesim` (a C++ application living in the gdpsw repository) is the IXPE full detector simulation based on the Geant4 framework.

`ixpesim` is fundamentally different from `ixpeobssim`, in that it follows the impinging photons one at a time, simulating the interaction with the detector at a microscopic level. As such, it provides a much richer information (e.g., real track images, with all the associated topological information), well beyond the ultimate capabilities of `ixpeobssim`, due to the fact that the latter only operates on a top-level description of the instrument (the response functions). It goes without saying, this does not come at no cost, and generating events with `ixpesim` is about 1000 times slower than the fast-simulation counterpart—while `ixpeobssim` can throw a million events in a few seconds, it takes an hour or so to `ixpesim` to achieve the same thing.

Note

Added in version 16.13.0: Over the years the inter-operation between `ixpeobssim` and `ixpesim` has improved at the point that it is now possible to simulate a simple celestial point source in `ixpesim` and feed the output back in to XSPEC for a fully-fledged spectro-polarimetric fitting.

Added in version 19.0.0: This is the first version with partial support of the concept of photon lists, i.e., list of photon energies, times and coordinates at the top of the Be window that can be fed into `ixpesim` to simulate arbitrary source models. Note this new functionality requires `ixpesim` version 13.6.0 or later.

Added in version 21.3.0: This version saw a complete refactoring of the photon list mechanism, and it is the first in which the latter can be considered fully functional, including the projection of the photon direction in the sky, with vignetting and dithering fully taken into account. This version requires `ixpesim` version 13.10.0 or later.

The basic work-flow for the inter-operation of `ixpeobssim` and `ixpesim` reads basically as follows:

- create a custom spectrum or, even better, a photon list to be fed into `ixpesim`;
- run a simulation within `ixpesim`;
- reconstruct the simulation with `ixperecon`;
- process the `ixpesim` output with *xpsimfmt* to make it compatible with XSPEC;
- feed all the ingredients in XSPEC for the spectro-polarimetic fit.

The various steps are further described in the remaining of this section.

27.1 Custom spectra

`ixpesim` was mainly developed to support the characterization, test, and calibration of the IXPE detector units, and, as such, is limited in scope to the simulation of the gas pixel detectors at the focal plane of the instrument—the simulation flow starts *right before the Berillium entrance window*.

Interestingly enough, `ixpesim` supports the use of custom input spectra in the form of simple, two-column text files (one line per energy, the overall normalization being irrelevant). This allows to incorporate the effect of all the elements upstream the GPD (i.e., the mirrors and the UV filter) in such custom spectra.

> **Warning**
>
> Admittedly, the flexibility of `ixpesim` in simulating celestial sources is limited—there are no real handles for the source morphology beyond a Gaussian beam, and the program is limited to energy-independent polarization degree and angle. This is, nonetheless, enough to perform a spectro-polarimetric fit within XSPEC. For more advanced applications, the use of photon lists is the obvious way to go.

`ixpeobssim` provides the capability of creating arbitrary custom spectra to be fed into `ixpesim` for simulating power-law sources, including the effect of the galactic absorption with an arbitrary H column density, through the *xpsimspec* application.

xpsimspec creates the convolution of:

- the intrinsic source spectrum;
- the galactic absorption;
- the mirror effective area;
- the transparency of the UV filter;
- the effect of the contaminants in the Be windows—which are accounted for in the response functions shipped with `ixpeobssim`, but are not simluated in `ixpesim`.

The output is a text file that can be used directly as a custom spectrum in `ixpesim`.

27.2 Photon lists

Since version 19.0.0 (but you are strongly encouraged to upgrade to version 21.3.0 or later, if you are playing this game) `ixpeobssim` provides the *xpphotonlist* application, that allows to generate a list of photon definitions (in the form of a FITS file) that can then be fed into `ixpesim` to be propagated through the detector and turned into a list of actual photo-electron tracks for the events triggering the detector.

> **Warning**
>
> The photon list mechanism is a relatively new feature that should be considered in beta testing mode. Since it has the potential for becoming our final weapon when it comes to detailed simulations of celestial sources, it is very important to proceed with all the necessary tests. You are welcome to try it and you are kindly encouraged to report as soon as possible any issue you might run into.

The photon lists contain all the relevant information about the photons impinging on the Be window of the GPD, i.e., time, energy, position and polarization degree and angle. The full format specification is as follows.

27.2.1 Header keywords

- TELESCOP: *Telescope name*
- INSTRUME: *Instrument name*
- DETNAM: *Name of the logical detector unit*
- DET_ID: *Name of the physical detector unit*
- TSTART: *Observation start time in MET*
- TSTOP: *Observation end time in MET*
- DATE-OBS: *Observation start datetime*
- DATE-END: *Observation end datetime*
- TELAPSE: *TSTOP-TSTART*
- TIMESYS: *Time system*
- TIMEUNIT: *Time units*
- TIMEREF: *Time reference*
- MJDREFI: *MJD ref day 01 Jan 2017 00:00:00 UTC*
- MJDREFF: *Frac part of MJD ref (32.184secs+37leapsecs)*
- TIMEZERO: *Zero time*
- ONTIME: *On source time*
- LIVETIME: *On source time corrected for dead time*
- DEADC: *Dead time correction*
- DEADAPP: *Has DEADC been applied to data*
- RA_OBJ: *[deg] R.A. Object*
- DEC_OBJ: *[deg] Dec Object*
- RA_PNT: *[deg] RA pointing*
- DEC_PNT: *[deg] Dec pointing*
- OBJECT: *Name of observed object*

27.2.2 Columns

- SEC (J): *Integral part of event time (MET)* [s]
- MICROSEC (J): *Fractional part of event time (MET)* [us]
- TIME (D): *Event time in seconds (MET)* [s]
- ENERGY (E): *Event energy in keV* [keV]
- RA (E): *Right Ascension of the photon* [deg]
- DEC (E): *Declination of the photon* [deg]
- DETX (E): *X position at the top of the Be window (GPD frame)* [mm]
- DETY (E): *Y position at the top of the Be window (GPD frame)* [mm]

- POL_DEG (E): *Polarization degree* []
- POL_ANG (E): *Polarization angle* [rad]
- SRC_ID (I): *Monte Carlo source identifier* [None]

The format is consistent with that expected with the full rewrite of the particle source class made specifically for this purpose in ixpesim, and requires gpdsw version 13.10.0 or later.

Note

Since not all the photons impinging on the Be window make it to the active gas volume and are absorbed, generating a photoelectron track with enough energy to trigger the detector, you should expect the event list in output from ixpesim to be significantly shorter than the photon list that is given as an input. The exact ratio depends on the energy spectrum of the source. As a rule of thumb, for a typical celestial source you should expect about 10% of the photons to generate a track.

As of ixpeobssim version 21.3.0 the photon list FITS files include by default a copy of the SC_DATA binary table that is used downstream to project the photons back in the sky taking into account the dithering of the observatory.

27.3 Fitting ixpesim data sets

The ixpesim output files are not readily usable in XSPEC. Beyond a few missing header keywords, the main problem is that the measured energy is expressed in (un-calibrated) PHA counts, and there is no concept of pulse invariant.

In real life level-1 files are processed by a series of tools, correcting for a number of effects (gain non-uniformity, secular gain variations, temperature effects, and charging). For simulated data we can use the same model that is used at the generation stage of the response functions.

xpsimfmt performs all the steps that are necessary to close the loop and have the ixpesim output inter-operate correctly with all the analysis tools shipped with ixpeobssim—namely:

- add some relevant keywords in the proper headers;
- add the PI, Q, U, RA, DEC, X, Y, and W_MOM columns in the EVENTS extension.

Warning

For the process to round-trip correctly, one should make sure that all the relevant adjustable simulation parameters (e.g., the GEM gain) are strictly the same as those used in the data sets upon which the response functions are based.

Care should be taken when simulating events with an ixpesim version different from that used for the generation of the response functions.

27.4 An exemplary use case

Below is some sample code that you might use to generate an `ixpesim` simulation of an arbitrarily complex source model, suitable for a spectro-polarimetric fit in XSPEC.

```
# Create the photon list to be fed into ixpesim. Be careful to the version of
# the IRF you are using, because you will need a non-diagonal response matrix.
# If you don't know what to do, use the default ``ixpe_mc_v9``
xpphotonlist.py --config path/to/srcmodel.py \
                --duration 5000\
                --irfname "ixpe:obssim:v10"

# Run the ixpesim simulation. A few notes:
# - if you want to simulate the three DUs, you will have to run them separately
# - to process the entire photon list, use a very large number for the -n switch
# - make sure you have the pressure right for the DU you are targeting.
#
# For version 9 of the IRFs, the asymtotic pressure values read:
# GPD_DME_ASYMPTOTIC_PRESSURE_DICT = {1: 645., 2: 631., 3: 638.}
ixpesim --src-photon-list ~/ixpeobssimdata/srcmodel_du1_photon_list.fits \
        --output-file ~/ixpeobssimdata/srcmodel_ixpesim_du1.fits \
        -n 1000000
        --dme-pressure 645.

# Run the event reconstruction.
ixperecon ~/ixpeobssimdata/srcmodel_ixpesim_du1.fits

# add the missing columns for the use in XSPEC
xpsimfmt.py ~/ixpeobssimdata/srcmodel_ixpesim_du1_recon.fits

# And if you want a reference ``ixpeobssim`` simulation to compare with...
xpobssim.py --config path/to/srcmodel.py \
            --duration 5000 \
            --deadtime 0. \
            --irfname ixpe_mc_v9
```

From this point on you should be able to bin the data and fit them in XSPEC as usual.

CHAPTER

TWENTYEIGHT

IMPLEMENTATION DETAILS

This section describes the technical details of the simulation flow for a *standard model component* and for the conversion of a *Chandra observation*. In order to take advantage of the efficient array manipulation capabilities provided by numpy, the entire implementation is vectorized, i.e. we don't have an explicit event loop in python.

28.1 Standard model component

The basic flow of the simulation for a single standard model component is coded in `ixpeobssim.srcmodel.roi.xModelComponentBase.rvs_event_list()`. Mathematically speaking, the simulation algorithm can be spelled out in the form of the following basic sequence:

1. Given the source spectrum $\mathcal{S}(E,t)$ and the effective area $A_{\rm eff}(E)$, we calculate the count spectrum as a function of the energy and time:

$$\mathcal{C}(E,t) = \mathcal{S}(E,t) \times A_{\rm eff}(E) \quad [\rm s^{-1}\ keV^{-1}].$$

   In case of a source with cosmological redshift Z or hydrogen column density n_H different from zero, these two quantities are used to correctly re-scale $\mathcal{C}(E,t)$:

$$\mathcal{C}'(E,t) = \mathcal{C}(E(1+E),t)\exp\left(-n_H\sigma(E)\right)$$

2. We calculate the light curve of the model component (in counts space) by integrating over the energy:

$$\mathcal{L}(t) = \int_{E_{\rm min}}^{E_{\rm max}} \mathcal{C}'(E,t)dE \quad [\rm Hz].$$

3. We calculate the total number of expected events $N_{\rm exp}$ by integrating the count rate over the observation time:

$$N_{\rm exp} = \int_{t_{\rm min}}^{t_{\rm max}} \mathcal{L}(t)dt$$

   and we extract the number of *observed* events $N_{\rm obs}$ according to a Poisson distribution with mean $N_{\rm exp}$.

4. We treat the count rate as a one-dimensional probability density function in the random variable t, we extract a vector $\hat{t}$ of $N_{\rm obs}$ values of t according to this pdf—and we sort the vector itself. (Here and in the following we shall use the hat to indicate vectors of lenght $N_{\rm obs}$.)

5. We treat the array of count spectra $\hat{\mathcal{C}}(E,\hat{t})$, evaluated at the time array $\hat{t}$, as an array of one-dimensional pdf objects, from which we extract a corresponding array $\hat{E}$ of (true) energy values. (In an event-driven formulation this would be equivalent to loop over the values t_i of the array $\hat{t}$, calculate the corresponding count spectrum

$$\mathcal{C}_i(E,t_i)$$

and treat that as a one-dimensional pdf from which we extract a (true) energy value E_i, but the vectorized description is more germane to what the code is actually doing internally.)

6. We treat the energy dispersion $\hat{\mathcal{D}}(\epsilon;\hat{E})$ as an array of one-dimensional pdf objects that we use to extract the measured energies $\hat{\epsilon}$ and the corresponding PHA values.

7. We extract suitable arrays of (true) $\hat{\text{RA}}$, $\hat{\text{Dec}}$ values and, similarly to what we do with the energy dispersion, we smear them with the PSF in order to get the correponding measured quantities.

8. We use the polarization degree P and angle α of the model component to calculate the visibility $\hat{\xi}$ and the phase $\hat{\phi}_0$ of the azimuthal distribution modulation, given the modulation factor $\mu(E)$ of the polarimeter

$$\hat{\xi} = \hat{P}(\hat{E},\hat{t},\hat{\text{RA}},\hat{\text{Dec}}) \times \mu(\hat{E})$$
$$\hat{\phi}_0 = \hat{\alpha}(\hat{E},\hat{t},\hat{\text{RA}},\hat{\text{Dec}}),$$

and we use these values to extract the directions of emission of the photoelectron.

(For periodic sources all of the above is done in phase, rather than in time, and the latter is recovered at the very end using the source ephemeris, but other than that there is no real difference.)

For source models involving more than one component, this is done for each component separately, and the different resulting phothon lists are then merged and ordered in time at the end of the process.

28.2 Conversion of a Chandra observation

In this case the starting point is a Chandra photon list, publicly available on the on-line database, which includes temporal, energetic and spatial information for each detected event. Being the Chandra angular and energetic resolution (respectively HPD $\sim$ 1 arcsec and FWHM $\sim$ 5% at 5.9 keV) much better than those expected for the IXPE mission), energies E_C and positions RA_C, Dec_C measured by Chandra can be assumed as Monte Carlo truth.

From the implementation standpoint, the conversion is coded in *ixpeobssim.srcmodel.roi.xChandraObservation.rvs_event_list()* and can be summarized in the following steps:

1. We compute the ratio T_R between the Chandra observation time T_C and the provided IXPE simulation time T_X:

$$T_R = T_C/T_X$$

2. We scale Chandra detected events by first concatenating the measured arrays (E_C, RA_C and Dec_C) as many times as the integer part of T_R (denoted $\lfloor T_R \rfloor$) and then appending to the end of the sequence a portion whose relative length is equal to the fractional part of T_R (called $\{T_R\}$). For instance, in case of the energy:

$$\hat{E} = \underbrace{E_C\ E_C \cdots E_C}_{\lfloor T_R \rfloor}\ \underbrace{E_C}_{\{T_R\}}.$$

3. We calculate the effective area ratio A_R between one IXPE telescope and Chandra as a function of energy and off-axis angle θ:

$$A_R(E,\theta) = \frac{A_{\text{eff},X}(E,\theta)}{A_{\text{eff},C}(E,\theta)}$$

4. This ratio allows to conveniently down-sample the Chandra time-scaled photon list. In practice, this is done by throwing an array of random numbers $\hat{r}$ between 0 and 1, and accepting as IXPE events only those that meet the condition:

$$\hat{r} < A_R(\hat{E},\hat{\theta})$$

5. We randomly extract the event times $\hat{t}$ uniformly between the starting and ending observation time.
6. From this point, once the arrays $\hat{E}$, $\hat{t}$, $\hat{RA}$ and $\hat{Dec}$ of true values are selected, the simulation goes ahead as in the standard model component case. In particular, we smear these quantities with the IXPE response functions and we extract the emission angle $\hat{\phi}$ based on the input polarization model.

In case of definition of multiple sub-regions of the ROI, the conversion is done separately for each of them (skipping those flagged as to be removed), and the resulting photon lists are merged and ordered in time at the end of the process.

CHAPTER

TWENTYNINE

API

29.1 Top-level indices

- genindex
- modindex
- search

29.2 Core Modules

Facilities related to FITS I/O.

ixpeobssim.core.fitsio.**copy_hdu_list**(*hdu_list*)

Create a copy in memory of a HDU list.

Note that the HDUList class provides a copy() method, but this is a shallow copy that is still tied to the underlying file the HDU list is read from: https://github.com/astropy/astropy/issues/6395

As a workaround we copy the HDUs one by one and assemble them back into a HDUList object.

ixpeobssim.core.fitsio.**find_column_index**(*hdu*, *col_name*)

Return the index corresponding to a given column name within a binary table, and None when the column does not exists.

Parameters

- **hdu** (*fits.BinTableHDU instance*) – The underlying HDU
- **col_name** (*str*) – The name of the target column.

ixpeobssim.core.fitsio.**read_hdu_list_in_memory**(*file_path*, *extension='fits'*)

Read a fits file and store the HDU list in memory.

This is a convenience function that is meant to open a file within a context manager (so that it is properly closed after the fact) and create a copy of the content in memory. This is less than trivial, as astropy typically keeps some kind of pointer back to the original file when dealing with FITS files, and it is not obvious how to make sure that all the resources are properly closed at runtime.

ixpeobssim.core.fitsio.**set_tlbounds**(*hdu*, *col_name*, *min_*, *max_*)

Set the proper TLMIN and TLMAX header keywords for a given column in a given HDU.

class ixpeobssim.core.fitsio.**xBinTableHDUBase**(*data=None*, *keywords=None*, *comments=None*)

Binary table HDU class.

This is a small wrapper around a standard binary table to facilitate customizations.

set_ext_name(*name*)

Set the extension name for the binary table.

classmethod spec_names()

Return the name of the data fields specified in the SPEC class member.

classmethod spec_names_and_types()

Return the name of the data fields specified in the SPEC class member.

class ixpeobssim.core.fitsio.**xFITSImageBase**(*file_path*)

Base class describing a FITS image.

Parameters

file_path (*string*) – The path to the FITS file containing the image.

add_circle(*x*, *y*, *radius*, *mode='radec'*, ***kwargs*)

Add a circle to the image.

Note that the x and y coordinates can be either RA and Dec in decimal degrees (if mode='radec') or relative to canvas coordinates (if mode='canvas').

Parameters

- **x** (*float*) – The x position of the circle center (Right Ascension in decimal degrees if relative=False, relative position in canvas coordinates if relative=True).
- **y** (*float*) – The y position of the circle center (Declination in decimal degrees if relative=False, relative position in canvas coordinates if relative=True).
- **radius** (*float*) – The radius of the circle in arcseconds.
- **mode** (*{'radec', 'canvas'}*) – Optional flag defining how the center of the circle is expresses: in right ascension and declination ('radec') or in relative canvas coordinates ('canvas').
- **kwargs** (*dict*) – All the keyword arguments passed to the matplotlib Circle patch.

add_circle_canvas(*ra*, *dec*, *radius*, ***kwargs*)

Convenience function, see add_circle().

add_circle_radec(*ra*, *dec*, *radius*, ***kwargs*)

Convenience function, see add_circle().

static add_label(*text*, *x=0.1*, *y=0.9*, ***kwargs*)

Add a label to an image.

This is a shortcut to have all the formatting defined.

Parameters

- **text** (*string*) – The label text
- **x** (*float*) – The x position of the label in relative coordinates
- **y** (*float*) – The y position of the label in relative coordinates
- ****kwargs** – All the keyword arguments passed to plt.text()

cdelt1()

Return the value of the CRVAL1 header keyword.

cdelt2()

Return the value of the CRVAL2 header keyword.

center()

Return the sky coordinates of the image center.

crpix1()

Return the value of the CRPIX1 header keyword.

crpix2()

Return the value of the CRPIX2 header keyword.

crval1()

Return the value of the CRVAL1 header keyword.

crval2()

Return the value of the CRVAL2 header keyword.

delta()

Return the pixel increment in the ra and dec coorinates.

get(*keyword*, *default=None*)

Retrieve the value of a primary header keyword.

inner_radius()

Return the inner radius of the image in degrees.

static make_plot(*data*, *wcs_*, *slices=None*, *zlabel=None*, *stretch='linear'*, *ticks=None*, *colorbar_format=None*, ***kwargs*)

Underlying plotting routine.

This is implemented as a staticmethod in such a way it can be called from external methods dealing with multi-dimensional data arrays, which are not readily supported in the base class. This is achieved by passing the slices argument, aling with the proper data slice.

Parameters

- **data** (*array_like*) – The underlying data to be plotted.
- **wcs** (*astropy.wcs object*) – The WCS object defining the transformation from pixel to sky coordinates.
- **slices** (*tuple, optional*) – Verbatim from the astropy.visualization documentation: For WCS transformations with more than two dimensions, we need to choose which dimensions are being shown in the 2D image. The slice should contain one x entry, one y entry, and the rest of the values should be integers indicating the slice through the data. The order of the items in the slice should be the same as the order of the dimensions in the WCS, and the opposite of the order of the dimensions in Numpy. For example, (50, 'x', 'y') means that the first WCS dimension (last Numpy dimension) will be sliced at an index of 50, the second WCS and Numpy dimension will be shown on the x axis, and the final WCS dimension (first Numpy dimension) will be shown on the y-axis (and therefore the data will be plotted using data[:, :, 50].transpose())
- **zlabel** (*str*) – The label for the z axis (e.g., the colorbar).
- **stretch** (*{'linear', 'sqrt', 'power', log', 'asinh'}, optional*) – The stretch function to apply to the image.

- **ticks** (*array_like or ticker, optional*) – The explicit setting for the colorbar ticks.
- **colorbar_format** (*{None, 'log', 'scilog'}, optional*) – Optional string for the colorbar formnatting.
- **kwargs** (*dict*) – All the keyword arguments passed to plt.imshow().

plot(*zlabel=None*, *stretch='linear'*, *ticks=None*, *colorbar_format=None*, ***kwargs*)

Plot the underlying FITS image.

Since ixpeobssim version 8.9.0 this not using aplpy under the hood anymore, see https://bitbucket.org/ixpesw/ixpeobssim/issues/272

Parameters

- **zlabel** (*string*) – The text label for the colorbar (use None for no colorbar)
- ****kwargs** – All the keyword arguments passed to plt.imshow()

plot_arrows(*nside*, *model*, *threshold=0.0*, ***kwargs*)

Overlay arrows on an image.

recenter(*ra*, *dec*, *radius*)

Recenter the image.

Parameters

- **ra** (*float*) – The RA position of the circle center in decimal degrees.
- **dec** (*float*) – The DEC position of the circle center in decimal degrees.
- **radius** (*float*) – The radius of the region in arcseconds.

shape()

Return the shape of the image.

sky_bounding_box()

Return the bounding box of the FITS image in sky coordinates, i.e., a 4-element tuple of the form (ra_min, dec_min, ra_max, dec_max).

square_sky_grid(*nside*, *ra0=None*, *dec0=None*, *half_size=None*)

Create a square, regular grid in ra and dec.

This is essentially an overloaded version of the utils.astro.square_sky_grid() method, where the arguments are inferred, when possible, from the underlying image structure.

class ixpeobssim.core.fitsio.**xHDUBase**

Base class for FITS HDU.

add_comment(*comment*)

Add a comment to the table header.

add_keyword(*key*, *value*, *comment=''*)

Add a keyword to the table header.

set_keyword(*key*, *value*)

Set a keyword into the table header.

set_keyword_comment(*keyword*, *comment*)

Set the comment for a header keyword.

setup_header(*keywords=None*, *comments=None*)

Update the table header with arbitrary additional information.

class ixpeobssim.core.fitsio.**xPrimaryHDU**(*data=None*, *header=None*, *creator='ixpeobssim'*, *keywords=None*, *comments=None*)

Class describing a primary HDU to be written in a FITS file.

This is initializing a standard astropy.io.fits.PrimaryHDU object and adding the creator and creation time fields.

Parameters

creator (*str*) – The application that created the header (defaults to *ixpeobssim*, and the ixpeobssim tag is automatically added.)

ixpeobssim.core.fitting.**fit**(*model*, *xdata*, *ydata*, *p0=None*, *sigma=None*, *xmin=-inf*, *xmax=inf*, *absolute_sigma=True*, *check_finite=True*, *method=None*, *verbose=True*, ***kwargs*)

Lightweight wrapper over the `scipy.optimize.curve_fit()` function to take advantage of the modeling facilities. More specifically, in addition to performing the actual fit, we update all the model parameters so that, after the fact, we do have a complete picture of the fit outcome.

Parameters

- **model** (`ixpeobssim.core.modeling.FitModelBase` instance callable) – The model function, f(x, ...). It must take the independent variable as the first argument and the parameters to fit as separate remaining arguments.
- **xdata** (*array_like*) – The independent variable where the data is measured.
- **ydata** (*array_like*) – The dependent data — nominally f(xdata, ...)
- **p0** (*None, scalar, or sequence, optional*) – Initial guess for the parameters. If None, then the initial values will all be 1.
- **sigma** (*None or array_like, optional*) – Uncertainties in *ydata*. If None, all the uncertainties are set to 1 and the fit becomes effectively unweighted.
- **xmin** (*float*) – The minimum value for the input x-values.
- **xmax** (*float*) – The maximum value for the input x-values.
- **absolute_sigma** (*bool, optional*) – If True, *sigma* is used in an absolute sense and the estimated parameter covariance *pcov* reflects these absolute values. If False, only the relative magnitudes of the *sigma* values matter. The returned parameter covariance matrix *pcov* is based on scaling *sigma* by a constant factor. This constant is set by demanding that the reduced *chisq* for the optimal parameters *popt* when using the *scaled sigma* equals unity.
- **method** (*{'lm', 'trf', 'dogbox'}, optional*) – Method to use for optimization. See *least_squares* for more details. Default is 'lm' for unconstrained problems and 'trf' if *bounds* are provided. The method 'lm' won't work when the number of observations is less than the number of variables, use 'trf' or 'dogbox' in this case.
- **verbose** (*bool*) – Print the model if True.
- **kwargs** – Keyword arguments passed to *leastsq* for `method='lm'` or *least_squares* otherwise.

ixpeobssim.core.fitting.**fit_gaussian_iterative**(*histogram*, *p0=None*, *sigma=None*, *xmin=-inf*, *xmax=inf*, *absolute_sigma=True*, *check_finite=True*, *method=None*, *verbose=True*, *num_sigma_left=2.0*, *num_sigma_right=2.0*, *num_iterations=2*, ***kwargs*)

Fit the core of a gaussian histogram within a given number of sigma around the peak.

This function performs a first round of fit to the data and then repeats the fit iteratively limiting the fit range to a specified interval defined in terms of deviations (in sigma) around the peak.

For additional parameters look at the documentation of the *ixpeobssim.core.fitting.fit_histogram()*

Parameters

- **num_sigma_left** (*float*) – The number of sigma on the left of the peak to be used to define the fitting range.
- **num_sigma_right** (*float*) – The number of sigma on the right of the peak to be used to define the fitting range.
- **num_iterations** (*int*) – The number of iterations of the fit.

ixpeobssim.core.fitting.**fit_histogram**(*model*, *histogram*, *p0=None*, *sigma=None*, *xmin=-inf*, *xmax=inf*, *absolute_sigma=True*, *check_finite=True*, *method=None*, *verbose=True*, ***kwargs*)

Fit a histogram to a given model.

This is basically calling *ixpeobssim.core.fitting.fit()* with some pre-processing to turn the histogram bin edges and content into x-y data. Particularly, the bin centers are taken as the independent data series, the bin contents are taken as the dependent data saries, and the square root of the counts as the Poisson error.

For additional parameters look at the documentation of the *ixpeobssim.core.fitting.fit()*

Parameters

- **model** (ixpeobssim.core.modeling.FitModelBase instance or) – callable The fit model.
- **histogram** (*ixpeHistogram1d instance*) – The histogram to be fitted.

Warning

We're not quite doing the right thing, here, as we should integrate the model within each histogram bin and compare that to the counts, but this is not an unreasonable first-order approximation. We might want to revise this, especially since we can probably provide an analytic integral for most of the model we need.

ixpeobssim.core.fitting.**fit_modulation_curve**(*histogram*, *p0=None*, *sigma=None*, *xmin=-inf*, *xmax=inf*, *absolute_sigma=True*, *check_finite=True*, *method=None*, *verbose=True*, *degrees=True*, ***kwargs*)

Fit a modulation curve.

For all the other parameters look at the documentation of the *ixpeobssim.core.fitting.fit_histogram()*

ixpeobssim.core.fitting.**linear_analytical_fit**(*x*, *y*, *dy=None*)

Simple vectorized, analytical linear fit.

See https://mathworld.wolfram.com/LeastSquaresFitting.html

ixpeobssim.core.fitting.**power_law_analytical_fit**(*x*, *y*)

Simple vectorized, analytical linear fit.

See https://mathworld.wolfram.com/LeastSquaresFitting.html

Geometry module.

class ixpeobssim.core.geometry.**xLine**(*begin*, *end*)

Class representing a line.

length()

Return the length of the line.

class ixpeobssim.core.geometry.**xPoint**(*x*, *y*, *z*)

Class representing a point in the three-dimensional space.

distance_to(*other*)

Return the distance to another point.

move(*length*, *ray*)

Move the point by a given length along a given ray.

norm()

Return the norm of the three-dimensional vector corresponding to the point.

unphysical()

Return True if the point is unphysical.

classmethod unphysical_point()

Return an unphysical point.

class ixpeobssim.core.geometry.**xRay**(*origin*, *theta*, *phi*)

Class representing a ray in the three-dimensional space.

xintersect(*x*)

Return the point where the ray intersects the plane at a given x.

yintersect(*y*)

Return the point where the ray intersects the plane at a given y.

zintersect(*z*)

Return the point where the ray intersects the plane at a given z.

Histogram facilities.

class ixpeobssim.core.hist.**xGpdMap2d**(*nside=10*, *zlabel='Entries/bin'*)

2-dimensional GPD map.

class ixpeobssim.core.hist.**xGpdMap3d**(*nside*, *zbins*, *zlabel=''*, *wlabel='Entries/bin'*)

Three-dimensional histogram where the first two axes represent the GPD active area in Physical coordinates.

class ixpeobssim.core.hist.**xHistogram1d**(*xbins*, *xlabel=''*, *ylabel='Entries/bin'*)

Container class for one-dimensional histograms.

errorbar(***kwargs*)

Plot the histogram as a scatter plot.

errorbar_data()

Return the x, y, dy arrays that can be used to build a scatter plot (with errors) from the histogram.

fit(*model*, *p0=None*, *sigma=None*, *xmin=-inf*, *xmax=inf*, *absolute_sigma=True*, *check_finite=True*, *method=None*, *verbose=True*, ***kwargs*)

Fit the histogram with a model.

gaussian_kde_smooth(*bandwidth=2.0*)

Create a copy of the histogram where the weights are smoothed with a gaussian kernel density estimatore.

Parameters

bandwidth (*float*) – The sigma of the gaussian kernel, in (fractional) number of bins.

scatter_plot()

Turn the histogram into a scatter plot.

> **Warning**
>
> This is to be removed.

class ixpeobssim.core.hist.**xHistogram2d**(*xbins*, *ybins*, *xlabel=''*, *ylabel=''*, *zlabel='Entries/bin'*)

Container class for two-dimensional histograms.

gaussian_kde_smooth(*bandwidth=(2.0, 2.0)*)

Create a copy of the histogram where the weights are smoothed with a gaussian kernel density estimatore.

> **Warning**
>
> This is essentially untested.

Parameters

bandwidth (*2-element tuple float*) – The sigma of the gaussian kernel, in (fractional) number of bins.

hbisect(*y*)

Return the horizontal slice corresponding to a given y value.

hslice(*bin_*)

Return the horizontal slice for a given bin.

hslices()

Return a list of all the horizontal slices.

vbisect(*x*)

Return the vertical slice corresponding to a given y value.

vslice(*bin_*)

Return the vertical slice for a given bin.

vslices()

Return a list of all the vertical slices.

class ixpeobssim.core.hist.**xHistogram3d**(*xbins*, *ybins*, *zbins*, *xlabel=''*, *ylabel=''*, *zlabel=''*, *wlabel='Entries/bin'*)

Container class for three-dimensional histograms.

class ixpeobssim.core.hist.**xHistogramBase**(*binning*, *labels*)

Base class for an n-dimensional histogram.

This interface to histograms is profoundly different for the minimal numpy/matplotlib approach, where histogramming methods return bare vectors of bin edges and counts. The main underlying ideas are

- we keep track of the bin contents, the bin entries and the sum of the weights squared (the latter for the purpose of calculating the errors);
- we control the axis label and the plotting styles;

- we provide two separate interfaces, fill() and set_content(), to fill the histogram from either unbinned or binned data;
- we support the basic arithmetics (addition, subtraction and multiplication by a scalar);
- we support full data persistence (I/O) in FITS format.

Note that this base class is not meant to be instantiated directly, and the interfaces to concrete histograms of specific dimensionality are defined in the sub-classes.

Parameters

- **binning** (*n-tuple of array*) – the bin edges on the different axes.
- **labels** (*n-tuple of strings*) – the text labels for the different axes.

bin_centers(*axis=0*)

Return the bin centers for a specific axis.

bin_widths(*axis=0*)

Return the bin widths for a specific axis.

static binning_col_name()

Column name for the axis binnings.

static binning_hdu_name(*axis*)

Extension name for the axis binnings.

static bisect(*binning*, *values*, *side='left'*)

Return the indices corresponding to a given array of values for a given binning.

copy()

Create a full copy of a histogram.

empty_copy()

Create an empty copy of a histogram.

static entries_hdu_name()

Extension name for the bin entries.

errors()

Return the bin errors.

fill(**data*, *weights=None*)

Fill the histogram from unbinned data.

Note this method is returning the histogram instance, so that the function call can be chained.

find_bin(**coords*)

Find the bin corresponding to a given set of "physical" coordinates on the histogram axes.

This returns a tuple of integer indices that can be used to address the histogram content.

find_bin_value(**coords*)

Find the histogram content corresponding to a given set of "physical" coordinates on the histogram axes.

classmethod from_file(*file_path*)

Load the histogram from a FITS file.

Note that we transpose the image data back at read time—see the comment in the save() method.

static label_keyword(*axis*)

Header keyword for the axis labels.

mean(*axis=0*)

Calculate the (binned) mean eastimate along a given axis.

num_entries()

Return the number of entries in the histogram.

plot(***kwargs*)

Plot the histogram.

rms(*axis=0*)

Calculate the (binned) rms eastimate along a given axis.

> **Warning**
>
> Mind this is not using anything clever (e.g. the Welford algorithm) and is not particularly numerically stable.

save(*file_path*, *overwrite=True*, ***header_keywords*)

Save the histogram (with edges) to a FITS file.

Note that all the image data are transposed so that the thing can be correctly visualized with standard FITS viewers—at least in two dimensions.

set_axis_label(*axis*, *label*)

Set the label for a given axis.

set_content(*content*, *entries=None*, *errors=None*)

Set the bin contents programmatically from binned data.

Note this method is returning the histogram instance, so that the function call can be chained.

set_errors(*errors*)

sum(*axis=None*)

return the sum of weights in the histogram.

static sumw2_hdu_name()

Extension name for the bin entries.

class ixpeobssim.core.hist.**xModulationCube2d**(*xbins*, *ybins*)

Specialized class for modulations cubes.

class ixpeobssim.core.hist.**xScatterPlot**(*x*, *y*, *dy=None*, *dx=None*, *xlabel=None*, *ylabel=None*)

Small class encapsulating a scatter plot.

Technically speaking, this would not belong here, as a scatter plot is not strictly related to any of the histogram classes, but a 1-dimensional histogram with errors can techincally be turned into a scatter plot, so we introduce the concept here for completeness.

> **Warning**
>
> Consider removing this class.

plot(***kwargs*)

Plot the scatter plot.

class ixpeobssim.core.modeling.**xConstant**

Constant model.

$$f(x; C) = C$$

cdf(*x*)

Overloaded cdf() method.

init_parameters(*xdata*, *ydata*, *sigma*)

Overloaded init_parameters() method.

static jacobian(*x*, *constant*)

Overloaded jacobian() method.

ppf(*q*)

Overloaded ppf() method.

static value(*x*, *constant*)

Overloaded value() method.

class ixpeobssim.core.modeling.**xExponential**

Exponential model.

$$f(x; N, \alpha) = Ne^{\alpha x}$$

static jacobian(*x*, *normalization*, *exponent*)

Overloaded jacobian() method.

static value(*x*, *normalization*, *exponent*)

Overloaded value() method.

class ixpeobssim.core.modeling.**xExponentialOffset**

Exponential model.

$$f(x; A, N, \alpha) = A + Ne^{\alpha x}$$

static jacobian(*x*, *offset*, *normalization*, *exponent*)

Overloaded jacobian() method.

static value(*x*, *offset*, *normalization*, *exponent*)

Overloaded value() method.

class ixpeobssim.core.modeling.**xFe55**

One-dimensional double gaussian model for Fe55 with 2 lines

$$f(x; A, \mu, \sigma) = Ae^{-\frac{(x-\mu)^2}{2\sigma^2}} + cose$$

init_parameters(*xdata*, *ydata*, *sigma*)

static jacobian(*x*, *amplitude0*, *amplitude1*, *peak*, *sigma*)

Overloaded jacobian() method.

plot(**parameters*, ***kwargs*)

Overloaded plot() method.

static value(*x*, *amplitude0*, *amplitude1*, *peak*, *sigma*)

Overloaded value() method.

class ixpeobssim.core.modeling.**xFitModelBase**

Base class for a fittable model.

This base class isn't really doing anything useful, the idea being that actual models that can be instantiated subclass xFitModelBase overloading the relevant class members.

The class features a number of static members that derived class should redefine as needed:

- `PARAMETER_NAMES` is a list containing the names of the model parameters. (It goes without saying thet its length should match the number of parameters in the model.)
- `PARAMETER_DEFAULT_VALUES` is a list containing the default values of the model parameters. When a concrete model object is instantiated these are the values being attached to it at creation time.
- `PARAMETER_DEFAULT_BOUNDS` is a tuple containing the default values of the parameter bounds to be used for the fitting. The values in the tuple are attached to each model object at creation time and are intended to be passed as the `bounds` argument of the `scipy.optimize.curve_fit()` function. From the `scipy` documentation: Lower and upper bounds on independent variables. Defaults to no bounds. Each element of the tuple must be either an array with the length equal to the number of parameters, or a scalar (in which case the bound is taken to be the same for all parameters.) Use np.inf with an appropriate sign to disable bounds on all or some parameters. By default models have no built-in bounds.
- `DEFAULT_PLOTTING_RANGE` is a two-element list with the default support (x-axis range) for the model. This is automatically updated at runtime depending on the input data when the model is used in a fit.
- `DEFAULT_STAT_BOX_POSITION` is the default location of the stat box for the model, see the `gpdswpy.matplotlib_.stat_box()` function for all the details.

In addition, each derived class should override the following things:

- the `value(x, *args)` static method: this should return the value of the model at a given point for a given set of values of the underlying parameters;
- (optionally) the `jacobian(x, *args)` static method. (If defined, this is passed to the underlying fit engine allowing to reduce the number of function calls in the fit; otherwise the jacobian is calculated numerically.)

Finally, if there is a sensible way to initialize the model parameters based on a set of input data, derived classes should overload the `init_parameters(xdata, ydata, sigma)` method of the base class, as the latter is called by fitting routines if no explicit array of initial values are passed as an argument. The default behavior of the class method defined in the base class is to do nothing.

See `gpdswpy.modeling.xGaussian` for a working example.

init_parameters(*xdata*, *ydata*, *sigma*)

Assign a sensible set of values to the model parameters, based on a data set to be fitted.

Note that in the base class the method is not doing anything, but it can be reimplemented in derived classes to help make sure the fit converges without too much manual intervention.

integral(*edges*)

Calculate the integral of the model within pre-defined edges.

Note that this assumes that the derived class provides a suitable `cdf()` method.

name()

Return the model name.

parameter_error(*name*)

Return the parameter error by name.

parameter_errors()

Return the vector of parameter errors.

parameter_status()

Return the complete status of the model in the form of a tuple of tuples (parameter_name, parameter_value, parameter_error).

Note this can be overloaded by derived classes if more information needs to be added.

parameter_value(*name*)

Return the parameter value by name.

parameter_values()

Return the vector of parameter values.

plot(**parameters*, ***kwargs*)

Plot the model.

Note that when this is called with a full set of parameters, the self.parameters class member is overwritten so that the right values can then be picked up if the stat box is plotted.

reduced_chisquare()

Return the reduced chisquare.

reset()

Reset all the necessary stuff.

This method initializes all the things that are necessry to keep track of a parametric fit.

- the parameter values are set to what it specified in `PARAMETER_DEFAULT_VALUES`;
- the covatiance matrix is initialized to a matrix of the proper dimension filled with zeroes;
- the minimum and maximum values of the independent variable (relevant for plotting) are set to the values specified in `DEFAULT_PLOTTING_RANGE`;
- the model bounds are set to the values specified in `PARAMETER_DEFAULT_BOUNDS`;
- the chisquare and number of degrees of freedom are initialized to -1.

rvs(*size=1*)

Return random variates from the model.

Note that this assumes that the derived class provides a suitable `ppf()` method.

set_parameter(*name*, *value*)

Set a parameter value.

set_parameters(**pars*)

Set all the parameter values.

Note that the arguments must be passed in the right order.

set_plotting_range(*xmin*, *xmax*)

Set the plotting range.

stat_box(*position=None*, *plot=True*, ***kwargs*)

Plot a ROOT-style stat box for the model.

static value(*x*, **parameters*)

Eval the model at a given point and a given set of parameter values.

> **Warning**
>
> This needs to be overloaded in any derived class for the thing to do something sensible.

class ixpeobssim.core.modeling.**xGaussian**

One-dimensional Gaussian model.

$$f(x; A, \mu, \sigma) = Ae^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

static der_amplitude(*x*, *amplitude*, *peak*, *sigma*)

Return the amplitude derivative of the function, to be used in the calculation of the Jacobian.

static der_peak(*x*, *amplitude*, *d_amplitude*, *peak*, *sigma*)

Return the peak derivative of the function, to be used in the calculation of the Jacobian.

Note that we pass the pre-calculated values of the amplitude derivatives in order not to repeat the same calculation more times than strictly necessary.

static der_sigma(*x*, *amplitude*, *d_amplitude*, *peak*, *sigma*)

Return the sigma derivative of the function, to be used in the calculation of the Jacobian.

Note that we pass the pre-calculated values of the amplitude derivatives in order not to repeat the same calculation more times than strictly necessary.

fwhm()

Return the absolute FWHM of the model.

init_parameters(*xdata*, *ydata*, *sigma*)

Overloaded init_parameters() method.

static jacobian(*x*, *amplitude*, *peak*, *sigma*)

Overloaded jacobian() method.

resolution()

Return the resolution of the model, i.e., the FWHM divided by the peak value.

resolution_error()

Return the error on the resolution.

static value(*x*, *amplitude*, *peak*, *sigma*)

Overloaded value() method.

class ixpeobssim.core.modeling.**xGeneralizedGaussian**

Generalized gaussian fitting model.

See https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.gennorm.html for implementation details.

static value(*x*, *norm*, *mean*, *sigma*, *beta*)

Overloaded value() method.

class ixpeobssim.core.modeling.**xHat**

Fit model representing a flat distribution truncated with an error function at both ends.

The ASYMMETRY class member controls the slope of the central part of the model.

static value(*x*, *norm*, *x1*, *sigma1*, *x2*, *sigma2*)

Overloaded value() method.

class ixpeobssim.core.modeling.**xLine**

Straight-line model.

$$f(x; m, q) = mx + q$$

static jacobian(*x*, *intercept*, *slope*)

Overloaded jacobian() method.

static value(*x*, *intercept*, *slope*)

Overloaded value() method.

class ixpeobssim.core.modeling.**xLogNormal**

Log-normal fitting model.

See https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.lognorm.html for more details on the implementation.

Note that our parametrization is done in terms of the mean and sigma of the distribution, in addition to the shape parameter.

static value(*x*, *norm*, *mu*, *sigma*, *shape*)

Overloaded value() method.

class ixpeobssim.core.modeling.**xLorentzian**

Lorentzian function

static jacobian(*x*, *normalization*, *peak*, *hwhm*)

Overloaded jacobian() method.

static value(*x*, *normalization*, *peak*, *hwhm*)

Overloaded value() method.

class ixpeobssim.core.modeling.**xModulationCurveDeg**

Modulation curve model (for fitting azimuthal distributions expressed in degrees).

$$f(x; A, m, \bar{\varphi}) = A(1 + m\cos(2(x - \bar{\varphi})))$$

init_parameters(*xdata*, *ydata*, *sigma*)

Overloaded init_parameters() method.

static jacobian(*x*, *amplitude*, *modulation*, *phase*)

Overloaded jacobian() method.

In addition to the degree-to-radians conversion of the inputs, here we have to multiply the last column of the jacobian (i.e., the phase derivatives) by pi / 180.

static value(*x*, *amplitude*, *modulation*, *phase*)

Overloaded value() method.

Here we are essentially calling the modulation curve model expressed in radians converting to radians the input values of the independent variable and the phase parameter.

class ixpeobssim.core.modeling.**xModulationCurveRad**

Modulation curve model (for fitting azimuthal distributions expressed in radians).

$$f(x; A, m, \bar{\varphi}) = A(1 + m\cos(2(x - \bar{\varphi})))$$

init_parameters(*xdata*, *ydata*, *sigma*)

Overloaded init_parameters() method.

static jacobian(*x*, *amplitude*, *modulation*, *phase*)

Overloaded jacobian() method.

static value(*x*, *amplitude*, *modulation*, *phase*)

Overloaded value() method.

class ixpeobssim.core.modeling.**xPixelPha**

Composite model for fitting the single-pixel pulse-height distributions.

This is essentially the sum of a gaussian centered at zero (and modeling the noise) and a power law with exponential cutoff (modeling the signal).

$$f(x; A_n, \sigma_n, N_s, \Gamma_s, x_0) = A_n e^{-\frac{x^2}{2\sigma_n^2}} + N_s x_s^{\Gamma} e^{-\frac{x}{x_0}}$$

static jacobian(*x*, *noise_amplitude*, *noise_sigma*, *signal_normalization*, *signal_index*, *signal_cutoff*)

Overloaded jacobian() method.

plot(**parameters*, ***kwargs*)

Overloaded plot() method.

static value(*x*, *noise_amplitude*, *noise_sigma*, *signal_normalization*, *signal_index*, *signal_cutoff*)

Overloaded value() method.

class ixpeobssim.core.modeling.**xPowerLaw**

Power law model.

$$f(x; N, \Gamma) = Nx^{\Gamma}$$

static jacobian(*x*, *normalization*, *index*)

Overloaded jacobian() method.

static value(*x*, *normalization*, *index*)

Overloaded value() method.

class ixpeobssim.core.modeling.**xPowerLawExpCutoff**

Power law with an exponential cutoff.

$$f(x; N, \Gamma, x_0) = N x^{\Gamma} e^{-\frac{x}{x_0}}$$

static jacobian(*x*, *normalization*, *index*, *cutoff*)

Overloaded jacobian() method.

static value(*x*, *normalization*, *index*, *cutoff*)

Overloaded value() method.

ixpeobssim.core.pipeline.**batch**()

Return the current value of the 'batch' rc parameter.

ixpeobssim.core.pipeline.**bootstrap_pipeline**(*model_name*)

Convenience bootstrap function.

ixpeobssim.core.pipeline.**figure**(*name*)

Return a matplotlib figure with the model name prepended to the actual name.

ixpeobssim.core.pipeline.**figure_name**(*name*)

Small convience function to enfore a minimum uniformity in the naming scheme for the output file.

This will prepeng the model name to any name the use passes as an argument, and change the latter to all lower case.

ixpeobssim.core.pipeline.**file_list**(**args*, ***kwargs*)

Create a file list from a series of patterns.

This method is very handy to retrieve the files generated by any given step of a pipeline in order to process them in the following step, and takes care automagically of looping over the three detector units.

The arguments can be either a string or a (str, int) tuple, in which case the string is a label and the integer is a uid attached to it (e.g., the identifier for a particular phase selection for a periodic source). The arguments are concatenated in the file name with an "_" character.

Assuming that we have a pipieline bootstrapped with a model called "mymodel", the resolution rules will yield: pipeline.file_list() -> '$IXPEOBSSIM_DATA/mymodel_du1/2/3.fits' pipeline.file_list(('sel', 1)) -> '$IXPEOBSSIM_DATA/mymodel_du1/2/3_sel0001.fits' pipeline.file_list(('sel', 1), 'cmap') -> '$IXPEOBSSIM_DATA/mymodel_du1/2/3_sel0001_cmap.fits'

The label "pha1*" is peculiar in that, if encountered it is automatically expanded to include all the flavors of the Stokes parameter spectra.

Note that, in order to maintain compatibility with Python 2, we opted for the current function signature over what would have been the most natural way to do things in Python 3, i.e., *file_list(*args, folder_path=IXPEOBSSIM_DATA, check_files=True)*

ixpeobssim.core.pipeline.**fit_ensamble_stokes_spectra**(*seed*, *normalized=False*, ***kwargs*)

Fit an ensamble file with XSPEC.

This is a small convenience function that, under the hood, collects the relevant file list, runs xpxspec, and return a list with the best-fit parameters and errors.

ixpeobssim.core.pipeline.**generate_ensamble**(*size=10*, *start_seed=0*, *processing_function=<function standard_ensamble_processing>*, *cleanup=True*, ***kwargs*)

Generate an ensable of simulations to study the a-posteriory statistical properties of the measurement setup.

Parameters

- **size** (*int*) – The number of independent realizations to be generated
- **start_seed** (*int*) – The first random seed to be used (will be incremented by one at each step)
- **cleanup** (*bool*) – If True, remove the (potentially large) event files after the processing.
- ****Kwargs** – All the keyword arguments to be passes to xpobssim.

ixpeobssim.core.pipeline.**glob_ensamble**(*seed*, **patterns*)

Glob an ensamble folder searching for the files matching a particular seed and pattern.

Note this is completely different in spirit wrt file_list(), as the list is not assembled a priori, but globbed directly from the file system (i.e., you are guaranteed that the files in the list do exist).

ixpeobssim.core.pipeline.**glob_ensamble_stokes_spectra**(*seed*, *normalized=False*)

Specialized function to get hold of a consistent set of Stokes spectra for a given seed.

If the normalized argument is False, this returns the PHA1, PHA1Q and PHA1U files for the three detector units, otherwise the PHA1QN and PHA1UN spectra are returned.

ixpeobssim.core.pipeline.**model**()

Return the current model name.

ixpeobssim.core.pipeline.**output_folder**()

Return the current value of the 'outfolder' rc parameter.

ixpeobssim.core.pipeline.**overwrite**()

Return the current value of the 'overwrite' rc parameter.

ixpeobssim.core.pipeline.**param**(*key*, *default=None*)

Retrive a given global configuration parameter.

ixpeobssim.core.pipeline.**params**()

Return the underlying rc param dictionary.

ixpeobssim.core.pipeline.**post_process_ensamble_pcubes**(*seed*, *label='pcube'*)

Post-process the polarization cubes for a given ensamble run.

ixpeobssim.core.pipeline.**reset**(*model_name*, ***kwargs*)

ixpeobssim.core.pipeline.**residual_figure**(*name*)

Create a figure for residual plots.

ixpeobssim.core.pipeline.**save**()

Return the current value of the 'save' rc parameter.

ixpeobssim.core.pipeline.**set_gcf_name**(*name*)

Set the canvas title for the current figure, following the same rules of the figure_name() method.

ixpeobssim.core.pipeline.**set_model**(*model_name*)

Set the pipeline model name.

By default this is the string that will be used, e.g., to create all the file path concatenations.

ixpeobssim.core.pipeline.**setup**(***kwargs*)

Setup the global pipeline params.

ixpeobssim.core.pipeline.**standard_ensamble_processing**(*file_list*, *mc=True*)

Standard processing routine.

ixpeobssim.core.pipeline.**suffix**(*label=None*, *index=None*)

Create a suffix to be appended to file names when the pipeline tools are created.

ixpeobssim.core.pipeline.**target**()

Return the function target for a particular pipeline run.

ixpeobssim.core.pipeline.**xpancrkey**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpbin**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpbinview**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpbkgtemplate**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpchrgmap**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpexposure**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpgrppha**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpmdp**(***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpobssim**(***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpophase**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpphase**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpphotonlist**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xppicorr**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xppimms**(***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpradialprofile**(***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpselect**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpsimfmt**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpsonify**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpstokesalign**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpstokesrandom**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpstokesshuffle**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpstokessmear**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpstripmc**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpvisibility**(**args*, ***kwargs*)

App wrapper.

ixpeobssim.core.pipeline.**xpxspec**(**args*, ***kwargs*)

App wrapper.

Custom random generation classes, mostly based on splines.

class ixpeobssim.core.rand.**xUnivariateAuxGenerator**(*rv*, *aux*, *pdf*, *bbox=None*, *kx=3*, *ky=3*, *xlabel='rv'*, *ylabel='aux'*, *zlabel='pdf'*)

Univariate random generator where the pdf of the random variable depends on an additional auxiliary variable.

Internally, a proper meshgrid of random and auxiliary variable values is created, and the pdf is evaluated on the grid to construct an interpolated bivariate spline, so that a horizontal slice of the bivariate spline at any given value of the auxiliary variable is the actual pdf of the random variable at that value of the auxiliary variable.

Note that the z function argument can either be an array-like object with shape (rv.size, aux.size), or a function that can be evaluated in the bi-dimensional phase space—see the base class xInterpolatedBivariateSpline for the implementation details.

Parameters

- **rv** (*array_like*) – Input values for the random variable (assumed to be sorted).
- **aux** (*array_like*) – Input values for the auxiliary variable (assumed to be sorted).
- **pdf** (*array_like of shape (x.size, y.size) or callable*) – Input pdf values, with shape (x.size,y.size).
- **bbox** (*array_like, optional*) – The boundary of the approximation domain.
- **kx** (*int*) – The spline order on the two axes.
- **ky** (*int*) – The spline order on the two axes.
- **xlabel** (*str, optional*) – A text label for the random variable.
- **ylabel** (*str, optional*) – A text label for the auxiliary variable.
- **zlabel** (*str, optional*) – A text label for the pdf values.

Note

A major backward-incompatible change was introduced here in version 3.0.0, in the context of https://bitbucket.org/ixpesw/ixpeobssim/issues/196, and the constructor signature of the class now reads as xUnivariateAuxGenerator(rv, aux, pdf...), instead of xUnivariateAuxGenerator(aux, rv, pdf...) as it used to be historically in all the previous versions of ixpeobssim.

(On a related note, since some of the arguments are passed by name in the base spline classes—mostly the axis labels—we decided to keep the argument names in synch with those of the base classes.)

The main driver for making this breaking change was that one the main use cases for this class is to generate time-dependent spectra, and the basic signature that we picked in this case, as specified in the documentation, is spec(E, t)—where E (the energy) is the random variable and t (the time) the auxiliary variable. This makes sense, as in most cases the spectra are time-independent, and keeping the energy in the first position allows to give the time a default value, simplifying all the signatures in this fairly common use case. This also makes the signature of the pdf(rv, aux) class method more reasonable.

After the fix to the creation of the underlying meshgrid implemented in https://bitbucket.org/ixpesw/ixpeobssim/commits/cdbd99a7545bb1b0fd09591b2c9b973fb56325d4 insisting on the signature as xUnivariateAuxGenerator(aux, rv, pdf...) would have forced us to swap the spectrum arguments spec(E, t) -> spec(t, E) all over the place (e.g., in srcmodel.spectrum).

(In brief: we have rotated the underlying spline representation by 90 degrees with respect to the historical implementation.)

rvs(*aux*)

Return random variates for a given array of values of the auxiliary variable.

Note that the sample size, here, is determined by the size of the aux argument.

slice(*aux*)

Return the one-dimensional pdf for a given value of the auxiliary variable (i.e., a horizontal slice of the underlying bivariate spline).

class ixpeobssim.core.rand.**xUnivariateGenerator**(*rv*, *pdf*, *w=None*, *bbox=None*, *k=3*, *ext=0*, *xlabel='rv'*, *ylabel='pdf'*)

Univariate random number generator based on a linear interpolated spline.

This class is adding very little on top of the xInterpolatedUnivariateSpline spline class: essentially we are building the ppf in the constructor and we are adding the pdf() method (simply an alias to the underlying __call__() operator, as well as an rvs() method to generate actual random numbers).

Parameters

- **rv** (*(N,) array_like*) – Array of points sampling the values of the random variable.
- **pdf** (*(N,) array_like*) – pdf values at the array rv.
- **w** (*(N,) array_like, optional*) – Weights for spline fitting.
- **bbox** (*(2,) array_like, optional*) – Bounduary of the approximation interval. .
- **k** (*int*) – The order of the spline (<=5, default 3, i.e., a cubic spline).
- **ext** (*int or string, optional*) – Controls the extrapolation mode for elements not in the interval defined by the knot sequence.
- **xlabel** (*str, optional*) – A text label of the random variable.
- **ylabel** (*str, optional*) – A text label for the pdf.

rvs(*size=1*)

Return random variates of arbitrary size.

Parameters
size (*int*) – The number of random numbers to be generated.

rvs_bounded(*size=1*, *rvmin=None*, *rvmax=None*)

Return random variates of arbitrary size between the specified bounds.

This is an alternative to rvs(), where the uniformly-distributed random array passed to the ppf is spanning the appropriate subset of the [0, 1] interval.

Although the default behavior of this method is identical to the standard rvs(), we prefer to have a separate function to avoid unnecessary complexity in what is by far the most common use case.

class ixpeobssim.core.rand.**xUnivariateGeneratorLinear**(*rv*, *pdf*, *ext=0*, *xlabel='rv'*, *ylabel='pdf'*)

Subclass of xUnivariateGenerator restricted to a linear spline.

Spline utility module, building on top of the scipy.interpolate modules.

ixpeobssim.core.spline.**interpolate**(*xa*, *ya*, *xb*, *yb*, *x*)

Simple two-point linear interpolation/extrapolation.

This is a convenience function that is pretty much only used in the optimize_grid_linear() below.

ixpeobssim.core.spline.**optimize_grid_linear**(*x*, *y*, *tolerance=0.0001*)

Optimize a pair of (x, y) arrays for the corresponding spline definition.

This loops over the input arrays and removes unnecessary data points to minimize the length of the arrays necessary to the spline definition.

Parameters

- **x** (*array*) – The input x-array.
- **y** (*array*) – The input y-array.
- **tolerance** (*float*) – The maximum relative difference between the generic yi value and the estrapolation of the two previous optimized data points for the point i to be removed.

class ixpeobssim.core.spline.**xInterpolatedBivariateSpline**(*x*, *y*, *z*, *bbox=None*, *kx=3*, *ky=3*, *xlabel=None*, *ylabel=None*, *zlabel=None*)

Bivariate interpolated spline on a rectangular grid.

This is somewhat similar in spirit to the corresponding univariate base class, except that the additional functionalities are, for the moment, limited to book-keeping and plotting facilities.

One handy facility that this class provides is that of allowing both an array like of shape (x.size, y.size) and a function as the z argument. This makes it possible to construct slides by passing either (x, y, z), as the signature of the underlying scipy's interpolator, and (x, y, f(x, y)), which is easier and more intuitive in most cases.

Parameters

- **x** (*array_like*) – Input x values (assumed to be sorted).
- **y** (*array_like*) – Input y values (assumed to be sorted).
- **z** (*array_like of shape (x.size, y.size) or callable*) – Input z values, with shape (x.size,y.size).
- **bbox** (*array_like, optional*) – The boundary of the approximation domain.
- **kx** (*int*) – The spline order on the two axes.

- **ky** (*int*) – The spline order on the two axes.
- **xlabel** (*str, optional*) – A text label for the quantity on the x-axis.
- **ylabel** (*str, optional*) – A text label for the quantity on the y-axis.
- **zlabel** (*str, optional*) – A text label for the quantity on the z-axis.

build_horizontal_ppf()

Create the horizontal percent point function (or inverse of cdf).

> **Warning**
>
> This really, really need to be fixed. Instead of grabbing a vertical slice at xmean, we should pass an argument to the function so that the subclasses can implement whatever is right for them.

hslice(*y*, *k=3*)

Return an horizontal slice at a given y of the bivariate spline.

Parameters

- **y** (*float*) – The y value at which the horizontal slice should be calculated.
- **k** (*int, optional*) – The degree of the resulting spline.

plot(*num_contours=75*, *logz=False*, ***kwargs*)

Plot the spline.

plot_color(*num_contours=75*, *logz=False*, ***kwargs*)

Color plot

plot_contours(*num_contours=10*, *colors='black'*, *fontsize='small'*, *logz=False*, *cfmt='%1.3f'*)

Contour plot.

scale(*scale_factor*, *zlabel=None*)

Scale the spline z values.

static transposed_meshgrid(*x*, *y*)

Return a suitable meshgrid to evaluate a function in the spline reference system.

This is needed because apparently meshgrid is using the matlab convention for the indices, which is inconsistent with most of the other interfaces in scipy. See: https://github.com/scipy/scipy/issues/3164

vslice(*x*, *k=3*)

Return a vertical slice at a given x of the bivariate spline.

Parameters

- **x** (*float*) – The x value at which the vertical slice should be calculated.
- **k** (*int, optional*) – The degree of the resulting spline.

xmax()

Return the maximum of the underlying x-array.

xmin()

Return the minimum of the underlying x-array.

ymax()

Return the maximum of the underlying y-array.

ymin()

Return the minimum of the underlying y-array.

class ixpeobssim.core.spline.**xInterpolatedBivariateSplineLinear**(*x*, *y*, *z*, *xlabel=None*, *ylabel=None*, *zlabel=None*)

Bivariate linear interpolated spline on a rectangular grid.

class ixpeobssim.core.spline.**xInterpolatedPiecewiseUnivariateSpline**(*x*, *y*, *breakpoints*, *w=None*, *bbox=None*, *k=3*, *ext=0*, *xlabel=None*, *ylabel=None*)

Pieacewise interpolated spline.

In addition to all the regular xInterpolatedUnivariateSpline parameters, this takes an additional breakpoints argument that allows to split the spline creation and evaluation into independent pieces. This is achieved by stories a list of independent splines under the hood.

class ixpeobssim.core.spline.**xInterpolatedUnivariateLogSpline**(*x*, *y*, *w=None*, *bbox=None*, *k=3*, *ext=0*, *xlabel=None*, *ylabel=None*)

Poor man's attempt at a spline in logarithmic space.

The class inherits from both xUnivariateSplineBase and scipy's InterpolatedUnivariateSpline, and the basic strategy, here, is twofold: we initialize xUnivariateSplineBase from the physical x and y values, but we construct an actual interpolator in logarithmic space under the hood, by passing log10(x) and log10(y) to the InterpolatedUnivariateSpline constructor. We then overload the __call__() method by tranforming the argument into its logarithm, performing the interpolation in logarithmic space, and raising the result to the power 10.

The class comes with some pretty obvious shortcomings, among which the difficulty of implementing sensible derivatives and integrals. As far as the latter is concerned, we proceed by brute force and create a second spline, this time in linear space, with a relatively large number of points to guarantee the accuracy of the integral. Admittedly, this is not very elegant nor very roboust. For the derivative issue, we simply do not provide the nu argument in the overloaded __call__ method. For all the other scipy's spline method that is not implemented in this context, we just overload it and raise a NotImplementedError exception.

At some point we even considered removing this class, but the truth is it is handy when one needs a quick way to extrapolate in log-log space. So the class is remaning, with all the caveates above.

antiderivative(*n=1*)

Overloaded method.

derivative(*n=1*)

Overloaded method.

derivatives(*x*)

Overloaded method.

integral(*a*, *b*)

Overloaded integral method.

The integral spline is calculated and cached the first time this method is called.

roots()

Overloaded method.

class ixpeobssim.core.spline.**xInterpolatedUnivariateLogSplineLinear**(*x*, *y*, *w=None*, *bbox=None*, *ext=0*, *xlabel=None*, *ylabel=None*)

Subclass of xInterpolatedUnivariateLogSpline with k=1.

antiderivative(*n=1*)

Overloaded method.

derivative(*n=1*)

Overloaded method.

derivatives(*x*)

Overloaded method.

roots()

Overloaded method.

class ixpeobssim.core.spline.**xInterpolatedUnivariateSpline**(*x*, *y*, *w=None*, *bbox=None*, *k=3*, *ext=0*, *xlabel=None*, *ylabel=None*)

Interpolated spline (i.e., passing through all the input points).

class ixpeobssim.core.spline.**xInterpolatedUnivariateSplineLinear**(*x*, *y*, *w=None*, *bbox=None*, *ext=0*, *xlabel=None*, *ylabel=None*)

Interpolate linear (k=1) spline.

class ixpeobssim.core.spline.**xStepFunction**(*x*, *y*, *xlabel=None*, *ylabel=None*)

Small convenience class describing a step function.

The basic rule, here, is that the value of the function between x[i] and x[i + 1] is exactly y[i].

Parameters

- **x** (*array_like*) – The array of x values.
- **y** (*array_like*) – The array of y values, with shape (N,), assuming that (N + 1,) is the x shape.
- **xlabel** (*str*) – The text label for the x axis.
- **ylabel** (*str*) – The text label for the y axis.

plot(*annotate=False*, *fmt='%.2f'*)

Plot the step function.

See the interesting discussion of a related problem at https://stackoverflow.com/questions/5347065 (For the reference, this is Will's solution.)

class ixpeobssim.core.spline.**xUnivariateSpline**(*x*, *y*, *w=None*, *bbox=None*, *k=3*, *s=None*, *ext=0*, *xlabel=None*, *ylabel=None*, *log=False*)

Wrapper around scipy's UnivariateSplineClass.

The basic idea is to keep track of the original arrays passed to the interpolator and to support arithmetic operations and plotting. We also allow the user to supply optional arguments to control the ranges and specify labels (e.g., names and units) for the quantities involved. We essentially maintain the original signature of the function, adding two optional arguments (xlabel and ylabel) at the end.

Note

Note that the interface to the base class has changed from numpy 0.14. An *ext* argument can be passed to the constructor starting with scipy 0.15 to control the extrapolation behavior and a *check_finite* argument is available in 0.16 to avoid *nans* in the input data. We currently do not expose either one and rely on the default parameters for backward compatibility.

Parameters

- **x** (*(N,) array_like*) – Input x values for the spline.
- **y** (*(N,) array_like*) – Input y values for the spline.
- **w** (*(N,) array_like, optional*) – Weights for spline fitting.
- **bbox** (*(2,) array_like, optional*) – Bounduary of the approximation interval. .
- **k** (*int*) – The order of the spline (<=5, default 3, i.e., a cubic spline).
- **s** (*float or None*) – The spline smoothing factor (o means no smoothing).
- **ext** (*int or string, optional*) – Controls the extrapolation mode for elements not in the interval defined by the knot sequence.
- **xlabel** (*str, optional*) – A text label for the quantity on the x-axis.
- **ylabel** (*str, optional*) – A text label for the quantity on the y-axis.
- **log** (*bool, default False*) – Flag to initialize the underline scipy spline in logarithmic space.

build_cdf(*ylabel='cdf'*, *spline_class=None*)

Create the cumulative distribution function.

Parameters

- **spline_class** (*class*) – The specific spline class (e.g., xInterpolatedUnivariateSpline) used to build the cdf
- **ylabel** (*str*) – The (optional) label for the y axis

Note

At this point the dafault behavior is to use a linear spline for the cdf by default. This is mainly for historical reasons, but maybe a better option would be to use a cubic spline, or to delegate this to call self.__class__?

Warning

Re-enable the check on y > 0!

build_ppf(*xlabel='q'*, *spline_class=None*)

Create the percent point function (or inverse of the cdf).

See the docstring for the build_cdf() method for more details.

inverse(*xmin=None*, *xmax=None*, ***kwargs*)

Calculate the inverse of the spline.

Note that the xmin and xmax arguments allow to invert the spline in a subset of its values.

Parameters

- **xmin** (*float*) – The minimum value for the x-axis of the inverse spline (or y-axis of the original one).

- **xmax** (*float*) – The maximum value for the x-axis of the inverse spline (or y-axis of the original one).

norm()

Return the integral over the entire spline domain.

plot(*num_points=1000*, *overlay=False*, *logx=False*, *logy=False*, *scale=1.0*, *offset=0.0*, *grids=False*, ***kwargs*)

Plot the spline.

Parameters

- **num_points** (*int, optional*) – The number of sampling points to be used to draw the spline.
- **overlay** (*bool, optional*) – If True, the original arrays passed to the spline are overlaid.
- **logx** (*bool, optional*) – If True, the spline is sampled and plotted with the log scale on the x axis.
- **logy** (*bool, optional*) – If True, the spline is plotted with the log scale on the y axis.
- **scale** (*float, optional*) – Optional scale factor for plotting (useful for overlaying splines with different ranges on the y axis).
- **offset** (*float, optional*) – Optional offset for plotting (useful for overlaying splines with different ranges on the y axis).
- ****kwargs** (*dict*) – Keyword arguments passed directly to the matplotlib.plot() method.

scale(*scale_factor*, ***kwargs*)

Return a different spline whose y values are scaled by a given factor wrt the original one.

xmax()

Return the maximum of the underlying x-array.

Note this works because the input array is sorted.

xmin()

Return the minimum of the underlying x-array.

Note this works because the input array is sorted.

class ixpeobssim.core.stokes.**xDataStokesParameters**

Small utility class to deal with the Stokes parameters in a data analysis context.

> **Warning**
>
> This is work in progress, and the class methods are known to fail by ZeroDivisionError in some circumstances.

static polarization_angle(*Q*, *U*, *dQ*, *dU*)

Return the polarization angle and proparate the uncertainties.

static polarization_degree(*I*, *Q*, *U*, *dI*, *dQ*, *dU*)

Return the polarization degree and proparate the uncertainties.

class ixpeobssim.core.stokes.**xModelStokesParameters**

Small utility class to deal with the Stokes parameters in a source model context.

Basically we provide conversion functions from Stokes parameters to polarizarion degree and angle and vice-versa.

Note that all the algebra, here, is coded in terms of the reduced Stokes parameters q = Q / I and u = U / I. The reason is twofold:

- when we simulate a model we typically decouple the definition of the spectrum from that of the polarization pattern;
- when converting polarization degree and angle to Stokes parameters we can only calculate, by definition, q and u—not Q and U.

For completeness: be aware that all the angles are measured in radians, and if you want to operate with degrees it is the user's responsibility to do the conversion outside this class.

static normalize(*QU*, *I*)

Calcualate the Stokes normalized Q parameter.

static pdpa_to_xy(*pol_deg*, *pol_ang*, *degrees=False*)

Convert polarization degree and angle into the x and y components of the polarization vector.

This is assuming that the position angle is measured starting from the celestial North, see https://bitbucket.org/ixpesw/ixpeobssim/issues/597 for more discussion about this.

> **Warning**
>
> This nd the following function are encapsulating our convention for measuring position angles, and should be probably better suited in a different module?

static polarization_angle(*q*, *u*)

Convert q and u to the corresponding polarization angle (in radians).

static polarization_degree(*q*, *u*)

Convert q and u to the corresponding polarization degree.

static q(*polarization_degree*, *polarization_angle*)

Convert a polarization degree and angle (in radians) into the corresponding q reduced Stokes parameter.

static qu_to_xy(*q*, *u*)

Conver the Stokes parameters into the x and y components of the polarization vector.

static u(*polarization_degree*, *polarization_angle*)

Convert a polarization degree and angle (in radians) into the corresponding u reduced Stokes parameter.

29.3 Binned data products

Base classes for binned data products.

ixpeobssim.binning.base.**broadcast_to_map_shape**(*v*, *shape*)

Broadcast an input array-like *v* to the target 3D *shape*.

If broadcasting fails, returns an array of zeros with the requested shape.

ixpeobssim.binning.base.**peek_binning_algorithm**(*file_path*)

Open a binned file and peek at the underlying algorithm

class ixpeobssim.binning.base.**xBinnedFileBase**(*file_path*)

Base class for binned files.

backscal()

Return the value of the BACKSCAL header keyword, if present.

binning_algorithm()

Return the binning algorithm used to create the file.

du_id()

Return the DU ID for the binned file.

Added in version 12.0.0.

> **Warning**
>
> This was added in version 12 of ixpeobssim after https://bitbucket.org/ixpesw/ixpeobssim/issues/327

classmethod from_file_list(*file_list*)

Method to instance the binned class from a list of files.

ontime(*scale=0.001*)

Return the nominal ontime for the binned file.

Parameters

scale (*float*) – A scale factor, by default 1.e-3 (i.e., ontime in ks).

plot(**args*, ***kwargs*)

Do nothing method.

set_data(*name*, *value*)

Add a (key, value) pair to the underlying __data_dict class member.

> **Warning**
>
> This is yet another side effect of the perverse __getattr__ / __setattr__ mechanism we have put in place for this class—if we calculate a quantity dinamically for a binned object and we still want to be able to access it with the same rules, we have to manually add it to the dictionary.

write(*file_path*, *overwrite=True*)

Write the binned data to the output file.

class ixpeobssim.binning.base.**xEventBinningBase**(*file_path*, ***kwargs*)

Base class for the event binning.

This is essentially opening an event file and keeping track of the xEventFile object, along all the keyword arguments passed to the constructor.

bin_()

Do-nothing method to be reimplemented in the derived classes.

static bin_centers(*bin_edges*)

Return an array of bin centers given an array of bin edges.

Parameters
bin_edges (*1-d array of length (n + 1).*) – The array with the bin edges.

Returns
The array with the values of the bin centers.

Return type
1-d array of length n.

static bin_widths(*bin_edges*)

Return an array of bin widths given an array of bin edges.

Parameters
bin_edges (*1-d array of length (n + 1).*) – The array with the bin edges.

Returns
The array with the values of the bin widths.

Return type
1-d array of length n.

build_primary_hdu(*data=None*, *header=None*)

Build the primary HDU for the output file.

check_pcube_weighting_scheme(*aeff*)

Simple check on the weighting scheme being used.

For more background information, see https://bitbucket.org/ixpesw/ixpeobssim/issues/573 and https://bitbucket.org/ixpesw/ixpeobssim/issues/613

static equipopulated_binning(*num_bins*, *data*, *min_val=None*, *max_val=None*)

Create an equipopulated binning based on the values of a generic data column.

Parameters

- **num_bins** (*int*) – The number of bins
- **data** (*array*) – The underlying data to be used for the binning
- **min_val** (*float (optional)*) – The minimum data value to be used for the binning
- **max_val** (*float (optional)*) – The maximum data value to be used for the binning

get(*key*, *default=None*)

Convenience method to address the keyword aguments.

load_aeff_for_polarization_analysis()

Load the proper arf file for a model-independent polarization analysis (i.e., to be used for polarization cubes, maps and map cubes).

This is loading the proper arf file making sure that, when weights are used, the SIMPLE weighting prescription is picked.

static make_binning(*bin_alg*, *min_val=None*, *max_val=None*, *num_bins=None*, *bin_list=None*, *bin_data=None*, *bin_file=None*)

Generic function to define a binning with several possible different algorithms.

static make_energy_binning(*bin_data=None*, ***kwargs*)

Small convenience function for energy binning.

process_image_ref_kwargs()

Set the xref and yref values for the output image (in sky) coordinates.

If either xref or yref are not specifiied, the center of the ROI specified in the proper extension is assumed.

process_kwargs()

Check the keyword arguments and if the output file is not set, create a path on the fly, based on the event file and the binning algorithm.

static read_binning_from_file(*file_path*)

Read a custom binning from file and return a numpy array.

set(*key*, *value*)

Convenience method to set keyword arguments.

weight_data()

Retrieve the weights from the underlying event file.

This encapsulates the simple logic used downstream by the binning algorithms supporting weights, i.e., if the command-line argument *weights* is False the weights are all one, while if it is True returns they are the content of the column indicated by the *weightcol* command-line argument.

write_output_file(*hdu_list*, *overwrite=True*)

Write the binned data to the output file.

Binning data products in detector coordinates.

class ixpeobssim.binning.detector.**xBinnedAreaEnergyFluxMap**(*file_path*)

Display interface to binned EFLUX files.

class ixpeobssim.binning.detector.**xBinnedAreaRateMap**(*file_path*)

Display interface to binned ARMAP files.

plot()

Plot the data.

class ixpeobssim.binning.detector.**xEventBinningARMAP**(*file_path*, ***kwargs*)

Class for ARMAP binning.

bin_()

Overloaded method.

process_data()

Convenience function factoring out the code in common with the corresponding EFLUX class—see the overloaded method in there.

Here we are binning the event position in detector coordinates and dividing by the livetime and the bin area. The function returns a n x n array of area rate values to be written in the output file.

class ixpeobssim.binning.detector.**xEventBinningEFLUX**(*file_path*, ***kwargs*)

Class for EFLUX binning.

process_data()

Overloaded method.

Here we are just multiplying by the average measured event energy.

facilities for exposure calculation.

ixpeobssim.binning.exposure.**create_psf_kernel**(*psf*, *pixsize*, *npix=21*)

Generate a binned kernel for the PSF convolution.

> **Warning**
>
> There is potential overlap with the calculate_psf_kernel() function in the evt.deconvolution module, but the latter is doing something slightly different—a Monte Carlo integration over the pixels, rather than a direct evaluation at the pixel center. We should probably think about whether which one we want.

class ixpeobssim.binning.exposure.**xBinnedLivetimeCube**(*file_path*)

Read-mode interface to a LTCUBE FITS file.

TODO: Temporarily most of this class is just a copy of xBinnedMDPMapCube.
We should refactor the common methods in a separate class.

ds9_region_mask(*region_list*, *region_slice=None*)

Return the (spatial) array map corresponding to a given ds9 region list.

If there is more than one region in the region list, by default the output mask corresponds to the logical or of all the regions. The region_slice optional argument allows to restrict the mask to a subset of the regions.

Parameters

- **region_list** – The region list defining the mask
- **region_slice** (*int or slice (optional)*) – An optional slice designator to select a subset of the region list.

map_shape()

Return the shape of the underlying sky-maps.

Mind the underlying arrays are all 3-dimensional cubes with the theta binning as the first dimension—so it's the last two that we care about. (And, for completeness: since the arrays are all the same, we use I for convenience.)

num_theta_layers()

Return the number of theta layers in the cube.

pixel_size()

Return the pixel size of the underlying spatial map.

plot(*prefix=None*)

Plot the data.

plot_livetime_map(*theta_layers=None*, *prefix=None*, ***kwargs*)

Plot the MDP map.

sum_pixels(*spatial_mask*, *theta_layer=0*)

Sum the relevant quantities over a given spatial mask and theta slice.

theta_binning()

Return the underlying theta binning in the form of a numpy array.

Note the array has one more element than the underlying THETA_LO and THETA_HI arrays, i.e., if the cube is binned in two theta layers, say 0–2 arcmin and 2–4 arcmin, the theta binning is [0. 2. 4.].

theta_centers()

Return the centers of the binning over theta.

theta_label(*theta_layer*)

Return a text label corresponding to the theta layer (e.g, to identify the theta range on a plot).

theta_range(*theta_layer*)

Return the theta value for a given theta layer.

class ixpeobssim.binning.exposure.**xEventBinningLTCUBE**(*file_path*, ***kwargs*)

Class for LTCUBE binning.

bin_()

Overloaded method.

process_kwargs()

Overloaded method.

class ixpeobssim.binning.exposure.**xExposureCube**(*exposure*, *ebounds=None*, *units=None*)

Structure for writing/reading exposure cubes.

classmethod empty(*shape*)

Create an empty exposure cube.

classmethod from_file(*file_path*)

Read an exposure cube from file.

write(*file_path*, *header*, *overwrite*)

Write the exposure cube to file.

> **Warning**
>
> Need to add all the keywords from the LTCUBE file

Format definitions for binned data products.

class ixpeobssim.binning.fmt.**xBinTableHDUEBOUNDS**(*data=None*, *keywords=None*, *comments=None*)

Binary table for storing energy bounds.

> **Warning**
>
> Can we just reuse the same extension from the response matrix?

class ixpeobssim.binning.fmt.**xBinTableHDULC**(*data=None*, *keywords=None*, *comments=None*)

Binary table for binned LC data.

class ixpeobssim.binning.fmt.**xBinTableHDUPCUBE**(*data=None*, *keywords=None*, *comments=None*)

Binary table for binned PCUBE data.

class ixpeobssim.binning.fmt.**xBinTableHDUPHA1**(*data=None*, *keywords=None*, *comments=None*)

Binary table for binned PHA1 data.

class ixpeobssim.binning.fmt.**xBinTableHDUPP**(*data=None*, *keywords=None*, *comments=None*)

Binary table for binned PP data.

class ixpeobssim.binning.fmt.**xBinTableHDUTHETABOUNDS**(*data=None*, *keywords=None*, *comments=None*)

Binary table for storing off-axis angle bounds.

Miscellanea binned products.

class ixpeobssim.binning.misc.**xBinnedLightCurve**(*file_path*)

Binned light curve.

plot(*mjd=False*, ***plot_opts*)

Overloaded plot method.

rate()

rate_error()

class ixpeobssim.binning.misc.**xBinnedMap**(*file_path*)

Display interface to binned CMAP files.

While this is essentially an xFITSImage, we need to inherit from xBinnedFileBase, as the latter provides the implementation of the from_file_list() slot. Instead of inheriting from xFITSImage, too, we preferred composition, instead.

> **Warning**
>
> I am more and more convinced that this is poor design, and maybe the pylint super-init-not-called is there for a reason. We should probably refactor the class and make the implementation cleaner.

average()

plot(***kwargs*)

Plot the data. The kwargs passed to plt.imshow().

class ixpeobssim.binning.misc.**xBinnedPulseProfile**(*file_path*)

Binned pulse profile.

plot(***kwargs*)

Overloaded plot method.

class ixpeobssim.binning.misc.**xEventBinningCMAP**(*file_path*, ***kwargs*)

Class for CMAP binning.

bin_()

Overloaded method.

process_kwargs()

Overloaded method.

class ixpeobssim.binning.misc.**xEventBinningLC**(*file_path*, ***kwargs*)

Class for LC binning.

bin_()

Overloaded method.

make_binning()

Build the light-curve binning.

process_kwargs()

Overloaded method.

class ixpeobssim.binning.misc.**xEventBinningPP**(*file_path*, ***kwargs*)

Class for pulse-profile binning.

bin_()

Overloaded method.

make_binning()

Build the light-curve binning.

Binned data structures pertaining to the (spectro-)polarimetric analysis.

class ixpeobssim.binning.polarization.**xBinnedCountSpectrum**(*file_path*)

Binned count spectrum.

plot(***kwargs*)

Overloaded plot method.

respfile()

Return the value of the RESPFILE keyword in the SPECTRUM extension header, stripped from the first part of the pathe (i.e., the file name only).

class ixpeobssim.binning.polarization.**xBinnedMDPMapCube**(*file_path*)

Read-mode interface to a MDPMAPCUBE FITS file.

average()

Compute the average over the map, then create an identical map filled with the average value.

ds9_region_mask(*region_list*, *region_slice=None*)

Return the (spatial) array map corresponding to a given ds9 region list.

If there is more than one region in the region list, by default the output mask corresponds to the logical or of all the regions. The region_slice optional argument allows to restrict the mask to a subset of the regions.

Parameters

- **region_list** – The region list defining the mask
- **region_slice** (*int or slice (optional)*) – An optional slice designator to select a subset of the region list.

energy_binning()

Return the underlying energy binning in the form of a numpy array.

Note the array has one more element than the underlying ENERG_LO and ENERG_HI arrays, i.e., if the cube is binned in two energy layers, say 2–4 keV and 4–8 keV, the energy binning is [2. 4. 8.].

energy_label(*energy_layer*)

Return a text label corresponding to the energy layer (e.g, to identify the energy range on a plot).

energy_range(*energy_layer*)

Return the energy range, i.e., (emin, emax) in keV, for a given energy layer.

map_shape()

Return the shape of the underlying sky-maps.

Mind the underlying arrays are all 3-dimensional cubes with the energy binning as the first dimension—so it's the last two that we care about. (And, for completeness: since the arrays are all the same, we use I for convenience.)

num_energy_layers()

Return the number of energy layers in the cube.

pixel_size()

Return the pixel size of the underlying spatial map.

plot(*prefix=None*)

Plot the data.

plot_mdp_map(*energy_layers=None*, *prefix=None*, ***kwargs*)

Plot the MDP map.

sum_pixels(*spatial_mask*, *energy_layer=0*)

Sum the relevant quantities over a given spatial mask and energy slice.

class ixpeobssim.binning.polarization.**xBinnedPolarizationCube**(*file_path*)

Read-mode interface to a PCUBE FITS file.

Added in version 12.0.0.

as_table()

Return the polarization cube as an astropy table.

plot(***kwargs*)

Default plotting for the polarization cubes.

> **Warning**
>
> This should be considered in evolution, and the API might change.

This is plotting the polarization cube in normalized Stokes parameters space, with custom grids to help reading out the polarization degree and angle.

A brief explanation of the supporte keywords arduments.

Parameters

- **side** (*float, optional*) – The absolute value of the maximum Q/I and U/I to be displayed (note the aspect ratio of the plot is set to 'equal', and the plot is assumed to be squared). By default this is driven by the maximum value of the polarization degree over the polarization cube.
- **pd_grid** (*array_like*) – The polarization degree radial grid in correspondence of which the gridding circumpherences are plotted to guide the eye.
- **pd_grid_label_angle** (*float*) – The angle at which the text labels for the PD grids are plotted.
- **pa_grid_step** (*float*) – The step of the tangential grid in the polarization angle.

- **sigma_levels** (*array_like*) – The sigma levels at which the ellipses repesenting the normalized Stokes parameter contours are plotted.
- **sigma_ls** (*tuple of the same size as sigma_levels*) – The line styles corresponding to sigma levels—by default the innermost ellipse is solid and the others are dashed.
- **colors** (*array_like or str, optional*) – The colors for the elliptical contours (by deafult the proper color for the specific DU in the polarization cube is picked).
- **label** (*str, optional*) – Optional label to be displayed in the legend—this is only used if the marker boolean flag is True and only associated to the first layer in the cube.
- **marker** (*bool*) – If True, a marker is plotted at the center of the elliptical contours.
- **marker_size** (*float*) – The size for the optional markers.
- **annotate_energies** (*bool*) – If true, the energy layers are annotated with nice arrows and text labels (note the positioning is still fragile).
- **setup_axes** (*bool*) – Call the underlying setup_gca_stokes() hook at the end. (This flag is provided to allow disingaging multiple grid plotting in case one wants to overlay multiple polarization cubes.)

plot_polarization_angle(***kwargs*)

Plot the polarization angle as a function of energy.

> **Warning**
>
> This is deprecated in favor of the standard plots of the Stokes parameters.

plot_polarization_degree(*min_num_sigmas=2.0*, ***kwargs*)

Plot the polarization degree as a function of energy.

> **Warning**
>
> This is deprecated in favor of the standard plots of the Stokes parameters.

polarization()

Return the polarization information in the form of a four element tuple (PD, PD_err, PA, PA_err) of arrays.

class ixpeobssim.binning.polarization.**xBinnedPolarizationMapCube**(*file_path*)

Read-mode interface to a PMAPCUBE FITS file.

align()

Align the polarization direction to the radial direction on a bin by bin basis.

> **Warning**
>
> This will need to be reviewed when we fix issue #597.

average()

Compute the average over the map, then create an identical map filled with the average value, broadcasting everything to the map shape.

calculate_significance_mask(*num_sigma=2.0*, *intensity_percentile=0.0*)

Utitlity function to calculate a mask on the underlying pixel cube where the polarization degree and angle can be reliably plotted.

The calculation is based on two different metrics:

- the value of I in each given pixel, compared with a given percentile of the overall I distribution across the cube;
- the number of sigma that the measured polarization degree differs from either 0 or 1 (if the measurement is too close to 0 or 1 and misses the physical bounds by less than a few error bars, then it is effectively not a measurement).

Note the logic is positive—i.e., we're passing the mask identifying the *good* pixels. This is typically negated downstream to set to zero all the other pixels.

Parameters

- **num_sigma** (*float*) – The number of standard deviations representing the significance of the measurement of the polarization degree.
- **intensity_percentile** (*float*) – the threshold on the I value for the pixels, based on the percentile of the I distribution across all the pixels.

convolve(*kernel*)

Convolve the polarization cube with a generic binned kernel.

normalized_stokes_parameters()

Calculate the normalized Stokes parameters, i.e., Q/I and U/I.

plot(*prefix=None*)

Plot everything.

plot_normalized_stokes_parameters(*energy_layers=None*, *num_sigma=1.0*, *intensity_percentile=0.05*, *prefix=None*, ***kwargs*)

Plot the normalized Stokes parameters.

plot_polarization_angle(*energy_layers=None*, *num_sigma=2.0*, *prefix=None*, ***kwargs*)

Plot the polarization angle.

plot_polarization_degree(*energy_layers=None*, *num_sigma=2.0*, *arrows=True*, *prefix=None*, ***kwargs*)

Plot the polarization degree.

plot_significance(*energy_layers=None*, *prefix=None*, ***kwargs*)

Plot the significance.

plot_stokes_parameters(*energy_layers=None*, *prefix=None*, ***kwargs*)

Plot the Stokes parameters.

radial_profile(*n=10*, *layer=0*)

Create a radial polarization profile of the source.

> **Warning**
>
> This needs to be cleaned up and properly documented.

save_to_ds9(*output_file_path*, *num_sigma=2.0*, *intensity_percentile=0.0*, *line_color='white'*)

Save the polarization arrows to a ds9 region.

sum_pixels(*spatial_mask*, *energy_layer=0*)

Sum the relevant quantities over a given spatial mask and energy slice.

class ixpeobssim.binning.polarization.**xEventBinningMDPMAP**(*file_path*, ***kwargs*)

Class for MDPMAP binning.

process_kwargs()

Overloaded method.

Note we have to call the method of the base class closest in the inheritance hierarchy, to make sure that all the ingredients for the WCS object generation are properly setup.

class ixpeobssim.binning.polarization.**xEventBinningMDPMAPCUBE**(*file_path*, ***kwargs*)

Class for MDPMAPCUBE binning.

bin_()

Overloaded method.

process_kwargs()

Overloaded method.

class ixpeobssim.binning.polarization.**xEventBinningPCUBE**(*file_path*, ***kwargs*)

Class for PCUBE binning.

Added in version 12.0.0.

bin_()

Overloaded method.

class ixpeobssim.binning.polarization.**xEventBinningPHA1**(*file_path*, ***kwargs*)

Original algorithm for PHA1 files.

binned_data()

Overloaded binning algorithm.

class ixpeobssim.binning.polarization.**xEventBinningPHA1Base**(*file_path*, ***kwargs*)

Base class for PHA1 binning.

bin_()

Overloaded method.

binned_data()

Method to be overloaded by derived classes.

class ixpeobssim.binning.polarization.**xEventBinningPHA1Q**(*file_path*, ***kwargs*)

Class for PHA1 binning.

binned_data()

Overloaded binning algorithm.

class ixpeobssim.binning.polarization.**xEventBinningPHA1QN**(*file_path*, ***kwargs*)

Subclass for creating normalized Stokes Q spectra.

binned_data()

Overloaded binning algorithm.

class ixpeobssim.binning.polarization.**xEventBinningPHA1U**(*file_path*, ***kwargs*)

Subclass for creating Stokes U spectra.

binned_data()

Overloaded binning algorithm.

class ixpeobssim.binning.polarization.**xEventBinningPHA1UN**(*file_path*, ***kwargs*)

Subclass for creating normalized Stokes U spectra.

binned_data()

Overloaded binning algorithm.

class ixpeobssim.binning.polarization.**xEventBinningPMAP**(*file_path*, ***kwargs*)

Class for PMAP binning.

process_kwargs()

Overloaded method.

Note we have to call the method of the base class closest in the inheritance hierarchy, to make sure that all the ingredients for the WCS object generation are properly setup.

class ixpeobssim.binning.polarization.**xEventBinningPMAPCUBE**(*file_path*, ***kwargs*)

Class for PMAPCUBE binning.

29.4 Event-level analysis

Facilities for the rotation of the polarization angle/Stokes parameters given an input model.

This module provides a series of functions facilitating the search for large-scale polarization signatures, such as radial/tangential polarization in extended sources such SNRs.

ixpeobssim.evt.align.**align_phi**(*phi*, *phi0*)

Rotate the photoelectron azimuthal angle and recalculate the event-by-event Stokes parameters.

Parameters

- **phi** (*array_like*) – The azimuthal angle for the input event list
- **phi0** (*array_like*) – The model polarization direction, calculated at the positions of the input events.
- **warning::** (..) – As we moved away from the used of azimuthal angles in the analysis, this function was added to the module for backward compatibility and testing purposes, but should no be used. Use align_stokes_parameters() below, instead.

ixpeobssim.evt.align.**align_stokes_parameters**(*q*, *u*, *q0*, *u0*)

Align the Stokes parameters according to an input polarization model.

Parameters

- **q** (*array_like*) – The event-by-event Q Stokes parameters.
- **u** (*array_like*) – The event-by-event U Stokes parameters.
- **q0** (*array_like*) – The input model Q Stokes parameters, calculated at the positions of the input events.
- **u0** (*array_like*) – The input model U Stokes parameters, calculated at the positions of the input events.

Animation utilities.

class ixpeobssim.evt.animate.**xMovingCircle**(*points*, *radius=25.0*, *kxy=1*, *kr=1*)

Small convenience class representing a time-dependent circular patch.

The moving patch is initialized given a set of (t, x, y) points, referred to the center of the target image. The interpolation is done via interpolating splines of order 1.

Parameters

- **points** (*iterable of three-elements (t, x, y) tuples*) – The points defining the path of the circle as a function of time. (The time is in seconds and the xy position is expressed as the delta with respect to the center of the image in arcminutes.)
- **radius** (*the circle radius in arcseconds*)

event_mask(*t*, *ra*, *dec*, *ra0*, *dec0*)

Return a boolean mask for a series of events.

Parameters

- **t** (*array_like*) – The array of event times.
- **ra** (*array_like*) – The array of the event R.A. positions.
- **dec** (*array_like*) – The array of the event Dec. positions.
- **ra0** (*float*) – The R.A. position of the target image center.
- **dec0** (*float*) – The Dec. position of the target image center.

frame_positions(*interval=100*)

Return a set of frame positions, referred to the center of the target image, for given animation interval.

The positions are in the form (dx, dy, radius), where dx and dy are the distances to the image center along the two orthogonal axes, in arcmin, and radius is the circle radius in arcseconds—this has to match the signature of the xSkyAnimation.show_roi() method, which is used as the updating slot of the animation.

See https://matplotlib.org/stable/api/animation_api.html for more information about the matplotlib animation APIs.

Parameters

interval (*the animation interval in ms.*)

class ixpeobssim.evt.animate.**xSkyAnimation**(*file_path*)

Small convenience class to describe an animation in sky coordinates.

add_roi(*roi*, ***kwargs*)

Add the ROI.

plot(***kwargs*)

Plot the underlying image.

Note that, in addition to dispatch the plot() call to the underlying xFITSImageBase object, this is overriding the return value, returning an AxesImage object, rather than the standard figure. This is needed to properly build the animation video.

run(*roi*, *interval=100*, *figsize=(12.0, 12.0)*)

Run the actual animation.

See https://matplotlib.org/stable/api/animation_api.html for more information about the matplotlib animation APIs.

See also https://holypython.com/how-to-save-matplotlib-animations-the-ultimate-guide/

Parameters

interval (*the animation interval in ms.*)

show_frame(*roi=(0.0, 0.0, 25.0)*, ***kwargs*)

Draw a circular ROI with a given position and radius.

Parameters

roi (*three-element tuple*) – The position and radius of the circular ROI in the form of a three-element tuple (dx, dy, radius).

Module encapsulating image deconvolution facilities.

ixpeobssim.evt.deconvolution.**calculate_inverse_kernel**(*intensity*, *true_intensity*, *kernel*)

Calculate the gargantuan (nx x ny x nk x nk) matrix that allows to deconvolve an (I, U or Q) image, given a proxy of the true intensity.

The basic idea, here, is that if you have a sensible proxy of the true I map (e.g., a high-resolution image from Chandra) you can use it to gauge the fraction of the measured intensity ending up in each true pixel, and you can actually use that to deconvole a map (with the same binning) of either Q or U. Phrased in a slighlty different way, while the PSF kernel for the direct true -> measured convolution is identical for all pixels, the kernel for the inverse measured -> true deconvolution is in general different for all the pixels and depends on the true intensity. This is why we need to precompute a store a specific kernel for each pixels. This can be in turn use to deconvole any measured quantity with the same spatial binning (Q, U, W2).

Warning

I am not positive this is right for the border pixels, where the kernel mask ends up out of the image—need to study that.

Parameters

- **intensity** (*2d array*) – The measured intensity map.
- **true_intensity** (*2d array*) – The proxy for the true intensity map.
- **kernel** (*2d array*) – The PSF kernel for the convolution.

ixpeobssim.evt.deconvolution.**calculate_psf_kernel**(*pixel_size*, *irf_name='ixpe:obssim20240101:v13'*, *du_id=1*, *max_radius=0.03*, *sample_size=5000000*)

Calculate the digital PSF kernel for image convolution.

This is essentially creating a square grid whose pixel size is set to the value of the corresponding function argument and whose half size is (loosely) determined by the max_radius arguments. A number of points is thrown in the grid according to the PSF shape and the corresponding probability for each pixel is estimated by numerical integration.

Parameters

- **pixel_size** (*float*) – The pixels size of the image that the kernel is operating on (in decimal degrees).
- **irf_name** (*str*) – The IRF name.
- **du_id** (*int*) – The DU id.
- **max_radius** (*float*) – The maximum radius (in decimal degrees) for defining the size of the kernel grid. This should be large enough to fully contain the PSF.

- **sample_size** (*int*) – The size of the random sample for the Monte Carlo integration of the psf profile over the square grid.

ixpeobssim.evt.deconvolution.**circular_kernel**(*nside*, *normalize=False*)

Build a simple circular kernel of a given side.

Parameters

- **nside** (*int*) – The side of the kernel array, assumed to be squared.
- **normalize** (*bool*) – If True, normalize to the kernel content (i.e., use for averaging rather than summing).

ixpeobssim.evt.deconvolution.**convolve_image**(*image*, *kernel*)

Convolve an image with the PSF kernel.

While I am sure there exist a more clever (and faster) way to do this with scipy, I retained the explicit Python loop, here, as I don't think this is a bottleneck in any way, and having a full implementation is useful for debugging.

Parameters

- **image** (*2d array*) – The input image to be convolved with the PSF.
- **kernel** (*2d array*) – The PSF kernel for the convolution.

ixpeobssim.evt.deconvolution.**deconvolve_image**(*image*, *kernel*, *K*)

Deconvolve a generic image (e.g., Q or U).

Parameters

- **image** (*2d array*) – The input image to be deconvoled.
- **kernel** (*2d array*) – The PSF kernel.
- **K** (*4d array*) – The inverse pixel-by-pixel kernel for the deconvolution.

Event display facilities.

This module provides a simple, top-level interface to track images in IXPE level-1 files.

class ixpeobssim.evt.display.**Recon**(*absorption_point: tuple[float, float]*, *barycenter: tuple[float, float]*, *track_direction: float*, *length: float*, *width: float*)

Container class encapsulating the event reconstruction.

draw_absorption_point()

Draw the reconstructed absorption point and track direction.

draw_barycenter()

Draw the reconstructed absorption point and track direction.

draw_track_direction(*line_width=1.0*, *length_ratio=0.5*)

Draw the track direction.

ixpeobssim.evt.display.**display_event**(*event*, *grid*, *threshold*, *dbscan*, *file_name=None*, *padding=False*, ***kwargs*)

Single-stop event display.

ixpeobssim.evt.display.**load_event_list**(*file_path*, *pivot_energy=8.0*, *interactive=False*, ***kwargs*)

Load the event data from the Level-2 event list for the purpose of the event display—these include, in order: mission elapsed time, energy, sky position and Stokes parameters.

This function has a few other functionalities, other than just loadinf the relevant colums from the the Level-2 files, and particularly:

- resample the input events with a given power-law spectral function;
- trimming the resampled colums to a target number of events preserving the time ordering and covering evengly the entire time span.

Parameters

- **file_path** (*str*) – The path to the input Level-2 file.
- **pivot_energy** (*float*) – The pivot energy for the resampling of the count spectrum.
- **interactive** (*bool*) – If True, show some debug plot with the output (resampled) spectrum.
- **kwargs** (*dict*) – The keyword arguments from the xDisplayArgumentParser.

class ixpeobssim.evt.display.**xDisplayArgumentParser**(*description*)

Specialized argument parser for the event display and related facilities.

This is placed here because if needs to be used by both the single-event display and the observation carousel.

class ixpeobssim.evt.display.**xDisplayCard**(*target_name*, *header*)

Specialize text card to display event information.

The basic idea, here, is that one initializes the card with the EVENTS header of a Level-2 file, and then updates the information on an event-by-event basis using the set_event_data() hook.

set_event_data(*met*, *energy*, *ra*, *dec*, *q*, *u*, *compact=True*)

Set the event data.

update_cumulative_statistics(*num_events*, *emin*, *emax*)

Set the card line with the basic cumulative statistics info.

class ixpeobssim.evt.display.**xHexagonCollection**(*x*, *y*, *radius=0.05*, *orientation=0.0*, ***kwargs*)

Collection of native hexagin patches.

Parameters

- **x** (*array_like*) – The x coordinates of the hexagon centers.
- **y** (*array_like*) – The y coordinates of the hexagon centers.
- **radius** (*float*) – The hexagon apothem.
- **orientation** (*float*) – The hexagon orientation in radians—zero means pointy topped.
- **kwargs** – The keyword arguments to be passed to the PatchCollection constructor.

set(***, *agg_filter=<UNSET>*, *alpha=<UNSET>*, *animated=<UNSET>*, *antialiased=<UNSET>*, *array=<UNSET>*, *capstyle=<UNSET>*, *clim=<UNSET>*, *clip_box=<UNSET>*, *clip_on=<UNSET>*, *clip_path=<UNSET>*, *cmap=<UNSET>*, *color=<UNSET>*, *edgecolor=<UNSET>*, *facecolor=<UNSET>*, *gid=<UNSET>*, *hatch=<UNSET>*, *in_layout=<UNSET>*, *joinstyle=<UNSET>*, *label=<UNSET>*, *linestyle=<UNSET>*, *linewidth=<UNSET>*, *mouseover=<UNSET>*, *norm=<UNSET>*, *offset_transform=<UNSET>*, *offsets=<UNSET>*, *path_effects=<UNSET>*, *paths=<UNSET>*, *picker=<UNSET>*, *pickradius=<UNSET>*, *rasterized=<UNSET>*, *sketch_params=<UNSET>*, *snap=<UNSET>*, *transform=<UNSET>*, *url=<UNSET>*, *urls=<UNSET>*, *visible=<UNSET>*, *zorder=<UNSET>*)

Set multiple properties at once.

Supported properties are

Properties:

agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array and two offsets from the bottom left corner of the image alpha: array-like or scalar or None animated: bool antialiased or aa or antialiaseds: bool or list of bools array: array-like or None capstyle: *.CapStyle* or {'butt', 'projecting', 'round'} clim: (vmin: float, vmax: float) clip_box: *~matplotlib.transforms.BboxBase* or None clip_on: bool clip_path: Patch or (Path, Transform) or None cmap: *.Colormap* or str or None color: **:mpltype:`color`** or list of RGBA tuples edgecolor or ec or edgecolors: **:mpltype:`color`** or list of **:mpltype:`color`** or 'face' facecolor or facecolors or fc: **:mpltype:`color`** or list of **:mpltype:`color`** figure: *~matplotlib.figure.Figure* gid: str hatch: {'/', '\', '|', '-', '+', 'x', 'o', 'O', '.', '*'} in_layout: bool joinstyle: *.JoinStyle* or {'miter', 'round', 'bevel'} label: object linestyle or dashes or linestyles or ls: str or tuple or list thereof linewidth or linewidths or lw: float or list of floats mouseover: bool norm: *.Normalize* or str or None offset_transform or transOffset: *.Transform* offsets: (N, 2) or (2,) array-like path_effects: list of *.AbstractPathEffect* paths: unknown picker: None or bool or float or callable pickradius: float rasterized: bool sketch_params: (scale: float, length: float, randomness: float) snap: bool or None transform: *~matplotlib.transforms.Transform* url: str urls: list of str or None visible: bool zorder: float

class ixpeobssim.evt.display.**xHexagonalGrid**(*num_cols*, *num_rows*, *pitch=0.05*, ***kwargs*)

Generic hexagonal grid.

Parameters

- **num_cols** (*int*) – The number of columns in the grid
- **num_rows** (*int*) – The number of rows in the grid
- **pitch** (*float*) – The grid pitch in mm.

static brightness(*color*)

Quick and dirty proxy for the brighness of a given array of colors.

See https://stackoverflow.com/questions/9733288 and also https://stackoverflow.com/questions/30820962 for how to split in columns the array of colors.

default_roi_side(*roi*, *min_side*, *pad=0.1*)

Return the default physical size of the canvas necessary to fully contain a given ROI.

draw_event(*event*, *num_clusters=1*, *offset=(0.0, 0.0)*, *min_canvas_side=2.0*, *indices=True*, *padding=True*, *zero_sup_threshold=None*, *values=False*, ***kwargs*)

Draw an actual event int the parent hexagonal grid.

This is taking over where the draw_roi() hook left, and adding the event part.

draw_roi(*roi*, *offset=(0.0, 0.0)*, *indices=True*, *padding=True*, ***kwargs*)

Draw a specific ROI of the parent grid.

pha_to_colors(*pha*, *zero_sup_threshold=None*)

Convert the pha values to colors for display purposes.

pixel_to_world(*col*, *row*)

Transform pixel to world coordinates.

Parameters

- **col** (*array_like*) – The input column number(s).
- **row** (*array_like*) – The input row number(s).

roi_center(*roi*)

Return the world coordinates of the physical center of a given ROI.

Note the small offsets that we apply serves the purpose of taking into account the fact that pixels are staggered in one direction.

static show_display(*file_path=None*, *dpi=100*, *batch=False*)

Convenience function to setup the matplotlib canvas for an event display.

Parameters

file_path (*str*) – Optional file path to save the image immediately before the plt.show() call.

class ixpeobssim.evt.display.**xL1Event**(*min_col: int*, *max_col: int*, *min_row: int*, *max_row: int*, *pha: array*, *trigger_id: int = 0*, *seconds: int = 0*, *microseconds: int = 0*, *timestamp: float = 0.0*, *livetime: int = 0*, *error_summary: int = 0*, *du_status: int = 0*, *recon:* Recon | *None = None*)

A fully fledged event.

This is building up on the core logic encapsulated in the xRegionOfInterest class, and is adding all the necessary event information on top of that, i.e., the pha values, and the time and error information.

Note the dataclass field are largely mapped over the columns of the corresponding EVENTS extension in the underlying FITS files.

highest_pixel(*absolute=True*)

Return the coordinates (col, row) of the highest pixel.

Parameters

absolute (*bool*) – If true, the absolute coordinates (i.e., those referring to the readout chip) are returned; otherwise the coordinates are intended relative to the readout window (i.e., they can be used to index the pha array).

run_clustering(*engine*)

Run the clustering on the track image.

class ixpeobssim.evt.display.**xL1EventFile**(*file_path*)

Simple interface to a level-1 file in FITS format.

Parameters

- **file_path** (*str*) – The path to the input event file.
- **padding** (*Padding instance*) – The ROI padding for the event file.

bisect_met(*met*)

Retrieve a specific event by its mission elapsed time.

Internally this is using a binary search on the time column, and in general it can be assumed that this O(log(N)) in complexity.

value(*col_name*)

Return the value of a given column for a given extension for the current event.

zero_sup_threshold()

Return the zero-suppression threshold, determined from the file header.

class ixpeobssim.evt.display.**xRegionOfInterest**(*min_col: int*, *max_col: int*, *min_row: int*, *max_row: int*)

Class describing a region of interest (ROI).

A region of interest is the datum of the logical coorinates of its two extreme corners, in the order (min_col, max_col, min_row, max_row).

at_border()

Return True if the ROI is on the border for a given chip_size.

column_indices()

Return an array with all the valid column indices.

coordinates_in_roi(*col*, *row*)

Return a boolean mask indicaing whether elements of the (col, row) arrays lie into the ROT area.

coordinates_in_rot(*col*, *row*)

Return a boolean mask indicaing whether elements of the (col, row) arrays lie into the ROT area.

row_indices()

Return an array with all the valid row indices.

serial_readout_coordinates()

Return two one-dimensional arrays containing the column and row indices, respectively, in order of serial readout of the ROI.

Example

```
>>> col, row = xRegionOfInterest(0, 4, 0, 3).serial_readout_coordinates()
>>> print(col)
>>> [0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4]
>>> print(row)
>>> [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3]
```

serial_readout_indices()

Return a two-dimensional array containing the readout index for each pixel of the ROI.

The ASIC serial readout starts from the top-left corner and proceeds one row at a time, i.e., for a toy 5 x 5 window starting at <0, 0> the readout indices look like +————————— | 0 1 2 3 4 | | 5 6 7 8 9 | | 10 11 12 13 14 | | 15 16 17 18 19 | | 20 21 22 23 24

It is worth noting that, provided that one loops over the row indices first and column indices last, the readout index can be determined by just accumulating a counter. This function is useful as it provides the right answer for any position in the event window, no matter what the loop order is.

class ixpeobssim.evt.display.**xXpolGrid**(***kwargs*)

XPOL grid.

Module encapsulating the event structure and related facilities.

ixpeobssim.evt.event.**intersect_gti**(*l2_file_path*, *l1_file_paths*, *gti_starts*, *gti_stops*, *tag='filtered'*)

Create a new observation file selecting only events that fall into the intersection of the initial GTIs and the given GTIs. The livetime is updated by resumming the LIVETIME column in level 1 files - this is not exact, but it should be a reasonable approximation for most uses. NOTE: the reason this is not a xEventFileFriend member function is that it is supposed to work on a single level 2 file, while an xEventFileFriend file can be instantiated with a list of level 2 files.

class ixpeobssim.evt.event.**xBaseEventList**(*time_=None*, *source_id=None*)

Base class for event lists.

This is the base class for objects that is created and filled at simulation time and then written to file in FITS format. It is essentially a dictionary containing all the column data for the output FITS file, providing additional facilities for addding, sorting in time and trimming event lists.

In order to be robust against name changes, subclasses should provide all the necessary interfaces to retrieve and set the column values (i.e., if the name of any of the columns changes, in principle it should not be necessary to make modifications beyond this file). Note that the colums related to times and source ID are encapsulated into this base class.

If the constructor is called with no argument an empty event list is created. Alternatively the user can initilize the event list with an array of event times (we did pick the time because typically this is the first dynamical variable being calculated). In that case the TRG_ID column is also filled right away and, if the optional source_id argument is not None, the SRC_ID as well. For completeness, this mechanism was added because we noticed that there were many calls to xEventList.fill_trigger_id() and xEventList.set_source_id() floating around in different places. (Note that, altough we need to overwrite the trigger identifier values at the end of the loop over the model components, after we have applied the deadtime, we do need to fill the corresponding column at the level of the single component because all the code downstream, e.g., the xEventList.trim() method, need *all* the columns to be present, and have the same length, to operate.)

apply_fiducial_area_base(*detx*, *dety*)

Trim out the events falling outside the detector active area.

Note that sub-classes are responsible for providing their own definition of the detector coordinates to apply the function.

apply_vignetting_base(*ra*, *dec*, *energy*, *vign*, *ra_pnt*, *dec_pnt*)

Base function to apply the effective area vignetting to the list.

Note that sub-classes are responsible for providing their own definition of the sky coordinates and energy to apply the function.

> **Note**
>
> Note that angular_separation returns values in degrees, while the vignetting function is expressed in terms of the off-axis angle expressed in arcminutes. See also https://bitbucket.org/ixpesw/ixpeobssim/issues/423

static array_is_sorted(*array*)

Return whether a given array is sorted.

This is an interesting topic per se, see https://stackoverflow.com/questions/47004506 I dropped this here assuming that we're not concerned by adding a step which is O(n) from an algorithmic standpoint.

empty()

Return whether the event list is empty.

num_events()

Return the number of events in the event list.

set_source_id(*src_id*)

Set the source ID.

set_time_columns(*time_*, *check_sorted=True*)

Populate the time-related columns.

sort()

Sort the event list based on the event time.

static split_event_time(*time_*)

Split the event time into its interger and fractional part.

While in hardware we will be measuring the seconds and microseconds separately, and then build the floating-point representation of the event time by summing the two, in the simulation realm we do the opposite, i.e., we extract the event times as floating-point numbers and then reverse-engineer the integral and fractional parts.

> **Warning**
>
> This is a crude implementation of the concept, where we essentially take the floor of the integral and fractional parts of the event time (modulo the conversion from seconds to microseconds for the second). In the future we might make this more complicated if we need to simulate the instrument timing in a more detailed fashion.

src_id()

Return the source ID.

time()

Return the event times.

trim(*mask*)

Trim all the data columns according to a generic input mask.

class ixpeobssim.evt.event.**xEventFile**(*file_path*)

Read/write interface to level-2 event files.

This is the main class providing access to the information into a FITS file containing an event list.

It supports minimal output facilities, such as adding columns and writing to file.

add_column(*ext_name*, *col_name*, *col_format*, *col_data*)

Add a column to the specified extension of the event file.

Parameters

- **ext_name** (*str*) – The name of the extension that the column should be added to.
- **col_name** (*str*) – The name of the column to be added.
- **col_format** (*str*) – The format for the new column (e.g., 'E').
- **col_data** (*array_like*) – The data for the new column.

add_columns(*ext_name*, **columns*)

Add multiple columns to the specified extension of the event file.

Compared to multiple calls to add_column(), and at the cost of a slight code duplication, this avoids create a new binary table at each call.

Parameters

- **ext_name** (*str*) – The name of the extension that the column should be added to.
- **columns** (*fits.Column instances*) – The columns to be added.

average_deadtime_per_event()

Calculate the average deadtime per event.

> **Warning**
>
> Note this is trickier than it seems, as the result might not be accurate if the underlying event file has already been filteres in time and/or phase.

backscal()

Return the value of the BACKSCAL header keyword, if present.

The keyword is written out by xpselect when appropriate. The function returns None when the keyword is missing in the event file.

close()

Close the underlying HDU.

copy_and_filter(*mask*, *history=None*)

Filter the event file according to a given mask of selected rows.

This is similar in spirit to the filter() method, except that the filtering is not done in place, but everything is copied beforehand, so that the original HDU list is not modified, and multiple selections can be applied in series, at the expense of a larger memory footprint.

deadtime_correction()

Return the deadtime correction.

det_position_data()

Return the detx and dety column data, in either the detector x and y measured coordinates on the detector.

ds9_region_file_mask(*file_path*, *mc=False*)

Convenience function filtering a photon list from a ds9 region file.

ds9_region_mask(**region_list*, *mc=False*)

Check which events are inside the logical or of all the regions within a given region list and return the corresponding array mask.

du_id()

Return the detector unit number used to run the simulation.

energy_data(*mc=False*)

Return the ra and dec column data, in either the Monte Carlo or the measured flavor depending on the keyword arguments passed to the binning class.

The try/except clause if needed for inter-operability with the ixpesim output files, where the column naming in the MONTE_CARLO extension is not in line with ixpeobssim, see https://bitbucket.org/ixpesw/gpdsw/issues/327

file_path()

Return the path to the underlying file.

filter(*mask*, *history=None*)

Filter the event file according to a given mask of selected rows.

This filtering happens in place, so it should be used when you don't need to apply subsequent selections on the same file multiple times.

get_gti_list()

Return a xGTIList object from the GTI extension

gti_data()

Return thr GTI data.

gti_mask()

Return a boolean mask selecting events falling inside GTI.

irf_name()

Return the name of the IRF set used to run the simulation.

Note the try/except block in the function body was added for ixpeobssim to inter-operate with the ixpesim output photon lists, which might have a MONTE_CARLO extension without the IRFNAME header keyword. In this case you have to set the keyword by hand, e.g., passing the proper command-line switch to xpbin.

livetime()

Return the livetime, i.e. the duration corrected for dead time.

livetime_data()

Return the LIVETIME column.

max_good_time()

Return the largest STOP time for the GTIs in the event file.

min_good_time()

Return the smaller START time for the GTIs in the event file.

num_events()

Return the total number of events in the event file.

octi_data()

Return the OCTI data.

phase_data()

Return the PHASE column.

phi_data()

Return the PHI column.

pi_data(*mc=False*)

Return the PI column.

pol_deg_weighted_average(*pol_deg_model*, *ebins=10*)

Calculate the weighted average of the polarization degree as a function of energy on the given event sample for a generic input polarization model.

This is achieved by calculating the input model of the polarization degree on an event by event basis (i.e., evaluating the model itself on the proper columns of the event file), creating a 2-dimensional histogram of the polarization degree vs. energy, and calculating the average of the polarization degree in vertical slices of energy.

> **Warning**
>
> The polarization degree is averaged coherently, i.e., under the assumption that the polarization angle is constant over all the event sample. Therefore using this function only makes sense when either the

input polarization model has a constant polarization angle, or after the photoelectron directions in the event file have been properly aligned to the given input model.

Parameters

- **pol_deg_model** (*callable*) – A Python function or an otherwise callable object with the proper signature returning the polarization degree as a function of energy, time and sky position (in this order).
- **ebins** (*int or arra-like (default None)*) – The energy binning for the weighted average. Loosely modeled on the numpy and scipy digitization functions, if bins is and integer, a logarithmically spaced binning between 2 and 8 keV is created.

Returns

- *An xScatterPlot object encoding the average polarization degree as a*
- *function of energy.*

primary_keywords()

Return a list of all the relevant keywords in the PRIMARY header.

This is used, e.g., to propagate all the necessary information (such as the ROI and the IRFs used) from the event files to the binned data files that are created from them.

Warning

This function needs to be refactored!

q_data()

Return the Q column.

remove_columns(*ext_name*, **col_names*)

Remove one or more columns from the specified extension of the event file.

Parameters

- **ext_name** (*str*) – The name of the extension that the column should be added to.
- ***col_names** (*str*) – The name(s) of the column(s) to be removed

sc_data()

Return the spacecraft data.

set_column(*ext_name*, *col_name*, *col_data*)

Overwrite the data for an existing column.

Parameters

- **ext_name** (*str*) – The name of the extension that the column should be added to.
- **col_name** (*str*) – The name of the column to be added.
- **col_format** (*str*) – The format for the new column (e.g., 'E').
- **col_data** (*array_like*) – The data for the new column.

sky_position_data(*mc=False*)

Return the ra and dec column data, in either the Monte Carlo or the measured flavor depending on the keyword arguments passed to the binning class.

Note that, since flight level-2 data do not contain RA and DEC columns, this uses internally X and Y.

srcid_data()

Return the SRC_ID column.

Note the try/except block in the function body was added for ixpeobssim to inter-operate with the ixpesim output photon lists, which come have a MONTE_CARLO extension without the SCR_ID column.

start_met()

Return the start MET of the observation.

stokes_data()

Return the Q and U columns.

stop_met()

Return the stop MET of the observation.

time_data()

Return the TIME column.

timeline_data()

Return the timeline data.

total_good_time()

Return the sum of all the GTIs in the event file.

trigger_id_data()

Return the TRG_ID column.

u_data()

Return the U column.

wcs_reference()

Return the reference point of the underlying WCS object.

Note that, since we are making sure that an event file *always* has a valid WCS associated with it (be it the actual content of the EVENTS header or one created on the fly given RA_SRC and DEC_SRC), this should be the preferred way to gauge the center of the field—compared, e.g., to using RA_SRC, DEC_SRC or RA_PNT, DEC_PNT directly.

write(*file_path*, *overwrite=False*)

Write the underlying HDU list to file.

write_fits_selected(*selection_mask*, *file_path*, *header_keywords=None*, *history=None*, *overwrite=True*, *filter_in_place=True*)

Write to file a subselection of events.

Parameters

- **selection_mask** (*array*) – A mask for the row selection.
- **file_path** (*string*) – The path to the output file.

xy_data()

Return the X and Y columns.

class ixpeobssim.evt.event.**xEventFileFriend**(*file_list2*, *file_list1=None*)

Read/write interface to put together level-1 and level-2 event files.

It supports minimal output facilities, mainly re-implementing functions from xEventFile class

build_livetime_histogram(*tbins*, *discard_bti=True*)

Create a time histogram of the livetime from the class l1, l2 files.

Returns
lvt_hist – Histogram of the livetime in the given time bins

Return type
xHistogram1d

build_rate_histogram(*tbins*, *discard_bti=True*, *selection_mask=None*)

Create a time histogram of the livetime corrected rate from the class l1, l2 files. Optionally provide a mask to select a subset of the events.

Returns
rate_hist – Histogram of the rate in the given time bins

Return type
xHistogram1d

det_position_data()

Wrap xEventFile.det_position_data() appending values for all the LV1 files.

energy_data(*mc=False*)

Wrap xEventFile.energy_data() appending values for all the LV2 files

energy_l1_data()

Rewrite xEventFile.energy_data() to get energy in keV for L1 data

gti_clip_mask(*all_events=False*, *lv1_gti=False*)

Create a mask for selecting the GTIs. Note that, even though by default the BTI filtering is already applied to LV2 files, this mask is not equivalent to taking the intersection between LV1 and LV2, as there are events in the GTI which are excluded from LV2 for other reasons. This can be useful, e.g., for computing the exposure.

gti_data(*lv1=False*)

Warning: This gets the GTI from either level 1 or level 2 file (default). The level 2 definition is more restrictive than other definitions of GTIs, and passes the following selection:

- Data has been received and processed at the SOC
- The spacecraft was actively pointing to the given target
- The detectors were properly configured to observe the target
- The spacecraft was outside the SAA model polygon
- The object was not occulted by the earth or moon

Note that we must handle the case where the GTIs are split over different files (which is usually the case for LV1, while LV2 GTIs should be identical, although we do not rely on this assumption here).

l1value(*val*, *all_events=False*)

l2value(*val*)

livetime(*time_mask=None*, *lv1_gti=False*)

Get the observation livetime by summing the LIVETIME columns after GTI filtering, possibly applying an additional time selection mask. If lv1_gti is True, use GTI from level 1 (default is 2). Note: the mask must be of the size of the full LV1 sample.

sky_position_data(*mc=False*)

Wrap xEventFile.sky_position_data() appending values for all the LV2 files.

start_met(*lv1=False*)

stop_met(*lv1=False*)

time_data(*all_events=False*)

class ixpeobssim.evt.event.**xEventList**(*time_=None*, *source_id=None*)

Class describing an ixpeobssim event list.

apply_charging(*irf_set*, ***kwargs*)

Apply the GEM charging to the event list.

apply_dead_time(*deadtime*)

Apply the dead time effect to the event list.

apply_fiducial_area()

Trim out the events falling outside the detector active area.

apply_vignetting(*vign*, *ra_pnt*, *dec_pnt*)

Apply the effective area vignetting to the event list.

detector_coordinates()

Return the event position in detector coordinates.

energy()

Return the reconstructed event energies.

fill_livetime(*gti_list*, *deadtime=0.0*)

Fill the LIVETIME column.

The basic rule is that the livetime for the first event is the time elapsed since the start run, and for all the other events we just calculate the time difference and subtract the deadtime per event.

Parameters

- **gti_list** (*xGTIList instance*) – The list of good time intervals for the observation.
- **deadtime** (*float*) – The deadtime per event.

fill_livetime_unphysical(*value=-1*)

Fill the LIVETIME column with an unphysical value.

This is called once when the event lists for the single source components are simulated in order to have the column in the event list itself filles and of the proper length. The actual trigger IDs are then set to the actual values after the source components are merged into the final event list and the deadtime is applied.

fill_trigger_id()

Fill the TRG_ID column with sequential values starting from 1.

fill_trigger_id_unphysical(*value=-1*)

Fill the TRG_ID column with an unphysical value.

This is called once when the event lists for the single source components are simulated in order to have the column in the event list itself filles and of the proper length. The actual trigger IDs are then set to the actual values after the source components are merged into the final event list and the deadtime is applied.

livetime()

Return the event livetimes.

mc_energy()

Return the Monte Carlo event energies.

mc_sky_coordinates()

Return the sky positions of the events.

phi()

Return the reconstructed azimuthal angle for the events.

pi()

Return the pulse invariant for the events.

q()

Return the Q Stokes parameter.

set_detector_position_columns(*detx*, *dety*)

Set all the columns related to the event position in the GPD frame.

set_energy_columns(*energy*, *pha*, *pi*)

Set all the columns related to the measured energy.

set_mc_energy_columns(*mc_energy*, *mc_pha*, *mc_pi*)

Set all the columns related to the Monte Carlo energy.

set_mc_gain_column(*gain=1.0*)

Set the column corresponding to the relative GEM gain.

set_mc_sky_position_columns(*mc_ra*, *mc_dec*, *mc_x*, *mc_y*)

Set all the columns related to the Monte Carlo sky position.

set_num_clusters(*value=1*)

Fill the NUM_CLU column with a fixed value.

Parameters

- **value** (*number or array*) – The values for the column to be filled with.
- **in** (*Note the control on whether value is a number or an array is performed*)
- **method.** (*the _set_column()*)

set_phi_columns(*phi*, *detphi*)

Set all the columns related to the photoelectron direction.

set_rec_columns(*pha*, *pi*, *energy*, *ra*, *dec*, *x*, *y*, *detx*, *dety*)

Set all the columns of the EVENT extension that imply a convolution of the MC truth with the instrument response functions.

set_seed_columns(*mc_energy*, *mc_pha*, *mc_pi*, *mc_ra*, *mc_dec*, *mc_x*, *mc_y*, *phi*, *detphi*)

Set all the columns for the seed event list.

These is the first set of columns being simulated in xpobssim, which is why we have this convenience function to set them all at once.

> **Note**
>
> Historically the columns set by this function included the time, but that was abandoned for the sake of consistency when we added the possibility of setting the event times at construction time. So the basic workflow is now:
>
> - initialize the event list using right after the event times have been extracted passing the times themselves and the source identifier to the constructor;
> - call set_seed_columns();
> - call set_rec_columns().

set_sky_position_columns(*ra*, *dec*, *x*, *y*)

Set all the columns related to the measured sky position.

set_weights(*value=1.0*)

Fill the W_MOM column.

Parameters

- **value** (*number or array*) – The values for the column to be filled with.
- **in** (*Note the control on whether value is a number or an array is performed*)
- **method.** (*the _set_column()*)

stokes_parameters()

Return the Stokes parameters.

u()

Return the U Stokes parameter.

write_fits(*creator*, *roi_model*, *irf_set*, ***kwargs*)

Write the event list and associated ancillary information to file.

Basic data format definitions for the event lists.

ixpeobssim.evt.fmt.**build_standard_wcs**(*ra0*, *dec0*)

Build the standard WCS object for event files programmatically.

See https://bitbucket.org/ixpesw/ixpeobssim/issues/552

ixpeobssim.evt.fmt.**set_object_header_keywords**(*hdu*, *ra_obj*, *dec_obj*, *name=None*)

Set the object-related header keywords for a given HDU.

ixpeobssim.evt.fmt.**set_standard_xy_header_limits**(*hdu*)

Set the TLMIN and TLMAX keywords for the X and Y columns for a given hdu.

ixpeobssim.evt.fmt.**set_telescope_header_keywords**(*hdu*, *du_id*)

Set the telescope-related header keywords for a given HDU.

ixpeobssim.evt.fmt.**set_time_header_keywords**(*hdu*, *start_met*, *stop_met*, *duration*, *ontime*, *livetime*, *deadc*)

Set the time-related header keywords for a given HDU.

ixpeobssim.evt.fmt.**set_version_keywords**(*hdu*, *version*)

Set the version-related header keywords for a given HDU.

ixpeobssim.evt.fmt.**set_wcs_header_keywords**(*hdu*)

Set the WCS-related header keywords for a given HDU.

ixpeobssim.evt.fmt.**standard_radec_to_xy**(*ra*, *dec*, *ra0*, *dec0*)

Convert sky coordinates to X and Y digital coordinates in the sky frame.

This is builfding on the fly a standard IXPE WCS object centered at the given sky position, and using it for the conversion.

Parameters

- **ra** (*array_like*) – The array of input Ra coordinates.
- **dec** (*array_like*) – The array of input Dec coordinates.
- **ra0** (*float*) – The Ra coordinate of the center of the field in the sky.
- **dec0** (*float*) – The Dec coordinate of the center of the field in the sky.

ixpeobssim.evt.fmt.**standard_xy_columns_kwargs**(*ra0*, *dec0*)

Return the appropriate keywords for the X and Y columns in event files.

See https://bitbucket.org/ixpesw/ixpeobssim/issues/552

ixpeobssim.evt.fmt.**standard_xy_to_radec**(*x*, *y*, *ra0*, *dec0*)

Convert X and Y digital coordinates to RA and DEC.

This is builfding on the fly a standard IXPE WCS object centered at the given sky position, and using it for the conversion.

Parameters

- **x** (*array_like*) – The array of input X coordinates.
- **y** (*array_like*) – The array of input Y coordinates.
- **ra0** (*float*) – The Ra coordinate of the center of the field in the sky.
- **dec0** (*float*) – The Dec coordinate of the center of the field in the sky.

class ixpeobssim.evt.fmt.**xBinTableHDUEvents**(*ra0*, *dec0*, *data=None*, *keywords=None*, *comments=None*)

Binary table description for the EVENTS extension of the observation output files.

class ixpeobssim.evt.fmt.**xBinTableHDUGTI**(*data=None*, *keywords=None*, *comments=None*)

Binary table for the good time intervals (GTI).

class ixpeobssim.evt.fmt.**xBinTableHDUMonteCarlo**(*data=None*, *keywords=None*, *comments=None*)

Binary table description for the MONTE_CARLO extension of the observation output files, including the additional Monte Carlo fields.

set_irf_name(*irf_name*)

Set the IRFNAME keyword.

class ixpeobssim.evt.fmt.**xBinTableHDUOCTI**(*data=None*, *keywords=None*, *comments=None*)

Binary table for the on-orbit calibration time intervals (OCTIs).

set_cal_stats(*num_runs*, *total_time*)

Set the calibration statistics keywords.

class ixpeobssim.evt.fmt.**xBinTableHDURoiTable**(*data=None*, *keywords=None*, *comments=None*)

Binary table for the good time intervals (GTI).

set_center(*ra0*, *dec0*)

Set the keywords for the ROI center.

class ixpeobssim.evt.fmt.**xBinTableHDUSpacecraftData**(*data=None*, *keywords=None*, *comments=None*)

Binary table for the spacecraft data.

set_roll_angle(*roll_angle*)

Set the roll angle.

class ixpeobssim.evt.fmt.**xBinTableHDUTimeline**(*data=None*, *keywords=None*, *comments=None*)

Binary table for the timeline data.

class ixpeobssim.evt.fmt.**xLvl2PrimaryHDU**(*data=None*, *header=None*, *creator='ixpeobssim'*, *keywords=None*, *comments=None*)

Level 2 primary header definition.

Data structures related to the good time intervals.

exception ixpeobssim.evt.gti.**UnexpectedEdgeType**

Custom exception raised when an unexpected sequence of GTI edges is encountered (e.g. two GTI start in a row without a GTI stop in between).

class ixpeobssim.evt.gti.**xGTIList**(*start_met*, *stop_met*, **gtis*)

Class representing a list of good time interval.

The interfaces are fairly minimal, but since we use this in quite different places, it was handy to collect the useful stuff in one place.

all_mets()

Return all the MET values corresponding to the start or stop of the GTIs in the list.

Note that Python supports the call to sum() with start set by keyword argument only since version 3.8, so we are refraining from that, here.

append_gti(*start*, *stop*)

Append a new GTI to the list.

complement()

Return the logical complement of the GTI list in the relevant time span, i.e., the list of time intervals between the start and the end of observation that are *not* good time intervals.

These are the ones where we do not take celestial data and we can, e.g., take calibration data.

See https://stackoverflow.com/questions/16789776/ for the use of the iterator over the list.

filter_event_times(*time_*)

Filter a given array of event times and return a reduced array only containing the times within the good time intervals in the list, along with the corresponding boolean mask. The latter, in turn, can be used to filter ancillary related columns, e.g., the phase in simulations of periodic sources.

classmethod from_arrays(*start_met*, *stop_met*, *tstarts*, *tstops*)

Initialize from two arrays (of start and stop). This is essentially converting the two arrays of start and stop to an array of tuples (start, stop), as required by the constructor of the class.

gti_mask(*time_*)

Return a boolean mask selecting the times falling inside the GTI in the list.

Parameters
time (*array of times to select*)

merge_gti(*new_gti_start*, *new_gti_stop*)

Update Good Time Intervals (GTIs) by performing the *intersection* with the given GTIs.

Parameters

- **new_gti_start** (*numpy.ndarray*) – Array of start times for GTIs.
- **new_gti_stop** (*numpy.ndarray*) – Array of stop times for GTIs.

Returns
Arrays of start and stop times for the intersection GTIs.

Return type
Tuple[numpy.ndarray, numpy.ndarray]

remove_bti(*bti_start*, *bti_stop*)

Update Good Time Intervals (GTIs) by removing Bad Time Intervals (BTIs) from them.

Parameters

- **bti_start** (*numpy.ndarray*) – Array of start times for BTIs.
- **bti_stop** (*numpy.ndarray*) – Array of stop times for BTIs.

Returns
Arrays of start and stop times for the new GTIs.

Return type
Tuple[numpy.ndarray, numpy.ndarray]

start_mets()

Return the value of the start MET values for the GTI.

stop_mets()

Return the value of the stop MET values for the GTI.

total_good_time()

Return the total good time.

class ixpeobssim.evt.gti.**xGTIListMergerHelper**

Helper class encapsulating most of the logic for combining two sets of GTIs. Provides internal flags for keeping track of whether we are inside the OLD/NEW GTIs and the logic for switching state based on edge detection.

class EdgeType(*value*)

Enumeration representing the types of edges that can occur in Good Time Intervals (GTIs) and Bad Time Intervals (BTIs).

Members:
OLD_GTI_START: Beginning of a good time interval. OLD_GTI_STOP: End of a good time interval. NEW_GTI_START: Beginning of a bad time interval. NEW_GTI_STOP: End of a bad time interval.

combine_intervals(*old_start*, *old_stop*, *new_start*, *new_stop*, *rule*)

Combine the given intervals based on the given rule. This function mostly prepare the input arrays for sweep_intervals().

sweep_intervals(*edge_times*, *edge_types*, *is_active*)

Generic sweep-line engine for interval algebra.

Intervals are half-open: [start, stop)

The user provided function is_active defines when the resulting GTIs should be considered active.

Parameters

- **edge_times** (*ndarray*)
- **edge_types** (*ndarray*)
- **is_active** (*callable() -> bool*) – Returns True when output GTI should be active.

Returns
start, stop

Return type
ndarray

update_state(*edge*)

Change the internal state based on the given edge type. Rise an appropriate error in case of erroneous transitions (e.g. encoutering a OLD_GTI_START when already inside a OLD_GTI).

class ixpeobssim.evt.gti.**xSimpleGTIList**(*start_met*, *stop_met*)

Subclass of xGTIList with a single GTI.

class ixpeobssim.evt.gti.**xUberGTIList**

Subclass of xGTIList with a single GTI including all the MET from -inf to inf.

Facilities to produce photon lists at the top of the window to be fed into ixpesim.

ixpeobssim.evt.ixpesim.**build_tow_response**(*irf_set*)

Build the response at the top of the Be window that is needed to generate the photon lists.

This is assembled "by hand" using the irfgen facilities, and to do this we need to know the proper files used to build the response functions in the first place. Unfortunately, for historical reasons, these are stored in the COMMENT field of the primary header of the FITS files, and therefore we have to resort to some string parsing to make this happen. Admittedly, this is fragile, and we are guaranteed to break it if we ever change the format of the comments. (Note, however, that we are starting with sensible defaults.)

class ixpeobssim.evt.ixpesim.**xBinTableHDUPhotons**(*data=None*, *keywords=None*, *comments=None*)

Binary table description for the PHOTONS extension in ixpesim event lists.

class ixpeobssim.evt.ixpesim.**xPhotonList**(*time_=None*, *source_id=None*)

Class describing a photon list.

This is, in many respects, the equivalent of the evt.event.xEventList class, except that it does not represent actual events in the detector, but photons at the top of the Be window.

apply_fiducial_area()

Trim out the events falling outside the detector active area.

apply_vignetting(*vign*, *ra_pnt*, *dec_pnt*)

Apply the effective area vignetting to the event list.

fill(*energy*, *ra*, *dec*, *detx*, *dety*, *pol_deg*, *pol_ang*)

Fill all the relevant columns of the event list.

write_fits(*creator*, *roi_model*, *irf_set*, ***kwargs*)

Write the photon list and associated ancillary information to file.

Polarization analysis as described in https://arxiv.org/pdf/1409.6214.pdf

class ixpeobssim.evt.kislat2015.**xModulationAnalysis**(*phi*, *weights=None*)

Small class implements the polarization analysis based on the event-by-event Stokes parameters described in Kislat et al. 2015, see https://arxiv.org/pdf/1409.6214.pdf

Parameters

phi (*array_like*) – The array of photoelectron azimuthal directions.

calculate_modulation(*degrees=False*)

Calculate the modulation.

class ixpeobssim.evt.kislat2015.**xStokesAnalysis**(*q*, *u*, *energy*, *modf*, *aeff*, *livetime*, *weights=None*, *acceptcorr=True*)

This class implements the polarization analysis based on the event-by-event Stokes parameters described in Kislat et al. 2015, see https://arxiv.org/pdf/1409.6214.pdf

The basic idea is that we pass to the constructor a list of photoelectron direction and energy arrays, along with the necessary response functions (and optional weights), and the proper functional combination are turned into the correponding (weighted) event-by-event Stokes parameters, that can then be easily summed in the proper energy range and turned into polarization degree and error.

Note that (modulo a factor of 2) all the book-keeping and processing is done in terms of the reconstructed Stokes parameters defined at the end of section 3 of the paper, i.e., the Stokes parameters are divided by the modulation factor on an event-by-event basis in the constructor.

Parameters

- **q** (*array_like*) – The array of event-by-event Q Stokes parameters.
- **u** (*array_like*) – The array of event-by-event U Stokes parameters.
- **energy** (*array_like*) – The array of the event energies (in KeV)—must have the same shape as q and u.
- **modf** (*xModulationFactor instance*) – The modulation factor—this is evaluated at the event energies and used to normalize q and u.
- **aeff** (*xEffectiveArea instance*) – The effective area—this is used for the energy flux calculation, and and to calculate the acceptance correction, if necessary.
- **livetime** (*float*) – The livetime (in s) for the energy flux calculation.
- **weights** (*array_like*) – Additional (optional) multiplicative event weights—must have the same shape as q and u.
- **acceptcorr** (*bool*) – If True, the Stokes parameters are weighted by the inverse of the acceptance.

W2(*mask*)

Return the sum of the squares of weights.

For an un-weighted analysis, and if the acceptance correction is not applied, the weights are all equal to unity, and this sum reduces to the number of events passing the cut, which is in turn equal to the sum of the I Stokes parameter.

average_energy(*emin*, *emax*)

Return the average energy over a given energy range.

static calculate_mdp99(*mu*, *I*, *W2*, *clip=True*)

Calculate the MDP based on equation (A.8) in the paper.

Parameters

- **mu** (*array_like*) – The effective modulation factor.
- **I** (*array_like*) – The I Stokes parameter.
- **W2** (*array_like*) – The sum of the weights squared.
- **clip** (*bool*) – If true, the MDP is clipped within the physical bounds 0–100%.

static calculate_n_eff(*counts*, *I*, *W2*)

Calculate the effective number of events.

static calculate_polarization(*I*, *Q*, *U*, *mu*, *W2=None*, *degrees=False*)

Calculate the polarization degree and angle, with the associated uncertainties, for a given q and u.

This implements equations (21), (36), (22) and (37) in the paper, respectively.

Note that the Stokes parameters passed as the input arguments are assumed to be normalized to the modulation factor (for Q and U) on an event-by-event basis and summed over the proper energy range.

Great part of the logic is meant to avoid runtime zero-division errors.

static calculate_polarization_sub(*I*, *Q*, *U*, *I_ERR*, *Q_ERR*, *U_ERR*, *degrees=False*)

Calculate the polarization degree and angle, along with the fellow associated uncertainties.

This is using the linear error propagation and is currently only used in the polarization cube subtraction.

static calculate_stokes_errors(*I*, *Q*, *U*, *mu*, *W2*)

Calculation of the errors on the Stokes parameters.

effective_mu(*emin*, *emax*)

Return the effective modulation factor weighted over the input events.

mdp_map_cube(*x*, *y*, *binning*)

Return the necessary cube to create a MDPMAPCUBE binned object.

Note the extensions names, here, are taken from the definition of the FITS file holding the data structure, and we have to be careful in passing out the actual values in the right order.

For reference, here is a snapshot the field, in the appropriate order: ['E_MEAN', 'COUNTS', 'MU', 'W2', 'N_EFF', 'FRAC_W', 'MDP_99', 'I']

property n

Simple property function to make the accumulation of the counts easy to read.

static normalized_stokes_parameters(*I*, *Q*, *U*, *I_ERR*, *Q_ERR*, *U_ERR*)

Return the normalized Stokes parameters QN = Q / I and UN = U /I properly handling possible zero-division error issues.

polarization(*emin*, *emax*, *degrees=False*)

Return the average polarization degree and angle, along with the corresponding statistical uncertainties, into a given energy range.

polarization_map_cube(*x*, *y*, *binning*)

Return the necessary cube to create a PMAPCUBE binned object.

polarization_table(*ebinning*, *degrees=True*)

Return a table with all the relevant polarization parameters.

Note the column names, here, are taken from the definition of the FITS file holding the data structure, and we have to be careful in passing out the actual values in the right order.

> **Warning**
>
> It would be nice to be able to grab the column definition from the proper file (binning.fmt) but for some reason I do get a circular import error, when I try and do that. It might be a sensible thing to understand why and do the proper refactoring.

static significance(*Q*, *U*, *Q_ERR*, *U_ERR*)

Calculate the significance of the polarization detection.

Here we take advantage of the fact that Q^2 / Var(Q) + U^2 / Var(U) is a chisquare with two degrees of freedom, a.k.a. an exponential, and we're using the corresponding cumulative function.

The significance in gaussian equivalent sigma is then calculated with the ppf of a normal distribution.

static stokes_covariance(*I*, *QN*, *UN*, *W2*)

Calculate the covariance between Q and U.

static stokes_i(*phi*, *weights=None*)

Convert the event azimuthal angle to the I Stokes parameter, see equations (9a) and (A.1a) in Kislat et al., 2015.

Parameters

- **phi** (*array_like*) – The array of azimuthal angles.
- **weights** (*array_like*) – Optional event weights.

static stokes_q(*phi*, *weights=None*)

Convert the event azimuthal angle to the Q Stokes parameter, see equations (9b) and (A.1b) in Kislat et al., 2015.

Note that, compared to equation (9b), we have an extra factor of 2, here, that renders the calculations downstream more natural. The rest of the class is implemented consistently.

This is factored out in a staticmethod so that it can be reused consistently in other places.

Parameters

- **phi** (*array_like*) – The array of azimuthal angles.
- **weights** (*array_like*) – Optional event weights.

static stokes_u(*phi*, *weights*)

Convert the event azimuthal angle to the U Stokes parameter, see equations (9c) and (A.1c) in Kislat et al., 2015.

Note that, compared to equation (9c), we have an extra factor of 2, here, that renders the calculations downstream more natural. The rest of the class is implemented consistently.

This is factored out in a staticmethod so that it can be reused consistently in other places.

Parameters

- **phi** (*array_like*) – The array of azimuthal angles.

- **weights** (*array_like*) – Optional event weights.

sum_stokes_parameters(*emin*, *emax*)

Sum the Stokes parameters into a given energy range.

Sonification utilities.

class ixpeobssim.evt.sonify.**ContolChangeParameter**

Most common contol change codes, see https://professionalcomposers.com/midi-cc-list/

class ixpeobssim.evt.sonify.**ProgramChangePrograms**

Program change programs, see https://www.recordingblogs.com/wiki/midi-program-change-message

ixpeobssim.evt.sonify.**midi_to_wav**(*midi_file_path*, *sound_font=''*)

Convert a MIDI file to audio using fluidsynth.

ixpeobssim.evt.sonify.**play_midi**(*midi_file_path*, *sound_font=''*)

Play a MIDI file file through fluidsynth.

ixpeobssim.evt.sonify.**stereo_to_mono**(*file_path*)

Convert a stereo wave file to a mono file containing the average of the two channels.

class ixpeobssim.evt.sonify.**xMidiEvent**(*type_*, *timestamp*, ***kwargs*)

Small classe describing a midi event.

This is intended as a temporary storage for actual MIDI messages expressed in absolute time and not necessarily time-ordered.

class ixpeobssim.evt.sonify.**xMidiFile**(*ticks_per_beat=480*)

Small wrapper around the midio.MidiFile class.

add_midi_track(*track_name=None*, *bpm=None*)

Add a track to the MIDI file.

static compute_note_number(*energy*, *mode='Ionian'*, *key='C'*, *pivot=4.0*, *dynamic_scaling=1.0*)

Convert an energy (in keV) into the corresponding note number.

This is accomplished assigninig the A4 note number to the pivot energy, and scaling the energies in octave space. Since the extended IXPE bandpass over which we calculate the response functions (1–15 keV) corresponds to roughly 4 octaves, this is not a terrible match for a sonification project. The nominal IXPE energy band (2–8 keV) is only 2 octaves, and in real life we might need to tweak things a little bit, which is the purpose of the dynamic_scaling argument (see the comments in the code for a detailed explanation of the algorithm).

Parameters

- **energy** (*array_like*) – The photon energy.
- **mode** (*str*) – The name of the musical scale to which the energy values should be snapped.
- **key** (*str*) – The key for the aforementioned musical scale.
- **pivot** (*float*) – The pivot energy (mapped to A4).
- **dynamic_scaling** (*float*) – Empirical parameter for scaling the energy in octave spaces.

static compute_pan(*phi*, *dynamic_scaling=1.0*)

Compute the pan.

This is simply mapping the photoelectron angle into the 0–127 phisical range.

Parameters

- **phi** (*array_like*) – The photoelectron angle.
- **dynamic_scaling** (*float*) – Empirical parameter to enhance the veocity dynamics.

static compute_velocity(*mc_energy*, *rec_energy*, *pivot=64.0*, *dynamic_scaling=1.0*)

Convert the energy measurement to note velocity.

The note velocity is determined by the ratio between the reconstructed and true energy, i.e., events in the left tail of the energy dispersion are rendered with a lower velocity.

Parameters

- **mc_energy** (*array_like*) – The true energy.
- **rec_energy** (*array_like*) – The reconstructed energy.
- **pivot_velocity** (*float*) – The pivot value for the note velocity, assigned when the reconstructed energy is equal to the true energy.
- **dynamic_scaling** (*float*) – Empirical parameter to enhance the veocity dynamics.

static control_change_message(*control*, *value*, *channel=0*)

Return a control change message.

fill(*file_path*, *roi=None*, ***kwargs*)

Fill the MIDI file with the event data in the proper format.

This is the main function where most of the action actually happens.

static pan_message(*value*, *channel=0*)

Return a pan message.

static plot_diagnostics(*timestamp*, *note*, *velocity*, *pan*, *duration*)

Create all the diagnostics plots.

static program_change_message(*program*, *channel=0*)

Return a program change MIDI message.

static set_tempo_meta_message(*bpm*, *time=0*)

Return a set_tempo MIDI meta-message.

static track_name_meta_message(*name*, *time=0*)

Return a track_name MIDI meta-message.

class ixpeobssim.evt.sonify.**xMidiNote**(*note_number*)

Small class encapsulating a MIDI note.

There's an infinite number of place on the web where one can find conversion tables between MIDI notes and physical characteristics, see, e.g., https://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies

A MIDI note is univiquely identified by a note number, ranging from 0 to 127. Here we shall restrict ourselves to the piano keys, i.e., note numbers from 21 to 127 (the lower notes are too low to be useful, anyway), with the understanding that:

- note 21 is A0, at 27.500 Hz;
- note 127 is G9, at 12543.854 Hz;
- note 69 is A4, at 440.000 Hz.

A MIDI note is initialized by its MIDI note number, and encapsulates all the facilities to calculate the frequency, note name and alike.

frequency()

Return the note frequency.

name()

Return the note name and octave.

note_name()

Return the note name.

octave()

Return the note octave.

class ixpeobssim.evt.sonify.**xMusicalScale**(*mode='Ionian'*, *key='C'*)

Simple class describing a musical scale.

A scale is essentilly given by a series (in the form of a numpy array) of note number covering the piano dynamic range.

snap(*values*)

Snap a series of notes to the scale.

Event list filtering facilities.

class ixpeobssim.evt.subselect.**xEventSelect**(*file_path*, ***kwargs*)

Base class for event subselection.

get(*key*, *default=None*)

Convenience method to address the keyword aguments.

mask_selected()

Return True if selecting with event mask directly,

phase_selected()

Return True if selecting in phase, i.e., phasemin and/or phasemax are not None.

select()

Select the events and write the output file.

set(*key*, *value*)

Convenience method to set keyword arguments.

time_selected()

Return True if selecting in time, i.e., tmin and/or tmax are not None.

ixpeobssim.evt.xspec_.**add_model_string**(*key*, *value*)

Add a key,value pair of strings to XSPEC's internal database. (simple wrapper around the xspec.Xset.addModelString() method.)

This database provides a way to pass string values to certain model functions (e.g., POW_EMIN and POW_EMAX) which are hardcoded to search for "key". (See the XSPEC manual description for the "xset" command for a table showing model/key usage.)

If the key,value pair already exists, it will be replaced with the new entries.

ixpeobssim.evt.xspec_.**add_model_strings**(***kwargs*)

Facility to add multiple model string to xspec.

ixpeobssim.evt.xspec_.**calculate_confidence_intervals**(*model_id=1*, *delta_fit_statistics=1.0*)

Calculate the confidence intervals for a fit.

This is running the error XSPEC command, see https://heasarc.gsfc.nasa.gov/xanadu/xspec/manual/node78.html

ixpeobssim.evt.xspec_.**compare_fit_data**(*fit_data*, *target_values*, *threshold=5.0*)

Compare the best-fit parameter values with the input model.

Parameters

- **fit_data** (*xXspecFitData instance*) – The fit output.
- **target_values** (*array_like*) – The corresponding values from the input model.
- **threshold** (*float*) – The threshold (in units of sigma) for determining the agreement between the best fit parameters and the model inputs.

Returns

- *Zero if the bes-fit parameters agree with the input, and an error code > 0*
- *otherwise.*

ixpeobssim.evt.xspec_.**current_fit_output**(*model_id=1*)

Return the current fit output.

ixpeobssim.evt.xspec_.**energy_flux**(*emin=2.0*, *emax=8.0*)

Return the value of the energy flux and of the photon flux between emin and emax.

ixpeobssim.evt.xspec_.**fit**(*stat_method='chi'*, *max_iterations=10*, *error=True*)

Perform a spectral fit.

ixpeobssim.evt.xspec_.**fix_parameter**(*par_index*, *par_value*, *model_id=1*)

Fix a model parameter to a given value.

ixpeobssim.evt.xspec_.**load_input_files**(**file_list*)

Read a set of PHA1 input files and create the corresponding xspec.Spectrum objects.

ixpeobssim.evt.xspec_.**load_local_models**(*package_name='ixpeobssim'*)

Load all the ixpeobssim local models.

We introduced this function, and called it automatically, version 12.0.0, based on the idea that whoever had XSPEC installed also had the XSPEC local models installed. This turned out to be a glaring usability issue, as there are many possible ways one could be unable to compile the local models, see https://bitbucket.org/ixpesw/ixpeobssim/issues/350 As of version 12.1.0 we wrap the XSPEC call in a try / except, and happily proceed even if the local models cannot be loaded.

ixpeobssim.evt.xspec_.**plot**(*figure_name='XSPEC spectrum'*, *logy=True*)

Custom, matplotlib-based implementation of the standard XSPEC spectral-fit plot.

ixpeobssim.evt.xspec_.**plot_normalized_counts**(*plot_data*, *du_id*, ***kwargs*)

Plot the normalized counts.

Note that we use full circles for positive values and empty circles for negative ones, so that we can plot Stokes parameters (that can go negative) in logarithmic scale.

ixpeobssim.evt.xspec_.**plot_residuals**(*plot_data*)

Plot the model residuals.

ixpeobssim.evt.xspec_.**reset**()

Global reset.

ixpeobssim.evt.xspec_.**retrieve_plot_data**(*bin_alg*, *du_id*)

Retrieve the plot data (and, if available, the corresponding model) for a given spectrum.

> **Warning**
>
> Note that the plot data within XSPEC are intented to be normalized wrt the energy—i.e., the input from the FITS table is divided by the width of the energy bin, and the units get an additional keV^{-1}.

ixpeobssim.evt.xspec_.**sample_spectral_model**(*expression*, *parameters*, *emin=1.0*, *emax=12.0*, *num_points=250*, *name=None*, *source_id=1*)

Sample the values for a generic XSPEC model, given an expression and a set of parameters.

Most notably, this is used to feed into ixpeobssim time-independent XSPEC spectral models.

Parameters

- **expression** (*str*) – The model expression string, using full component names.
- **name** (*str*) – The model name.
- **source_id** (*int*) – The source number.
- **parameters** (*dict or list*) – The model parameters.
- **emin** (*double*) – Energy limits, in keV.
- **emax** (*double*) – Energy limits, in keV.
- **num_points** (*int*) – The number of points to sample the spectrum.

ixpeobssim.evt.xspec_.**select_energy_range**(*emin=None*, *emax=None*)

Apply a global mask on the energy value for the channels to be fitted.

ixpeobssim.evt.xspec_.**set_parameter**(*par_index*, *par_value*, *model_id=1*)

Set the model parameter value corresponding to provided index.

ixpeobssim.evt.xspec_.**set_parameter_range**(*par_index*, *min_value*, *max_value*, *model_id=1*)

Set the range for a parameter.

ixpeobssim.evt.xspec_.**setup_fit_model**(*expression*, *par_values=None*)

Create a model for spectral fitting in XSPEC.

Note we abort if the model cannot be created—this can happen, e.g., if one is trying to use local models without having compiled them. This seems as the only sane option, at this point, as the result of moving on would depend on the internal XSPEC state (is there any active model?) and this would be likely more confusing to the user than stating straight away what the problem is.

class ixpeobssim.evt.xspec_.**xXspecFitData**(*model*)

Small containter class to store the output of an XSPEC spectral fit.

This includes the parameter names, best-fit values and errors, the test statistics and the number of degrees of freedom.

next()

Alternative method for Python 2 compatibility.

par_error(*identifier*)

Return the parameter error at a given index.

par_value(*identifier*)

Return the parameter value at a given index.

stat_box(***kwargs*)

Return a stat box that can be overlaid onto a given plot.

Note that when the error command has been run after the fit, we do include the average of the (asymmetric) error bars in the stat box.

class ixpeobssim.evt.xspec_.**xXspecPlotData**(*energy*, *data*, *errors*, *model=None*)

Small container class storing the XSPEC plot data.

Plot data are typically retrieved by function calls to the global xspec.Plot object, and this contained provides a convenient interface to cache plot data for later use. The container encapsulates the x and y values, the errors on the y values and, if available, the model calculated in correspondence of the x array.

save(*file_path*)

To be implemented.

class ixpeobssim.evt.xspec_.**xXspecSpectrumManager**

Small facility to keep track of the spectra being loaded into XSPEC.

This is essentially a dict where the type of the binned spectra and the DU Ids are mapped into the data group in the XSPEC memory, so that we can easily retrieve the plot data at any time.

data_group(*bin_alg*, *du_id*, *default=None*)

Return the XSPEC data group mapping to a particular binning algorithm and DU ID.

register(*file_path*, *data_group*)

Register a spectrum.

Note we peek into the binned file to get an hold on the binning algorithm and the DU ID.

spectrum_types()

Return a tuple with all the binning algorithms containing at least a spectrum.

29.5 Instrument

Charging-related facilities.

ixpeobssim.instrument.charging.**asymptotic_gain_delta**(*energy_flux*, *k_c=120000.0*, *tau_d=110000.0*, *delta_max=0.067*)

Return the asymptotic gain decrease for the GEM in an arbitrary state.

ixpeobssim.instrument.charging.**calculate_gain**(*time_*, *energy_flux*, *initial_gain=1.0*, *k_c=120000.0*, *tau_d=110000.0*, *delta_max=0.067*)

Calculate the gain as a function of time for a given input energy flux.

The time grid and the corresponding energy flux are passed as numpy arrays (of the same shape). The system is integrated using finite differences.

ixpeobssim.instrument.charging.**charging_tau**(*energy_flux*, *k_c=120000.0*)

Return the time constant for the GEM charging in absence of discharge.

ixpeobssim.instrument.charging.**create_charging_map_extension**(*fast_map*, *slow_map=None*, *start_date='N/A'*, *start_time='N/A'*, *version=1*)

Create the CHRG_MAP extension for a charging map file.

ixpeobssim.instrument.charging.**effective_tau**(*energy_flux*, *k_c=120000.0*, *tau_d=110000.0*)

Return the effective time constant for the GEM in an arbitrary state.

ixpeobssim.instrument.charging.**gain_profile**(*energy_flux*, *k_c=120000.0*, *tau_d=110000.0*)

Return a lambda to generate a gain profile, given some model parameters.

to plot the actual time-profile of the gain.

ixpeobssim.instrument.charging.**read_charging_map**(*file_path*)

Open a charging map file and read the values for the slow and fast component. We get the size of the arrays from the header.

ixpeobssim.instrument.charging.**read_charging_parameters**(*input_file_path*)

Open a charging parameters calibration file, read the parameters and return them.

Parameters

input_file_path (*string*) – the path to the input FITS file storing the charging parameters

class ixpeobssim.instrument.charging.**xBinTableHDUCharging**(*data=None*, *keywords=None*, *comments=None*)

Binary table description for the CHRG_MAP extension of the observation output files.

class ixpeobssim.instrument.charging.**xChargingPrimaryHDU**(*data=None*, *header=None*, *creator='ixpeobssim'*, *keywords=None*, *comments=None*)

Charging map file primary header.

class ixpeobssim.instrument.charging.**xEnergyFluxCube**(*nside*, *tedges*, *zlabel='Mission Elapsed Time [s]'*, *wlabel='Energy flux [keV mm^{-2} s^{-1}]'*)

Main data structure for the charging-induced gain correction.

calculate_gain_data(*k_c_fast=120000.0*, *tau_d_fast=110000.0*, *delta_max_fast=0.067*, *initial_dg_fast=None*, *k_c_slow=0.0*, *tau_d_slow=0.0*, *delta_max_slow=0.0*, *initial_dg_slow=None*)

Calculate the expected space-time profile of the gain for a given illumination history. We are adopting a charging model with two components: one fast and one slow. Both are regulated by the same equation, which includes a rate-dependent charging and a constant discharging, with the caracteristic time for the two processes (fast and slow) separated by roughly an order of magnitude. The system is integrated using finite differences. The returned gain profile is computed on the times defined by self.tbinning() and has the same spatial dimensions as self.content

Parameters

- **k_c_fast** (*float*) – the charging constant for the fast process (ADC counts/mm^2)
- **tau_d_fast** (*float*) – the discharge time constant for the fast process (seconds)
- **delta_max_fast** (*float*) – maximum fractional gain drop for the fast process
- **initial_dg_fast** (*float or array*) – initial fraction of gain drop due to the fast process. If an array, the value is given for each spatial bin - thus the dimension must match the spatial dimensions of self.content. If None, we assume initial full discharge for the fast component.

- **k_c_slow** (*float*) – as k_c_fast, but for the slow process
- **tau_d_slow** (*float*) – as tau_d_fast, but for the slow process
- **delta_max_slow** (*float*) – as delta_max_fast, but for the slow process
- **initial_dg_slow** (*float*) – as initial_dg_fast, but for the slow process

fill(*detx*, *dety*, *time_*, *energy*, *deadtime_correction=None*, ***kwargs*)

Overloaded fill() method.

Here we essentially take care of the energy weighting and the normalization by the elapsed time and the pixel area. This is not completely trivial as the mean energy must be calculated in the same time slices used to bin the original histogram, to cope with the possibility that the source spectrum—or the event flux on the detector, for whatever reason—changes with time.

> **Warning**
>
> The deadtime correction is passed as a single number, i.e., averaged over the full data set.

gain(*detx*, *dety*, *time_*, ***charging_params*)

Return the expected corrected gain for a list of events at specific detector positions and times.

gain_map(*t*)

Return a two-dimensional histogram with the gain map at a given time.

static step(*flux*, *dt*, *dg*, *k_c*, *delta_max*, *tau_d*)

Compute a step of the charging model (for either the slow or fast process). For a description of the charging parameters see the calculate_gain_data() method

Parameters

- **flux** (*float or array*) – the value of the input energy flux for this step (ADC counts/mm^2/s)
- **dt** (*float*) – the time step (in seconds)
- **dg** (*float or array*) – the current gain drop due to this process (fractional)

tbinning()

Return the histogram binning on the time axis (aka z-axis).

time_slice(*tmin=None*, *tmax=None*)

Return a time slice of the irradiation history, between generic bounds.

By default this is returning the average energy flux per unit area over the entire input sample.

Note that the times are adjusted to the closest bih edges.

xbinning()

Return the histogram binning on the detx axis.

ybinning()

Return the histogram binning on the dety axis.

ixpeobssim.instrument.du.**det_name_to_du_id**(*det_name*)

Convert the logical name of a given DU to the actual DU ID.

ixpeobssim.instrument.du.**du_logical_name**(*du_id*)

Return the logical name for a given DU (i.e., the content of the DETNAME header keyword).

ixpeobssim.instrument.du.**du_physical_name**(*du_id*)

Return the physical name for a given DU (i.e., the content of the DET_ID header keyword).

ixpeobssim.instrument.du.**du_rotation_angle**(*du_id*, *roll_angle=0.0*, *modulo=True*)

Return the rotation angle for a given DU (modulo 2 pi).

Basic GPD-related constants.

ixpeobssim.instrument.gpd.**detphi_to_phi**(*detphi*, *du_id*, *roll_angle*)

Convert the phi angle from the GPD reference frame to the focal plane (i.e., sky) reference.

Parameters

- **detphi** (*float or array*) – The azimuthal angle in the GPD reference frale in radians
- **du_id** (*int*) – The detector unit hosting the GPD
- **roll_angle** (*float*) – The telescope roll angle in decimal degrees

ixpeobssim.instrument.gpd.**fiducial_area**(*half_side_x=6.6*, *half_side_y=6.8*)

Return the area of the fiducial rectangle.

This is essentially calling the base rectangle_area() function, with the proper default values for the fiducial cut.

Parameters

- **half_side_x** (*float*) – The half side of the fiducial rectangle along the x coordinate in mm.
- **half_side_y** (*float*) – The half side of the fiducial rectangle along the y coordinate in mm.

ixpeobssim.instrument.gpd.**gpd_map_binning**(*half_side_x*, *half_side_y*, *num_bins_x*, *num_bins_y=None*)

Return a numpy array with an appropriate binning for a bidimensional map over the GPD area.

Note this function was refactored to adapt to a generic rectangle in response to https://github.com/lucabaldini/ixpeobssim/issues/668, so this now returns two independent arrays representing the binng on the x and y axes.

Parameters

- **half_side_x** (*float*) – The half side of the histogram binning along the x coordinate (in mm).
- **half_side_y** (*float*) – The half side of the histogram binning along the y coordinate (in mm).

ixpeobssim.instrument.gpd.**phi_to_detphi**(*phi*, *du_id*, *roll_angle*)

Convert the azimuthal angle from the focal plane (i.e., sky) reference frame to the GPD reference.

Parameters

- **phi** (*float or array*) – The azimuthal angle in the sky reference frame in radians
- **du_id** (*int*) – The detector unit hosting the GPD
- **roll_angle** (*float*) – The telescope roll angle in decimal degrees

ixpeobssim.instrument.gpd.**rectangle_area**(*half_side_x*, *half_side_y*)

Small convenience function to calulate the area of a rectangle, given the half sides along the two coordinates.

Parameters

- **half_side_x** (*float*) – The half side of the rectangle along the x coordinate (in mm).
- **half_side_y** (*float*) – The half side of the rectangle along the y coordinate (in mm).

ixpeobssim.instrument.gpd.**rotate_detxy**(*x*, *y*, *du_id*, *roll_angle=0.0*, *inverse=False*)

Convert the (x, y) position from the focal plane reference frame to the the gpd reference frame based on the DU id and roll angle.

> **Warning**
>
> This would be more consistent with the rest of the code base if the direct and inverse rotations were implemented in two different functions rather than one function with an additional argument.

Parameters

- **x** (*float or array*) – The x position or array of x positions in mm
- **y** (*float or array*) – The y position or array of y positions in mm
- **du_id** (*int*) – The detector unit hosting the GPD
- **roll_angle** (*float*) – The telescope roll angle in decimal degrees

ixpeobssim.instrument.gpd.**within_fiducial_rectangle**(*x*, *y*, *half_side_x=6.6*, *half_side_y=6.8*)

Return wheter an (x, y) position in mm is within a fiducial rectangle of given half-sides on the two coordinates.

Parameters

- **x** (*float or array*) – The x position or array of x positions in mm
- **y** (*float or array*) – The y position or array of y positions in mm
- **half_side_x** (*float*) – The half side of the fiducial rectangle along the x coordinate (in mm).
- **half_side_y** (*float*) – The half side of the fiducial rectangle along the y coordinate (in mm).

ixpeobssim.instrument.gpd.**within_gpd_physical_area**(*x*, *y*)

Return wheter an (x, y) position in mm is within the GPD physical area.

Parameters

- **x** (*float or array*) – The x position or array of x positions in mm
- **y** (*float or array*) – The y position or array of y positions in mm

ixpeobssim.instrument.mma.**apply_dithering**(*time_*, *ra_pnt*, *dec_pnt*, *dither_params=None*)

Apply dithering correction directly to pointing direction in celestial coordinates. In this way the correction gets propagated down the chain and the appropriate IRFs are used. See convolve_event_list in mma.py

Parameters

- **time** (*float or array*) – The event time
- **ra_pnt** (*float or array*) – The pointing ra in decimal degrees
- **dec_pnt** (*float or array*) – The pointing dec in decimal degrees
- **dither_params** (*(amplitude, pa, px, py) tuple, optional*) – The parameters for the dithering of the observatory. The dithering is not applied if this is set to None.

ixpeobssim.instrument.mma.**fiducial_backscal**(*half_side_x*, *half_side_y*)

Calculate the BACKSCALE value for a give fiducial rectangle in detector coordinates.

This function was introduced in response to the issue https://github.com/lucabaldini/ixpeobssim/issues/668 in place of the old FIDUCIAL_BACKSCAL constant.

Parameters

- **half_side_x** (*float*) – The half side of the fiducial rectangle along the x coordinate in mm.
- **half_side_y** (*float*) – The half side of the fiducial rectangle along the y coordinate in mm.

ixpeobssim.instrument.mma.**gpd_to_sky**(*detx*, *dety*, *time_*, *ra_pnt*, *dec_pnt*, *du_id*, *roll_angle*, *dither_params=None*)

Convert an array of (detx, dety) positions in detector coordinates to an array of (ra, dec) positions in the sky. This is supposed to be the inverse of sky_to_gpd(), so that the two operations, applied in series, should preserve the original data.

Parameters

- **detx** (*float or array*) – The detx position in detector coordinates in mm
- **dety** (*float or array*) – The dety position in detector coordinates in mm
- **time** (*float or array*) – The event time
- **ra_pnt** (*float or array*) – The pointing ra in decimal degrees
- **dec_pnt** (*float or array*) – The pointing dec in decimal degrees
- **du_id** (*int*) – The detector unit id
- **roll_angle** (*float*) – The telescope roll angle in decimal degrees
- **dither_params** (*(amplitude, pa, px, py) tuple, optional*) – The parameters for the dithering of the observatory. The dithering is not applied if this is set to None.

ixpeobssim.instrument.mma.**parse_dithering_kwargs**(***kwargs*)

Parse the keyword arguments related to the dithering.

ixpeobssim.instrument.mma.**sky_to_gpd**(*ra*, *dec*, *time_*, *ra_pnt*, *dec_pnt*, *du_id*, *roll_angle*, *dither_params=None*)

Convert an array of (ra, dec) positions in the sky to an array of (x, y) positions onto the gpd reference frame. This involves essentially three different steps: first we project the sky coordinates onto the focal plane reference frame, then we apply the dithering pattern (if enabled) based on the event times and finally we rotate the coordinates according to the DU id and the telescope roll angle.

Warning

Change x and y to detx and dety!

Parameters

- **ra** (*float or array*) – The ra position in the sky in decimal degrees
- **dec** (*float or array*) – The dec position in the sky in decimal degrees
- **time** (*float or array*) – The event time
- **ra_pnt** (*float or array*) – The pointing ra in decimal degrees

- **dec_pnt** (*float or array*) – The pointing dec in decimal degrees
- **du_id** (*int*) – The detector unit id
- **roll_angle** (*float*) – The telescope roll angle in decimal degrees
- **dither_params** (*(amplitude, pa, px, py) tuple, optional*) – The parameters for the dithering of the observatory. The dithering is not applied if this is set to None.

Spacecraft-related facilities.

ixpeobssim.instrument.sc.**create_ineclipse_gtis**(**hk_file_paths*, *complement=False*)

Create arrays of start and stop times defining Good Time Intervals (GTIs) corresponding to spacecraft eclipse conditions.

By default (complement=False), the function returns intervals when the spacecraft is not affected by the sun.

The INECLIPSE region is defined by the sign of the angles ADSEC2SUN and ADSEC2ECL, the former refering to the position of sun relative to the spacecraft, the latter to the condition of Earth-occultation. The region is defined as: (ADSEC2SUN < 0) & (ADSEC2ECL >= 0): such selection has been confirmed by spectral analysis of observation data. Every other combination of signs shows contamination of solar-induced background.

ixpeobssim.instrument.sc.**dithering_pattern**(*amplitude=1.6*, *period_a=907.0*, *period_x=101.0*, *period_y=449.0*)

Implementation of the full dithering pattern as per the report by Allyn Tennant and Kurt Dietz linked in the issue page: https://bitbucket.org/ixpesw/ixpeobssim/issues/193

Note this returns an anonymous function that can be evaluated into a generic array of time values, the basic usage being:

```
>>> t = numpy.linspace(0., 10000., 10001)
>>> dithering = dithering_pattern()
>>> x, y = dithering(t)
```

Parameters

- **amplitude** (*float*) – The dithering amplitude in arcminutes.
- **period_a** (*float*) – The main dither period.
- **period_x** (*float*) – The x dithering period.
- **period_y** (*float*) – The y dithering period.

ixpeobssim.instrument.sc.**mask_to_gti**(*time*, *mask*, ***, *t_start=None*, *t_end=None*)

Convert a boolean mask into half-open GTIs [t_start, t_stop).

If t_start or t_end are provided, they define the observation boundaries. Otherwise, time[0] and time[-1] are used.

Parameters

- **time** (*ndarray*) – 1D monotonically increasing time array.
- **mask** (*ndarray of bool*) – True inside GTIs.
- **t_start** (*float or None, optional*) – Observation start time. Must satisfy t_start <= time[0].
- **t_end** (*float or None, optional*) – Observation end time. Must satisfy t_end >= time[-1].

Returns
tstart, tstop – Start and stop times of GTIs (half-open).

Return type
ndarray

ixpeobssim.instrument.sc.**period_to_omega**(*period*)

Convert a characteristic period (in s) into the corresponding omega (in rad/s).

ixpeobssim.instrument.sc.**pointing_direction**(*sc_data*, *met*)

Return the pointing direction at a given array of MET.

ixpeobssim.instrument.sc.**pointing_splines**(*sc_data*)

Build a pair of R.A. and Dec dithering splines starting from as set of spacecraft data.

This can be used to recover the actual pointing as a function of time, given a SC_DATA binary table.

ixpeobssim.instrument.sc.**pow_triangular_wave**(*x*, *amplitude*, *period*, *exponent=0.5*)

Modified triangual wave, where the relative values are raised to a generic exponent, in such a way that the values of the maxima and minima are preserved.

ixpeobssim.instrument.sc.**spiral_dithering_pattern**(*amplitude=1.6*, *period_theta=100.0*, *period_r=970.0*, *exponent=0.5*)

Alternative, spiral-like dithering pattern.

ixpeobssim.instrument.sc.**triangular_wave**(*x*, *amplitude*, *period*)

Basic description of a (symmetric) triangular wave with a given period and amplitude.

The basic expression is taken from https://en.wikipedia.org/wiki/Triangular_distribution and, in this form, the function evaluates to 0 for x = 0.

Spacecraft trajecory related functions

class ixpeobssim.instrument.traj.**xIXPETrajectory**(*ephemeris_file_name='de430t_2000_2040.bsp'*, *min_sun_angle=65.0*, *max_sun_angle=115.0*, *builtin_timescale=True*)

A class to characterize the spacecraft's orbit and trajectory for ixpeobssim.

This is supposed to keep track of the necessary elements to keep track of the state of the observatory and generate the GTI.

The TLE data are read from the proper file in $IXPEOBSSIM_INSTRUMENT_DATA (which satellite precisely is controlled by the satellite_name argument).

The planetary and lunar ephemeris are taken from JPL. While JPL distributes and maintain an extensive set of files, in order to avoid large downloads at runtime, we have packaged or own trimmed versions of DE405 and DE430t within the 2000–2040 time span into the ixpeobssim/instrument/data folder. The trimming has been performed using the jplephem utility described at https://rhodesmill.org/skyfield/planets.html

The generation of the timescale information takes advantage of all the skyfield machinery. Note that, since the default skyfield method do not reasily offer any flexibility in terms of where the ancillary files are located, we have implemented our own Loader class that behaves as the the original with two notable exceptions:

- the builtin timescale ancillary files (when necessary) are loaded from IXPEOBSSIM_INSTRUMENT_DATA, rather than from the sjyfield installation;
- the updated timescale ancillary files (when necessary) are downloaded (and read from) IXPEOBSSIM_DATA.

Parameters

- **ephemeris_file_name** (*str*) – The ephemeris file name (in the IXPEOBSSIM_INSTRUMENT_DATA folder).
- **builtin_timescale** (*bool*) – Flag passed to the skyfield timescale generation routine (if True the static files shipped with ixpeobssim are used, otehrwise they are downloaded).

static complement_epochs(*epochs*, *start_met*, *stop_met*)

Return the complement to a list of epochs.

earth_occultation_epochs(*start_met*, *stop_met*, *target_ra*, *target_dec*, *initial_step=100.0*, *precision=0.001*)

Calculate the epochs when a given target is occulted by the Earth.

earth_occultation_mets(*start_met*, *stop_met*, *target_ra*, *target_dec*, *initial_step=100.0*, *precision=0.001*)

Return the list of METS for which the Earth occultation status is changing.

elevation(*met*)

Return the altitude of the satellite (in km) at a given MET.

static equispaced_met_grid(*start_met*, *stop_met*, *approximate_step*)

Return an equally-spaced grid of MET values according to the input argument.

Note that the array returned ranges exactly from start_met to stop_met, and the step size is rounded to guarantee that.

geocentric(*met*)

Return the geocentric position at a given MET.

gti_list(*start_met*, *stop_met*, *target_ra*, *target_dec*, *saa=True*, *occult=True*, *initial_step=100.0*, *precision=0.001*)

Calculate the GTIs, i.e., the ephocs where we are not in the SAA and the target is not occulted by the Earth.

While this method seems (and is, in fact) quite convoluted, we decided to calculate separately the SAA and Earth occultation epochs, when the instrument is not taking data, and merge them at the end to extract the good time intervals. The reasoning is that, while the former epochs are quite regular over the timescale of an observation—which allows to perform a sensible binary search starting from a reasonably corse grid (say with a 100 s step)—when we put in AND the two we get all kind of resonances, to the point that the GTIs can be arbitrarily small, and we would need a much finer (and computationally intensive) initial grid in order not to introduce measurable errors.

Parameters

- **start_met** (*float*) – The initial MET for the observation.
- **stop_met** (*float*) – The final MET for the observation.
- **target_ra** (*float*) – The right ascension of the target source in decimal degrees.
- **target_dec** (*float*) – The declination of the target source in decimal degrees.
- **saa** (*bool*) – Consider the SAA passages in the GTI calculation.
- **occult** (*bool*) – Consider the Earth occultation in the GTI calculation.
- **initial_step** (*float*) – The step (in s) for the initial equispacedtime time grid used to seed the binary search. (This can slow things down when too small.)
- **precision** (*float*) – The worst-case precision of the GTI calculation (i.e., the stop condition for the binary search)

in_saa(*met*)

Return whether the satellite is in the SAA at a given time.

static load_tle(*tle_file_name='tle_data_science.txt'*, *satellite_name='AGILE'*)

Load TLE data for a given satellite. Note the tle() method is deprecated with an entertaining comment DEPRECATED: in a misguided attempt to be overly convenient, this routine builds an unweildy dictionary of satellites with keys of two different Python types: integer keys for satellite numbers, and string keys for satellite names. It even lists satellites like `ISS (ZARYA)` twice, in case the user wants to look them up by a single name like `ZARYA`. What a mess. Users should instead call the simple `tle_file()` method, and themselves build any dictionaries they need.

Parameters

- **tle_file_name** (*str*) – The name of the file (in the IXPEOBSSIM_INSTRUMENT_DATA folder) containing the TLE data.
- **satellite_name** (*str*) – The name of the satellite to be looked for in the TLE data ancillary files. In absence of sensible TLE data for IXPE, NUSTAR and AGILE are two scientific satellites in near-equatorial orbit that can be used for our purpose.

met_to_ut(*met*, *use_iau2000b=True*)

Convert MET(s) to skyfield.Time object(s), using the proper underlying timescale.

See https://github.com/skyfielders/python-skyfield/issues/373 for all the gory details of the iau2000b magic! That really saved the day.

position(*met*)

Return the position (longitude and latitude in decimal degrees) of the satellite at a given MET.

saa_epochs(*start_met*, *stop_met*, *initial_step=100.0*, *precision=0.001*)

Calculate the SAA epochs.

saa_mets(*start_met*, *stop_met*, *initial_step=100.0*, *precision=0.001*)

Return the list of METs for which we cross the SAA boundaries within a given time interval.

sun_constrained(*met*, *target_ra*, *target_dec*)

Return whether a given target at a given time cannot be observed due to the Sun constraints.

sun_constraint_epochs(*start_met*, *stop_met*, *target_ra*, *target_dec*, *initial_step=1000.0*, *precision=0.001*)

Calculate the epochs when observations are inhibited by the Sun constraints.

sun_constraint_mets(*start_met*, *stop_met*, *target_ra*, *target_dec*, *initial_step=1000.0*, *precision=0.001*)

Calculate the list of METs for which the Sun observability changes.

target_moon_angle(*met*, *target_ra*, *target_dec*)

Return the angular separation between a target RA and DEC and the Moon at a given MET.

target_occulted(*met*, *target_ra*, *target_dec*, *limiting_altitude=200.0*)

Return whether a given target is occulted at a given time.

target_planet_angle(*met*, *target_ra*, *target_dec*, *planet_name*)

Return the angular separation between a target RA and DEC and a planet at a given MET time.

> **Warning**
>
> For now the angular separation is calculated from a geocentric (i.e., not spacecraft) position, which should be sufficient for our needs.

target_sun_angle(*met*, *target_ra*, *target_dec*)

Return the angular separation between a target RA and DEC and the Sun at a given MET.

timeline_mets(*start_met*, *stop_met*, *target_ra*, *target_dec*, *saa*, *occult*, *initial_step=100.0*, *precision=0.001*)

Return a list of all the relevant MET values for the calculation of the observation timeline, i.e., those that pertain to the SAA and to the Earth occultation.

The calculation is started with the observaton start and stop met, and the SAA and Earth occultation MET values are addedd if necessary.

The function returns separate lists for the SAA and Earth occultation MET values, as well as a sorted logical OR of the two with no, diplicates, since all is needed downstream to calculate the timeline.

class ixpeobssim.instrument.traj.**xObservationTimeline**(*start_met*, *stop_met*, *target_ra*, *target_dec*, *saa*, *occult*)

Small container class encapsulating the detailed planning, in terms of GTIs, calibration intervals and SAA passages, for a specific observation (i.e., a portion of the mission with a specific, fixed pointing).

This is complete rewrite of the code that used to live into the gti_list() method of the xIXPETrajectory class, ini response to https://bitbucket.org/ixpesw/ixpeobssim/issues/417 The xIXPETrajectory class is still responsible for all of the complex logics and calculation to detect the relevant MET values for the SAA passages and the Earth occultation, and these values are turned into a list of xTimelineEpoch objects to be consumed downstream.

This class provides a coherent and unified interface to extract good time intervals and onboard calibration intervals, with optional padding on either end and with the possibility of enforcing a minimum duration (after the padding).

epoch_data()

Return the epoch data in a form that can be used to fill the TIMELINE extension in the output FITS files.

filter_epochs(*min_duration=0.0*, *start_padding=0.0*, *stop_padding=0.0*)

Filter the epochs by duration.

gti_list(*min_duration=0.0*, *start_padding=0.0*, *stop_padding=0.0*)

Return the list of the good time intervals (GTI).

The GTIs are defined as those when we are not in the SAA and we are not occulted by the Earth.

Note the return value of this function is not a plain old list, but a xGTIList object.

octi_list(*min_duration=0.0*, *start_padding=0.0*, *stop_padding=0.0*)

Return the list of the onboard calibration time intervals (OCTI).

The OCTIs are defined as those when we are occulted by the Earth, but outside the SAA.

sc_data(*time_step*, *dither_params=None*, *saa=True*, *occult=True*)

Return the spacecraft data to be written into the SC_DATA optional extension of the output photon list.

Spacecraft data are instantaneous data (e.g., longitude, latitude, elevation) evaluated on a regular grid with constant spacing. Rather than with the start or the stop of the observation, the grid is aligned with MET = 0, and padded as necessary on both ends to include the entire span of the observation.

The relevant data are returned into the form of a Python dictionary of the form {column_name: value_array} that can be used directly to fill the columns of the output binary table.

29.5.1 Arguments:

time_step: float
Interval used to save the SC information

dither_params
[(amplitude, pa, px, py) tuple, optional] The parameters for the dithering of the observatory. The dithering is not applied if this is set to None.

saa
[bool] Flag indicating whether the time intervals in the SAA should be flagged—the corresponding column with be identically zero if this is false.

occult
[bool] Flag indicating whether the time intervals when the target is occulted should be flagged—the corresponding column with be identically zero if this is false.

class ixpeobssim.instrument.traj.**xSAABoundary**(*file_name='saa_polygon.txt'*)

Small container class representing the SAA boundary.

The main purpose of this class is to encapsulate the algorithm to figure out whether any given point on the Earth surface is inside the SAA polygon or not. You can glance through the related issue at https://bitbucket.org/ixpesw/ixpeobssim/issues/232 to get an account of the underlying discussion, but essentially we opted for the SAA class to be a subclass of matplotlib.path.Path since the latter provides all the functionality that we need (includeing plotting), implemented efficiently and without bringing in yet another dependence.

contains(*lon*, *lat*)

Return a boolean mask signaling the pairs of coordinates inside the polyogon.

plot(*plot_vertices=True*, ***kwargs*)

Plot the polygon.

class ixpeobssim.instrument.traj.**xSkyfieldLoader**(*verbose=True*, *expire=True*)

timescale(*delta_t=None*, *builtin=False*)

Open or download the three timescale files, returning a *Timescale*.

This is an almost verbatim re-implementation of the timescale() method in skyfield/iokit.py, the only difference being the locations from where we retrieve the data.

class ixpeobssim.instrument.traj.**xTLE**

Small utility function to create TLE strings for the purpose of our simulation.

See https://en.wikipedia.org/wiki/Two-line_element_set for some background information.

Mind this is not meant to be a complete interface, which would be silly—what we are after is really the ability to generate a fictional TLE for a simple circular orbit with arbitrary altitude and inclination. We purposedly do not provide the ability to write ouput file in order not to pollute the environment—this is something that should remain internal to the simulation.

And, for completeness, here is the IXPE TLE pulled out from https://www.n2yo.com/satellite/?s=49954 shortly after launch.

1 49954U 21121A 21351.00640149 .00001120 00000-0 35770-4 0 9994 2 49954 0.2300 281.7657 0011347 134.4260 303.9164 14.90740926 1166

static lines(*inclination=0.23*, *altitude=601.1*, *catalog_no=49954*, *launch_year=2021*, *launch_no=121*, *launch_piece='A'*, *launch_day=351.00640149*, *ballistic_coefficient='.00001120'*, *drag_term='35770-4'*, *element_number=9994*, *ascending_node=281.7657*, *eccentricity='0011347'*, *perigee=134.426*, *mean_anomaly=303.9164*)

class ixpeobssim.instrument.traj.**xTimelineEpoch**(*start_met*, *stop_met*, *in_saa*, *occulted*)

Small convenience class to encapsulate an "epoch" within a given observation.

In this context an epoch is the datum of a start_met, a stop_met, and a series of bit flags indicating whether the observatory is, e.g., in the SAA or occulted by the Earth.

See https://bitbucket.org/ixpesw/ixpeobssim/issues/417

bitmask()

Return a bit mask corresponding to the observatory status during the epoch.

isgti()

Return True if the epoch is a good time interval, i.e., we're not in the SAA and we're not occulted by the Earth.

isocti()

Return True of the epoch is a good on-orbit calibration time interval, i.e., we are occulted by the Earth but not in the SAA.

shrink(*start_padding*, *stop_padding*)

Create a new epoch where the bounds are shrinked by given amounts on either side. The "shrinked" epoch is inhering the SAA and occultation flags.

Note that the original object is left untouched, i.e., the new epoch is created in place.

29.6 Response Functions

Read interface to effective area files.

class ixpeobssim.irf.arf.**xEffectiveArea**(*file_path*)

VALID_WEIGHTING_SCHEMES = (None, 'UNWEIGHTED', 'NEFF', 'SIMPLE')

Class describing the on-axis effective area.

Parameters
file_path (*str*) – The path to the .arf FITS file containing the effective area table.

plot()

Plot the effective area.

weighting_scheme()

Return the weighting scheme used to calculate the effective area.

See https://bitbucket.org/ixpesw/ixpeobssim/issues/573/ for more information about why we do need to keep track of this.

The basic purpose that this hook serves is to be able to make sure that weights are used consistently across binned data products and response files, e.g., we don't want for people to be able to create weighted polarization cubes with the effective area correction unless the arf file is created with the 'SIMPLE' prescription.

For backward compatibility this is returning None if the response file is missing the *STOKESWT* keyword in the *SPECRESP* extension. If this is the case, the user should not make assumptions about the actual weighting scheme used.

class ixpeobssim.irf.arf.**xTowEffectiveArea**(*file_path*)

Class describing the on-axis effective area at the top of the window.

Parameters
file_path (*str*) – The path to the FITS file containing the effective area table.

Base classes for all the read interfaces to response files.

The classes in this module represent the base of the hyerarchy for all the classes descibing response files, and particularly:

- xResponseBase acts as a base class for *all* the response files;
- xSpecRespBase acts as a base class for the (on-axis) effective area, the modulation factor and the modulation respose function.

The latter (i.e., the classes inheriting from xSpecRespBase) all have in common being one-dimensional functions of the (true) energy, and have a SPECRESP extension containing the response in bins of energy.

The xSpecRespBase class is also equipped to allow for systematic errors on the response values, according to the proper OGIP standard.

class ixpeobssim.irf.base.**xResponseBase**(*file_path*, *extension*)

Base class for reading response data from FITS files.

Parameters

- **file_path** (*str*) – The path to the input FITS file.
- **extension** (*str*) – The extension of the input FITS file (typically fits, arf or rmf)

field(*col_name*, *ext_index=1*)

Return a view over a column of the specified extension as an array.

Since most of the response files have exactly one relevant (binary table) extension, having the extension index defaulting to one is a handy shortcut to retrieve the column data in a general fashion.

Parameters

- **col_name** (*str*) – The name of the target column in the binary table extension.
- **ext_index** (*int*) – The index of the target binary table extension.

header_comments()

Return the content of the COMMENT keyword of the primary header, split into lines.

This is essentially only used in ixpeobssim.evt.ixpesim to reverse-engineer the value of the GPD pressure for the purpose of building the telescope response at the top of the Be window.

class ixpeobssim.irf.base.**xSpecRespBase**(*file_path*, *extension*, *k=2*, *pad=True*)

Derived class describing a spectral response, i.e., effective area, modulation response function, or modulation factor.

has_sys_errors()

Return True if the object instance has both negative and positive systematic errors defined.

plot_base(***kwargs*)

Plot the spectral response.

plot_sys_envelope()

Plot the envelope of the systematic errors.

plot_sys_errors()

Plot the fractional systematic errors.

sys_envelope()

Return the envelope of the systematic uncertainties.

Basic IRF naming conventions and CALDB-related facilities.

ixpeobssim.irf.caldb.**irf_file_name**(*base*, *du_id*, *irf_type*, *intent*, *version*, *simple_weighting=False*, *gray_filter=False*)

Return the file name for a specific response function.

Parameters

- **base** (*str*) – Base name of the set of response functions.
- **du_id** (*int*) – Identifier of the specific DU.
- **irf_type** (*str*) – Identifier for the calibration data.
- **intent** (*str*) – The specific intent for the response file.
- **version** (*int*) – Version number.
- **simple_weighting** (*bool*) – If True, load the response file with the SIMPLE weighting scheme.
- **gray_filter** (*bool*) – If True, load the response files apppropriate for the gray filtes (where) this makes sense.
- **is** (*The basic naming convention scheme used for the CALDB by HEASARC*)
- **ixpe_[instrument]_[datatype]_[date]_[ver].[ext]**
- **where**
- **(d1/d2/d3);** (** [instrument] indicates the detector*)
- **data;** (** [datatype] provides an identifier for the calibration*)
- **file;** (** [date] indicates the first date of validity of the*)
- **file.** (** [ver] is the CALDB version number for that*)
- **conventions** (*We approximately follow the HEASARC*)
- **that** (*except for the fact*)
- **date.** (*we don't really have a concept of validity*)
- **filter** (*Note that all the logic for handling the simple weighting and the gray*)
- **https** (*was moved here following*)

ixpeobssim.irf.caldb.**irf_file_path**(*irf_name*, *du_id*, *irf_type*, *caldb_path=None*, *check_file=True*, *simple_weighting=False*, *gray_filter=False*)

Return the full file path to a particular IRF file.

ixpeobssim.irf.caldb.**irf_folder_path**(*irf_type*, *caldb_path='/home/docs/checkouts/readthedocs.org/user_builds/ixpeobssim/checkouts/*

Return the CALDB folder for a particular IRF type.

ixpeobssim.irf.caldb.**parse_irf_name**(*irf_name*, *delimiter=':'*)

Parse a generic IRF name and return the basic field values it encapsulates.

The `irf_name` is an internal designation that ixpeobssim uses to identitify a full (self-consistent) set of response functions.

Modulation-related facilities.

class ixpeobssim.irf.modf.**xAzimuthalResponseGenerator**

Random number generator for the azimuthal response of the polarimeter.

Here is the basic underlying math. Typically the response of a polarimeter to monochromatic, linearly polarized incident radiation is of the form:

$$N(\phi) = A + B\cos^2(\phi - \phi_0).$$

This can be conveniently rewritten in term of the overall normalization (i.e., the total number of events) and the modulation, defined as

$$m = \frac{N_{\rm max} - N_{\rm min}}{N_{\rm max} + N_{\rm min}} = \frac{B}{2A + B}$$

(the modulation can also be characterized as the fraction of modulated events in a given distribution, as can be readily demonstrated, and it is the quantity that the detector is effectively measuring). The angular response can be rewritten as

$$N(\phi) = \frac{N_0}{2\pi}\left[1 + m\cos(2(\phi - \phi_0))\right].$$

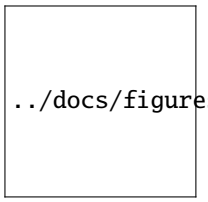
For completeness, the modulation, the modulation factor and the polarization degree for a monocromatic source are related to each other through:

$$m(E, t) = P(E, t) \times \mu(E)$$

(i.e., at any given energy the modulation factor is the modulation of the detector response for 100% linearly polarized incident radiation).

In terms of throwing random numbers, the phase is a trivial constant that can be added after the fact (modulo 2pi), so effectively the relevant probability density function is

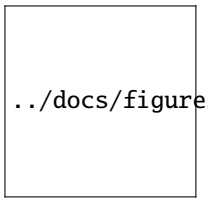
$$\text{pdf}(\phi) = \frac{1}{2\pi}\left[1 + m\cos(2\phi)\right],$$



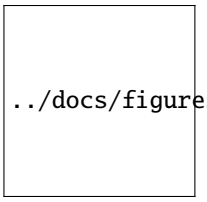
The corresponding cumulative distribution function is

$$\text{cdf}(\phi) = \frac{1}{2\pi}\left(\phi + \frac{m}{2}\sin\left(2\phi\right)\right),$$

and it is unfortunate that it cannot be inverted, otherwise we would have no need to interpolate for generating random numbers according to this distribution.



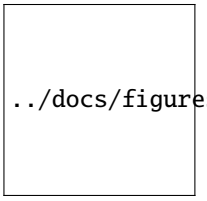
From the prospecive of the code, this generator is a standard *xUnivariateAuxGenerator* where the azimuthal angle is our random variable and the modulation is our auxiliary variable. For any given value of the modulation, a vertical slice is providing the corresponding one-dimensional pdf.



The class also provide facilities to fit a histogram to recover the underlying modulation and phase.

Example

```
>>> import numpy
>>> from ixpeobssim.utils.matplotlib_ import plt
>>> from ixpeobssim.irf.modf import xAzimuthalResponseGenerator
>>>
>>> generator = xAzimuthalResponseGenerator()
>>> modulation = numpy.full(1000000, 0.5)
>>> phase = numpy.radians(45.)
>>> phi = generator.rvs_phi(phase, modulation)
>>> hist = plt.hist(phi, bins=numpy.linspace(0, 2*numpy.pi, 100),
                    histtype='step', label='Random angles')
>>> fit_results = generator.fit_histogram(hist)
>>> fit_results.plot()
>>> plt.show()
```



build_horizontal_ppf()

Overloaded method.

This is essentially done for a few reasons:

- we do have an analytical form of the cdf that we can use to reduce the numerical noise;
- this random generator is peculiar in that the pdf is linear in the auxiliary variable m;
- this is possibly the most important random engine in the framework and its accuracy must be tested carefully.

There are many parameters that can be optimized, here, in order to maximize the accuracy of the ppf representation. (This accuracy is characterized in test/test_azimuthal_response.py in terms of standard and maximum relative error). Among these are the grids sampling the modulation and quantile values, and the order of the various splines involved. We put some thoughts into getting this right, but we cannot exclude that this can be improved.

For the modulation values we just take a constant-pitch grid, while for the quantiles we calculate the cdf (for m = 1, which is where the deviations from a straight lines are largest) on a constant-pitch grid in angle.

We empirically found that k=3 is the best spline index for the ppf at any given modulation value, while we get the maximum accuracy for kx = ky = 1 in the actual bivariate spline representing the ppf.

static cdf(*phi*, *m*)

Evaluate the underlying one-dimensional cdf for a given value of the modulation, and assuming that the phase is zero.

Parameters

- **phi** (*float or array*) – The (independent) azimuthal angle variable, in [-pi, pi].
- **m** (*float or array*) – The modulation, in [0, 1].

static pdf(*phi*, *m*)

Evaluate the underlying one-dimensional pdf for a given value of the modulation, and assuming that the phase is identically zero.

Parameters

- **phi** (*float or array*) – The (independent) azimuthal angle variable, in [-pi, pi].
- **m** (*float or array*) – The modulation, in [0, 1].

rvs_phi(*m*, *phase*)

Generate random variates for any modulation and phase values.

This is essentially calling the underlying xUnivariateAuxGenerator.rvs() method passing the modulation array as an argument and adding the phase array (modulo 2pi) to the output.

Parameters

- **m** (*array*) – An array of modulation values. (The function returns an equal-length array of phi values.)
- **phase** (*float or array*) – The phase of the modulation. (This can either be a vector or an array of the same length as *modulation*.)

class ixpeobssim.irf.modf.**xModulationFactor**(*file_path*, *extension='fits'*, *k=1*)

Class describing the modulation factor.

The effective area is essentially a linear spline, with built-in facilities for evaluation and plotting.

To zero-th order, an *xModulationFactor* instance is an object capable of evaluating itself in a point or in a grid of points, and capable of plotting itself.

More interestingly, it can generate random *phi* values, given a vector of event energies and corresponding vectors (or simple floats) representing the polarization degree and angle corresponding to the energies themselves. Internally, any *xModulationFactor* has an *xAzimuthalResponseGenerator* object and when the *xModulationFactor.rvs_phi()* method is called, the polarization degree is multiplied by the modulation factor of the detector, evaluated at the right energy, and converted into a modulation value, after which the underlying *xAzimuthalResponseGenerator.rvs_phi()* is called.

plot()

Overloaded method for the xpirfview application.

rvs_phi(*energy*, *polarization_degree*, *polarization_angle*)

Return random variates for a given array of values of energy, polarization degree and polarization angle.

Parameters

- **energy** (*array*) – An array of energy values. (The function returns an equal-length array of phi values.)
- **polarization_degree** (*array or float*) – The polarization degree, in [0–1]. (This can either be a vector or an array of the same length as *energy*.)

- **polarization_angle** (*array or float*) – The polarization angle, in radians. (This can either be a vector or an array of the same length as *energy*.)

weighted_average(*energy*)

Return the weighted average of the mudulation factor given an array of energies.

Parameters

energy (*array*) – An array of energy values.

Modulation respose function.

class ixpeobssim.irf.mrf.**xModulationResponse**(*file_path*)

Class describing the modulation response, i.e., the product of the effective area times the modulation factor.

plot()

Plot the modulation response.

PSF parametrization.

ixpeobssim.irf.psf.**gauss_king**(*r*, *W*, *sigma*, *N*, *r_c*, *eta*)

Functional representation of the Gaussian plus King PSF profile described in Fabiani et al., 2014, equation (2):

$$\mathrm{PSF}(r) = W\exp^{-(\frac{r^2}{2\sigma^2})} + N\left(1+\left(\frac{r}{r_c}\right)^2\right)^{-\eta}$$

Parameters

- **r** (*float or array*) – The radial distance from the true source position is arcsec.
- **W** (*float*) – Normalization of the Gaussian component.
- **sigma** (*float*) – Width of the Gaussian component.
- **N** (*float*) – Normalization of the King component.
- **r_c** (*float*) – Characteristic radius of the King component.
- **eta** (*float*) – Exponent of the King component.

ixpeobssim.irf.psf.**gauss_king_eef_at_infinity**(*W*, *sigma*, *N*, *r_c*, *eta*)

Return the value of the Encircled Energy Fraction (EEF) at infinity, given the parameters of the functional representation, see equation (4) of Fabiani et al., 2014.

$$\mathrm{EEF}(\infty) = 2\pi W\sigma^2 + \pi\frac{r_c^2 N}{\eta - 1}$$

Parameters

- **r** (*float or array*) – The radial distance from the true source position is arcsec.
- **W** (*float*) – Normalization of the Gaussian component.
- **sigma** (*float*) – Width of the Gaussian component.
- **N** (*float*) – Normalization of the King component.
- **r_c** (*float*) – Characteristic radius of the King component.
- **eta** (*float*) – Exponent of the King component.

class ixpeobssim.irf.psf.**xPointSpreadFunction**(*file_path*)

Class describing a (simplified, energy independent) PSF.

The effective area is essentially a linear spline, with built-in facilities for evaluation and plotting.

Parameters
psf_file_path (*str*) – The path to the .psf FITS file containing the PSF parameters.

> **Note**
>
> The parametrization is taken from Fabiani et al., 2014, table 2.

build_eef()

Build the Encircled Energy Fraction (EEF) as a function of r.

And, while we're at it, we also calculate and cache the HEW.

delta(*size=1*)

Return an array of random offset (in ra, dec or L, B) due to the PSF.

Note the output is converted in degrees.

draw_psf_circle(*image*, *x=0.8*, *y=0.8*, *label='HPD'*, *hpd_value=False*, *color='white'*, *lw=1.5*)

Add the PSF circle to the image with labels.

Parameters

- **image** (*xFITSImageBase instance*) – The parent xFITSImageBase object
- **x** (*float*) – The position of the psf circle in relative canvas coordinates
- **y** (*float*) – The position of the psf circle in relative canvas coordinates

plot()

Overloaded plot method.

class ixpeobssim.irf.psf.**xPointSpreadFunction2d**(*file_path*)

Two-dimensional version (i.e., non azimuthally symmetric) version of the PSF.

Parameters
file_path (*str*) – The file path for the PSF image

build_eef()

Calculate the (azimuthally averaged) encircled energy fraction as a function of r.

delta(*size=1*)

Return an array of random offset (in ra, dec) due to the PSF.

class ixpeobssim.irf.psf.**xPointSpreadFunctionBase**(*gpd*)

Base virtual class for the PSF data structures.

Added in version 28.1.0: This was added in version 27.1.0 as a base class for the old-style PSF (azimuthally symmetric and including the position resolution of the GPD) and the new-style PSF necessary to study the Stokes cross-talk (non azimuthally symmetric and limited to the X-ray optics).

This class holds a single boolean flag inficating whether the GPD position resolution is included in the PSF. Sub-classes should reimplement the delta() hook, returning random arrays of (ra, dec) offsets.

Parameters
gpd (*bool*) – Boolean flag indicating whether a given PSF includes the effect of the GPD position resolution.

delta(*size=1*)

Return an array of random offset (ra, dec) due to the PSF.

smear(*ra*, *dec*)

Smear a pair of arrays of coordinates.

smear_single(*ra*, *dec*, *size=1*)

Smear a single pair of coordinates for an arbitrary number of times.

Definition of the response matrix.

class ixpeobssim.irf.rmf.**xEnergyDispersion**(*file_path*)

Class representing the energy dispersion.

Parameters
file_path (*str*) – The path to the .rmf FITS file containing the energy dispersion tables.

channel_to_energy(*channel*)

Convenient proxy to the underlying xEnergyDispersionBounds method.

convolve_energy(*mc_energy*)

Convolve an array of mc_energy with the energy dispersion.

This function is less trivial than the name might suggest, and it is performing several different operations at the same time, namely:

- call xEnergyDispersionMatrix.rvs2() to smear the true energy and get the measured energy (in channel space) both before and after the digitization;
- convert the measured energy before digitization to keV;
- convert the pha to pulse invariant.

energy_to_channel(*energy*)

Convenient proxy to the underlying xEnergyDispersionBounds method.

pha_analysis(*energy*)

Perform a pulse-height analysis on a given array of energies.

static pha_to_pi(*pha*)

Simple placeholder method to convert a PHA value to pulse invariant.

> **Note**
>
> Since we're not handling the gain corrections, yet, this is simply converting the values from integer to floating point numbers.

plot()

Plot the energy dispersion.

static scale_energy(*energy*)

Hook to allow to apply distorsions to the energy response of the detector.

In this default implementation this is just returning the energy, untouched, but the method can be overridden to simulate a variety of systematic effects.

class ixpeobssim.irf.rmf.**xEnergyDispersionBounds**(*hdu*)

Class encapsulating the bounds for the energy dispersion matrix, as stored in the EBOUNDS extension of a .rmf file.

This is essentially a univariate spline with the channel number on the x axis and the average energy of the corresponding interval on the y axis. At the same time we do store the bounds of the intervals themselves.

channel_to_energy(*channel*)

Convert a channel number to energy.

energy_to_channel(*energy*)

Convert a physical energy (in keV) into a channel number.

This is using a simple binary search on the underlying energy bounds.

max()

Return the maximum energy of the last channel.

min()

Return the minimum energy of the first channel.

num_channels()

Return the number of channels.

class ixpeobssim.irf.rmf.**xEnergyDispersionMatrix**(*hdu*)

Class encapsulating the energy dispersion matrix, as stored in the MATRIX extension of a .rmf file.

We don't downsample the matrix to avoid the failure of the fit procedure in XSPEC.

Parameters
hdu (*FITS hdu*) – The MATRIX hdu in the .rmf FITS file.

static digitize(*value*, *dtype=<class 'numpy.int16'>*)

Digitize an energy value.

This is essentially rounding an array to the nearest integer and cast the result to the specified type.

rvs(*aux*)

Overloaded method.

This is the original rvs() overloade method that was initially implemented in the energy dispersion code, and it is digitizing the measured energy before returning it, just like the detector would do in real life.

We discovered along the way that it is handy to keep track of the measured (smneared) energy *before* the digitization (i.e., with all the significant digits), as this allows to apply corrections (e.g., for the GEM charging) without running into rounding problems. This is why we added the rvs2() class method, which is returning both.

rvs2(*aux*)

Alterntative rvs() implementation where we return the measured energy both *before* and *after* the digitization.

Note that the energy before the digitization is still in the pha space, and needs to be explicitely converted to physical units (keV), if that it what one is interested. When that's the case, the operation is deferred to the xEnergyDispersion class, who is aware of both the energy dispersion matric and the energy bounds.

PSF parametrization.

class ixpeobssim.irf.vign.**xVignetting**(*file_path*)

Class describing vignetting.

The vignetting is essentially a bivariate linear spline, with built-in facilities for evaluation and plotting.

Parameters
file_path (*str*) – The path to the vign.fits file containing the vignettting table.

plot()

Plot the vignetting.

29.7 IRF Generation

Small module providing simulation tools for the alpha particles emitted by the Be window.

ixpeobssim.irfgen.alphas.**gas_cell_intersection_points**(*ray*)

Calculate the intersecting points of a given ray with the gas cell.

This is a horrible list of conditions, and should be

ixpeobssim.irfgen.alphas.**generate_decays**(*alpha_energy=5.0*, *num_alphas=1000000*, *plot=True*, *energy_cut=0.1*)

Generate the initial alpha particles in the Be window volume.

This is returning a list of xRay object and a numpy array of energies describing the alpha particles that emerge from the window in a direction that is intersecting the fiducial cylinder enclosing the GPD active gas volume.

This first part of the program is vectorized, as only a small fraction of the particles survive the initial cut, and the whole thing would be excruciatingly slow, if stuffed into a plain Python loop.

ixpeobssim.irfgen.alphas.**generate_distributions**(*alpha_energy=5.0*, *num_alphas=10000000*)

ixpeobssim.irfgen.alphas.**plot_distributions**(*alpha_energy=5.0*, *num_alphas=1000000*)

ixpeobssim.irfgen.alphas.**plot_event**(*ray*, *enery*, *track_start=None*, *track_end=None*)

Rudimentary single-event display.

ixpeobssim.irfgen.alphas.**plot_raindrops**(*alpha_energy=5.0*, *num_alphas=2000*, *max_roi_size=5000*)

ixpeobssim.irfgen.alphas.**plot_rectangle**(*width*, *height*, *center=(0.0, 0.0)*, ***kwargs*)

Facility to plot a generic rectangle.

ixpeobssim.irfgen.alphas.**simulate**(*alpha_energy=5.0*, *num_alphas=100000*, *plot=False*, *energy_cut=0.1*)

Run the actual simulation.

ixpeobssim.irfgen.astar.**load_alpha_stopping_power_data**(*identifier*, *density=None*)

Load the stopping power data for a given element or compound.

ixpeobssim.irfgen.astar.**load_dme_alpha_stopping_power_data**(*temperature=20.0*, *pressure=800.0*)

Load the stopping power data for a given element or compound.

class ixpeobssim.irfgen.astar.**xAlphaStoppingPowerTable**(*identifier*, *density=None*)

Basic interface to the output files of the ASTAR database.

bragg_curve(*energy=10.0*, *energy_step=0.01*)

Return a spline representing the Bragg curve for the element or compound.

energy_loss(*max_energy=10.0*, *energy_step=0.01*)

process_data()

Do all the post-processing of the underlying data.

class ixpeobssim.irfgen.astar.**xDMEAlphaStoppingPowerTable**(*temperature*, *pressure*)

Specialized subclass for the DME compound.

Library for producing auxiliary files for the IRF generation.

ixpeobssim.irfgen.auxiliary.**absorption_type_masks**(*mc_absz*, *min_dme_absz=0.832*, *max_dme_absz=10.83*)

Return a set of three masks for the window, DME and GEM absorption.

This is based on the Monte Carlo z coordinate of the absorption point, and it is not the most elegant solution, as the correctness of the mask relies on the fact that we use the right bounds for the thickness of the absorption gap used in the simulations. (Since the latter is essentially never changed, this is less horrible than it might seems at a first glance.)

Moving forward, fixing https://bitbucket.org/ixpesw/gpdsw/issues/222 and use the exact information might be the way to go.

ixpeobssim.irfgen.auxiliary.**allx_edisp_file_path**(*abs_label*, *weight_name*, *aux_version=3*)

Return the path to the allx edisp data file path for a given absorption type and set of weights.

ixpeobssim.irfgen.auxiliary.**allx_pha_model_file_path**(*aux_version=3*)

Return the path to the file with the PHA model.

ixpeobssim.irfgen.auxiliary.**allx_qeff_file_path**(*weight_name*, *aux_version=3*)

Return the path to the allx lines data file path for a given absorption type and set of weights.

ixpeobssim.irfgen.auxiliary.**allx_rmf_file_path**(*abs_label*, *weight_name=None*, *aux_version=3*)

Return the path to the FITS file(s) to be used to compose the response matrix.

ixpeobssim.irfgen.auxiliary.**calculate_efficiency**(*n*, *N*)

Calculate the efficiency, given a number of trials and success, and attach the classical binomial error to it.

ixpeobssim.irfgen.auxiliary.**calculate_pi**(*mc_energy*, *pha*)

Perform the conversion from PHA to PI.

This is necessary because the PHA values in the ixpesim simulation are expressed in the terms of the native detector ADC counts, and the scale factor to the actual energy channels in the response functions is arbitrary. In real life the conversion will be based on the ground calibration with monochromatic sources, and this is meant to be a sensible proxy for that process for operating on Monte Carlo data sets.

Note this operates differently depending on whether we're working with line data or a continoum spectrum.

Parameters

- **mc_energy** (*array_like*) – The array of (true) energies in keV
- **pha** (*array_like*) – The array of measured PHA values.

Returns

- **rec_energy** (*array_like*) – The array of reconstructed energies (in keV) corresponding to the input PHAs.
- **pi** (*array_like*) – The array of pulse invariants corresponding to the input PHAs.
- **pha_model** (*xInterpolatedUnivariateSpline object*) – The model to operate the conversion between PHAs and PIs.

ixpeobssim.irfgen.auxiliary.**calculate_pi_broadband**(*mc_energy*, *pha*, *energy_binning=None*)

Calculate the event pulse invariant for a broadband simulation.

This is trying to account for the bulk of the non linearities by fitting the normalized PHA in different energy slices and then fitting the scale factor as a function of the energy with an erf function. (The effect is of the order of 2% from 2 keV to 12 keV, at about 3000 ADC counts per keV nominal.)

ixpeobssim.irfgen.auxiliary.**calculate_pi_line**(*pha*, *aux_version=3*)

Convert the pha column into pulse invariant for a line simulation.

Note that, for line simulations, we use the PHA model derived from the allx data set.

ixpeobssim.irfgen.auxiliary.**event_weights**(*event_data*, *label='alpha075'*)

Book-keeping function mapping text labels to weights.

ixpeobssim.irfgen.auxiliary.**lines_edisp_file_path**(*abs_label*, *weight_name*, *aux_version=3*)

Return the path to the lines edisp data file path for a given absorption type and set of weights.

ixpeobssim.irfgen.auxiliary.**lines_qeff_file_path**(*weight_name*, *aux_version=3*)

Return the path to the lines qeff data file path for a given absorption type and set of weights.

ixpeobssim.irfgen.auxiliary.**lines_rmf_file_path**(*abs_label*, *weight_name=None*, *aux_version=3*)

Return the path to the FITS file(s) to be used to compose the response matrix.

ixpeobssim.irfgen.auxiliary.**load_allx_edisp_data**(*abs_label*, *weight_name=None*, *aux_version=3*)

Load the (raw) response matrix data created with mkauxallx.py.

ixpeobssim.irfgen.auxiliary.**load_allx_rmf_hist**(*abs_label*, *weight_name=None*, *aux_version=3*)

Load the post-processed histogram with the rmf data.

ixpeobssim.irfgen.auxiliary.**load_event_data**(**file_list*, *absorption_masks=True*, *pi=True*)

Load the data from the (reconstructed) event files generates with ixpesim.

This is chaining together all the data in the input files—for allx simulations you should pass all the files to a single function call, while for lines you should be open them one by one.

The event data are returned in the form of a dictionary indexed by the corresponding column names in the EVENTS and MONTE_CARLO extensions of the photon lists. Metadata are stored into a separate info dictionary.

ixpeobssim.irfgen.auxiliary.**load_lines_edisp_data**(*abs_label*, *weight_name=None*, *aux_version=3*)

Load the quantum efficiency data from file.

ixpeobssim.irfgen.auxiliary.**load_lines_rmf_hist**(*abs_label*, *weight_name=None*, *aux_version=3*)

Load the post-processed histogram with the rmf data.

ixpeobssim.irfgen.auxiliary.**load_pha_model**(*aux_version=3*)

Load a PHA to PI model from file.

ixpeobssim.irfgen.auxiliary.**load_pressure_scan_table**(*file_path*, *col_name='MU2'*)

Load the pressure scan data from file.

This is reading the binary table stored in the FITS files and arranging them in an order that is suitable for interpolation in pressure and energy.

ixpeobssim.irfgen.auxiliary.**load_qeff_table**(*file_path*)

Load the quantum efficiency data from file.

This is reading the binary table stored in the FITS files and returning the columns in the form of a Python dictionary indexed by column name.

ixpeobssim.irfgen.auxiliary.**pscan_modf_file_path**(*weight_name*, *aux_version=3*)

Return the path to the modulation factor data from the analysis of the pressure scan.

ixpeobssim.irfgen.auxiliary.**write_pha_model**(*file_path*, *model*, *info*)

Write the PHA to PI conversion model table to file.

ixpeobssim.irfgen.auxiliary.**write_pressure_scan_table**(*file_path*, *data*, *info*)

Write a pressure table to file.

ixpeobssim.irfgen.auxiliary.**write_qeff_table**(*file_path*, *data*, *info*)

Write a quantum efficiency table to file.

class ixpeobssim.irfgen.auxiliary.**xEdispModelGenGauss**

Energy dispersion fitting model.

This is the combination of a generalized Gaussian distribution for the main peak, and a hat-shaped bridge for the tail, with the possible addition of a low-energy bump.

static value(*x*, *norm*, *peak_frac*, *peak*, *sigma*, *beta*, *bump_frac*)

Overloaded value() method.

class ixpeobssim.irfgen.auxiliary.**xEdispModelLogNormal**

Energy dispersion fitting model.

This is the combination of a log-normal distribution for the main peak, and a hat-shaped bridge for the tail, with the possible addition of a low-energy bump.

static value(*x*, *norm*, *peak_frac*, *peak*, *sigma*, *shape*, *endpoint*, *bump_frac*, *bump_sigma*)

Overloaded value() method.

Base module for all the facilities related to the creation of response functions.

ixpeobssim.irfgen.base.**make_arf**(*irf_name*, *du_id*, *pressure*, *creator*, *comments*, *weight_name=None*, *aux_version=3*)

Write the IXPE effective area response function.

ixpeobssim.irfgen.base.**make_modf**(*irf_name*, *du_id*, *pressure*, *creator*, *comments*, *weight_name=None*, *aux_version=3*)

Write the IXPE modulation factor response function.

ixpeobssim.irfgen.base.**make_mrf**(*irf_name*, *du_id*, *pressure*, *creator*, *comments*, *weight_name=None*, *aux_version=3*)

Write the IXPE effective area response function.

ixpeobssim.irfgen.base.**make_psf**(*irf_name*, *du_id*, *params*, *creator*, *comments*, *weight_name=None*, *aux_version=3*)

Write the IXPE PSF response function.

ixpeobssim.irfgen.base.**make_rmf**(*irf_name*, *du_id*, *pressure*, *creator*, *comments*, *weight_name=None*, *aux_version=3*)

Write the IXPE edisp response function.

The specifications are described at page ~15 of the following document: ftp://legacy.gsfc.nasa.gov/caldb/docs/memos/cal_gen_92_002/cal_gen_92_002.ps

ixpeobssim.irfgen.base.**make_vign**(*irf_name*, *du_id*, *pressure*, *creator*, *comments*, *weight_name=None*, *aux_version=3*)

Write the IXPE vignetting response function.

ixpeobssim.irfgen.du.**uv_filter_transparency**(*energy*)

Return the transparency of the UV filter evaluated on a given grid of energy points.

ixpeobssim.irfgen.du.**uv_filter_transparency_spline**()

Return a spline with the transparency of the UV filter as a function of the energy.

ixpeobssim.irfgen.gpd.**dme_photoemission_frac_spline**(*pressure=800.0*)

Return a spline representing the fraction of photoelectons extracted from the DME as a function of energy.

This is achieved by calculating the probability of extracting a photoelectron from the Be window and the GEM using the Geant 4 Monte Carlo simulation.

ixpeobssim.irfgen.gpd.**load_ixpesim_ancillary_data**(*file_name='ixpesim_stdlines_20191109.txt'*)

Load the ancillary data calculated with the full Geant 4 Monte Carlo simulation to inform the generation of the instrument response functions.

The columns represent, in order:

- Energy [keV];
- Trigger efficiency;
- Probability of extraction of a photoelectron from the Be window;
- Probability of extraction of a photoelectron from the GEM Cu upper layer;
- Simulated value of the modulation factor.

ixpeobssim.irfgen.gpd.**modulation_factor**(*energy*, *pressure*, *weight_name=None*, *aux_version=3*)

Return the modulation factor calculated on a grid of points.

ixpeobssim.irfgen.gpd.**photoabsorption_efficiency**(*energy*, *temperature*, *pressure*, *thickness=1.0*)

Return the photoabsorption efficiency of the GPD absorption gap.

ixpeobssim.irfgen.gpd.**quantum_efficiency**(*energy*, *temperature*, *pressure*, *weight_name=None*, *aux_version=3*)

Return the overall quantum efficiency of the GPD.

Parameters

- **energy** (*array-like*) – The energy array where the quantum efficiency is calculated
- **temperature** (*float*) – The GPD filling temperature
- **pressure** (*float*) – The GPD gas pressure
- **weight_name** (*str*) – The label for the weights to be used.
- **aux_version** (*int*) – The version of the auxiliary data products used to model the passive conversions.

ixpeobssim.irfgen.gpd.**window_al_transparency**(*energy*, *thickness=5.3e-06*)

Return the transparency of the 53 nm alumination of the Be window.

ixpeobssim.irfgen.gpd.**window_be_transparency**(*energy*, *contaminants={'Ag': 10.0, 'Al': 300.0, 'B': 3.0, 'BeO': 6000.0, 'C': 400.0, 'Ca': 100.0, 'Cd': 2.0, 'Co': 10.0, 'Cr': 100.0, 'Cu': 100.0, 'Fe': 800.0, 'Li': 3.0, 'Mg': 490.0, 'Mn': 100.0, 'Mo': 20.0, 'Ni': 100.0, 'Pb': 20.0, 'Si': 300.0, 'Th': 2.0, 'U': 140.0}*, *thickness=0.005*)

Return the transparency of the 50 um Be window.

Parameters

- **energy** (*array-like*) – The array of energy values where we calculate the transparency.
- **contaminants** (*dict, optional*) – A dictionary containing the list and concentration of contaminants in the Be window.

ixpeobssim.irfgen.gpd.**window_transparency**(*energy*, *contaminants={'Ag': 10.0, 'Al': 300.0, 'B': 3.0, 'BeO': 6000.0, 'C': 400.0, 'Ca': 100.0, 'Cd': 2.0, 'Co': 10.0, 'Cr': 100.0, 'Cu': 100.0, 'Fe': 800.0, 'Li': 3.0, 'Mg': 490.0, 'Mn': 100.0, 'Mo': 20.0, 'Ni': 100.0, 'Pb': 20.0, 'Si': 300.0, 'Th': 2.0, 'U': 140.0}*)

Return the overall window transparency.

class ixpeobssim.irfgen.gpd.**xEdispDataInterface**(*weight_name=None*, *aux_version=3*)

Basic interface to the post-processed `ixpesim` output that is necessary to create the response matrix.

combine(*temperature=20.0*, *pressure=800.0*, *contaminants={'Ag': 10.0, 'Al': 300.0, 'B': 3.0, 'BeO': 6000.0, 'C': 400.0, 'Ca': 100.0, 'Cd': 2.0, 'Co': 10.0, 'Cr': 100.0, 'Cu': 100.0, 'Fe': 800.0, 'Li': 3.0, 'Mg': 490.0, 'Mn': 100.0, 'Mo': 20.0, 'Ni': 100.0, 'Pb': 20.0, 'Si': 300.0, 'Th': 2.0, 'U': 140.0}*)

Combine the window, DME and GEM response functions in the proper proportions (depending on the target temperature and pressure, as well as the window contaminants) to create the actual response function.

class ixpeobssim.irfgen.gpd.**xModfDataInterface**(*weight_name=None*, *aux_version=3*)

Basic interface to the post-processed `ixpesim` output parametrizing the modulatuion factor as a function of the pressure.

pressure_slice(*pressure*)

Return a slice at the given pressure, in the form of a piecewise interpolated spline.

class ixpeobssim.irfgen.gpd.**xQeffDataInterface**(*weight_name=None*, *aux_version=3*)

Basic interface to the post-processed `ixpesim` output that is necessary to create the arf response functions.

This is essentially reading the FITS file with the quantum efficiency tabulated as a function of the energy for the different conversion types (window, DME, and GEM) and caching the necessary arrays in a form that is convenient for later use.

In addition, the class is parametrizing the extraction probabilities for the window and the GEM, so that we can readily scale all the relevant quantities at different internal pressure in a self-consistent fashion without the need to re-run the original simulations.

Warning

When we updated this to allow for weights, it became clear that the efficiency of the active gas volume could not be simply computed from the analystic formula (which was the original approach).

We therefore decided to cache the (energy-dependent) weighting efficiency along with the other quantities, based on the consideration that its dinamic range is comparatively small, and it is therefore more easily amenable to interpolation.

dme_absorption_prob(*energy*, *temperature=20.0*, *pressure=800.0*, *contaminants={'Ag': 10.0, 'Al': 300.0, 'B': 3.0, 'BeO': 6000.0, 'C': 400.0, 'Ca': 100.0, 'Cd': 2.0, 'Co': 10.0, 'Cr': 100.0, 'Cu': 100.0, 'Fe': 800.0, 'Li': 3.0, 'Mg': 490.0, 'Mn': 100.0, 'Mo': 20.0, 'Ni': 100.0, 'Pb': 20.0, 'Si': 300.0, 'Th': 2.0, 'U': 140.0}*)

Return the probabilty for a photon of being absorbed in the DME at a given energy (or array of energies).

dme_quantum_efficiency(*energy*, *temperature=20.0*, *pressure=800.0*, *contaminants={'Ag': 10.0, 'Al': 300.0, 'B': 3.0, 'BeO': 6000.0, 'C': 400.0, 'Ca': 100.0, 'Cd': 2.0, 'Co': 10.0, 'Cr': 100.0, 'Cu': 100.0, 'Fe': 800.0, 'Li': 3.0, 'Mg': 490.0, 'Mn': 100.0, 'Mo': 20.0, 'Ni': 100.0, 'Pb': 20.0, 'Si': 300.0, 'Th': 2.0, 'U': 140.0}*)

Return the quantum efficiency of the DME.

gem_absorption_prob(*energy*, *temperature=20.0*, *pressure=800.0*, *contaminants={'Ag': 10.0, 'Al': 300.0, 'B': 3.0, 'BeO': 6000.0, 'C': 400.0, 'Ca': 100.0, 'Cd': 2.0, 'Co': 10.0, 'Cr': 100.0, 'Cu': 100.0, 'Fe': 800.0, 'Li': 3.0, 'Mg': 490.0, 'Mn': 100.0, 'Mo': 20.0, 'Ni': 100.0, 'Pb': 20.0, 'Si': 300.0, 'Th': 2.0, 'U': 140.0}*)

Return the probabilty for a photon of being absorbed in the GEM at a given energy (or array of energies).

gem_extraction_prob(*energy*)

Return the photoelectron extraction probability for the GEM at a given energy.

Mind we clip the output of the spline interpolation to the [0, inf] interval, as the spline might accidentally go below zero at low energy due to the noise.

gem_quantum_efficiency(*energy*, *temperature=20.0*, *pressure=800.0*, *contaminants={'Ag': 10.0, 'Al': 300.0, 'B': 3.0, 'BeO': 6000.0, 'C': 400.0, 'Ca': 100.0, 'Cd': 2.0, 'Co': 10.0, 'Cr': 100.0, 'Cu': 100.0, 'Fe': 800.0, 'Li': 3.0, 'Mg': 490.0, 'Mn': 100.0, 'Mo': 20.0, 'Ni': 100.0, 'Pb': 20.0, 'Si': 300.0, 'Th': 2.0, 'U': 140.0}*)

Return the quantum efficiency of the GEM.

Note that, while the GEM extraction probability is pressure-independent, this depends on the pressure due to the effect of the absorptions in the gas, changing the number of X-ray photons reaching the GEM top.

plot_extraction_probability()

Plot the probability of extracting a photoelectron from either the window or the GEM as a function of energy.

plot_quantum_efficiency(*temperature=20.0*, *pressure=800.0*)

Plot the quantum efficiency as a function of the energy, disaggregated by conversion type.

quantum_efficiency(*energy*, *temperature=20.0*, *pressure=800.0*, *contaminants={'Ag': 10.0, 'Al': 300.0, 'B': 3.0, 'BeO': 6000.0, 'C': 400.0, 'Ca': 100.0, 'Cd': 2.0, 'Co': 10.0, 'Cr': 100.0, 'Cu': 100.0, 'Fe': 800.0, 'Li': 3.0, 'Mg': 490.0, 'Mn': 100.0, 'Mo': 20.0, 'Ni': 100.0, 'Pb': 20.0, 'Si': 300.0, 'Th': 2.0, 'U': 140.0}*)

Return the ovearall GPD quantum efficiency, including all the absoprtion types.

weight_eff_dme(*energy*)

Return the DME weighting efficiency at a given array of energies.

weight_eff_gem(*energy*)

Return the GEM weighting efficiency at a given array of energies.

weight_eff_win(*energy*)

Return the window weighting efficiency at a given array of energies.

window_absorption_prob(*energy*, *temperature=20.0*, *pressure=800.0*, *contaminants={'Ag': 10.0, 'Al': 300.0, 'B': 3.0, 'BeO': 6000.0, 'C': 400.0, 'Ca': 100.0, 'Cd': 2.0, 'Co': 10.0, 'Cr': 100.0, 'Cu': 100.0, 'Fe': 800.0, 'Li': 3.0, 'Mg': 490.0, 'Mn': 100.0, 'Mo': 20.0, 'Ni': 100.0, 'Pb': 20.0, 'Si': 300.0, 'Th': 2.0, 'U': 140.0}*)

Return the probabilty for a photon of being absorbed in the window at a given energy (or array of energies).

window_extraction_prob(*energy*)

Return the photoelectron extraction efficiency for the Be window at a given energy.

Mind we clip the output of the spline interpolation to the [0., 1.] interval, as the spline might accidentally go below zero at low energy due to the noise.

window_quantum_efficiency(*energy*)

Another name for the same thing :-)

The window sees all the photons we generate with ixpesim, so the denominator is 1, and the quantum efficiency is exactly equal to the extraction probability.

ixpeobssim.irfgen.xcom.**create_energy_grid**(*emin=0.001*, *emax=0.015*, *num_points=100*)

Create a text file with a list of additional energies to query the XCOM databse, to be passed to the XCOM web interface.

By default this is writing out a grid of logarithmically-spaced energy values between 1 and 15 keV—one energy per line, in MeV.

ixpeobssim.irfgen.xcom.**load_xsection_data**(*identifier*)

Load the cross-section data for a given element or compound.

29.8 Source Models

Background model components.

class ixpeobssim.srcmodel.bkg.**xCelestialBkgBase**(*name*, *ra*, *dec*, *photon_spectrum*, *identifier=None*)

Base class for the celestial background components.

Celestial background is assumed to be uniform within the field of view, and is therefore modeled as a uniform disk with a radius larger than the field of view itself—the normalization should be computed self-consistently for the full disk, and when the GPD fiducial cut is applied the photons that do not hit the detector are simply thrown away.

(Other than the fact that the normalization is normalized to the solid angle of the component, this is essentially identical to any other celestial component.)

Note that, by construction, the component is unpolarized, and the column density and redshift are identically zero.

static solid_angle(*radius*)

Return the solid angle subtended by a cone of aperture radius.

See https://en.wikipedia.org/wiki/Solid_angle

Parameters
radius (*float*) – The half-apex of the cone in decimal degrees.

class ixpeobssim.srcmodel.bkg.**xExtragalacticBkg**(*ra*, *dec*, *identifier=None*)

Derived class for the extragalactic X-ray background.

All the basic formalism is taken from Gruber et al., 1999 https://iopscience.iop.org/article/10.1086/307450/pdf

Essentially in our energy band we model the extragalactic background as a broken power law with index 1.29 and break energy 41.13 keV.

Note 1.29 is the index of the photon spectrum, while the 0.29 in the paper refers to the energy spectrum, see https://bitbucket.org/ixpesw/ixpeobssim/issues/544

class ixpeobssim.srcmodel.bkg.**xGalacticBkg**(*ra*, *dec*, *rosat_r7_bkg_rate*, *identifier=None*)

Derived class for the galactic X-ray background.

A good reference for this is Tanaka, 2002: https://www.aanda.org/articles/aa/pdf/2002/06/aah3204.pdf

Operationally, I think the most convenient mean to gauge the intensity of the Galactic background is the HEASARC background tool https://heasarc.gsfc.nasa.gov/cgi-bin/Tools/xraybg/xraybg.pl and, particularly, the ROSAT count rate in the 1–2 keV energy band (R7).

class ixpeobssim.srcmodel.bkg.**xInstrumentalBkg**(*name*, *photon_spectrum*, *radial_slope=0.0*, *identifier=None*, *convolve_energy=False*)

Base class for the instrumental background

In addition to the base class xModelComponentBase, this class has the spectrum member to fully describe the spectrum of the instrumental background.

This essentially corresponds to a uniform, unpolarized source in the GPD reference frame, whose events are not convolved with any of the instrument response functions, except for the energy dispersion, if explicitely requested.

Parameters

- **name** (*str*) – The name of the component
- **photon_spectrum** (*callable*) – The photon spectrum in photons cm-2/s-1/keV-1 for the bkg source.
- **radial_slope** (*float*) – The radial slope for the distribution in detector coordinates.
- **identifier** (*int, optional*) – The source identifier.
- **convolve_energy** (*bool*) – Convolve the source spectrum with the energy dispersion (default is False).

rvs_event_list(*parent_roi*, *irf_set*, ***kwargs*)

Overloaded method.

rvs_photon_list(*parent_roi*, *irf_set*, ***kwargs*)

Extract a random photon list.

class ixpeobssim.srcmodel.bkg.**xPowerLawInstrumentalBkg**(*norm=0.0004*, *index=1.0*, *radial_slope=0.0*)

Specialized instrumental background source with a power-law spectrum.

The default values for the spectral parameters are set after Bunner et al. 1978 (1978ApJ...220..261B), where the authors provide the non X-ray background rates for their three detectors. We are using values for the Neon filled detector in Table 3 of the paper. We fit the three points (energy bins: 0.76–1.6, 1.6–3.0, 3.0–6.0) with a power law. The best-fit values for the index and normalization of this power-law are 1.0 and 4.e-4, respectively.

class ixpeobssim.srcmodel.bkg.**xRadialBackgroundGenerator**(*radial_slope*, *num_points=100*)

Univariate generator sampling the radial coordinate on the detector surface for a background component whose radial distribution, normalized by the area, depends linearly on the radius.

This was introduced in https://github.com/lucabaldini/ixpeobssim/issues/663

The need for an instrumental background component that is not uniform in detector coordinates stems from the analysis of a number of observations. Interestingly enough, the radial slope of the background counts per unit area seems to be different for different observation.

> **Warning**
>
> While initially I was hoping to code this by sampling two independent random variables, within the proper bounds, on the x and y coordinates, it turned out that an azimuthally symmetric bivariate pdf cannot be expressed as the product of two independent variables on a square, and we had to resort to sampling r over a circle and trimming in the fiducial rectangle after the fact. This is hugly, as we need to guess in advance how much random numbers we have to throw so that we end up with enough counts after the trimming, but so life goes.

This is essentially an univariate generator whose pdf is a slight generalization of the function f(r) = 2r (for r = 1) that one would use to throw random numbers uniformly distributed in a circle:

$$p(r) = \left(1 - \frac{\alpha}{2}\right) \frac{r}{h} + \alpha \left(\frac{r}{h}\right)^2$$

The radial slope alpha repesents the fractional half-excursion of the variation across the size h of the fiducial rectangle. For alpha = 0 the detector position are distributed uniformly over the fiducial rectangle. For alpha = 2 the radial dependence is maximal, and the density of events is zero at the center of the detector. For alpha = -1 (and assuming a square fiducial region) the pdf approaches zero at the boundary of the circle.

Parameters

radial_slope (*float*) – The slope of the radial profile, that is, the fractional half-excursion of the variation across the size of the fiducial rectangle.

static average_oversample_fraction(*radial_slope*)

Return an heuristic for the average oversample fraction for a given radial slope of the profile.

This is a purely geometric factor that is easy to calculate for a flat profile (slope = 0), in which case it reads pi/2 but not trivial in the general case. Our approach is to generate events within the smallest circle ciscumscribed to the fiducial rectangle on a grid of radial slope values and measure the fraction that ends up in the fiducial rectangle itself. For completeness, this is calculated in tests/test_radial_bkg.py.

Note that this is accurate within a few % in the limit of infinite statistics.

oversampled_size(*size*, *radial_slope*, *safety_factor=1.2*, *min_size=1000*)

Return the oversampled size for a given target size and radial slope.

This is essentially the function that determines how many random numbers we need to throw to be sure that, after trimming to the fiducial region, we end up to enough events. This is purely heuristic and is based on the average_oversample_fraction() function when the statistics is large enough, with a minimum bound to be sure we are not killed by statistical fluctuation in the small number regime.

static pdf(*r*, *radial_slope*)

Small function encapsulating the underlying pdf for the random generator.

Note we are using the average physical size of the GPD along the x and y directions as an effective value for the radial parametrization.

static polar_to_cartesian(*r*, *phi*)

Convert an array of polar coordinates in the plane into the corresponding cartesian coordinates.

rvs_xy(*size*)

Extract a set of arrays of coordinate detectors.

class ixpeobssim.srcmodel.bkg.**xRosatPSPCResponseMatrix**

Simple interface to the ROSAT PSPC response matrix.

The FITS files was downloaded from the ROSAT HEASARC page, and the content seems qualitatively in agreement with the expectations based on the XRT collecting area https://heasarc.gsfc.nasa.gov/docs/rosat/ruh/handbook/node39.html#figXMAareas and the PSPC quantum efficiency https://heasarc.gsfc.nasa.gov/docs/rosat/ruh/handbook/node56.html#SECTION00730000000000000000

The on-axis effective area is included in the response matrix, and to extract it we just sum the (un-normalized) pdf in each energy bin.

The effective area is stored internally as an interpolated spline of order k=1.

class ixpeobssim.srcmodel.bkg.**xTemplateInstrumentalBkg**(*file_path='/home/docs/checkouts/readthedocs.org/user_builds/i*
emin=0.1, *emax=15.0*, *k=1*,
radial_slope=0.0)

Instrumental background based on a template.

In-flight calibration sources and calibration-related facilities.

class ixpeobssim.srcmodel.calibsrc.**xCalC**(*rate*)

Class describing the "Cal C" onboard calibration source.

rvs_detxy(*size*)

Overloaded method.

class ixpeobssim.srcmodel.calibsrc.**xCalibrationROIModel**(*source*)

Class describing a custom ROI (region of interest) model for calibration sources.

In contrast to celestial ROI models, calibration ROI models only handle one source.

rvs_event_list(*irf_set*, ***kwargs*)

Extract an event list for the full ROI.

class ixpeobssim.srcmodel.calibsrc.**xCalibrationSourceBase**(*rate*)

Base class for all (on-ground and on-orbit) calibration sources.

The common denominator to all the calibration sources is:

- the event rate is constant in time and the counting statistics is Poisson;
- a specific identifier (>=100) is assigned to each source, and set once and for all, independently on the ROI machinery (this is different from the celestial sources, where the identifier is typically assigned depending on the order in which the ROI is populated);
- there is no concept of parent ROI, as the calibration sources are supposed to be operated one at a time, and independently from the satellite pointing;
- there is no concept of GTI, the start_met and duration being literally all we care about when it comes to decide whether the source is activated or not.

rvs_detphi(*size*)

Extract the photoelectron emission direction in detector coordinates.

Do-nothing method to be re-implemented in derived classes.

rvs_detxy(*size*)

Extract the event positions in detector coordinates.

Do-nothing method to be re-implemented in derived classes.

rvs_energy(*size*)

Extract the event energies.

Do-nothing method to be re-implemented in derived classes.

rvs_event_list(*irf_set*, ***kwargs*)

Main interface to generate random events from the calibration source.

Note that the signature here is different from the celestial sources, as we don'y need the reference to the parent ROI.

Also, since the calibration sources are not meant to be combined, the deadtime and the trigged id are calculated right away in the body function.

rvs_time(*start_met*, *duration*)

Extract the event times.

Here we essentially throw a random number with a Poisson distribution based on the expected mean of events, and then extract the event times at random—uniformly distributed between the start and stop met.

This is supposed to be the common paradigm for all the calibration sources.

class ixpeobssim.srcmodel.calibsrc.**xCalibrationSourceImage**(*file_path*)

Small convenience class descring the morphology of a calibration source in detector coordinates.

> **Warning**
>
> This functionality has significant overlap with the xFITSImage class in srcmodel.roi and the two should be refactored. Also: the very concept of a map in detector coordinates stored in the form of a FITS file has a lot in common with the stuff on the spurious modulation branch, and all of these things should be accommodated in a consistent fashion.

The HALF_SIDE top-level class member was added in response to issue https://github.com/lucabaldini/ixpeobssim/issues/668 and was set to the old value of the fiducial half-side. This is now self-contained to the calibration source machinery, and we might have to generalize the code to arbitrary fiducial cuts in the future, if we ever want to heavily use this functionality (which has not been the case, up to now).

plot()

Plot the source image.

rvs_coordinates(*size=1*)

Generate random coordinates based on the image map.

class ixpeobssim.srcmodel.calibsrc.**xMonochromaticUnpolarizedCalibrationSourceBase**(*rate*, *energy*)

Specialized base class for monochromatic unpolarized calibration sources.

rvs_detphi(*size*)

Overloade method.

rvs_detxy(*size*)

Do-nothing method to be re-implemented in derived classes.

rvs_energy(*size*)

Overloade method.

class ixpeobssim.srcmodel.calibsrc.**xMonochromaticUnpolarizedFlatField**(*rate*, *energy*)

Unpolarized square flat field at a given energy.

rvs_detxy(*size*)

Overloaded method.

Module containing ephemeris utilities, classes and functions

ixpeobssim.srcmodel.ephemeris.**get_earth_barycentric_ephemeris**(*met*)

Return earth barycentric vector given a set of terrestrial times referred to MET in seconds.

Parameters
met (*array_like*) – Array of terrestrial times in MET

Returns
Return the earth barycentric vector

Return type
array

ixpeobssim.srcmodel.ephemeris.**get_eccentric_anomaly**(*t_*, *orb_eph*, *periods=100*, *samples=10*, *plot=False*)

At different time of emission of pulsar we calculate eccentric anomaly of binary system.
Eccentric Anomaly in radiants.

Parameters

- **t** (*array float*) – Array of times at which evaluate the eccentric anomaly
- **orb_eph** (*class xOrbitalEphemeris*) – Orbital ephemeris
- **periods** (*int*) – Number of periods to use to evaluate the eccentric anomaly
- **samples** (*int*) – Number of bins in the eccentric anomaly evaluation
- **plot** (*Boolean*) – If True plot the inverse function

Returns
returns the eccentric anomaly as a function of pulsar emission time

Return type
class spline

ixpeobssim.srcmodel.ephemeris.**phase_function**(*met*, *met0*, *nu0*, *nudot0=0.0*, *nuddot=0.0*)

Return the rotational phase for a given observational time.

Warning

This function is deprecated in favor of the xEphemeris.met_to_phase() class method, and is (temporarily) kept for backward compatibility.

ixpeobssim.srcmodel.ephemeris.**read_par_file**(*parfile_path*)

Method to read a parameter file and to get pulsar parameters.

The following parameters are accepted:

- PEPOCH Epoch of period/frequency (MJD)
- POSEPOCH Epoch of position measuremet (MJD)
- F0 Pulsar rotation frequency (s^-1)
- F1 Pulsar rotation frequency derivative (s^-2)
- F2 Pulsar rotation frequency second derivative (s^-3)
- P0 Pulsar period (s).
- P1 Pulsar period derivative (10^-15).

- DM Dispersion measure (pc cm^-3)
- A1 Projected pulsar semi-major axis of orbit (lt-s)
- E/ECC Eccentricity of orbit
- T0 Epoch of periastron passage of orbit (MJD)
- TASC Epoch of ascending node passage (MJD)
- PB Period of orbit (days)
- OM Longitude of periastron passage (deg)
- EPS1 First Laplace parameter [E x sin(OM)]
- EPS2 Second Laplace parameter [E x cos(OM)]
- EPS1DOT Time derivative of EPS1
- EPS2DOT Time derivative of EPS2
- OMDOT Rate of periastron advance (deg/yr)
- PBDOT Rate of change of orbital period
- XDOT Rate of change of projected semi-major axis
- ECCDOT Rate of change of eccentricity

For further information refer to the TEMPO2 user manual.

Note

Parameter files are commonly used in pulsar timing analysis, because they contain precise information about pulsar position, spin-down rate, orbital parameters (if in binary systems), etc. The structure matches with a dictionary fill-in method.

Parameters
parfile_path (*str*) – Path to the .par file containing the ephemeris

Returns
Return a dictionary containing the pulsar parameters

Return type
dictionary

`ixpeobssim.srcmodel.ephemeris.time_list`(*pulse_shape*, *start_met*, *ephemeris*, *n_events*, *duration*)

Time events extracted with the acceptance rejection method.

Parameters

- **pulse_shape** – Pulse profile shape from pulse_pol_from_harmonics_spline in ixpeobssim.srcmodel.polarization
- **start_met** (*float*) – Starting time
- **ephemeris** (*xOrbitalEphemeris object*) – The orbital ehemeris to be used.
- **n_events** (*integer*) – Number of events to simulate
- **duration** (*float*) – Time of observation

class ixpeobssim.srcmodel.ephemeris.**xEphemeris**(*met0*, *nu0*, *nudot0=0.0*, *nuddot=0.0*)

Convenience class encapsulating the ephemeris for a periodic source.

This is meant to capture effects up to the second time derivative of the frequency.

The basic task of the class is to handle the conversion from mission elapsed time to phase and vice versa. This is performed through the met_to_phase(), and phase_to_met() interfaces. Note that, in this context, by *phase* we mean a free-running quantity, while by *pulse phase* we mean the fractional part of the former, in the [0, 1[interval.

The direct transformation (met to phase or pulse phase) is performed through a simple taylor expansion

$$\phi = \nu_0(t - t_0) + \frac{1}{2}\dot{\nu}_0(t - t_0)^2 + \frac{1}{6}\ddot{\nu}(t - t_0)^3$$

while the inverse (phase to met) is done through an interpolating spline.

In addition, given a pulse shape in the form of a xUnivariateGenerator, the class is able to extract a set of random arrays of met and pulse phase via the rvs() method. This is used in ixpeobssim.srcmodel.roi for simulating periodic sources.

Parameters

- **met0** (*float*) – The reference mission elapsed time.
- **nu0** (*float*) – Rotational frequency at met0.
- **nudot0** (*float*) – First detivative of the rotational frequency at met0.
- **nuddot** (*float*) – Second derivative of the rotational frequency (constant).

dict()

Return the ephemeris content in the form of a dictionary.

This is useful, e.g., when calling xpphase passing an ephemeris object to set the fields programmatically. The reason why we're wrapping this and not using __dict__ directly is to be able to accommodate possible future changes in either the class layout or the options for the external programs.

fold(*met*, *start_met*, *phi0=0.0*)

Fold the MET values to return the corresponding arrays of pulse phase.

> **Warning**
>
> This is creating a new xEphemeris object with the reference MET set to start_met, and the frequency derivatives changed accordingly, and then calling the met_to_phase() method of this new object. This is supposed to roundtrip with the rvs() method to a decent accuracy, and is therefore the interface used in the xpphase application. Note, however, that there is a lot of numerical (more or less random) rounding going on under the hood, and this is not intended for "professional" use. (In a sense, the function internals are tweaked specifically for the numerical noise to play in such a way that we're making approximately the same errors in rvs() and fold()).

classmethod from_file(*parfile_path*)

Read a set of attributes given a parameter file.

met_to_phase(*met*)

Return the phase for a given value of array of MET values.

Note the phase, here, is free-running, i.e. not bound between 0 and 1. For a simple ephemeris with zero time derivatives this is returning a straight line with a slope equal to the (constant) frequency.

nu(*met*)

Return the source frequency at a given time.

nudot(*met*)

Return the source first derivative of frequency at a given time.

period(*met*)

Return the source period at a given time.

phase_spline(*start_met*, *duration*, *num_points=1000*)

Return an interpolated univariate spline with the phase values for a given time range.

This is mainly used in the inverted fashion to convert from phase to met.

phase_spline_inverse(*start_met*, *duration*, *num_points=1000*)

Return the inverse of the phase spline for a given time range.

This can effectively be used for calculating the proper mission elapsed times starting from a bunch of phase values.

phase_to_met(*phase*, *start_met=None*, *duration=None*, *num_points=1000*)

Return the mission elapsed times for given value or array of phases.

This is essentially the inverse of phase(), and the two are supposed to roundtrip with each other. The reason why this function is slightly more complicated is that the former cannot be inverted analytically, and since the source frequency changes with time, it is not trivial to map directly the minimum and maximum phase to the corresponding met values, which is needed to create the interpolating spline for the conversion.

The start_met and stop_met arguments are provided to control the spline operating the conversion, and should be provided if available—in this case the span of the spline is optimal and it is guaranteed that no extrapolation is needed. Both arguments default to None, in which case a zero-order guess is made as to what their value shoule be.

Parameters

- **phase** (*array_like*) – The phase value(s) to be converted in met.
- **start_met** (*float*) – The starting met value for the spline operating the conversion.
- **duration** (*float*) – The overall span for the spline operating the conversion.
- **num_points** (*int*) – The number of points for spline operating the conversion.

rvs(*pulse_profile*, *start_met*, *duration*, *num_events*)

Return two (sorted) arrays of pulse phase and met for a given pulse profile, observation interval and number of events.

Parameters

- **pulse_profile** (*xUnivariateGenerator instance*) – The pulse target profile.
- **start_met** (*float*) – The initial met for the observation.
- **duration** (*float*) – The duration of the observation.
- **num_events** (*float*) – The total number of events to be simulated.

class ixpeobssim.srcmodel.ephemeris.**xOrbitalEphemeris**(*met0*, *nu0*, *t_orbital*, *p_orbital*, *axis_proj*, *ecc=0.0*, *lon_periast=0.0*, *nudot0=0.0*, *nuddot=0.0*, *p_orbitaldot=0.0*, *eps1=None*, *eps2=None*, *model=None*)

Convenience class encapsulating an orbital ephemeris.

Parameters

- **met0** (*float*) – Epoch of period/frequency (MET)
- **nu0** (*float*) – Pulsar rotation frequency (s^-1)
- **t_orbital** (*float*) – Epoch of ascending node passage or Epoch of periastron passage of orbit (MET)
- **p_orbital** (*float*) – Period of orbit (s)
- **axis_proj** (*float*) – Projected pulsar semi-major axis of orbit (lt-s)
- **ecc** (*float*) – Eccentricity of orbit
- **lon_periast** (*float*) – Longitude of periastron passage (deg)
- **nudot0** (*float*) – Pulsar rotation frequency derivative (s^-2)
- **nuddot** (*float*) – Pulsar rotation frequency second derivative (s^-3)
- **p_orbitaldot** (*float*) – Rate of change of orbital period
- **eps1** (*float*) – First Laplace parameter [E x sin(lon_periest)]
- **eps2** (*float*) – Second Laplace parameter [E x cos(lon_periest)]
- **model** (*Binary model (BT/ELL1/DD/MSS)*)

classmethod from_file(*parfile_path*)

Read a set of attributes given a parameter file.

omega_orb(*met*)

Return the inverse of the orbit period at a given time.

ixpeobssim.srcmodel.gabs.**mapped_column_density_HI**(*ra*, *dec*, *map_name='LAB'*)

Return the mapped H_I column density at a given position in the sky.

The value is read from one of the available input maps. Note that the data in the maps are stored in Galactic coordinates, while we're giving Celestial coordinates in input, here—the transformation is handled internally.

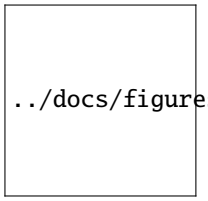
Parameters

- **ra** (*float*) – Right ascension of the source (in decimal degrees).
- **dec** (*float*) – Declination of the source (in decimal degrees).
- **map** (*str*) – The HI column density map to use. Can be either 'LAB' (LAB survey) or 'DL' (Dickey & Lockman).

class ixpeobssim.srcmodel.gabs.**xInterstellarAbsorptionModel**(*num_samples=1000*)

Class implementing the insterstellar absorption model using the Wisconsin (Morrison and McCammon; ApJ 270, 119) cross-sections.

Here we essentially read the numbers in table 2 from the paper and build an interpolated univariate linear spline with the photoabsorption cross section values. The cross section is parametrized as a set of piecewise quadratic functions fitted to the data in energy bins—see the figure below.



Example

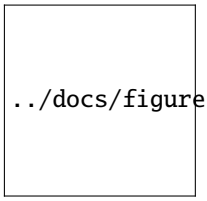
```
>>> from ixpeobssim.srcmodel.gabs import xInterstellarAbsorptionModel
>>> model = xInterstellarAbsorptionModel()
# Build a spline with the interstellar transmission factor as a function
# of the photon energy
>>> trans = model.transmission_factor(1.e22)
>>> trans.plot()
```

transmission_factor(*column_density*)

Return the transmission factor for a given column density.

This is essentially returning

$$\varepsilon = \exp(-n_H \sigma)$$



Parameters

column_density (*float*) – The column density at which the transmission factor should be calculated.

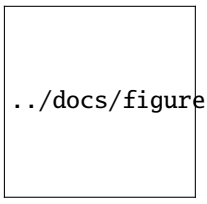
Warning

We do have an issue with the extrapolation, here, as at the current stage there is no guarantee that the result of the spline evaluation would be <= 1. We could set the ext class member of the spline to 3 before reurning it, but event that would not be right in general. This is probably not a huge issue, but it should be addressed properly.

xsection_ecube()

Return a spline with the values of absorption cross section multiplied by the cube of the energy, as a function of the energy.

This is essentially the same plot in Figure 1 of the original paper, as illustrated in the following figure



FITS image support.

class ixpeobssim.srcmodel.img.**xFITSImage**(*file_path*)

Class describing a FITS image equipped to extract random coordinates.

rvs_coordinates(*size=1*, *randomize=True*)

Generate random coordinates based on the image map.

Parameters

- **size** (*int*) – The number of sky coordinates to be generated.
- **randomize** (*bool*) – If true, the positions are randomized uniformely within each pixel.

Interface to the magnetar tabular models by Roberto Turolla and Roberto Taverna.

ixpeobssim.srcmodel.magnetar.**package_model_table**(*root_folder_path*, *model_name*)

Build a model table package in OGIP-compliant FITS format given an archive of text files containing the Stokes spectra as a function of energy and phase.

ixpeobssim.srcmodel.magnetar.**parse_data_file**(*file_path*, **target_params*, *num_phase_bins=9*, *num_energy_bins=49*)

Parse the content of an input data file.

The basic data structure is as follows:

- a first line with the 4 parameter values;
- a line with (phase_bin, phase_min, phase_max)
- a series of lines with (logemin, logemax, I, Q/I, U/I)

```
    89.9000      89.9000      1.40000     0.200000
  1    0.0010000000      0.69913174
-0.300000    -0.273469      852.250    0.0217493   -0.0900019
-0.273469    -0.246939      725.750    0.0282958   -0.0814848
...
 0.946939     0.973469      78.0000    0.0389240   -0.0909807
 0.973469      1.00000      198.250    0.0437026   -0.0840274
  2      0.69913174       1.3972635
...
```

ixpeobssim.srcmodel.magnetar.**parse_legacy_data**(*file_path*, *pad_bounds=True*, *emin=1.0*, *emax=12.0*)

Parse a legacy data file, predating the glorious, full model table.

> **Warning**
>
> This is obsolete and, in principle, there should be no reason to use it except for testing.

Here is an example of the underlying data format:

```
89.9000      60.0000     0.500000     0.340000
0   0.00100000     0.699132
-0.300000    -0.273469      4456.25     0.776281      73.0709
-0.273469    -0.246939      834.500     0.786651      73.3261
```

The basic rules are:

- the first line contains the model parameters: chi, xi, delta_phi, and beta;
- then comes a variable number of blocks where the first (3-elements) row indicates the index and range of the bin phase and the following (5-element) rows encapsulate the spectral and polarimetric properties.

More specifically, each 5-element row includes:

- log10(energy_lo);

- log10(energy_hi);
- flux (arbitrary units—the number of counts in the underlying MC);
- polarization degree;
- polarization angle in decimal degrees.

class ixpeobssim.srcmodel.magnetar.**xMagnetarModelsT2020**(*value*)

Enum class encapsulating the physical models in Taverna et al. 2020 https://ui.adsabs.harvard.edu/link_gateway/2020MNRAS.492.5057T/EPRINT_PDF

- ATMOSPHERE = 1
- BLACKBODY = 2
- SOLID_SURFACE_FREE_IONS = 3
- SOLID_SURFACE_FIXED_IONS = 4

classmethod has_value(*value*)

Convenience method that can be used downstrem to check the validity of a model passed as an integer.

https://stackoverflow.com/questions/43634618

class ixpeobssim.srcmodel.magnetar.**xMagnetarTableModelComponentT2020**(*file_path*, *normalize=False*)

Interface to the magnetar tabular models.

The models are coming as three separate FITS files for the I, Q, and U Stokes parameters, the basic structure of each one being, e.g.:

```
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      15   ()
  1  PARAMETERS    1 BinTableHDU     47   6R x 10C   [12A, J, E, E, E, E, E, E, J,␣
↪PE(13)]
  2  ENERGIES      1 BinTableHDU     21   49R x 2C   [D, D]
  3  SPECTRA       1 BinTableHDU     21   314496R x 2C   [6E, 49E]
```

(The actual dimensions of the cards might change, but the very fundamental structure of the files is fixed by the OGIP standard, I believe.)

The PARAMETERS binary table encapsulate the following model parameters:

- *Model*: the underlying Physical model, see the MagnetarModel enum
- *chi*: angle between the line of sight and the rotation angle of the star.
- *xi*: angle between the magnetic axis ans the rotation angle of the star.
- *delta_phi*: twist angle.
- *beta*: electron velocity in units of c.
- *phase*: the phase bins.

For illustration purposes, in the initial implementation we have 4 physical models x 13 chi x 8 xi x 12 delta_phi x 7 beta x 9 phase bins, that is, 314496 combinations. Spelling things out explicitely:

```
Model     -> [1, 2, 3, 4]
chi       -> [0.1, 15, 30, 45, 60, 75, 89.9, 105, 120, 135, 150, 165, 179.9]
xi        -> [0.1, 15, 30, 45, 60, 75, 85, 89.9]
delta_phi -> [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4]
```

(continues on next page)

(continued from previous page)

```
beta      -> [0.2, 0.3, 0.34, 0.4, 0.5, 0.6, 0.7]
phase     -> [0.350066, 1.0482, 1.74633, 2.44446, 3.14259, 3.84072,
              4.53886, 5.23699, 5.93512]
```

The ENERGY binary extensions encapsulates the energy binning, and in the initial implementations has 49 bins between ~0.5 and 10 keV.

Finally, the glorious SPECTRA extensions contains the spectra for all the possible configurations. It comes with two columns, the first of which contains the 6 model parameters and the second the 49 spectral values.

Note the SPECTRA extension is filled by looping over the model parameters in reverse order, i.e., from *phase* to *Model*.

interpolate(**params*)

Interpolate the underlying SPECTRA table to a given set of parameters.

interpolate_indices(**params*, *threshold=1e-06*)

Interpolate the underlying SPECTRA extension at a given set of input parameters.

This performs a simple linear interpolation, where the first parameter (the actual model) is treated in a special fashion in that is not interpolated at all—in fact the function requires that the value passed as an argument to the function is one of the four valis values.

For all the other parameters, the corresponding grids of tabulated values are bisected at the target value and the two nearest elements are weighted proportinally to the relative distance of the target.

nearest(**params*)

Return the slice of the underlying multi-dimensional array corresponding to the model that come closest to the input parameters.

nearest_indices(**params*)

Return the indices of the slice of the underlying multi-dimensional array corresponding to the model that comes closest to the input parameters.

num_phase_bins()

Return the number of phase bins.

parameters(**indices*)

Retrieve the parameter values for a given set of indices.

spectrum_data(**indices*)

Retrieve the spectrum data for a given series of indices.

Note that the function requires to pass *all* the indices, *except* for the last one, i.e., the one referring to the bin phase, and the return value contains the data for all the bin phases.

weighted_average(*indices*, *weights*)

Calculate the weighted average for a given set of entries in the underlying SPECTRA table.

class ixpeobssim.srcmodel.magnetar.**xMagnetarTableModelT2020**

Full description of a magnetar spectro-polarimetric model.

classmethod auxfiles_missing()

Convenience function to check for missing auxiliary files.

Note this is a classmethod, so it can be invoked without an instance of the class, which can be handy, e.g., for skipping unit tests.

energy_spectrum(*model*, *chi*, *xi*, *delta_phi*, *beta*, *integral_flux*, *emin=2.0*, *emax=10.0*)

Return a bivariate spline representing the energy spectrum.

This is essentially retrieving the photon spectrum (as a bivariate spline) and multiplying the z array with the proper tiling of the x array (i.e., the energy).

interpolate(*model*, *chi*, *xi*, *delta_phi*, *beta*, *integral_flux*, *emin=2.0*, *emax=10.0*)

Return a set of wraps of the interpolated splines ready to be used in ixpeobssim.

interpolated_splines(*model*, *chi*, *xi*, *delta_phi*, *beta*, *integral_flux*, *emin=2.0*, *emax=10.0*, *pad_phase=True*, *pad_energy=True*)

Return a set of model splines interpolating the underlying model table to the target parammeters.

nearest(*model*, *chi*, *xi*, *delta_phi*, *beta*, *integral_flux*, *emin=2.0*, *emax=10.0*)

Return a set of wraps of the nearest splines ready to be used in ixpeobssim.

nearest_splines(*model*, *chi*, *xi*, *delta_phi*, *beta*, *integral_flux*, *emin=2.0*, *emax=10.0*, *pad_phase=True*, *pad_energy=True*)

Return a set of model splines for the nearest set of parameters in the underlying model table.

phase_resolved_integral_flux(*phase_binning*, *model*, *chi*, *xi*, *delta_phi*, *beta*, *integral_flux*, *emin=2.0*, *emax=10.0*)

Return the integral flux of the model (in erg/cm^2) in arbitrary phase bins.

class ixpeobssim.srcmodel.magnetar.**xMagnetarTableModelT2020QedOff**

Full description of a magnetar spectro-polarimetric model no QED effects.

class ixpeobssim.srcmodel.magnetar.**xParameterSpaceT2020**

Small namespace to encapsulate the standard grid of model parameters.

Facilities related to modeling of the polarization properties.

ixpeobssim.srcmodel.polarization.**broadband_pol_ang**(*spec*, *pol_ang*, *emin=2.0*, *emax=8.0*, *num_points=500*, *degrees=True*)

Calculate the broadband polarization angle for a given spectrum and polarization angle vs. energy.

ixpeobssim.srcmodel.polarization.**broadband_pol_deg**(*spec*, *pol_deg*, *emin=2.0*, *emax=8.0*, *num_points=500*)

Calculate the broadband polarization degree for a given spectrum and polarization degree vs. energy.

ixpeobssim.srcmodel.polarization.**constant**(*C*)

Simple wrapper returning a constant, independently of the input arguments.

Parameters

- **C** (*int or float*) – The target constant.
- **x.x.x** (*New in version*)
- **addition** (*to be able to call the function with no argument at all. In*)
- **when**
- **(i.e.** (*the first argument*)
- **returning** (*the energy) is a numpy array we are now*)
- **target** (*an array of the same length (whose elements are all equal to the*)
- **constant).**

ixpeobssim.srcmodel.polarization.**constant_spline**(*xmin*, *xmax*, *C*, *xlabel='Energy [keV]'*, *ylabel=None*)

Convenience function to generate a constant spline on a generic x-axis interval. (Note that underlying array only have two points, as in this case the interpolation is trivial.)

Parameters

- **x** (*numpy array*) – The grid for the x-axis of the output spline.
- **C** (*int or float*) – The target constant
- **a** (*We typically use this to express a polarization degree or angle as*)
- **energy** (*function of the*)
- **of** (*which is the reason for the default argument*)
- **x-axis.** (*the name and units on the*)

ixpeobssim.srcmodel.polarization.**fourier_series_factory**(**params*)

Simple factory class for a generic Fouries series.

Given an arbitrary number of coefficients of a generic Fuorier expansion, this is returning a function that can be evaluated on an arbitrary grid of points.

Note that the Fourier expansion is done in the pulse-phase space, i.e., the output function is expecting an argument in the [0., 1.[interval (peridically repeating itself.)

Parameters

params (*2-element tuple(s)*) – The input set of Fourier coefficients describing the amplitude and phase lag of, in the form of 2-element (amplitude, phase) tuples.

Example

```
>>> import numpy
>>> from ixpeobssim.srcmodel.polarization import fourier_series_factory
>>> factory = fourier_series_factory((0.2, 0.5), (0.3, 0.5))
>>> x = numpy.linspace(0., 1., 100)
>>> y = factory(x)
```

ixpeobssim.srcmodel.polarization.**harmonic_addition**(**params*)

Basic implementation of the harmonic addition theorem, see http://mathworld.wolfram.com/HarmonicAdditionTheorem.html

Parameters

- **params** (*3-element tuple(s)*) – The input set of parameters describing the modulation curves that need to be added, in the form of an arbitrary number of 3-element tuples (F, m, delta) containing the normalization (i.e., the source flux associated to the component) the modulation (i.e., the degree of polarization of the source) and the phase.
- **that** (*You should note*)
- **of** (*since the modulation curve is defined in terms*)
- **phi)** (*cos(2 **)
- **cos(phi)** (*rather than*)
- **are** (*all the angles in the calculation*)
- **two** (*multiplied by*)

- **in** (*and the final arctan is multiplied by 1/2. Keep this*)
- **the** (*mind when you compare the implementation below with the formula at*)
- **link.** (*mathworld*)

ixpeobssim.srcmodel.polarization.**harmonic_component_addition**(**components*)

Harmonic component addition.

ixpeobssim.srcmodel.polarization.**pulse_pol_from_harmonics_spline**(*mean_flux*, *mean_pol*, **params*)

Build phase-dependent luminosity and polarization degree, using Fourier harmonics composition. This is important for modeling millisecond pulsar emission geometry, see: Eq.(45) in K. Viironen and J. Poutanen, A&A 426, 985-997 (2004)

mean_flux must be a scalar in $cm^{-2}s^{-1}keV{-1}$ units; mean_pol must be between zero and one.

For parameters look at Fourier_harmonics function.

class ixpeobssim.srcmodel.polarization.**xPolarizationFieldBase**(*ra0*, *dec0*, *radial_profile=1.0*)

Virtual base class describing a generic, azimuthally simmetric polarization field.

Parameters

- **ra0** (*float*) – The right ascension of the center of the field in decimal degrees.
- **dec0** (*float*) – The declination of the center of the field in decimal degrees.
- **radial_profile** (*float or callable with a signature radial_profile(r, E, t)*) – The radial profile for the polarization degree, expressed as a function of the distance from the center in decimal degrees, energy and time.

polarization_angle(*ra*, *dec*)

Do nothing method to be reimplemented in derived classes.

polarization_angle_model()

Convenience function adapting the signature of the polarization_angle() hook to the needs of xpobssim.

polarization_degree(*E*, *t*, *ra*, *dec*)

Return the polarization degree as a function of the standard dynamical variables.

polarization_degree_model()

Convenience function adapting the signature of the polarization_angle() hook to the needs of xpobssim.

class ixpeobssim.srcmodel.polarization.**xRadialPolarizationField**(*ra0*, *dec0*, *radial_profile=1.0*)

polarization_angle(*ra*, *dec*)

Return the azimuthal angle in the tangential plane, measured in radians from the celestial North, at the position (ra, dec) for a purely radial field centered at (ra0, dec0).

class ixpeobssim.srcmodel.polarization.**xStokesSkyCube**

Class describing a Stoke sky-cube, i.e., a coherent collection of Stokes sky maps in different energy layers.

add_layer_pda(*pd_file_path*, *pa_file_path*, *emin*, *emax=None*)

Add a layer for polarization degree and angle.

add_layer_qu(*q_file_path*, *u_file_path*, *emin*, *emax=None*)

Add a layer for Q and U maps.

layer(*i*)

Return a generic cube layer by index.

plot(***kwargs*)

Plot all the layers.

plot_input_data()

Plot the input data for all the layers.

polarization_angle(*ra*, *dec*, *energy*)

Return the polarization angle value given a ra, dec and energy.

polarization_angle_model()

Convenience method to adapt the signature of the __call__ class method to the one needed at simulation time.

polarization_degree(*ra*, *dec*, *energy*)

Return the polarization degree value given a ra, dec and energy.

polarization_degree_model()

Convenience method to adapt the signature of the __call__ class method to the one needed at simulation time.

xy_grid_bounds()

Return the spatial bounds (in pixel space) for the underlying interpolating grid.

Note that, since we make sure at fill time that all the layers have the same WCS, here we take the bounds from the first layer.

class ixpeobssim.srcmodel.polarization.**xStokesSkyCubeLayer**(*qdata*, *udata*, *wcs_*, *input_file_paths*, *input_labels*)

Small convenience class representing a layer of an xStokesSkyCube object.

This is essentially a lightweight wrapper of the xStokesSkyMap class, on top of which we add a few extra members (e.g., the energy bounds) that are necessary when a map is used in the context of a cube.

energy_range_label()

Return a text label encoding the proper energy range for the layer.

label()

Return a generic text label for the layer.

set_energy_range(*emin*, *emax=None*)

Set the minimum and maximum energy for the layer.

If emax is None it is assumed that the layer has been calculated at a specified energy (as opposed to an energy range), and is treated as such—the minimum and maximum energies are the same.

class ixpeobssim.srcmodel.polarization.**xStokesSkyMap**(*qdata*, *udata*, *wcs_*, *input_file_paths*, *input_labels*)

Class representing a Stokes map in sky coordinates.

This is the basic interface to polarization models for extended sources. The basic idea is that we store two bivariate splines for the Q and U Stokes parameters, along with the corresponding WCS.

The class constructor takes raw numpy arrays of Q and U values sampled on a regular rectangular grid, but in fact the class is designed in such a way that it should rarely be necessary to instantiate a class object from the constructor—the model maps will be typically loaded from images stored in FITS files, using the convenience class methods. The class supports in a transparent way input in three different forms:

- Q and U;
- x and y polarization components;
- polarization degree and angle.

The class is callable, and returns the Q and U value at given sky coordinates (ra and dec).

Example

```
>>> map_ = xStokesSkyMap.load_from_qu(qpath, upath)
>>> map_ = xStokesSkyMap.load_from_pda(pdpath, papath)
```

static compare_wcs(*wcs1*, *wcs2*)

Compare two WCS objects to make sure they are identical.

I am sure we can do this in a better way, but the way the information is stored in the WCS seems really convoluted. At the zero order we are trying to make sure that two WCS objects have the same center, axes and extension.

classmethod load_from_pda(*pd_file_path*, *pa_file_path*)

Load a sky map of Stokes parameters from polarization degree and angle.

classmethod load_from_qu(*q_file_path*, *u_file_path*)

Load a sky map of Stokes parameters from Q and U data.

static map_grid(*shape*)

Return grid for map interpolation.

The spline is performed in logical space, interpolating the values at the center of the bins, and the transformation from sky coordinates is handled at __call__() time.

> **Warning**
>
> This is tricky, as the pixels in the FITS maps start from 1, while the underlying numpy array is zero-indexed. Ideally we want to take the interpolation at the center of the bin, and therefore the necessary offset is -1 + 0.5 = -0.5. (All of this TBC.)
>
> See ixpeobssim.srcmodel.img.xFitsImage.__call__() for more information.

static pixel_shape(*wcs_*)

Small convenience function to retrieve the pixel shape of a WCS object.

We need this because the private attributes "_naxis1" and "_naxis2" have been deprecated since astropy version 3.1 in favor of the "pixel_shape" property.

plot_arrows(*nside=25*, *ra0=None*, *dec0=None*, *radius=None*, *threshold=0.0*, ***kwargs*)

Overlay the polarization arrows on top of a map.

plot_input_data(*suffix=None*)

Plot the underlying input arrays used to build the sky map.

plot_polarization_angle(*vmin=None*, *vmax=None*, *cmap='gist_heat'*, *degrees=True*)

Plot the polarization angle as a function of the sky direction.

plot_polarization_degree(*vmin=None*, *vmax=None*, *cmap='gist_heat'*)

Plot the polarization degree as a function of the sky direction.

plot_q(*vmin=None*, *vmax=None*, *cmap='gist_heat'*)

Plot Q as a function of the sky direction.

plot_u(*vmin=None*, *vmax=None*, *cmap='gist_heat'*)

Plot U as a function of the sky direction.

polarization_angle(*ra*, *dec*)

Return the polarization angle value given a ra and dec.

polarization_angle_model()

Convenience method to adapt the signature of the __call__ class method to the one needed at simulation time.

polarization_degree(*ra*, *dec*)

Return the polarization degree value given a ra and dec.

polarization_degree_model()

Convenience method to adapt the signature of the __call__ class method to the one needed at simulation time.

polarization_vector(*ra*, *dec*)

Return the x and y components of the polarization vector at given sky coordinates. .

static read_map(*file_path*)

Read an input FITS file containg map data.

This is a generic interface to parse arbitrary bidimensional data (depending on the context, these can be in the U, Q or x, y projections or polarization degree, angle space).

The FITS file is supposed to contain an image as the first extension, along with the WCS information.

static wcs_center(*wcs_*)

Return the coordinates of the center of a given WSC object.

static wcs_radius(*wcs_*, *default=0.1*)

Return the approximate radius of a given WCS object.

The FITS specification dictates that when CD is present, CDELT should be set to 1, and is effectively ignored, see https://docs.astropy.org/en/stable/api/astropy.wcs.Wcsprm.html

class ixpeobssim.srcmodel.polarization.**xTangentialPolarizationField**(*ra0*, *dec0*, *radial_profile=1.0*)

polarization_angle(*ra*, *dec*)

Return the azimuthal angle in the tangential plane, measured in radians from the celestial North, at the position (ra, dec) for a purely tangential field centered at (ra0, dec0).

Model component and region of interest definition.

class ixpeobssim.srcmodel.roi.**xBinarySource**(*name*, *ra*, *dec*, *photon_spectrum*, *polarization_degree*, *polarization_angle*, *ephemeris*, *column_density=0.0*, *redshift=0.0*, *identifier=None*)

Class representing a binary source (e.g., a pulsar in a binary system).

See *ixpeobssim.srcmodel.roi.xModelComponentBase* for the signature of the base class.

Parameters

- **ra** (*float*) – The right ascension of the source (in decimal degrees).
- **dec** (*float*) – The declination of the source (in decimal degrees).
- **ephemeris** (ixpeobssim.srcmodel.roi.xEphemeris object) – The source ephemeris.

rvs_photon_list(*parent_roi*, *irf_set*, ***kwargs*)

Placeholder.

class ixpeobssim.srcmodel.roi.**xCelestialModelComponentBase**(*name*, *photon_spectrum*, *polarization_degree*, *polarization_angle*, *column_density=0.0*, *redshift=0.0*, *identifier=None*)

Base class for the source object.

Note that the source identifier defaults to none and is typically assign after the fact when the source itself is added to a source model.

Parameters

- **name** (*string*) – The name of the source.
- **photon_spectrum** (*function*) – The function object representing the photon spectrum.
- **polarization_degree** (*function*) – The function object representing the polarization degree.
- **polarization_angle** (*function*) – The function object representing the polarization angle.
- **column_density** (*float*) – The value of the column density (in cm^-2) used to calculate the Galactic absorption. Defaults to 0.
- **redshift** (*float*) – The source redshift. Defaults to 0.
- **identifier** (*int*) – A unique identifier of the source within a ROI model. Defaults to None and is, generally, automatically assigned when building the ROI (i.e., you don't have to worry about it, but it's handy to have in the output event list).

build_intensity_map(*wcs_*, *num_points=1000000*)

Base function to calculate the underlying intensity map over a regular sky grid.

In the base class this is implemented via brute force, i.e., we throw a bunch or random sky directions via a rvs_sky_coordinates() call and then we bin them on the approriate wcs object. In reality, for most of the source classes, this can be done much more effectively by other means, with two major advantages:

- the calculation can be made faster;
- the numerical noise due to the Monte Carlo integration can be entirely avoided.

Note that, whenever possible, this function is supposed to return the actual model values on a grid of points, rather than a brute-force Monte Carlo approximation. In practice, we should strive to reimplement this method in all sub-classes, with the only exception of extended sources based on FITS images.

calculate_average_polarization(*egrid*, *tgrid*, *degrees=False*)

Calculate the average polarization degree and angle in a given energy and time (or phase) interval.

Warning

This is deprecated in favor of the model_pol_average facilities.

calculate_integral_flux(*emin=2.0*, *emax=8.0*, *t=0.0*, *num_points=250*, *erg=True*)

Return the integral source flux at a generic time.

This is achieved by taking a "slice" of the source spectrum at that time and integrating between a minimum and maximum energy.

Parameters

- **emin** (*float*) – The minimum integration energy (default 2 keV).
- **emax** (*float*) – The maximum integration energy (default 8 keV).
- **t** (*float*) – The time (default is 0).

classmethod convolve_event_list(*event_list*, *parent_roi*, *irf_set*, ***kwargs*)

Convolve a Monte Carlo event list with a set of instrument response functions.

Note that we factor this functionality as a separate class method so that it can be used in different context, e.g., in the Chandra to IXPE converter.

create_count_spectrum(*aeff*, *time_grid*, ***kwargs*)

Create the count spectrum object at the base of the source simulation.

This is the bit where we convolve the source spectrum with the effective area.

Parameters

- **aeff** (*xEffectiveArea object*) – The effective area as a function of the energy—this can be a generic callable accepting the energy as the first (and only) argument.
- **time_grid** (*array_like*) – The time (or phase) grid over which the count spectrum is created.

rvs_event_list(*parent_roi*, *irf_set*, ***kwargs*)

Extract a random event list for the model component.

rvs_photon_list(*parent_roi*, *irf_set*, ***kwargs*)

Extract a random photon list.

rvs_sky_coordinates(*size=1*)

Generate random coordinates for the model component.

This is a do-nothing function and should be re-implemented by each derived class.

Parameters

size (*float*) – The number of sky coordinate pairs to be generated.

sampling_time_grid(*start_met*, *duration*)

Return the time grid used to sample the lightcurve when generating events.

set_count_spectrum_params(*ny=200*, *kx=3*, *ky=3*)

Adjust the parameters used at runtime for the creation of the count spectrum.

This is the main hook to control how the count spectrum for a given source component is created, see https://bitbucket.org/ixpesw/ixpeobssim/issues/55

setup(*photon_spectrum*, *polarization_degree*, *polarization_angle*)

Setup the model component in terms of photon spectrum and polarization degree and angle.

class ixpeobssim.srcmodel.roi.**xChandraObservation**(*name*, *polarization_degree*, *polarization_angle*, *region=None*, *exclude=False*, *identifier=None*)

Class representing a source taken from a Chandra observation.

Parameters

- **name** (*string*) – The name of the source.
- **polarization_degree** (*function*) – The function object representing the polarization degree.
- **polarization_angle** (*function*) – The function object representing the polarization angle.
- **region** – The optional region to select the photon list from.
- **exclude** (*bool*) – The optional flag to exclude the selected region from the simulation.
- **identifier** (*int*) – A unique identifier of the source within a ROI model. Defaults to None and is, generally, automatically assigned when building the ROI (i.e., you don't have to worry about it, but it's handy to have in the output event list).

rvs_event_list(*parent_roi*, *irf_set*, ***kwargs*)

Extract a random event list for the model component.

rvs_photon_list(*parent_roi*, *irf_set*, ***kwargs*)

Extract a random photon list.

class ixpeobssim.srcmodel.roi.**xChandraROIModel**(*evt_file_path*, *acis*)

Class describing a Chandra ROI (region of interest) model.

This is essentially an (ordered) collection of component objects (i.e., instances of classes inheriting from xCelestialModelComponentBase) than can be accessed by source name.

Parameters

evt_file_path (*string*) – The path to the FITS file containing the Chandra event list.

filter_events(*region*, *exclude=False*)

Return the filtered event arrays with coordinates inside the given region.

Parameters

- **region** (*region instance or None*) – The region to select the photon list from (None to take the whole remaining area).
- **exclude** (*bool*) – Flag to exclude the selected region from the simulation.

class ixpeobssim.srcmodel.roi.**xExtendedSource**(*name*, *img_file_path*, *photon_spectrum*, *polarization_degree*, *polarization_angle*, *column_density=0.0*, *redshift=0.0*, *identifier=None*)

Class representing an extended source.

See *ixpeobssim.srcmodel.roi.xCelestialModelComponentBase* for the signature of the base class.

Parameters

img_file_path (*string*) – The path to the FITS file containing the image of the source.

rvs_sky_coordinates(*size=1*)

Generate random coordinates for the model component.

Parameters

size (*float*) – The number of sky coordinate pairs to be generated.

class ixpeobssim.srcmodel.roi.**xGaussianDisk**(*name*, *ra*, *dec*, *sigma*, *photon_spectrum*, *polarization_degree*, *polarization_angle*, *column_density=0.0*, *redshift=0.0*, *identifier=None*)

Class representing a (azimuthally simmetric) gaussian disk.

See *ixpeobssim.srcmodel.roi.xCelestialModelComponentBase* for the signature of the base class.

Parameters

- **ra** (*float*) – The right ascension of the disk center (in decimal degrees).
- **dec** (*float*) – The declination of the disk center (in decimal degrees).
- **sigma** (*float*) – The root mean square of the disk (in degrees).

build_intensity_map(*wcs_*)

Overloaded method.

rvs_sky_coordinates(*size=1*)

Generate random coordinates for the model component.

This is returning an array of the proper length with identical values.

Parameters

size (*float*) – The number of sky coordinate pairs to be generated.

class ixpeobssim.srcmodel.roi.**xModelComponentBase**(*name*, *identifier=None*)

Base class for all the model components (i.e., the sources) populating a generic region of interest for a simulation.

Historically this base class was tailored to celestial sources, but since different use cases emerged along the way that do not really fit in the original picture (e.g., instrumental background and calibration flat fields, where everything happens within the detector, and there is no concept of mirror effective area nor vignetting, just to name a few) the decision was taken to split the very fundamental features of the object into this minimal base class and have a separate base class for the celestial objects.

In addition to some fundamental members and method, the class includes a handful of static methods that encapsulate useful operations of general interest, such as generating event times and energy according to simple analytical distributions.

Parameters

- **name** (*string*) – The name of the source.
- **identifier** (*int, optional*) – A unique identifier of the source within a ROI model. Defaults to None and is, generally, automatically assigned when building the ROI (i.e., you don't have to worry about it, but it's handy to have in the output event list).

classmethod convolve_sky_direction(*mc_ra*, *mc_dec*, *parent_roi*, *psf*)

Smear the sky position with the PSF and calculate the corresponding digitized x and y values.

static parse_time_kwargs(***kwargs*)

Parse the keyword arguments related to the duration of the observation.

static poisson(*average*)

Return a number extracted from a Poisson distribution with a given average.

rvs_event_list(*parent_roi*, *irf_set*, ***kwargs*)

Return an event list for the model component.

Parameters

- **parent_roi** (*xROIModel instance*) – The parent region of interest (i.e., the xROIModel instance the model component belongs to).

- **irf_set** (*xIRFSet instance*) – The set of detector response functions to be used for the convolution of the physical quantities.
- ****kargs** (*keyword arguments*) – At runtime this is precisely the complete set of command-line options that xpobssim is run with (i.e., you do get all the goodies describing the configuration of the simulation, such as the start mission elapsed time and the simulation duration).

> **Note**
>
> This is a do-nothing function that should be re-implemented in every derived class.
>
> This is also the main xModelComponentBase method, since all the Physics of the model itself is encapsulated here. In addition, the method is aware of the parent region of interest object and the instrument response functions, which are essential ingredients to create the output event list.
>
> As a rule of thumb:
>
> - all the operations that makes sense to perform on a component by component basis (e.g., the convolution with the instrument response functionsm, including the vignetting) happen in the body of this function;
> - all the operations (such as the application of the deadtime) that need to be performed on the final event list after all the model model components have been merged togethere are deferred to the downstream code.

rvs_photon_list(*parent_roi*, *irf_set*, ***kwargs*)

Return a photon list for the model component.

static uniform_phi(*size*)

Return an array of a given size of azimuthal angles distributed between -pi and pi.

This can be used to replace the fully-fledged xAzimuthalResponse generator in cases where one knows since the beginning that a flat distribution is needed (it goes without saying that this is much less demanding from a computational standpoint).

static uniform_rectangle(*size*, *half_side_x*, *half_side_y*)

Return two arrays of a given size with random x and y coordinates uniformly distributed within a rectangle of a given half-sides centered in 0, 0.

static uniform_square(*size*, *half_side*)

Specialized function generating random coordinates uniformly distributed within a square.

static uniform_time(*size*, *start_met*, *duration*)

Return a sorted array of a given size of event times uniformly distributed between a minimum and a maximum time.

class ixpeobssim.srcmodel.roi.**xPeriodicPointSource**(*name*, *ra*, *dec*, *photon_spectrum*, *polarization_degree*, *polarization_angle*, *ephemeris*, *column_density=0.0*, *redshift=0.0*, *identifier=None*)

Class representing a periodic point source (e.g., a pulsar).

See *ixpeobssim.srcmodel.roi.xCelestialModelComponentBase* for the signature of the base class.

Parameters

- **ra** (*float*) – The right ascension of the source (in decimal degrees).

- **dec** (*float*) – The declination of the source (in decimal degrees).
- **ephemeris** (ixpeobssim.srcmodel.roi.xEphemeris object) – The source ephemeris.

ephemeris_info()

Return all the ephemeris information in a form that is suitable for display and pretty-printing.

rvs_photon_list(*parent_roi*, *irf_set*, ***kwargs*)

Extract a random photon list.

sampling_time_grid()

Return the phase grid used to sample the lightcurve when generating events.

class ixpeobssim.srcmodel.roi.**xPointSource**(*name*, *ra*, *dec*, *photon_spectrum*, *polarization_degree*, *polarization_angle*, *column_density=0.0*, *redshift=0.0*, *identifier=None*)

Class representing a steady point source.

See *ixpeobssim.srcmodel.roi.xCelestialModelComponentBase* for the signature of the base class.

Parameters

- **ra** (*float*) – The right ascension of the source (in decimal degrees).
- **dec** (*float*) – The declination of the source (in decimal degrees).

build_intensity_map(*wcs_*)

Overloaded method.

rvs_sky_coordinates(*size=1*)

Generate random coordinates for the model component.

This is returning an array of the proper length with identical values.

Parameters
size (*float*) – The number of sky coordinate pairs to be generated.

class ixpeobssim.srcmodel.roi.**xROIModel**(*ra_center*, *dec_center*, **sources*)

Class describing a full ROI (region of interest) model.

This is essentially an (ordered) collection of component objects (i.e., instances of classes inheriting from xCelestialModelComponentBase) than can be accessed by source name.

Parameters

- **ra_center** (*float*) – The right ascension of the center of the ROI (in decimal degrees).
- **dec_center** (*float*) – The declination of the center of the ROI (in decimal degrees).

add_source(*source*)

Add a source to the ROI.

add_sources(**sources*)

Add an arbitrary number of sources to the ROI.

first_source()

Return the first source (by insertion) in the ROI.

rvs_event_list(*irf_set*, ***kwargs*)

Extract an event list for the full ROI.

Parameters

irf_set (*ixpeobssim.irf.xIRFSet` object.*) – The set of instrument response functions to be used.

> **Warning**
>
> The sampling_time should not be the same for all sources, and each source should be able to decide its own in a sensible way. (See issue #44.)

rvs_photon_list(*irf_set*, ***kwargs*)

Extract a photon list for the full ROI.

This was added to support xpphotonlist.py.

source_by_id(*uid*)

Retrieve a source by insertion index.

source_by_name(*name=None*)

Retrieve a source by name.

If None (or no argument is passed) this returns the first source in the ROI.

class ixpeobssim.srcmodel.roi.**xUniformAnnulus**(*name*, *ra*, *dec*, *rmin*, *rmax*, *photon_spectrum*, *polarization_degree*, *polarization_angle*, *column_density=0.0*, *redshift=0.0*, *identifier=None*)

Class representing a uniform annulus.

See *ixpeobssim.srcmodel.roi.xCelestialModelComponentBase* for the signature of the base class.

Parameters

- **ra** (*float*) – The right ascension of the annulus center (in decimal degrees).
- **dec** (*float*) – The declination of the annulus center (in decimal degrees).
- **rmin** (*float*) – The minimium radius of the annulus (in degrees).
- **rmax** (*float*) – The maximium radius of the annulus (in degrees).

build_intensity_map(*wcs_*)

Overloaded method.

rvs_sky_coordinates(*size=1*)

Generate random coordinates for the model component.

This is returning an array of the proper length with identical values.

The algorithm is taken from http://mathworld.wolfram.com/DiskPointPicking.html

Parameters

size (*float*) – The number of sky coordinate pairs to be generated.

class ixpeobssim.srcmodel.roi.**xUniformDisk**(*name*, *ra*, *dec*, *radius*, *photon_spectrum*, *polarization_degree*, *polarization_angle*, *column_density=0.0*, *redshift=0.0*, *identifier=None*)

Class representing a uniform disk.

See *ixpeobssim.srcmodel.roi.xCelestialModelComponentBase* for the signature of the base class.

Parameters

- **ra** (*float*) – The right ascension of the disk center (in decimal degrees).
- **dec** (*float*) – The declination of the disk center (in decimal degrees).
- **radius** (*float*) – The radius of the disk (in degrees).

build_intensity_map(*wcs_*)

Overloaded method.

rvs_sky_coordinates(*size=1*)

Generate random coordinates for the model component.

This is returning an array of the proper length with identical values.

The algorithm is taken from http://mathworld.wolfram.com/DiskPointPicking.html

Parameters

size (*float*) – The number of sky coordinate pairs to be generated.

Spectral facilities.

ixpeobssim.srcmodel.spectrum.**cutoff_power_law**(**args*)

Wrapper definition.

> **Warning**
>
> We might use some functools magic, here, to make sure the decorated model factories preserve the correct function signature when queried for help.

ixpeobssim.srcmodel.spectrum.**cutoffpl**(**args*)

Wrapper definition.

> **Warning**
>
> We might use some functools magic, here, to make sure the decorated model factories preserve the correct function signature when queried for help.

ixpeobssim.srcmodel.spectrum.**gauss**(*norm*, *mean*, *sigma*)

Gaussian line spectral component.

ixpeobssim.srcmodel.spectrum.**gaussian_line**(*norm*, *mean*, *sigma*)

Gaussian line spectral component.

ixpeobssim.srcmodel.spectrum.**highecut**(**args*)

Wrapper definition.

> **Warning**
>
> We might use some functools magic, here, to make sure the decorated model factories preserve the correct function signature when queried for help.

ixpeobssim.srcmodel.spectrum.**highecut_power_law**(**args*)

Wrapper definition.

> **Warning**
>
> We might use some functools magic, here, to make sure the decorated model factories preserve the correct function signature when queried for help.

ixpeobssim.srcmodel.spectrum.**int_eflux2pl_norm**(*integral*, *emin*, *emax*, *index*, *erg=True*)

Convert an integral energy flux into the corresponding power-law normalization.

ixpeobssim.srcmodel.spectrum.**integral_energy_flux**(*spectrum*, *emin*, *emax*, *column_density=None*, *erg=True*)

Return the integral energy flux within a given energy interval for a given spectral model.

Parameters

- **spectrum** (*callable*) – The spectral model (must be able to evaluate itself onto an array of energies)
- **emin** (*float*) – The minimum energy for the integral
- **emax** (*float*) – The minimum energy for the integral
- **column_density** (*float (optional)*) – The value of the column density (in cm^-2) used to calculate the Galactic absorption.
- **erg** (*bool*) – If True, convert the output from keV to erg

ixpeobssim.srcmodel.spectrum.**integral_flux**(*spectrum*, *emin*, *emax*, *column_density=None*)

Return the integral flux within a given energy interval for a given spectral model.

Parameters

- **spectrum** (*callable*) – The spectral model (must be able to evaluate itself onto an array of energies)
- **emin** (*float*) – The minimum energy for the integral
- **emax** (*float*) – The minimum energy for the integral
- **column_density** (*float (optional)*) – The value of the column density (in cm^-2) used to calculate the Galactic absorption.

ixpeobssim.srcmodel.spectrum.**load_spectral_spline**(*file_path*, *emin=1.0*, *emax=12.0*, *energy_col=0*, *flux_col=1*, *delimiter=None*, ***kwargs*)

Convenience function to load spectral data from a txt file.

Since we happen to load spectral data from text files a lot, this is an attempt to provide a facility that is generic enough to be effectively reused.

ixpeobssim.srcmodel.spectrum.**pl_integral**(*norm*, *index*, *emin*, *emax*)

Return the integral of a generic power law in a given energy range.

ixpeobssim.srcmodel.spectrum.**pl_integral_flux**(*norm*, *index*, *emin*, *emax*)

Return the integral flux for a generic power law in a given energy range.

ixpeobssim.srcmodel.spectrum.**pl_norm**(*integral*, *emin*, *emax*, *index*, *energy_power=0.0*)

Return the power-law normalization resulting in a given integral flux (or integral energy flux, or more in general integral of the flux multiplied by a generic power of the energy) between the minimum and maximum energies assuming a given spectral index.

More specifically, given a power law of the form

$$\mathcal{S}(E) = C\left(\frac{E}{E_0}\right)^{-\Gamma} \quad [\mathrm{keV}^{-1}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}],$$

(where $E_0 = 1$ keV) we define $\beta = (1 + p - \Gamma)$ and calculate

$$I_p = \int_{E_{\rm min}}^{E_{\rm max}} E^p \mathcal{S}(E) dE = \begin{cases} \frac{CE_0^\Gamma}{\beta}\left(E_{\rm max}^\beta - E_{\rm min}^\beta\right) & \beta \neq 0 \\ CE_0^\Gamma \ln\left(E_{\rm max}/E_{\rm min}\right) & \beta = 0 \end{cases} \quad [\mathrm{keV}^p\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}].$$

Hence

$$C = \begin{cases} \frac{I_p \beta}{E_0^\Gamma\left(E_{\rm max}^\beta - E_{\rm min}^\beta\right)} & \beta \neq 0 \\ \frac{I_p}{E_0^\Gamma \ln(E_{\rm max}/E_{\rm min})} & \beta = 0. \end{cases}$$

Parameters

- **integral** (*float or array*) – The value of the integral flux or energy-to-some-power flux
- **emin** (*float*) – The minimum energy for the integral flux
- **emax** (*float*) – The maximum energy for the integral flux
- **index** (*float*) – The power-law index
- **energy_power** (*float*) – The power of the energy in the integral
- **erg** (*bool*) – if True, convert erg to keV in the calculation.

ixpeobssim.srcmodel.spectrum.**power_law**(**args*)

Wrapper definition.

> **Warning**
>
> We might use some functools magic, here, to make sure the decorated model factories preserve the correct function signature when queried for help.

ixpeobssim.srcmodel.spectrum.**powerlaw**(**args*)

Wrapper definition.

> **Warning**
>
> We might use some functools magic, here, to make sure the decorated model factories preserve the correct function signature when queried for help.

ixpeobssim.srcmodel.spectrum.**wrap_spectral_model**(*model*)

Simple decorator to simplify the definition of spectral models.

This is essentially wrapping all the model parameters with the wrap_spectral_parameter() function and calling the input models with the wrapped parameters, so that we don't have to worry about whether each of them is time-dependent or time-independent.

ixpeobssim.srcmodel.spectrum.**wrap_spectral_parameter**(*parameter*)

Wrap a spectral parameter in order to handle time- (or phase-) dependence.

This is a small, handy trick to handle in a consistent fashion spectral models where the underlying parameters can be either constant or time-dependent: once they are wrapped, the parameters can be called with a given time as an argument even when they are time-independent.

Note that, in all cases, the t argument defaults to None, to allow time-independent spectral models to be called omitting it.

Parameters

parameter (*float or callable*) – The spectral parameter (e.g., the normalization or the index of a power law). This can be either a scalar or a callable with the signature parameter(t).

Returns

- *An anonymous function that, called with a given time as the only*
- *arguments, returns the parameter evaluated at that time (or the scalar*
- *if the parameter is time-independent).*

Example

```
>>> from ixpeobssim.srcmodel.spectrum import wrap_spectral_parameter
>>>
>>> param = wrap_spectral_parameter(1.)
>>> print(param(1000.))
>>> 1.0
>>> print(param())
>>> 1.0
```

class ixpeobssim.srcmodel.spectrum.**xCountSpectrum**(*spectrum*, *aeff*, *t*, *column_density=0.0*, *redshift=0.0*, *scale_factor=1.0*, *emin=None*, *emax=None*, *kx=3*, *ky=3*)

Class representing a count spectrum, i.e., the convolution of the source photon spectrum and the detector effective area

$$\mathcal{C}(E,t) = \mathcal{S}(E,t) \times A_{\rm eff}(E) \quad [\rm s^{-1}\ keV^{-1}].$$

Parameters

- **spectrum** (*callable*) – The source spectrum, i.e., a Python function that can be called with two numpy arrays E and t and returns the differential energy spectrum of the source at the appropriate enenrgy and time value(s).
- **aeff** (*ixpeobssim.irf.xEffectiveArea instance*) – The instrument effective area.
- **t** (*array_like*) – The array of time values where we're sampling the light curve of the source.
- **column_density** (*float, defaults to 0.*) – The H column density to calculate the insterstellar absorption.
- **redshift** (*float, defaults to 0.*) – The source redshift.
- **scale_factor** (*float, defaults to 1.*) – An optional scale factor for the output count spectrum (see the warning below).
- **emin** (*float (optional)*) – The minimum value for the energy grid.

- **emax** (*float (optional)*) – The maximum value for the energy grid.
- **kx** (*int (optional)*) – The degree of the bivariate spline on the x axis.
- **ky** (*int (optional)*) – The degree of the bivariate spline on the y axis.

Warning

The scale_factor parameter was a workaround for running the MDP script on a pulsar, where we sample in phase and then have to multiply by the observation time. The functionality should be implemented in a more roboust fashion.

build_mdp_table(*ebinning*, *modulation_factor*)

Calculate the MDP values in energy bins, given the modulation factor of the instrument as a function of the energy.

Parameters

- **ebinning** (*array*) – The energy binning
- **modulation_factor** (*ixpeobssim.irf.modf.xModulationFactor instance*) – The instrument modulation factor as a function of the energy.

num_expected_counts(*tmin=None*, *tmax=None*, *emin=None*, *emax=None*)

Return the number of expected counts within a given time interval and energy range

$$\int_{t_{\rm min}}^{t_{\rm max}} \int_{E_{\rm min}}^{E_{\rm max}} \mathcal{C}(E,t) dt dE$$

rvs_event_times(*size*)

Extract a series of random event times from the count spectrum.

Note the array is sorted in place.

class ixpeobssim.srcmodel.spectrum.**xSmearingMatrix**(*irf_name*, *du_id*, *spectral_index*, *gray_filter=False*, *column_density=0.0*)

Class representing a smearing matrix, i.e., the convolution of the response matrix with a given source spectrum and effective area.

class ixpeobssim.srcmodel.spectrum.**xSourceSpectrum**(*E*, *t*, *spectrum*, *column_density=0.0*, *redshift=0.0*, *kx=3*, *ky=3*)

Class representing a source spectrum,

$$\mathcal{S}(E,t) \quad [\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{keV}^{-1}].$$

At the top level this is essentially a bivariate spline with the energy running on the x-axis, the time running on the y-axis, and the differential source spectrum on the x-axis.

Parameters

- **E** (*array_like*) – The energy array (in keV) where the spectrum is evaluated.
- **t** (*array_like*) – The time array (in s) where the spectrum is evaluated.
- **spectrum** (*callable*) – The source spectrum, i.e., a Python function with the signature spectrum(E, t) returning the differential energy spectrum of the source at the appropriate enenrgy and time value(s).

- **column_density** (*float, defaults to 0.*) – The H column density to calculate the insterstellar absorption.
- **redshift** (*float, defaults to 0.*) – The source redshift.
- **kx** (*int (optional)*) – The degree of the bivariate spline on the x axis.
- **ky** (*int (optional)*) – The degree of the bivariate spline on the y axis.

build_energy_integral(*emin=None*, *emax=None*)

Build the energy-integrated spectrum between the given bounds.

The output is stored in the form of a xUnivariateGenerator, featuring all the spline facilities, along with the capability of extracting random numbers.

build_light_curve(*emin=None*, *emax=None*)

Build the light curve, i.e., the spectrum integrated over the entire energy range, as a function of time.

build_time_average(*tmin=None*, *tmax=None*)

Build the time-averaged spectrum.

build_time_integral(*tmin=None*, *tmax=None*)

Build the time-integrated spectrum between the give bounds.

The output is stored in the form of a xUnivariateGenerator, featuring all the spline facilities, along with the capability of extracting random numbers.

emax()

Return the maximum energy for which the spectrum is defined.

emin()

Return the minimum energy for which the spectrum is defined.

energy_slice(*E*)

Return a one-dimensional slice of the spectrum at a given energy.

time_slice(*t*)

Return a one-dimensional slice of the spectrum at a given time.

tmax()

Return the maximum time for which the spectrum is defined.

tmin()

Return the minimum time for which the spectrum is defined.

class ixpeobssim.srcmodel.spectrum.**xXspecModel**(*expression*, *parameters*, *w=None*, *bbox=None*, *k=3*, *ext=0*, *emin=1.0*, *emax=12.0*, *num_points=250*, *correct=True*)

Basic interface to a generic time-independent XSPEC model.

This is essentially a univariate interpolated spline that is constructed from a regular-grid sample of a generic XSPEC model, defined by an expression and a (full) set of parameters.

The model can be loaded from a text file in a suitable form using the xXspecModel.from_file() facility, and the data points can be exported to a text file using the save_ascii() class method.

Parameters

- **expression** (*str*) – The model expression string, using full XSPEC component names.

- **parameters** (*dict or list*) – The model parameters. This can take the form of either a list, where all the parameters are passed (in the right order), or a dictionary indexed by the serial identifier of the parameter itself. The second form allows for passing along only a subset of the parameters and mimics the behavior of the XSPEC Python bindings described in https://heasarc.gsfc.nasa.gov/docs/xanadu/xspec/python/html/model.html

  Note that any attempt of implementing a structure where the parameters are passed by name is made non trivial by the possibility of compound models featuring multiple instances of the same parameter names (for different components), and the questionable benefit provided by this idea was deemed not worth the additional complications connected with that.
- **w** – Set of parameters controlling the spline
- **bbox** – Set of parameters controlling the spline
- **k** – Set of parameters controlling the spline
- **ext** – Set of parameters controlling the spline
- **emin** (*double*) – Energy limits, in keV.
- **emax** (*double*) – Energy limits, in keV.
- **num_points** (*int*) – The number of points to sample the spectrum.
- **correct** (*bool*) – If True, we attempt at converting the integral values from XSPEC into actual fluxes at the bin centers. (This is achieved via a numerical integration of a cubic spline.)

classmethod from_file(*file_path*, *w=None*, *bbox=None*, *k=3*, *ext=0*, *emin=1.0*, *emax=12.0*, *num_points=250*)

Create a class instance from file.

The basic file format is the following. (Note that it is up to the user to make sure that the components are defined in the same order in which they first appear in the model expression string, and that the order of the parameters within each component is exactly the one that XSPEC is expecting.)

```
model     phabs*powerlaw
COMP1         phabs
nH    3.
COMP2     powerlaw
PhoIndex  1.
norm      100.
```

save_ascii(*file_path*)

Save a txt file with spectrum data points.

Module containing functions and classes to manage time delays

ixpeobssim.srcmodel.tdelays.**SSB = {'all': <function alls_f>, 'einstein': <function einsteins_f>, 'roemer': <function roemers_f>, 'shapiro': <function shapiros_f>}**

Dictionary for delays in Binary System (BS)

ixpeobssim.srcmodel.tdelays.**allb_f**(*t_*, *eph*, ***kwargs*)

All delays referred to Binary System.

ixpeobssim.srcmodel.tdelays.**alls_f**(*t_*, ***kwargs*)

All delays referred to Solar System Barycenter.

ixpeobssim.srcmodel.tdelays.**einsteinb_f**(*t_*, ***kwargs*)

Einstein delays formula referred to Binary System.

> **Warning**
>
> Not implemented yet.

ixpeobssim.srcmodel.tdelays.**einsteins_f**(*t_*, ***kwargs*)

Einstein delays formula referred to Solar System Barycenter.

> **Warning**
>
> First implementation of satellite location, temporarily avoided.

ixpeobssim.srcmodel.tdelays.**roemerb_f**(*t_*, *eph*, ***kwargs*)

Roemer delays formula referred to Binary System.

> **Warning**
>
> IXPE satellite ephemeris not implemented yet.

Parameters

- **t** (*float*) – Terrestrial time in MET
- **eph** (*xOrbitalEphemeris object*)
- ****kwargs** – ra : Right ascension of the source dec : Declination of the source

Returns

Return the earth barycentric vector

Return type

array float

ixpeobssim.srcmodel.tdelays.**roemers_f**(*t_*, ***kwargs*)

Roemer delays formula referred to Solar System Barycenter.

> **Warning**
>
> IXPE satellite ephemeris not implemented yet.

Parameters

- **t** (*float*) – Terrestrial time in MET
- ****kwargs** – ra : Right ascension of the source dec : Declination of the source

Returns

Return the earth barycentric vector

Return type

array float

ixpeobssim.srcmodel.tdelays.**shapirob_f**(*t_*, ***kwargs*)

Shapiro delays formula referred to Binary System.

> **Warning**
>
> Not implemented yet.

ixpeobssim.srcmodel.tdelays.**shapiros_f**(*t_*, ***kwargs*)

Shapiro delays formula referred to Solar System Barycenter.

> **Warning**
>
> Not implemented yet.

class ixpeobssim.srcmodel.tdelays.**xTDelays**(*mjdvalue*, *name=''*, *unit='MJD'*)

Convenience class encapsulating times handling.

Default time in input is in mjd unit, both mjd and met time scale are managed.

Parameters

- **mjdvalue** (*array float*) – array of time values in MJD unit
- **metvalue** (*array float*) – array of time values in MET unit
- **unit** (*string*) – time unit in MET or MJD
- **name** (*string*) – name of the model

apply_decorr(*ephemeris*, *ra*, *dec*, *samples=500*, *delay='all'*, *binary=False*)

Return delay effected times.

Parameters

samples (*int*) – Number of samples per period

apply_delay(*ephemeris*, *ra*, *dec*, *delay='all'*, *binary=False*)

Return barycentered times, given an ephemeris and the source coordinates.

Parameters

- **ephemeris** (*xOrbitalEphemeris object*)
- **ra** (*float*) – Right ascension of the source
- **dec** (*float*) – Declination of the source
- **delay** (*string*) – define the delay function to use: roemer, einstein, shapiro or all
- **binary** (*Boolean*) – When true time delay is applied in the binary system, when false in the solar system

met_max()

Return the maximum in met unit.

met_min()

Return the minimum in met unit.

mjd_max()

Return the maximum in mjd unit.

mjd_min()

Return the minimum in mjd unit.

29.9 Utilities

class ixpeobssim.utils.argparse_.**xArgumentFormatter**(*prog*, *indent_increment=2*, *max_help_position=24*, *width=None*)

Do nothing class combining our favorite formatting for the command-line options, i.e., the newlines in the descriptions are preserved and, at the same time, the argument defaults are printed out when the –help options is passed.

The inspiration for this is coming from one of the comments in https://stackoverflow.com/questions/3853722

class ixpeobssim.utils.argparse_.**xArgumentParser**(*prog=None*, *usage=None*, *description=None*)

Light-weight wrapper over the argparse ArgumentParser class.

This is mainly intended to reduce boilerplate code and guarantee a minimum uniformity in terms of how the command-line options are expressed across different applications.

> **Warning**
>
> Mind you should refrain adding options containing a hyphen, as that will break the pipeline code. (This is a consequence of the fact that argparse tranforms, for good reasons, hyphens into underscores at parse time.)

add_auxversion(*default=3*)

add_batch()

Custom option.

add_boolean(*name*, *default*, *help*)

Add a boolean argument.

add_charging(*default=False*)

Add all the charging-related options.

add_column_density(*default=0.0*)

add_configfile(*default=None*, *required=True*)

Custom option.

add_deadtime(*default=0.00108*)

Custom option.

add_dithering(*default=True*)

Add all the dithering-related options.

add_duration(*default=1000*)

Custom option.

add_ebinning(*default_emin=2.0*, *default_emax=8.0*, *ebin_algs=['FILE', 'LIN', 'LOG', 'LIST']*, *default_ebinalg='LOG'*, *default_ebins=4*)

Custom options.

add_ebounds(*default_emin=2.0*, *default_emax=8.0*)

Custom options.

add_eef(*default=1.0*)

Custom option for the encircled energy fraction factor.

This is a multiplicative factor (less or equal to 1) used in sensitivity calculation to account for the effect of the far tails of the PSF, which will be removed by any sensible spatial cut to reduce the background.

add_file()

Custom option.

add_filelist()

Custom option.

add_grayfilter()

Custom option.

add_gti_settings(*default_min_duration=0.0*, *default_start_pad=0.0*, *default_stop_pad=0.0*)

Custom option.

add_irfname(*default='ixpe:obssim20240101:v13'*, *required=False*)

Custom option.

add_mc(*default=False*)

Custom option.

add_objname(*default=None*)

add_on_orbit_calibration()

Add the options related to the onboard calibration.

add_outfile(*default=None*)

Custom option.

add_outfolder(*default='/home/docs/ixpeobssimdata'*)

Custom option.

add_overwrite(*default=True*)

Custom option.

Note we are doing this mess with the eval and choices so that this can be conveniently wrapped into the pipeline class.

add_phasebounds(*default_phasemin=None*, *default_phasemax=None*)

Custom options.

add_phi0(*default=0.0*)

add_pl_index(*default=2.0*)

add_pl_norm(*default=0.0045023*)

add_redshift(*default=0.0*)

add_roll(*default=0.0*)

Custom option.

add_save()

Custom option.

add_sc_data()

add_seed(*default=None*)

Custom option.

The default for the default argument was changed from 0 to None as a consequence of issue #189. It is intended that, if the seed is None, the downstream code will generate a random random seed instead.

add_srcid(*default=0*)

Custom option for the identification of a source into the ROI.

add_srcname(*default=None*)

Custom option for the identification of a source into the ROI.

add_startdate(*default='2022-04-21'*)

Custom option.

Note the Easter egg—the default is Martin's birthday!

add_stopdate(*default='==SUPPRESS=='*)

Custom option.

add_stretch(*default='linear'*)

add_suffix(*default=None*)

Custom option.

add_target_source()

Add the option for the target source.

add_tbounds(*default_tmin=None*, *default_tmax=None*)

Custom options.

add_timeline()

add_trajectory(*default=False*)

Add the trajectory-related options.

add_vignetting(*default=True*)

add_vrange()

add_weightcol(*default='W_MOM'*)

add_weightname(*default='alpha075'*)

add_weights(*default=False*)

static optional_int(*value*)

Small convenience funtion to convert 'None' into None when parsing arguments.

parse_args(*args=None*, *namespace=None*)

Overloaded method.

Here we print the ixpeobssim start message, in addition to the standard help output.

print_help()

Overloaded method.

Here we print the ixpeobssim start message, in addition to the standard help output.

class ixpeobssim.utils.argparse_.**xPipelineParser**(*prog=None*, *usage=None*, *description=None*)

Specialized argument parser for analysis pipelines.

class ixpeobssim.utils.argparse_.**xSourceModelArgumentParser**(*prog=None*, *usage=None*, *description=None*)

Specialized argument parser for source models.

Utilities for the Chandra to IXPE conversion.

ixpeobssim.utils.chandra.**arf_ratio**(*ixpe_arf*, *ixpe_vign*, *chandra_arf*, *chandra_vign*, *emin=1.0*, *emax=10.0*, *thetamax=8.5*)

Make the ratio between IXPE and Chandra effective area taking in account the vignetting.

This is returning an interpolated bivariate spline in the energy-off-axis angle plane (the energy is measured in keV and the off-axis angle in arcmin).

ixpeobssim.utils.chandra.**gti**(*gtidata*)

Return the sum of the GTI in the corresponding extension data.

ixpeobssim.utils.chandra.**livetime**(*header*)

Return the livetime of the observation.

ixpeobssim.utils.chandra.**load_arf**(*detname='ACIS-I'*)

Load the Chandra effective area data from file.

ixpeobssim.utils.chandra.**load_vign**()

Load the Chandra vignetting data from file.

ixpeobssim.utils.chandra.**pointing**(*header*)

Return the pointing direction.

ixpeobssim.utils.codestyle.**square**(*x*)

Return the square of x.

Parameters
x (*number or array*) – The input x value (can be either a numeric type or an array).

Returns
The square of the input value

Return type
float

Examples

```
>>> x = 3.
>>> y = square(x)
>>> print(y)
```

Note

Yes, this is a pretty dumb function, but watch out for the use of doctrings.

class ixpeobssim.utils.codestyle.**xSampleClass**(*name*, *description=None*)

An example class, doing nothing useful.

Parameters

- **name** (*string*) – The instance name.
- **description** (*string*) – An optional description.

Examples

```
>>> from codestyle import xSampleClass
>>> c1 = xSampleClass('hello')
>>> c2 = xSampleClass('world!')
>>> print(c1 + c2)
```

ixpeobssim.utils.fmtaxis.**axis_label**(*name*, *units=None*)

class ixpeobssim.utils.fmtaxis.**fmtaxis**

Dumb containter class for axis formats.

class ixpeobssim.utils.fmtaxis.**label**

Dumb containter class for axis labels

class ixpeobssim.utils.fmtaxis.**units**

Dumb container class for units.

ixpeobssim.utils.logging_.**abort**(*message=''*)

Abort the execution (via a sys.exit) with a message.

Use this with care, and opt for custom exceptions whenever possible.

ixpeobssim.utils.logging_.**startmsg**()

Print the start message.

class ixpeobssim.utils.logging_.**xTerminalFormatter**(*fmt=None*, *datefmt=None*, *style='%'*, *validate=True*)

Logging terminal formatter class.

format(*record*)

Overloaded format method.

ixpeobssim.utils.math_.**decimal_places**(*val*)

Calculate the number of decimal places so that a given value is rounded to exactly two signficant digits.

Note that we add epsilon to the argument of the logarithm in such a way that, e.g., 0.001 is converted to 0.0010 and not 0.00100. For values greater than 99 this number is negative.

ixpeobssim.utils.math_.**decimal_power**(*val*)

Calculate the order of magnitude of a given value,i.e., the largest power of ten smaller than the value.

ixpeobssim.utils.math_.**fold_angle_deg**(*phi*)

Fold an azimuthal angle (in degrees) within [-180, 180] to [-90, 90].

Parameters
phi (*array_like or scalar*) – The input angle or array of values.

ixpeobssim.utils.math_.**fold_angle_rad**(*phi*)

Fold an azimuthal angle (in degrees) within [-pi, pi] to [-pi/2, pi/2].

Parameters
phi (*array_like or scalar*) – The input angle or array of values.

ixpeobssim.utils.math_.**format_value**(*value*, *precision=3*)

Format a number with a reasonable precision

ixpeobssim.utils.math_.**format_value_error**(*value*, *error*, *pm='+/-'*, *max_dec_places=6*)

Format a measurement with the proper number of significant digits.

ixpeobssim.utils.math_.**modulo_2pi**(*phi*)

Compute the modulo operation bringing the output angles (in radians) into the interval [-pi, pi].

Parameters
phi (*array_like or scalar*) – The input angle or array of values.

ixpeobssim.utils.math_.**weighted_average**(*values*, *weights=None*)

Return the weighted average of an array of values.

This is simply a wrapper over the numpy.average() function.

ixpeobssim.utils.matplotlib_.**add_slider**(*val_func*, *img_obj=None*, *ax=None*, *coords=(0.2, 0.05, 0.6, 0.03)*, *label=None*, *valmin=0.0*, *valmax=1.0*, *num_step=10*, *color='lightgoldenrodyellow'*, ***kwargs*)

Add a slider object to a plot. This can be used e.g. to naviagte a 3-dimensional histogram by plotting a slice of it at a given bin on its last axis, which the user can change by scrolling the slider. Requires as input a function that returns the new image data based on the current value of the slider and the graphical object which is linked to the slider (e.g. the one produced by plot() or imgshow()). Default is the current image.

ixpeobssim.utils.matplotlib_.**color_wheel_ar**(*i*)

Support for a custom color wheel.

This was originally defined in evt.colorselection as setEnergyColor() and was moved here for consistency, see issue #251.

ixpeobssim.utils.matplotlib_.**color_wheel_mpr**(*i*)

Support for a custom color wheel.

This was originally defined in __init__ as xpColor() and was moved here for consistency, see issue #252.

ixpeobssim.utils.matplotlib_.**context_no_grids**()

Setup the current figure with no grids.

ixpeobssim.utils.matplotlib_.**context_two_by_two**(*scale=1.9*)

Setup the current figure for a 2x2 panel.

ixpeobssim.utils.matplotlib_.**draggable_colorbar**(*mappable=None*, *cax=None*, *ax=None*, ***kwargs*)

Create a draggable colorbar

ixpeobssim.utils.matplotlib_.**du_color**(*du_id*)

Return the default color for a give DU.

ixpeobssim.utils.matplotlib_.**labeled_marker**(*x*, *y*, *label*, *dx=0*, *dy=0*, ***kwargs*)

Draw a marker and a label at a specified point.

ixpeobssim.utils.matplotlib_.**last_line_color**(*default='black'*)

Return the color used to draw the last line

ixpeobssim.utils.matplotlib_.**marker**(*x*, *y*, ***kwargs*)

Draw a marker at a specified point.

ixpeobssim.utils.matplotlib_.**metplot**(*met*, *values*, *display='dt'*, **args*, ***kwargs*)

ixpeobssim.utils.matplotlib_.**nlog_errorbars**(*x*, *y*, *dy*, ***kwargs*)

Plot an errorbar by taking the absolute value of y and flagging the negative part with hollow markers.

This is useful to plot Stokes parameters (that can go negative) in log scale.

ixpeobssim.utils.matplotlib_.**plot_arrows**(*grid*, *field*, *threshold=0.0*, ***kwargs*)

Plot a 2-dimensional field of arrows.

This is a lightweight wrapper upon the plt.quiver() method, that is taylored to the overlay of arrow fields upon polarizaion degree maps.

Parameters

- **grid** (*array_like*) – The underlying grid for the display
- **field** (*array_like or callable*) – Anything that can be called on the grid and returns the components of the arrow field in the grid points, or an array matching the grid that can be used directly
- **threshold** (*float, optional*) – Optional threshold on the magnitude of the field at any given point (points below the threshold are set to zero)
- **kwargs** (*dict*) – All the keyword arguments passed to plt.quiver()

ixpeobssim.utils.matplotlib_.**plot_circle**(*center*, *radius*, ***kwargs*)

Plot a circle.

ixpeobssim.utils.matplotlib_.**plot_ellipse**(*center*, *width*, *height*, ***kwargs*)

Plot an ellipse.

ixpeobssim.utils.matplotlib_.**rc_param**(*key*)

Return a given matplotlib configuration property.

ixpeobssim.utils.matplotlib_.**residual_plot**(*figure_name=None*, *separation=0.3*, *padding=0.01*, *ylabel_offset=-0.085*, *figsize=(8.0, 6.0)*, *sharex=False*)

Create a new figure with two axes objects for residual plots.

Parameters

- **figure_name** (*str*) – The name of the figure.
- **separation** (*float*) – The vertical separation point between the two axes.

ixpeobssim.utils.matplotlib_.**save_all_figures**(*output_folder*, *file_extensions=('png', 'pdf')*)

Save all the figures in memory to a given output folder.

ixpeobssim.utils.matplotlib_.**save_gcf**(*output_folder='/home/docs/ixpeobssimdata'*, *file_name=None*, *file_extensions=('pdf', 'png')*)

Save the current matplotlib figure.

If the current figure has a sensible name, it will be used to construct the path to the output file—we just make everything lower case and replace spaces with _.

Examples

```
>>> from ixpeobssim.utils.matplotlib_ import *
>>> plt.figure('Test figure')
>>> # ... do something.
>>> # This will create `output_folder/test_figure.png`.
>>> save_gcf('output_folder')
```

Parameters

- **output_folder** (*string*) – The path to the output folder (default to pwd).
- **file_name** (*string*) – The figure name (the name of the output files will be name.extension).
- **file_extensions** (*list of strings*) – A list of extensions the figure needs to be saved into.

Returns

The list of paths to the file(s) being created by the function.

Return type

list

ixpeobssim.utils.matplotlib_.**setup**()

Basic system-wide setup.

The vast majority of the settings are taken verbatim from the matplotlib 2, commit 5285e76: https://github.com/matplotlib/matplotlib/blob/master/matplotlibrc.template

Note that, although this is designed to provide an experience which is as consistent as possible across different matplotlib versions, some of the functionalities are not implemented in older versions, which is why we wrap each parameter setting into a _set_rc_param() function call.

ixpeobssim.utils.matplotlib_.**setup_gca**(*xlabel=None*, *ylabel=None*, *xmin=None*, *xmax=None*, *ymin=None*, *ymax=None*, *logx=False*, *logy=False*, *grids=False*, *xticks=None*, *yticks=None*, *legend=False*)

Setup the axes for the current plot.

ixpeobssim.utils.matplotlib_.**setup_gca_stokes**(*side=1.0*, *pd_grid=array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])*, *pd_grid_label_angle=45.0*, *pa_grid_step=30.0*, *line_width=0.75*, ***kwargs*)

Setup the current axis object in a way that is appropriate for plotting Stokes parameters, i.e., for the (Q/I, U/I) phase space.

Parameters

- **side** (*float*) – The absolute value of the minimum and maximum Q/I and U/I values for the axes ranges.
- **pd_grid** (*array like*) – The polarization degrees values for which we plot the reference circles.
- **pd_grid_label_angle** (*float, optional*) – The angle (in decimal degrees) at which the pd level labels are rendered.
- **pa_grid_step** (*float*) – The azimuthal angle step (in degrees) for the diagonal lines.
- **kwargs**

class ixpeobssim.utils.matplotlib_.**xDraggableColorbar**(*cbar*, *mappable*)

Interactive colorbar than can be dragged and zoomed. Most of the code is taken from: http://www.ster.kuleuven.be/~pieterd/python/html/plotting/interactive_colorbar.html

Input arguments: a standard colorbar and a mappable object (e.g. an image created with pyplot.imshow())

cmap_index()

Return the index of the current color map

cmap_name()

Return the name of the current color map

connect()

Connect all the events we need

disconnect()

Disconnect all the stored connection ids

key_press(*event*)

We connect three butons: 'up' and 'down' to change the colormap and 'r' to restore the original view (colormap and limits)

on_motion(*event*)

On motion we check if the mouse is over the colorbar. If the mouse LEFT button is pressed we shift the extrema of the colorbar, if the RIGHT button is pressed we zoom in/out

on_press(*event*)

On button press we will see if the mouse is over the colorbar and store the coordinates of the click

on_release(*event*)

On release we reset the press data

redraw()

Redraw the colorbar and the related mappable object

restore()

Restore the original color map, vmin and vmap

set_label(*zlabel*)

Set the colorbar label.

show_cmap_name()

Show the color map name on top of the color bar

update_cmap(*index*)

Set the color map to the one at the given index

update_limits(*vmin*, *vmax*)

Update the extrema (vmin and vmax) of the colorbar and update the mappable color normalization to it

update_norm()

Update the color normalization of the mappable object to reflect that of the colorbar

class ixpeobssim.utils.matplotlib_.**xStatBox**(*position='upper left'*, *halign='left'*, *valign='top'*)

Base class describing a text box, to be used for the histogram and fit stat boxes.

In the initial implementation this was wrapped into a simple function, but we immediately run into problems in cases where we needed to setup a stat box in differen steps before drawing it on the canvas (e.g., when a FitModel subclass needs to customize the stat box of the base class). The class approach is more flexible, although one needs a few more lines of code to add entries and plot the thing.

Parameters

- **position** (*str of tuple*) – It can either be a two-element tuple (in which case the argument is interpreted as a position in absolute coordinates, with the reference corner determined by the alignment flags), or a string in the set ['upper left', 'upper right', 'lower left', 'lower rigth']. If position is a string, the alignment flags are ignored.
- **halign** (*str*) – The horizontal alignment ('left' | 'center' | 'right')
- **valign** (*str*) – The vertical alignment ('top' | 'center' | 'bottom')

add_entry(*label*, *value=None*, *error=None*)

Add an entry to the stat box.

plot(***kwargs*)

Plot the stat box.

Parameters

****kwargs** (*dict*) – The options to be passed to *plt.text()*

set_position(*position*, *halign='left'*, *valign='top'*)

Set the position of the bounding box.

class ixpeobssim.utils.matplotlib_.**xTextCard**

Small class reperesenting a text card.

This is essentially a dictionary that is capable of plotting itself on a matplotlib figure in the form of a multi-line graphic card.

draw(*x0=0.1*, *y0=0.9*, *line_spacing=0.1*, *spacing_ratio=0.9*)

Draw the card.

Parameters

- **x0** (*float*) – The absolute coordinates of the top-left corner of the card.
- **y0** (*float*) – The absolute coordinates of the top-left corner of the card.
- **line_spacing** (*float*) – The line spacing in units of the total height of the current axes.
- **spacing_ratio** (*float*) – The fractional line spacing assigned to the key label.

set_line(*key*, *value*, *fmt='%s'*, *units=None*)

Set the value for a given key.

Parameters

- **key** (*str*) – The key, i.e., the explanatory text for a given value.

- **value** (*number or str*) – The actual value (if None, a blank line will be added).
- **fmt** (*str*) – The string format to be used to render the value.
- **units** (*str*) – The measurement units for the value.

Miscellaneaous utilities.

ixpeobssim.utils.misc.**pairwise**(*iterable*)

Iterate over a binning vector.

This will give you the bin edges for each bin, see https://stackoverflow.com/questions/5764782

ixpeobssim.utils.misc.**pairwise_enum**(*iterable*)

Iterate over a binning vector keeping track of the indices.

ixpeobssim.utils.misc.**process_file_list**(*processing_function*, *file_list*, **args*, ***kwargs*)

Filter a series of files with a given processing function and return the list of the of the outputs from the processing, filtering out the None values.

This is aimed at reducing the biolerplate code in the apps, where we typically make the same processing on a list of files and return the list of the processed files. Since for this application None is a signal for an existing file that does not get overwritten, filtering it out is helpful downstream.

Parameters

- **processing_function** (*callable*) – The processing function to be called on the file—this should take the file path as a first argument.
- **file_list** (*iterable*) – The list of paths to the files to be processed.
- ***args** – Positional arguments to be passed to the processing function.
- ****kwargs** – Keyword arguments to be passed to the processing function.

ixpeobssim.utils.os_.**check_input_file**(*file_path*, *extension=None*)

Make sure that an input file exists (and, optionally, has the right extension). We accept a list of extensions, specifically to support the case where a file can be given in compressed form (astropy handles 'fits' and 'gz' transparently).

Note that we abort the execution with no mercy if anything fails.

ixpeobssim.utils.os_.**check_output_file**(*file_path*, *suffix*, *overwrite=False*, *extension='fits'*)

Small utility function to manage the I/O in the ixpeobssim applications.

This basically build the path to the output file, given that to the input file, given a small set of rules. Among other things, the function verifies that the input file exists and has the right extension. If the output file exists and the overwite flag is not set, the function is returning None, as a mean for consumer functions to be aware that the aformentioned files should not be overwritten.

Parameters

- **file_path** (*str*) – The path to the input file.
- **suffix** (*str*) – The suffix to be attached to the output file.
- **overwrite** (*bool*) – Flag to automatically overwite existing files.
- **extension** (*str*) – The extension of the input and output files.

ixpeobssim.utils.os_.**cp**(*source*, *dest*, *create_tree=False*)

Copy a file.

Return 0 upon succesfull operation, 1 otherwise.

ixpeobssim.utils.os_.**filter_input_file_list**(*file_list*)

Out of a list of files return only those with valid input file extensions, .e. '.fits', or '.fits.gz'

ixpeobssim.utils.os_.**mkdir**(*dir_path*)

Create a directory (unless it already exists).

Return 0 upon succesfull operation, 1 otherwise.

ixpeobssim.utils.os_.**mv**(*source*, *dest*)

Move a file.

Return 0 upon succesfull operation, 1 otherwise.

ixpeobssim.utils.os_.**rm**(*file_path*)

Remove a file.

Return 0 upon succesfull operation, 1 otherwise.

ixpeobssim.utils.os_.**rmdir**(*dir_path*)

Remove an entire (empty or non empty) folder.

ixpeobssim.utils.packaging_.**parse_version_string**(*version_string*)

Small utility function to parse the version string of a generic package.

Note that we only parse the major, minor and patch fields (i.e., the release segment) of the version string.

ixpeobssim.utils.packaging_.**retrieve_version**(*package*)

Retrieve the version for a generic package.

class ixpeobssim.utils.packaging_.**xPackageVersion**(*major*, *minor*, *patch=None*)

Small class encapsulating a package version.

This was introduced when fixing issue #280, and the whole mechanism is inspired by the wonderful packaging Python package (since the use of the version information we are doing is fairly limited we didn't want to add yet another dependence on ixpeobssim).

ixpeobssim.utils.profile.**MB**(*size*)

Convert from B to MB.

ixpeobssim.utils.profile.**psavailable**()

Return the amount of available memory (in MB), as provided by psutil.

ixpeobssim.utils.profile.**psfree**()

Return the amount of free memory (in MB), as provided by psutil.

ixpeobssim.utils.profile.**psmem**()

Return the virtual memory profile (in MB), as provided by psutil.

ixpeobssim.utils.profile.**psstatus**()

Return a string representing the status of the memory.

ixpeobssim.utils.profile.**timing**(*f*)

Small decorator to time a generic function.

class ixpeobssim.utils.profile.**xChrono**

Small chronometer class.

A chronometer essentially measures the elapsed time since it has been started and is equipped to print itself to the standard output. (Note the chronometer is reset unpon the instantiation of a class object.)

Examples

```
>>> from ixpeobssim.utils.profile import xChrono
>>> c = xChrono()
>>> # ... do something.
>>> print(c)
```

reset()

Reset the chronometer.

class ixpeobssim.utils.profile.**xMemoryProfiler**

Small utility class to help profiling the allocated memory.

classmethod available()

Return the available memory.

Collection of system-related utilities.

ixpeobssim.utils.system_.**cleanup**(*dir_path*)

Remove all the files in a given folder.

ixpeobssim.utils.system_.**cmd**(*command*, *verbose=False*, *log_file_path=None*, *log_file_mode='w'*, *dry_run=False*)

Exec a system command.

This uses subprocess internally and returns the subprocess status code (if the dry_run option is true the function will just print the command out through the logger and returns happily).

By default the stdout and the stderr are redirected into subprocess pipes so that the output can be effectively used by the logger. It the log_file_path parameter is different from None the stdout is redirected to file instead. The rules are:

- if verbose is True the command output is logged onto the terminal one line at a time;
- if the status code is zero we just aknowledge that before returning it;
- upon error we try and log out both the error code and the error message.

ixpeobssim.utils.system_.**import_module**(*file_path*)

Import a module programmatically (Python 3 version).

Parameters
file_path (*str*) – The file path corresponding to the module to be imported.

Module for time conversions and related utilities.

class ixpeobssim.utils.time_.**UTCTimezone**

Derived tzinfo object to support the UTC timezone with Python 2.

See https://docs.python.org/2/library/datetime.html#tzinfo-objects for more details.

dst(*dt*)

datetime -> DST offset as timedelta positive east of UTC.

tzname(*dt*)

datetime -> string name of time zone.

utcoffset(*dt*)

datetime -> timedelta showing offset from UTC, negative values indicating West of UTC

ixpeobssim.utils.time_.**current_datetime_string**(*tzinfo=datetime.timezone.utc*, *fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Return a string with the current date and time.

Parameters
tzinfo (*a datetime.timezone instance or None*) – The timezone info (use None for local time).

Returns
The current date and time string.

Return type
string

ixpeobssim.utils.time_.**current_datetime_string_local**(*fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Return a string with the current UTC date and time.

Parameters
fmt (*string*) – An optional format specifier for non standard input string

Returns
The current local date and time string.

Return type
string

ixpeobssim.utils.time_.**current_datetime_string_utc**(*fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Return a string with the current UTC date and time.

Parameters
fmt (*string*) – An optional format specifier for non standard input string

Returns
The current UTC date and time string.

Return type
string

ixpeobssim.utils.time_.**current_met**()

Return the current mission elapsed time.

Returns
The current mission elapsed time.

Return type
float

ixpeobssim.utils.time_.**current_time**()

Return the current unix time.

Returns
The current Unix time.

Return type
float

ixpeobssim.utils.time_.**days_to_seconds**(*days*)

Convert days to seconds.

ixpeobssim.utils.time_.**met_to_jd**(*met*)

Convert a MET in the corresponding Julian date.

Parameters
met (*float*) – The mission elapsed time.

Returns
The Julian Date.

Return type
float

ixpeobssim.utils.time_.**met_to_mjd**(*met*)

Convert a MET in the corresponding Modified Julian date.

Parameters
met (*float*) – The mission elapsed time.

Returns
The Modified Julian Date.

Return type
float

ixpeobssim.utils.time_.**met_to_num**(*met*)

Convenience conversion factor to turn a MET into a number that matplotlib can interpret natively as a datetime.

According to https://matplotlib.org/3.1.1/api/dates_api.html matplotlib represents dates using floating point numbers specifying the number of days since 0001-01-01 UTC, plus 1.

ixpeobssim.utils.time_.**met_to_string**(*met*, *tzinfo=datetime.timezone.utc*, *fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Convert a MET to a string expressing time and date.

Parameters

- **met** (*float*) – The input mission elapsed time.
- **tzinfo** (*a datetime.timezone instance or None*) – The timezone info (use None for local time).
- **fmt** (*string*) – The format for the output string

Returns
The string corresponding to the input time

Return type
string

ixpeobssim.utils.time_.**met_to_string_local**(*met*, *fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Convert a MET to a string representing local date and time.

Parameters

- **met** (*float*) – The input mission elapsed time.
- **fmt** (*string*) – The format for the output string

Returns
The string corresponding to the input time

Return type
string

ixpeobssim.utils.time_.**met_to_string_utc**(*met*, *fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Convert a MET to a string representing UTC date and time.

Parameters

- **met** (*float*) – The input mission elapsed time.
- **fmt** (*string*) – The format for the output string

Returns
The string corresponding to the input time

Return type
string

ixpeobssim.utils.time_.**met_to_unix**(*met*)

Convert a MET to a Unix time.

Parameters
met (*float*) – The input mission elapsed time.

Returns
The Unix time corresponding to the input mission elapsed time.

Return type
float

ixpeobssim.utils.time_.**mjd_to_met**(*mjd*)

Convert a MJD in the corresponding Mission Elapsed Time.

Parameters
mjd (*float*) – The time ref to Modified Julian Day.

Returns
The mission elapsed time.

Return type
float

ixpeobssim.utils.time_.**seconds_to_days**(*seconds*)

Convert seconds to days.

ixpeobssim.utils.time_.**seconds_to_years**(*seconds*)

Convert seconds to days.

ixpeobssim.utils.time_.**string_to_met**(*string*, *tzinfo=datetime.timezone.utc*, *fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Convert a string expressing time and date to a MET.

Parameters

- **string** (*string*) – The input datetime string.
- **tzinfo** (*a datetime.timezone instance or None*) – The timezone info (use None for local time).
- **fmt** (*string*) – An optional format specifier for non standard input string

Returns
The mission elapsed time corresponding to the input string.

Return type
float

ixpeobssim.utils.time_.**string_to_met_local**(*string*, *fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Convert a string expressing a local time and date to a MET.

Parameters

- **string** (*string*) – The input datetime string.
- **fmt** (*string*) – An optional format specifier for non standard input string

Returns
The mission elapsed time corresponding to the input string.

Return type
float

ixpeobssim.utils.time_.**string_to_met_utc**(*string*, *lazy=False*, *fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Convert a string expressing a UTC time and date to a MET.

Parameters

- **string** (*string*) – The input datetime string.
- **lazy** (*bool*) – Flag to attempt and auto-fix partially formed datetime strings.
- **fmt** (*string*) – An optional format specifier for non standard input string

Returns
The mission elapsed time corresponding to the input string.

Return type
float

ixpeobssim.utils.time_.**string_to_unix**(*string*, *tzinfo=datetime.timezone.utc*, *fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Convert a string expressing time and date to a Unix time.

Parameters

- **string** (*string*) – The input datetime string.
- **tzinfo** (*a datetime.timezone instance or None*) – The timezone info (use None for local time).
- **fmt** (*string*) – An optional format specifier for non standard input string

Returns
The Unix time corresponding to the input string.

Return type
float

ixpeobssim.utils.time_.**string_to_unix_local**(*string*, *fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Convert a string expressing a local time and date to a Unix time.

Parameters

- **string** (*string*) – The input datetime string.
- **fmt** (*string*) – An optional format specifier for non standard input string

Returns
The Unix time corresponding to the input string.

Return type
float

ixpeobssim.utils.time_.**string_to_unix_utc**(*string*, *fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Convert a string expressing a UTC time and date to a Unix time.

Parameters

- **string** (*string*) – The input datetime string.
- **fmt** (*string*) – An optional format specifier for non standard input string

Returns
The Unix time corresponding to the input string.

Return type
float

ixpeobssim.utils.time_.**unix_to_met**(*ut*)

Convert a Unix time to a MET.

Parameters
ut (*float*) – The input Unix time.

Returns
The mission elapsed time corresponding to the input Unix time.

Return type
float

ixpeobssim.utils.time_.**unix_to_string**(*ut*, *tzinfo=datetime.timezone.utc*, *fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Convert a Unix time to a string expressing time and date.

Parameters

- **ut** (*float*) – The input Unix time.
- **tzinfo** (*a datetime.timezone instance or None*) – The timezone info (use None for local time).
- **fmt** (*string*) – The format for the output string

Returns
The string corresponding to the input time

Return type
string

ixpeobssim.utils.time_.**unix_to_string_local**(*ut*, *fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Convert a Unix time to a string representing local date and time.

Parameters

- **ut** (*float*) – The input Unix time.
- **fmt** (*string*) – The format for the output string

Returns
The string corresponding to the input time

Return type
string

ixpeobssim.utils.time_.**unix_to_string_utc**(*ut*, *fmt='%Y-%m-%dT%H:%M:%S.%f'*)

Convert a Unix time to a string representing UTC date and time.

Parameters

- **ut** (*float*) – The input Unix time.
- **fmt** (*string*) – The format for the output string

Returns
The string corresponding to the input time

Return type
string

class ixpeobssim.utils.time_.**xTimeInterval**(*start_met*, *stop_met*)

Small convenience class to encapsulate the very concept of a time interval.

This was added after https://bitbucket.org/ixpesw/ixpeobssim/issues/417 in an attempt to avoid code duplications wherever we have objects (e.g., GTIs, observation epochs or calibration runs) that have a start and a stop time.

Note that all the times are assumed to be in MET.

bounds()

Return the bounds of the epoch in the form of a two-element tuple (start, stop).

property duration

Return the total duration of the time interval in seconds.

ixpeobssim.utils.time_.**years_to_seconds**(*years*)

Convert years to seconds.

Facilities for conversions across different units.

ixpeobssim.utils.units_.**arcmin_to_arcsec**(*val*)

Convert arcminutes to arcseconds.

ixpeobssim.utils.units_.**arcmin_to_degrees**(*val*)

Convert arcminutes to degrees.

ixpeobssim.utils.units_.**arcsec_to_arcmin**(*val*)

Convert arcseconds to arcminutes.

ixpeobssim.utils.units_.**arcsec_to_degrees**(*val*)

Convert arcseconds to degrees.

ixpeobssim.utils.units_.**atm_to_mbar**(*val*)

Convert atm to mbar.

ixpeobssim.utils.units_.**celsius_to_kelvin**(*val*)

Convert Celsius degrees to Kelvin degrees.

ixpeobssim.utils.units_.**degrees_to_arcmin**(*val*)

Convert degrees to arcminutes.

ixpeobssim.utils.units_.**degrees_to_arcsec**(*val*)

Convert degrees to arcseconds.

ixpeobssim.utils.units_.**erg_to_keV**(*val*)

Convert erg to keV.

ixpeobssim.utils.units_.**ergcms_to_mcrab**(*val*)

Convert a flux in erg per cm square into mcrab.

ixpeobssim.utils.units_.**keV_to_erg**(*val*)

Convert keV to erg.

ixpeobssim.utils.units_.**mbar_to_atm**(*val*)

Convert mbar to atm.

CHAPTER

THIRTY

CODE DEVELOPMENT

This page includes some useful information and pointers for people willing to contribute changes to ixpeobssim. We assume that you have read the *Installation* page before landing here.

Warning

This page is fairly Linux-centric, in that most of the development, this days, happens under GNU-Linux. That said, all the components and tools that we use are instrinsically cross-platform, and ixpeobssim is known to be working on Windows and Mac. Feel free to edit this page making it more friendly for Windows and Mac users.

30.1 Up and running with github

git is a distributed version control system and github is the web hosting service that we use to develop the public version of ixpeobssim. Here is the entry point for the git documentation, in case you want to have a feeling of what git is doing and how you use it.

Mind that, in order to be able to push back changes to the remote repository you will need to tell git on your machine who you are, i.e.:

```
git config --global user.name "Your Name"
git config --global user.email you@example.com
```

30.2 Cloning the repository

In order to clone the repository, go to the webpage ixpeobssim, and click on the top right "fork" icon. Create your fork, then clone it on your local device, by typing:

```
git clone git@github.com:github_username/ixpeobssim.git
```

(mind this will create an ixpeobssim folder in the current directory, so cd to the appropriate place in your filesystem first).

If you get an error message along the lines of

```
Permission denied (publickey).
fatal: Could not read from remote repository.
Please make sure you have the correct access rights and the repository exists.
```

that simply means that you have to exchange your public SSH key with the server. In order to do that, click on your github profile icon on the top-right of the github webpage, select "settings", "SSH and GPG keys", "New SSH key" (top right) and paste in the form the content of the local (i.e. on the machine you are cloning the repository into) ~/.ssh/id_rsa.pub file.

If you don't have a public ssh key, you can generate it by typing

```
ssh-keygen
```

(press ENTER a couple of times and here is your public key in ~/.ssh/id_rsa.pub)

30.3 Basic git workflow

The `ixpeobssim` public repository is intentionally protected, meaning that nobody is allowed to push changes directly to it.

Everybody can merge changes onto the public repository, via pull requests provided that there's a least one approval. This scheme makes it for a fairly horizontal development approach, where everybody can contribute changes more or less independently, but forces people to do so in a coordinate fashion, and gives eveybody else a chance to look at the code before changes are actually merged.

The basic workflow we want to stick to is essentially the following. Whenever you are ready to start making a set of modifications, click on the "contribute" icon on your fork webpage (i.e. https://github.com/github_username/ixpeobssim), (on top of the "code" list), then "open pull request"; fill in the request and open it.

It is recommended to open a pull request from a branch of your fork, rather than from the "main", in order to be able to work on more parallel tasks. Create a new branch to work into and check it out (if you haven't already done so):

```
git branch fixing_something
git checkout fixing_something
```

It goes without saying that it is highly recommended to name the branch making clear its intent (e.g., mybranch is not a very expressive name).

At this point you are in the new branch, and you can start doing your modifications. Make sure your modifications do not break existing unit tests (scroll down below for more information about that) and, if you are writing brand new code, consider adding more unit tests covering the new territory. Once you're happy with the changes, commit them

```
git commit -m "Some expressive message" file1 file2 ... filen
git push
```

Mind that the first time you push on the new branch you will get an error message along the lines of

```
git push

   fatal: The current branch fixing_something has no upstream branch.
   To push the current branch and set the remote as upstream, use
   git push --set-upstream origin fixing_something
```

Follow the instructions and you should be all set.

Once you are done with your consistent set of modifications, go ahead on the repository web interface and create a pull request. Click on the menu icon top left of the code list, on your fork webpage, in order to select the right branch you want to make a pull request from (default in this menu is "main"); then create and open your pull request, as described above. You're all set! Wait for the comments of the reviewer, and finally merge the branch on the master (or, even better, have somebody else doing it for you).

30.4 Coding guidelines

Though we'll never be able to follow any set of coding conventions religiously, PEP 0008 is our starting point. Take a second to give a look to this short recap of the most salient guidelines:

- Use 4 spaces for indentation level (no TABS).
- Limit all lines to 79 characters.
- Surround top-level function and class definitions with two blank lines. Method definitions inside a class are surrounded by a single blank line. Use blank lines in functions, sparingly, to indicate logical sections.
- Use one import per line, right at the top of the module.
- Use single quote characters for strings and double quotes characters for triple-quoted strings.
- Avoid extraneous white spaces, and especially avoid more than one space around an assigment.
- Don't use spaces around the = sign when used to indicate a keyword argument or a default parameter value.
- Modules should have short, all-lowercase names.
- Class names should normally use the CapWords convention (for ixpeobssim starting with a *x*).
- Function and member names should be lowercase, with words separated by underscores as necessary to improve readability.
- Constants are usually defined on a module level and written in all capital letters with underscores separating words.
- Always use a *def* statement instead of an assignment statement that binds a *lambda* expression directly to an identifier.

An example module, illustrating the basic guidelines, is available on the repository at [github]/ixpeobssim/utils/codestyle.py.

30.5 Documenting the code

We use sphinx to generate the ixpeobssim documentation (which is the same big projects like Scipy, astropy and Python itself are using). We use the Napoleon extension in the Numpy flavor, and creating inline documentation essentially boils down to

- providing suitable docstrings with the appropriate syntax in the source files;
- creating suitable .rst files in the *doc/modules* folder.

In addition to *Napoleon*, you also will need *programoutput* and *sphinx_rtd_theme* sphinx extensions. You can easily get them with `pip` running:

```
python -m pip install sphinxcontrib-napoleon
python -m pip install sphinxcontrib-programoutput
python -m pip install sphinx_rtd_theme
```

Make sure also to have on your machine the dvipng package able to render math equations via `LaTeX`.

It won't take more than a few minutes to get aquainted to the basic rules, and the [github]/ixpeobssim/utils/codestyle.py module, along with its fellow [github]/doc/modules/utils.codestyle.rst file, provide a minimal working example that, compiled with sphinx, would be rendered like *ixpeobssim.utils.codestyle*.

You can compile and view the ixpeobssim documentation locally by doing

```
cd docs
make htmlall
htmlview _build/html/index.html
```

which is useful to make sure everything is in order when writing and documenting code.

Documentation is available online: *<https://ixpeobssim.readthedocs.io/en/latest/overview.html>*

> **Warning**
>
> We should update this section once the documentation is uploaded on the wiki and we have made up our mind about the access details.

30.6 Unit testing

We use the Python unittest module for the purpose (the documentation includes a whole bunch of good examples). While, again, we'll never be religious about this, it'd be great to provide as many unit tests as we can, while we develop code.

We collect the unit tests in the [github]/tests folder; [github]/tests/test_codestyle.py is the simplest possible unit test, while [github]/tests/test_spline.py is an actual working example. The file names for all the unit-testing python modules should start with *test_*, because that is the pattern that the test discovery will look for.

To run the full suite:

```
make test
```

CHAPTER

THIRTYONE

RELEASE NOTES

ixpeobssim (33.0.0) - Fri, 29 May 2026 06:29:25 +0200

- Merging pull request https://github.com/lucabaldini/ixpeobssim/pull/747
- Adding new response files for calendar years 2025 and 2026, that should match precisely what is available on the HEASARC CALDB for the same time intervals (modulo the fact that ixpeobssim uses the arf files, while ixpecalcarf recalculates everything starting from the modulation factor and the quantum efficiency).
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/738

ixpeobssim (32.0.1) - Tue, 14 Apr 2026 12:32:28 +0200

- Merging pull request https://github.com/lucabaldini/ixpeobssim/pull/746
- Adding a __init__.py file to the ixpeobssim.bkg and adding the entire folder to the MANIFEST.in to make sure that the background templates are included in the package when installed via pip.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/745

ixpeobssim (32.0.0) - Fri, 10 Apr 2026 12:29:34 +0200

- Merging pull request https://github.com/lucabaldini/ixpeobssim/pull/670
- This is major release with new functionalities related to handling the background. There should be no breaking changes.
- New tool xpsun.py to separate time windows when the detector is illuminated/not illuminated by the sun (see the corresponding help for usage).
- New flags in xpbin.py for creating flare-subtracted binned files.
- New, improved static background template, based on data collected during occultation, used by default.
- Arithmetics implemented for the xBinnedMap class.
- New functionalities to manipulate GTIs.
- Updated documentation.

ixpeobssim (31.1.1) - Mon, 29 Sep 2025 13:51:39 +0200

- Merging pull request https://github.com/lucabaldini/ixpeobssim/pull/742
- Merging pull request https://github.com/lucabaldini/ixpeobssim/pull/743
- Tentative fix to get the package up and running again on PyPI.
- **Issue(s) closed:**

  - https://github.com/lucabaldini/ixpeobssim/issues/740

ixpeobssim (31.1.0) - Thu, 29 May 2025 17:33:30 +0200

- Merging pull request https://github.com/lucabaldini/ixpeobssim/pull/732
- `__isub__` and `__imul__` dunder methods implemented for xBinnedCountSpectrum, xBinnedMDPMapCube and xBinnedPolarizationMapCube.
- Unit tests added.
- Continuous integration re-enabled on github.

ixpeobssim (31.0.3) - Thu, 03 Oct 2024 12:17:38 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/732
- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/734
- Small bug fix in the visualization of polarization cubes.
- Year-2 target list updated.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/728
  - https://github.com/lucabaldini/ixpeobssim/issues/733

ixpeobssim (31.0.2) - Thu, 22 Aug 2024 16:07:35 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/731
- Small bug fix in the significance calculation, germane to that fixed in issue #709.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/727

ixpeobssim (31.0.1) - Fri, 08 Mar 2024 08:38:15 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/726
- Fix for a critical bug that was preventing from using weights in a model-independent analysis with the latest iteration of the response files.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/725

ixpeobssim (31.0.0) - Thu, 07 Mar 2024 15:21:06 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/723
- Initial import of version 13 of the response files, in synch with the official IXPE CALDB shipped with HEASoft 6.33.
- Default IRF name bumped to one of the new sets of response files (more specifically, 'ixpe:obssim20240101:v13') now provided in 6-months time intervals.
- Documentation updated.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/724

ixpeobssim (30.6.4) - Tue, 12 Dec 2023 14:22:42 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/718

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/722
- New xSmearingMatrix class added, and a couple of unit tests along with it.
- Fix for issue #721 (compatibility issue with scipy 6x)
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/721

ixpeobssim (30.6.3) - Tue, 19 Sep 2023 16:33:03 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/718
- Small fix in the significance calculation for polarization cubes. (This won't make a huge difference for any sensible detection, but the old calculation was wrong, and you should definitely update.)
- Unit test added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/709

ixpeobssim (30.6.2) - Mon, 18 Sep 2023 18:56:54 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/717
- Extrapolated data points at 1 and 12 keV added when building the quadratic splines for specresp response files (effective area, modulation response function, modulation factor). This avoid the low-energy extrapolation problems that we had with the gray filter.
- Unit test added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/713

ixpeobssim (30.6.0) - Sun, 17 Sep 2023 21:06:28 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/716
- Added proper hooks to load the existing response files with the gray filter.
- –grayfilter option added to xpobssim and xpphotonlist.
- –grayfilter option added to xpbin for all the algorithms that are pertinent to polarization (this should allow to perform out of the box a spectro-polarimetric analysis in XSPEC, as well as a model-independent analysis).
- All the convolved logic for handling the simple weighting scheme in the file naming moved to ixpeobssim.irf.caldb.irf_file_name(), along with the (even more convoluted) logic for handling the gray filter.
- toy_point_source_grayfilter example added.
- Unit tests added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/710
  - https://github.com/lucabaldini/ixpeobssim/issues/711
  - https://github.com/lucabaldini/ixpeobssim/issues/712

ixpeobssim (30.5.0) - Wed, 24 May 2023 14:50:05 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/704
- Now the BACKSCAL keyword is written out by xpselect for an arbitary ds9 region file.
- **Issue(s) closed:**

  - https://github.com/lucabaldini/ixpeobssim/issues/543

ixpeobssim (30.4.0) - Wed, 24 May 2023 10:24:44 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/702
- Back-filling the v12 response files with the new arf and mrf appropriate for use with the gray filter (the same files are being released in the May 2023 update of the CALDB at HEASARC).
- Small update to the structure of the pseudo-caldb to make room for the new QE files and the response at the top of the window to be used in the photon-list workflow.
- Back-filling the v12 response files with the effective area at the top of the window (not yet used in the photon list workflow).

ixpeobssim (30.3.1) - Wed, 24 May 2023 08:47:40 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/706
- Small fix for the Q_N and U_N error calculation in the model-independent analysis.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/705

ixpeobssim (30.3.0) - Thu, 18 May 2023 12:53:13 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/700
- Added proper hooks to load the arf and mrf weighted response files with the SIMPLE weighting prescription
- xpbin now able to produce PCUBE (and PMAP and PMAPCUBE) binned files with weights.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/573

ixpeobssim (30.2.3) - Sat, 18 Mar 2023 10:58:48 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/695
- As run target list rendering for year 1 final update.

ixpeobssim (30.2.2) - Wed, 15 Feb 2023 10:22:34 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/692
- Minor fix in the population of the output file list for xpstokesalign.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/691

ixpeobssim (30.2.1) - Thu, 26 Jan 2023 12:31:20 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/689
- Fixing the generation of documentation on readthedocs.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/683

ixpeobssim (30.2.0) - Thu, 26 Jan 2023 11:52:59 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/689
- Explicitly setting TLMIN and TLMAX for the PI column in both the event files output by xpobssim and the files formatted by xpsimfmt—this is to ensure interoperability with the FTOOLS environment.

- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/688
  - https://github.com/lucabaldini/ixpeobssim/issues/588

ixpeobssim (30.1.1) - Tue, 24 Jan 2023 15:40:53 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/687
- IRFGEN stripmode removed from xpsimfmt, in order to avoid any interaction with the ixpeirfgen package.

ixpeobssim (30.1.0) - Mon, 23 Jan 2023 17:27:34 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/686
- New command-line switches added to xpsimfmt to create smaller output files, facilitating the photon-list and IRF generation code paths.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/685

ixpeobssim (30.0.1) - Wed, 11 Jan 2023 16:23:48 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/682
- *from_future_import_annotations* added to support the dataclasses in the evt/display with Python versions prior to 3.9
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/681

ixpeobssim (30.0.0) - Sat, 17 Dec 2022 07:18:16 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/680
- New v12 response files shipped, and now default for simulation and analysis.
- Docs updated.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/669

ixpeobssim (29.7.1) - Fri, 16 Dec 2022 14:57:10 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/679
- xpsimfmt.py modified to support (again) the proper generation of reponse files.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/678

ixpeobssim (29.7.0) - Sun, 11 Dec 2022 10:12:24 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/677
- New xpobsdisplay to create animated displays with track images and cumulative histograms out of Level-1 and Level-2 data files.
- Single event display revamped.

ixpeobssim (29.6.0) - Thu, 01 Dec 2022 15:05:28 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/673
- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/674

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/675
- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/676
- Several new command-line switches added to xpdisplay.
- Diagnostic events properly handled in the event display.
- New xLorentzian model in ixpeobssim.core.modeling
- New xpradialprofile.py application to help gauging the level of background in a given observation.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/664
  - https://github.com/lucabaldini/ixpeobssim/issues/665
  - https://github.com/lucabaldini/ixpeobssim/issues/667

ixpeobssim (29.5.0) - Thu, 24 Nov 2022 13:20:07 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/666
- Constants related to the GPD physical size and default fiducial size fully revamped, passing from a square to a rectangle wherever appropriate.
- Possibility for a radial dependence added to the instrumental background classes.
- Large refactoring and cleanup.
- Documentation updated.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/668
  - https://github.com/lucabaldini/ixpeobssim/issues/663

ixpeobssim (29.4.0) - Thu, 17 Nov 2022 11:50:54 +0100

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/657
- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/659
- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/660
- Fix in the error propagation in the histogram sum
- Better handling of the parameter bounds when summing fitting models.
- New interface to level-1 files and single-event display.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/639
  - https://github.com/lucabaldini/ixpeobssim/issues/656
  - https://github.com/lucabaldini/ixpeobssim/issues/658

ixpeobssim (29.3.1) - Thu, 06 Oct 2022 22:12:32 +0200

- Minor changes to the README file.

ixpeobssim (29.3.0) - Thu, 06 Oct 2022 22:08:22 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/645
- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/648

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/649
- Fix for GTI in light curves.
- Fix grids behavior when setup_gca() is called multiple times on the same figure.
- Facilities for the display of the as-run target list largely refactored.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/647

ixpeobssim (29.2.1) - Wed, 28 Sep 2022 09:31:59 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/644
- REAMDE.md revamped.
- setup.py updated to include the package description on PyPI.
- Installation instructions largely revised.

ixpeobssim (29.2.0) - Tue, 27 Sep 2022 19:25:48 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/641
- Github actions added for the upload on PyPI and the CI.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/624
  - https://github.com/lucabaldini/ixpeobssim/issues/625

ixpeobssim (29.1.4) - Tue, 27 Sep 2022 11:45:36 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/641
- Small fix for a matplotlib 3.6.0 incompatibility.
- Switching to draggable colorbars for two-dimensional histograms.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/631

ixpeobssim (29.1.3) - Sat, 24 Sep 2022 21:21:14 +0200

- Some cosmetics on the public guthub interface.

ixpeobssim (29.1.2) - Sat, 24 Sep 2022 19:09:22 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/633
- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/636
- A few bits of the documentation on RTD fixed, and all references to the old, private repository on bitbucket fixed.
- Versioning infrastructure refactored to allow for installation via pip.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/620
  - https://github.com/lucabaldini/ixpeobssim/issues/632

ixpeobssim (29.1.1) - Thu, 22 Sep 2022 20:17:42 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/630
- Data file required by a magnetar unit test restored.

- Path to a CALDB file fixed in order to be able to run xpobssim with the charging option enabled.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/619
  - https://github.com/lucabaldini/ixpeobssim/issues/629

ixpeobssim (29.1.0) - Wed, 21 Sep 2022 21:40:16 +0200

- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/1
- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/2
- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/3
- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/4
- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/623
- Merging in pull request https://github.com/lucabaldini/ixpeobssim/pull/628
- Autogenerated XSPEC files removed from the repository (this was simply a leftover from a previous installation of the private repo.)
- Links to the old bitbucket issues redirected to the new github ones.
- As run target list updated.
- Minor fixes to the docs.
- Fix for a bug in the chandra-to-IXPE photon-list workflow (issue 618).
- Small fix for a bug in the photon-list creation for periodic sources (issue 621).
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/618
  - https://github.com/lucabaldini/ixpeobssim/issues/621
  - https://github.com/lucabaldini/ixpeobssim/issues/622

ixpeobssim (29.0.0) - Thu, 08 Sep 2022 11:19:50 +0200

- First, glorious version for public release.

ixpeobssim (28.4.0) - Sat, 27 Aug 2022 10:59:55 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/271
- New linearized error propagation in the polarization cube subtraction.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/614

ixpeobssim (28.3.0) - Fri, 26 Aug 2022 12:43:50 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/270
- Structure refactored to facilitate splitting the IRF generation code out of ixpeobssim.
- New irf.ebounds module added, with the energy grid definition.
- irfgen import removed from the **argparse_** module.
- GPD filling temperature and pressure moved to irfgen.gpd
- Names and path for the MMA effective area and vignetting files moved to irfgen.mma

- UV filter file naming moved to irfgen.du
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/612

ixpeobssim (28.2.0) - Wed, 24 Aug 2022 07:10:03 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/269
- Creation of weighted polarization cubes, maps and map cubes with acceptance correction disengaged until we have proper arf files with the *SIMPLE* weighting scheme.
- New weighting_scheme() hook, defaulting to None, added to the xEffectiveArea class.
- xStokesAnalysis constructor signature changed, in preparation of the addition of the energy flux to the polarization cubes.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/613

ixpeobssim (28.1.0) - Tue, 23 Aug 2022 18:37:19 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/268
- New infrastructure in place for generating and using 2d (as in "non azimuthally symmetric") PSF.

ixpeobssim (28.0.1) - Fri, 19 Aug 2022 07:58:59 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/267
- Docs entry page revamped.
- As-run target list updated.

ixpeobssim (28.0.0) - Thu, 28 Jul 2022 08:20:14 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/266
- Massive refactoring of the IRF files, updated to the latest structure and naming conventions.
- Obsolete response files removed altogether.
- Code resolving the paths for the response files largely simplified.
- caldb folder ixpe/mma moved to ixpe/xrt/bcf to match the real CALDB.
- Unit tests updated.
- IRF docs completely revised.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/564

ixpeobssim (27.0.0) - Thu, 21 Jul 2022 16:31:29 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/263
- `rotate` keyword argument removed from all the functions in srcmodel.polarization (from now on, when passing input maps in angle or PD/PA components, it is assumed that the angle is measured from the celestial North).
- xy mode for reading in polarization maps and aligning Stokes parameters removed altogether—we're still accepting PD/PA, but we should really encourage people to work in normalized Q/U space for extended sources.
- Origin of coordinates for measuring the position angle now correctly set to the celestial North, for both the visualization and the Stokes alignment.

- New data structures in srcmodel.polarization for radial and tangential polarization fields with arbitrary radial profiles.
- New toy_radial_disk and toy_tangential_disk examples, illustrating the new functionality.
- `casa` example renamed to ``toy_casa``, and fully revamped.
- Clocking direction of the DUs fixed.
- Additional 90 degree rotations added in the photon-list generation and in xpsimfmt to fix the orientation of the polarization patterns in the e2e workflow.
- Unit tests added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/349
  - https://github.com/lucabaldini/ixpeobssim/issues/596
  - https://github.com/lucabaldini/ixpeobssim/issues/597

ixpeobssim (26.6.1) - Thu, 21 Jul 2022 07:11:18 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/264
- Small additions for the final version of the Software X paper.

ixpeobssim (26.6.0) - Wed, 13 Jul 2022 12:37:11 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/257
- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/259
- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/261
- Photon list mechanism implemented for the xChandraObservation model component class.
- Vignetting now correctly applied in the photon list workflow.
- `DETX` and `DETY` columns added in the xpsimfmt output files.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/593
  - https://github.com/lucabaldini/ixpeobssim/issues/606
  - https://github.com/lucabaldini/ixpeobssim/issues/607

ixpeobssim (26.5.0) - Wed, 13 Jul 2022 11:28:26 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/261
- Script to display the as-run target list added.
- New configuration file for an unpolarized point source.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/600

ixpeobssim (26.4.0) - Wed, 13 Jul 2022 11:22:07 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/260
- Brute force workaround for a regression introduced in numpy 1.22.0
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/608

ixpeobssim (26.3.3) - Mon, 30 May 2022 12:07:45 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/256
- Small fix in the polarization cube subtraction and multiplication (credits: Lawrence P.)

ixpeobssim (26.3.2) - Wed, 18 May 2022 15:40:05 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/255
- Minor fix to the zlabel for binned count maps.

ixpeobssim (26.3.1) - Wed, 18 May 2022 13:59:39 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/254
- Bug fix in xpphase for ephemeris referred to times before the start of the observation.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/601

ixpeobssim (26.3.0) - Wed, 18 May 2022 13:29:57 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/252
- New ixpeobssim.event.evt.xEventFileFriend class added to handle synchronized pairs of level-1 and level-2 data.
- xpselect generalized to accept a binary selection mask.

ixpeobssim (26.2.0) - Wed, 18 May 2022 12:27:15 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/250
- Polarization map cubes equipped with a general-purpose convolution routine and a plotting hook for the significance.
- Circular sum kernel facility added.
- Alignment and radial profile of polarization maps for polarization map cubes.

ixpeobssim (26.1.1) - Thu, 12 May 2022 17:28:16 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/253
- Fix for issue 599
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/599

ixpeobssim (26.1.0) - Sat, 07 May 2022 21:07:09 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/249
- (This is an intermediate release with the specific purpose of keeping track of the analysis for the magnetar discovery paper, and we are not closing any of the related issues, just yet.)
- Added fiducial backscal value.
- Inhibit circle/annuli and ds9 region file selections, and writing the BACKSCAL header keyword for the first.
- Propagating the BACKSCAL value to the binned polarization cubes.
- Initial implementation of the PCUBE subtraction.
- Script for the 4u analysis for the Science paper.

ixpeobssim (26.0.1) - Mon, 02 May 2022 16:06:51 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/248/

- Small test script added.

ixpeobssim (26.0.0) - Mon, 02 May 2022 15:45:08 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/247/
- Default IRF name bumped to ixpe:obssim:v11
- PSF parametrization changed: maximum radius pushed out to 480 arcseconds, manual scale factors removed, and parametrizations set for the three MMAs separately.
- Docs updated.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/158
  - https://github.com/lucabaldini/ixpeobssim/issues/580
  - https://github.com/lucabaldini/ixpeobssim/issues/428

ixpeobssim (25.8.0) - Thu, 28 Apr 2022 09:25:27 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/246
- Deterministic implementation of xppicorr.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/595

ixpeobssim (25.7.0) - Fri, 22 Apr 2022 19:15:16 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/245
- Supporting regions version 0.6.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/589

ixpeobssim (25.6.5) - Fri, 01 Apr 2022 12:14:21 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/244
- Specific target for testing the local installation added to the Makefile.
- Minor bug fix.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/590

ixpeobssim (25.6.4) - Fri, 01 Apr 2022 12:10:16 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/242
- Rendering of the LTP recast in terms of the TWGs.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/578

ixpeobssim (25.6.3) - Wed, 09 Mar 2022 15:36:06 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/243
- Removing all time-dependent PI correction files, as they now live in a separate repository: https://bitbucket.org/ixpesw/pi_corr_caldb/
- **Issue(s) closed:**

  - https://github.com/lucabaldini/ixpeobssim/issues/579

ixpeobssim (25.6.2) - Mon, 07 Mar 2022 15:09:52 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/241
- Method to calculate the energy spectrum added to the magnetar model interface.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/388

ixpeobssim (25.6.1) - Tue, 01 Mar 2022 10:09:07 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/240
- Minor changes.

ixpeobssim (25.6.0) - Mon, 28 Feb 2022 14:43:32 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/239
- New xpstokesrandom and xpstokesshuffle implemented, and added to the docs and wrapped in the pipeline.
- Unit tests added.

ixpeobssim (25.5.0) - Mon, 28 Feb 2022 14:13:39 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/238
- New plotting style for the polarization cubes (the API should be considered experimental and might evolve as we learn to use the new features).
- New features in the **ixpeobssim.utils.matplotlib_** module to support the new plotting of polarization cubes.

ixpeobssim (25.4.0) - Mon, 28 Feb 2022 13:59:49 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/237
- Proper calculation of the detection significance added to polarization cubes, maps and map cubes.
- Additional fields P_VALUE and CONFID added to the polarization cubes, maps and map cubes.
- A few fix for zero-division errors.
- Small fix in summing the values of N_EFF and FRAC_W across polarization cubes, maps and map cubes.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/467

ixpeobssim (25.3.4) - Fri, 25 Feb 2022 14:15:43 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/236
- New PI correction files for the ~complete Cas A observation 01001301

ixpeobssim (25.3.3) - Fri, 25 Feb 2022 09:36:45 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/235
- Fix for issue #574.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/574

ixpeobssim (25.3.2) - Thu, 24 Feb 2022 06:20:16 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/234

- (Yet another) fix for bug #567.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/567

ixpeobssim (25.3.1) - Wed, 23 Feb 2022 14:42:40 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/233
- De-correction for the GPD quantum efficiency applying when simulating photon lists for the instrumental background—see, however all the caveats in the relevant issue.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/517

ixpeobssim (25.3.0) - Wed, 23 Feb 2022 14:04:03 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/231
- Full refactoring of the code dealing with the response files.
- All format specifications moved to ixpeobssim.irfgen.fmt
- New xSpecRespBase class added, acting as a base class for the effective area, modulation factor and modulation response function, and equipped to use the SYS_MIN and SYS_MAX columns, when available.
- Vignetting factored out of the effective area class.
- pylinted.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/568
  - https://github.com/lucabaldini/ixpeobssim/issues/313

ixpeobssim (25.2.1) - Tue, 22 Feb 2022 16:15:04 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/232
- Bug fix for issue #567
- Phase and time grids in the photon list now driven by the proper class members.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/567

ixpeobssim (25.2.0) - Fri, 18 Feb 2022 11:35:38 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/230
- xppiscale.py renamed as xppicorr.py and generalized to global scale and offset corrections, as well as generic time-dependent corrections driven from a FITS file.
- Initial PI correction for the first chunck of Cas A observation (v02) added.
- Unit tests added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/568

ixpeobssim (25.1.0) - Fri, 18 Feb 2022 10:36:58 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/229
- New facilities related to the exposure calculation.

- Livetime cube binning algorithm (LTCUBE) added—it saves a ltcube with information on the elapsed time in each theta bin in each pixel of the map.
- LTCUBE supported in xpbinview
- New xpancrkey and xpexposure apps added.
- New toy_offaxis configuration file and associated analysis pipeline illustrating the new exposure functionality.
- IN_SAA and TARGET_OCCULT columns in the SC_DATA extensions now driven by whether we are taking data—they are identically zero if the –saa and/or the –occult flags are set to False from command line.
- Bug fix in the __iadd__() slot for xBinnedAreaRateMap objects.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/433
  - https://github.com/lucabaldini/ixpeobssim/issues/572
  - https://github.com/lucabaldini/ixpeobssim/issues/562

ixpeobssim (25.0.0) - Thu, 17 Feb 2022 13:46:16 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/223
- Adding support for weights in binned MDP and polarization maps.
- Added –acceptcorr option in xpbin for polarization data products.
- Small tweak to the binning functions to accept tuples in addition to lists.
- Fix for issue #443.
- Modulation cubes and associated classes removed.
- Major change to the format of the polarization cubes and the MDP and polarization maps and map cubes to keep track of all the necessary figures for a correct calculation of the MPD, as well as for holding errors on the Stokes parameters and the significance of a polarization measurement.
- Supporting errors on Stokes parameters in polarization cubes and maps.
- Bug fix in xpevtstat.py
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/443
  - https://github.com/lucabaldini/ixpeobssim/issues/540
  - https://github.com/lucabaldini/ixpeobssim/issues/565
  - https://github.com/lucabaldini/ixpeobssim/issues/566
  - https://github.com/lucabaldini/ixpeobssim/issues/444

ixpeobssim (24.0.0) - Tue, 08 Feb 2022 15:15:19 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/227
- Massive cleanup of the ixpeobssim.evt.event module, with parts moved out to the new ixpeobssim.evt.fmt and ixpeobssim.evt.gti modules, and a few obsolete interfaces, such as _radec_to_xy_int(), removed.
- Major cleanup of the xEventList class, with obsolete interfaces removed, and WCS information used consistently throughout.
- All WCS-related header keywords are now consistently set through the proper keyword arguments of the fits.Column objects, as opposed to manually hacking the header itself.

- WCS information added to the output xpsimfmt files, that should be now properly displayed in ds9.
- Comprehensive revision of the binary table headers for simulated files.
- ixpeobssim.evt.ixpesim streamlined.
- build_wcs() signature changed for consistency.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/518
  - https://github.com/lucabaldini/ixpeobssim/issues/523
  - https://github.com/lucabaldini/ixpeobssim/issues/526
  - https://github.com/lucabaldini/ixpeobssim/issues/538
  - https://github.com/lucabaldini/ixpeobssim/issues/548
  - https://github.com/lucabaldini/ixpeobssim/issues/550
  - https://github.com/lucabaldini/ixpeobssim/issues/552

ixpeobssim (23.8.1) - Tue, 08 Feb 2022 11:32:43 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/228
- Documentation on the binned data products fully revamped.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/478
  - https://github.com/lucabaldini/ixpeobssim/issues/551

ixpeobssim (23.8.0) - Tue, 08 Feb 2022 10:15:04 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/226
- Bug fix for the phase folding returning values outside the interval [0, 1]
- Avoid applying the vignetting twice in the Chandra-to-IXPE converter.
- Modified Chandra-to-IXPE workflow using the exposure.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/131
  - https://github.com/lucabaldini/ixpeobssim/issues/488

ixpeobssim (23.7.0) - Fri, 04 Feb 2022 16:09:45 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/225
- Added granular invert command line switches to xpselect; this allow to invert (i.e., take the logical not) of any of the selections applied.
- Small bug fix.
- Unit tests added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/549

ixpeobssim (23.6.1) - Thu, 03 Feb 2022 10:26:54 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/224

- Minor change to toy_pollin to match the polling definition in XSPEC.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/547

ixpeobssim (23.6.0) - Wed, 02 Feb 2022 17:10:33 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/221
- energy_spectrum changed to photon_spectrum throughout.
- Index for the EXB changed.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/544

ixpeobssim (23.5.0) - Wed, 02 Feb 2022 16:56:24 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/222
- Small refactoring in the binning routines, with all the I/O dictionaries moved into the binning module, so that they can be effectively used in the apps.
- Mechanism for building the path to the output file fixed for consistency with the other apps in xpphase.py and xpophase.py
- New set_column() class member added to evt.event.xEventFile.
- New app xppiscale.py added, and included in the pipeline facilities.
- New toy_ms_pulsar configuration file and associated pipeline added.
- Small tweak to the header keywords in xpsimfmt.py.
- xpphotonlist added to the pipeline facilities.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/439

ixpeobssim (23.4.0) - Wed, 02 Feb 2022 08:34:21 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/219
- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/220
- Bug fix for filtering mismatch in weighted polarization analysis.
- Minor tweaks to the rendering of bivariate splines.
- Added a small macro with preliminary plots for the ixpeobssim paper.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/541

ixpeobssim (23.3.0) - Fri, 28 Jan 2022 15:03:30 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/218
- xStokesAnalysis modified to filter out malformed events and events outside the 0–15 keV energy where the response functions can be sensibly extrapolated.
- Improved diagnostics in xStokesAnalysis.
- More sensible error message from xpbin.py when the input file is not found.
- **Issue(s) closed:**

    - https://github.com/lucabaldini/ixpeobssim/issues/539

ixpeobssim (23.2.1) - Wed, 26 Jan 2022 18:49:36 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/217
- Improved errore messages when failing consistency check in summing binned products.
- Using TSTART and TSTOP as default values for LC binning bounds (as opposed to the extremes of the GTIs, which are generally different for the three detectors in the same observation).
- **Issue(s) closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/537

ixpeobssim (23.2.0) - Wed, 26 Jan 2022 17:04:18 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/216
- New background PHA1 files created from the first observation of SMC X-1, and code added to perform the proper scaling to create usable templates.
- New xTemplateInstrumentalBkg class for generating template-driven background spectra.
- Added (and enabled by default) an option to prevent the convolution with the instrumental background spectrum with the energy dispersion, and modified the handling of the energy bounds for the simulation in the two cases.
- Docs added.
- Sample configuration file (instrumental_bkg_smcx1) added to illustrate the new functionality.
- Realistic instrumental background added to the Cas A configuration file.
- **Issue(s) closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/535

ixpeobssim (23.1.0) - Wed, 26 Jan 2022 10:30:25 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/215
- ixpeobssim.evt.subselect refactored to interoperate with filtered, level-2 data.
- Livetime correction disabled by default in xpselect.
- Docs and unit tests updated.
- **Issue(s) closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/536

ixpeobssim (23.0.1) - Tue, 25 Jan 2022 18:49:28 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/214
- LTP updated.
- Last update label added.
- **Issue(s) closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/533

ixpeobssim (23.0.0) - Mon, 24 Jan 2022 13:34:04 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/213
- This is the first release capable of operating on flight data, and most of the changes originate from the very first experience with the Cas A data.

- Using OBJ_RA and OBJ_DEC (rather than PNT_RA and PNT_DEC, that are not present in the level-2 file primary header) as a default value for centering WCS objects.
- Kislat analysis re-cast in Q and U, as opposed to PHI.
- EXPOSURE keyword removed from the event lists, and added at the xpbin.py level to allow for fitting in XSPEC.
- Pixel grid definition for X and Y changed from 900 x 900 pixels at 2 arcsec steps to 600 x 600 pixels at 2.6 arcsec steps.
- Physical energy in keV retrieved via the PI column in event files.
- RA and DEC coordinates retrieved via X and Y in event files.
- xpselect refactoring to handle with the fact that the LIVETIME columns is not included in filtered level-2 event lists.
- Minor changes.
- Docs updated.
- Unit tests added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/531
  - https://github.com/lucabaldini/ixpeobssim/issues/528
  - https://github.com/lucabaldini/ixpeobssim/issues/532
  - https://github.com/lucabaldini/ixpeobssim/issues/530
  - https://github.com/lucabaldini/ixpeobssim/issues/529

ixpeobssim (22.0.0) - Sun, 23 Jan 2022 09:21:26 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/210
- Full refactoring of ixpeobssim.evt.subselect in order to correctly propagate the livetime through the time and phase selections.
- xEventFile.average_deadtime_per_event() hook added.
- –phimin, –phimax and –invert options removed from xpselect.
- Added livetime-correction options to xpselect.
- Headers properly updated in xpselect.
- Time-related keywords added to the MONTE_CARLO extension.
- Infrastructure to control the count spectrum spline (ny, kx and ky) in place.
- New xStepFunction class added.
- New livetime examples (with selection in time and phase) revised.
- Documentation section about xpselect added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/378
  - https://github.com/lucabaldini/ixpeobssim/issues/55
  - https://github.com/lucabaldini/ixpeobssim/issues/514
  - https://github.com/lucabaldini/ixpeobssim/issues/169

ixpeobssim (21.4.0) - Sun, 23 Jan 2022 07:46:55 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/212
- New xpstokessmear application added to test the effect of the spurious modulation correction via a gaussian smearing of the Stokes parameters.
- New xpaddmofweights application added to process a level-2 file adding a new column with weights based on the modulation factor as a function of the energy.
- Output support enhanced in the xEventList class, via the addition of the add_column(), add_columns(), remove_columns() and write() methods.
- General binary search method to locate bin and bin values in multi-dimensional histograms added.
- xpstokessmear and xpaddmodfweights added to the reference docs.
- xpaddmofweights and xpstokessmear added to the pipeline.
- Minor refactoring of the basic app structure.
- Unit tests added.
- Copyright notice updated.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/512
  - https://github.com/lucabaldini/ixpeobssim/issues/521

ixpeobssim (21.3.3) - Thu, 20 Jan 2022 22:02:12 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/211
- Minor doc update.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/506

ixpeobssim (21.3.2) - Wed, 19 Jan 2022 20:39:07 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/209
- Bug fix for having xpsimfmt inter-operate with event files with no MONTE_CARLO extension.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/515

ixpeobssim (21.3.1) - Tue, 18 Jan 2022 05:53:58 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/208
- Emergency patch for issue #513 (photon-list mechanism not working with instrumental background).
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/513

ixpeobssim (21.3.0) - Mon, 17 Jan 2022 18:39:51 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/207
- Sorting the photon list before writing them out to FITS, fixing a fairly serious flaw in the mechanism.
- Roll angle added to the SC_DATA extension.
- SC_DATA extension added to the photon lists.

- Adding RA, DEC, X and Y to the xpsimfmt output file.
- Properly handling dithering, vignetting and fiducial cut in the photon lists.
- Using scdata=False for the test_instrumental_background test.
- pointing_ra/dec changed to ra/dec_pnt throughout.
- Polarization angle in the photon lists rotated to the GPD reference frame, and inverse transformation implemented in xpsimfmt
- DET_ID overwritten by xpsimfmt
- Added option to use MC/reconstructed absorption point in xpsimfmt.
- Documentation updated and unit test added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/494
  - https://github.com/lucabaldini/ixpeobssim/issues/498
  - https://github.com/lucabaldini/ixpeobssim/issues/500

ixpeobssim (21.2.0) - Sat, 15 Jan 2022 10:48:30 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/203
- Implemented dithering directly to the pointing direction, so that it gets propagated to the SC_DATA binary table.
- Vignetting now correctly applied.
- Moved parse_dithering_kwargs() method to the ixpeobssim.instrument.mma module.
- Command-line options refactored.
- Added facility to recover the pointing direction from the SC_DATA table.
- LAUNCH_DATE and LAUNCH_MET added in the **time_** module.
- Added unit tests.
- Added documentation.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/431
  - https://github.com/lucabaldini/ixpeobssim/issues/432

ixpeobssim (21.1.2) - Thu, 13 Jan 2022 16:34:48 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/206
- Bugged commit https://bitbucket.org/ixpesw/ixpeobssim/commits/264dad9b5b1549ec83d9a2dfb874491ee3901045 reverted.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/487
  - https://github.com/lucabaldini/ixpeobssim/issues/497

ixpeobssim (21.1.1) - Thu, 13 Jan 2022 15:19:10 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/205
- XSPEC version parsing removed.
- Added unit test.

- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/507

ixpeobssim (21.1.0) - Thu, 13 Jan 2022 14:59:08 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/204
- Default suffix for xpstokesalign changed from '_phialign' to '_stokesalign'
- Preventing xpstokesalign from changing the DEPHI column, if present in the input event list.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/504
  - https://github.com/lucabaldini/ixpeobssim/issues/505

ixpeobssim (21.0.1) - Tue, 11 Jan 2022 20:55:54 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/201
- Emergency fix for the setuptools files after the refactoring of the CALDB.
- ixpeobssim/srcmodel/par_files moved to ixpeobssim/srcmodel/parfiles.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/499

ixpeobssim (21.0.0) - Mon, 10 Jan 2022 17:49:05 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/201
- Full re-organization of the pseudo CALDB to match the structure of the real CALDB.
- IRF-name separator changed from "_" to ":" to allow CALDB-like file names and properly support weights.
- File name conventions for the IRF files aligned with the real CALDB starting from version 10.
- New keywords (and checksum) added to all the response files.
- COMMENT fields pertaining to the version and weight for response functions removed from all the headers, since this information is now tracked in proper keywords.
- IRF documentation fully revamped.
- Default IRF name set to "ixpe:obssim:v10".
- Modulation response function added to the xIrfSet class.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/496
  - https://github.com/lucabaldini/ixpeobssim/issues/462
  - https://github.com/lucabaldini/ixpeobssim/issues/468
  - https://github.com/lucabaldini/ixpeobssim/issues/479
  - https://github.com/lucabaldini/ixpeobssim/issues/492

ixpeobssim (20.2.0) - Fri, 07 Jan 2022 16:23:47 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/200
- Added facilities to display the observation plan.
- **Issue(s) closed:**

  - https://github.com/lucabaldini/ixpeobssim/issues/495

ixpeobssim (20.1.0) - Fri, 07 Jan 2022 16:16:00 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/199
- Polarization alignment according to an input model re-casted in terms of the Stokes parameters.
- xpphialign.py renamed as xpstokesalign.py
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/493

ixpeobssim (20.0.0) - Fri, 17 Dec 2021 16:02:54 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/198
- **All multiplicative polarization models renamed and in synch with the XSPEC repository:**
  - `constpol` is now `polconst`
  - `linpol` is now `pollin`
  - `powpol` is now `polpow`
  - `quadpol` model has bee removed.
  - (Note that the parameter names have been changed, as well)
- *Do not forget to cleanup and recompile the ixpeobssim local models!*
- All model names changed in the codebase.
- Documentation updated.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/489

ixpeobssim (19.7.0) - Fri, 17 Dec 2021 10:18:46 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/197
- IXPE TLE updated with the first post-launch values.
- Docs updated.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/490

ixpeobssim (19.6.1) - Fri, 17 Dec 2021 08:49:42 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/196
- Minor fixes to the documentation.

ixpeobssim (19.6.0) - Tue, 30 Nov 2021 10:10:23 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/195
- Initial implementation of the animation module.
- xpsonify improved to support animation.
- Docs updated.

ixpeobssim (19.5.3) - Tue, 30 Nov 2021 10:06:23 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/194

- Fix a runtime zero-division error in xpbin.py
- Fix a runtime error in core.fitsio.py
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/470
  - https://github.com/lucabaldini/ixpeobssim/issues/483

ixpeobssim (19.5.2) - Thu, 25 Nov 2021 18:17:54 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/193
- Docs tweaked.

ixpeobssim (19.5.1) - Wed, 24 Nov 2021 15:58:08 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/192
- Bug fix.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/482

ixpeobssim (19.5.0) - Wed, 24 Nov 2021 13:44:05 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/191
- New sonification module.
- New application to transform a photon list into an MIDI and/or audio file.
- Documentation updated.

ixpeobssim (19.4.1) - Mon, 22 Nov 2021 18:39:23 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/190
- Unit test added for issue #179 (invalid)
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/179

ixpeobssim (19.4.0) - Mon, 22 Nov 2021 16:26:09 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/189
- xInstrumentalBackground class refactored to support photon lists.
- Energy bounds for the instrumental background now correctly inferred from the simulation setup.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/481

ixpeobssim (19.3.3) - Mon, 22 Nov 2021 11:50:31 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/188
- Toy response functions removed.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/452

ixpeobssim (19.3.2) - Mon, 22 Nov 2021 11:29:01 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/187

- main() entry point added to xpphotonlist
- Unit test added to ensure that all the apps have appropriate entry points to run in user mode.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/480

ixpeobssim (19.3.1) - Mon, 22 Nov 2021 09:29:03 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/186
- A couple of typos fixed.
- Energy spectrum changed to photon spectrum throughout.
- Docs on binary systems added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/430
  - https://github.com/lucabaldini/ixpeobssim/issues/261
  - https://github.com/lucabaldini/ixpeobssim/issues/384
  - https://github.com/lucabaldini/ixpeobssim/issues/386

ixpeobssim (19.3.0) - Sun, 21 Nov 2021 20:37:10 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/185
- highecut_power_law spectral model added.
- xpchrgmap added to the pipeline, and corresponding command-line parser modified accordingly.
- Simplified observing plan simulation implemented.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/449
  - https://github.com/lucabaldini/ixpeobssim/issues/450

ixpeobssim (19.2.5) - Sat, 20 Nov 2021 17:21:56 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/184
- XSPEC headers moved to a separate file to facilitate supporting multiple XSPEC version.
- Collecting PyXspec and XSPEC version strings.
- Conditional compilation for the XSPEC headers, to support the new include layout in XSPEC version 12.12.0.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/472

ixpeobssim (19.2.2) - Sat, 20 Nov 2021 09:05:34 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/183
- Fix for malformed TLE in sgp4 version 2.20
- TLE epoch changed from January 1, 2021 to December 9, 2021.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/473

ixpeobssim (19.2.1) - Fri, 19 Nov 2021 11:19:28 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/182
- Minor fixes to the docs.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/476

ixpeobssim (19.2.0) - Fri, 19 Nov 2021 09:50:30 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/180
- xpchrgcorr.py removed in favor of the official tool available in gpdsw.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/475

ixpeobssim (19.1.0) - Fri, 19 Nov 2021 09:32:43 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/181
- CI Python version changed from 3.9.6 to 3.6.15.
- A couple of tweaks to support Python 3.6.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/477

ixpeobssim (19.0.0) - Thu, 18 Nov 2021 13:54:55 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/179
- First implementation of the photon list mechanism.
- xBaseEventList class added, and xEventList refactored to support the new xPhotonList.
- Keeping track of the primary header comments in the IRF files.
- xEventList.filled_array() method removed.
- Refactoring of the roi module.
- Docs updated.

ixpeobssim (18.1.1) - Thu, 18 Nov 2021 10:56:19 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/178
- main() entry point added for consistency to all the apps, see https://github.com/lucabaldini/ixpeobssim/issues/469

ixpeobssim (18.1.0) - Wed, 17 Nov 2021 14:14:14 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/176
- Bug fix in the flux normalization for the magnetar models.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/471

ixpeobssim (18.0.1) - Wed, 17 Nov 2021 13:22:41 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/176
- Bug fix in the xpbin.py pixsize command-line switch.
- –dpi option added to xpbinview.py

- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/454

ixpeobssim (18.0.0) - Wed, 13 Oct 2021 15:59:48 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/175
- W_MOM column added to the event lists (provisionally set to 1).
- DET_Q and DET_U changed into Q and U.
- X and Y changed from int to floats.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/415
  - https://github.com/lucabaldini/ixpeobssim/issues/424

ixpeobssim (17.3.0) - Fri, 08 Oct 2021 09:29:13 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/174
- New MMA effective area curves with a refined analysis of the MMA calibration data.
- IRF v9 created.
- Version number added to the IRF heders as a comment.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/460

ixpeobssim (17.2.0) - Thu, 07 Oct 2021 14:49:07 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/173
- xpbin.py generalized to support weights.
- xpsimfmt.py generalized to support weights and added to the pipeline, with some tweaks to allow for a full spectro-polarimetric fit in XSPEC.
- xpcustomirf.py generalized to support weights, and now generating a nominal vignetting function to allow the loading of the effective area.
- Bug fix in xpevtstat.py when running on input files with no MONTE_CARLO extension.
- v8 iteration of the response functions added (but not the default, yet). Note this is the version passed over to the SOC to start populating the CALDB, and includes the first set of response functions with weights.
- DET_ID added to the primary header of the IRF files.
- Keeping track of N_EFF and FRAC_W in the Stokes analysis a la Kislat.
- Major restructuring of the irfgen code to support the generation of response functions with weights.
- Small fix for the modified XSPEC errors.
- N_EFF and FRAC_W columns added in the polarization and modulation cubes.
- All references to the standard cut efficiency removed.
- Secular pressure values updated.
- Minor tweak to the `utils.argparse_` module.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/464

  - https://github.com/lucabaldini/ixpeobssim/issues/463
  - https://github.com/lucabaldini/ixpeobssim/issues/459

ixpeobssim (17.1.0) - Thu, 02 Sep 2021 12:49:35 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/172
- New folder `obsdata` added to the hierarchy to hold observation-specific files (e.g., charging maps)
- Added vanilla charging maps, with all the values set to zero, to be used in observations where the detectors are initially completely discharged (and to be used by default).
- Charging parameters now read from the proper file in the preudo-CALDB.
- Charging-specific command-line switches modified (note that chrgtsteps is now chrgtstep, and we're setting the width of the step, rather than the number of steps).

ixpeobssim (17.0.0) - Wed, 01 Sep 2021 15:06:39 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/171
- GTI calculation completely refactored.
- OCTI calculation revamped, now inhibiting the activation of the calibration sources in the SAA.
- New `TIMELINE`, `SC_DATA` and `OCTI` extensions added to the output files to keep track of the status of the instrument along the orbit.
- New xpobsview application added for a quick look of a given observation timeline.
- xpobssim command-line switched tweaked for consistency, and new switches to control the minimum duration and the padding of the GTIs ans the OCTIs added.
- Livetime-related keywords fixed when the on-orbit calibration sources are activated (issue 457).
- Docs updated.
- Data format documentation regenerated as part of the docs creation (issue #429).
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/429
  - https://github.com/lucabaldini/ixpeobssim/issues/409
  - https://github.com/lucabaldini/ixpeobssim/issues/417
  - https://github.com/lucabaldini/ixpeobssim/issues/425
  - https://github.com/lucabaldini/ixpeobssim/issues/457

ixpeobssim (16.17.0) - Tue, 24 Aug 2021 07:56:22 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/170
- Confidence interval calculation added in XSPEC fitting, and enabled by default.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/346

ixpeobssim (16.16.0) - Thu, 19 Aug 2021 10:09:20 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/169
- Models for all the data challenge 1 sources added.
- Source documentation added.

- New gaussian line spectral model.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/410

ixpeobssim (16.15.0) - Sat, 14 Aug 2021 19:14:23 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/168
- Added an optional 'side' argument to the bisect function in hist.py, matching the signature of numpy.searchsorted (default is 'left', as in numpy.searchsorted, so the change is backward-compatible).
- Small change in how events are assigned to the correct gain value by the gain() function in charging.py, to match the fact that the the self.__gain_data attribute now has the dimension of its time axis increased by one, matching exactly the time binning of the energy flux cube.
- Implemented the slow charging process—for now its parameters are hard-coded to zero, so that only the fast charging is actually active. We will fully enable the slow charging process when charging parameters will be taken from a CALDB file, see https://github.com/lucabaldini/ixpeobssim/issues/449/set-the-parameters-for-the-charging-model
- Writing the slow charging map to the CHRG_MAP extension.

ixpeobssim (16.14.0) - Sat, 14 Aug 2021 11:32:29 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/164
- Added an option to provide a list of input charging maps to xpobssim.py, along with the facilities to parse them.
- Modified the charging model in charging.py to accept the input charging map
- Added in charging.py two classes representing the PRIMARY and CHRG_MAP extension of the FITS charging map files
- Modified most of the functions in `utils.time_` to optionally accept a custom format (defaulting to DATE-TIME_FMT).
- Small app added for extracting charging maps from observation files and save them in a dedicated file.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/445

ixpeobssim (16.13.0) - Fri, 13 Aug 2021 21:00:50 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/167
- Major restructuring of the auxiliary infrastructure for the response functions, but no change in any of the standard applications.
- Enhanced support for ixpeobssim-ixpesim inter-operation.
- AUX_VERSION bumped to version 3.
- New xpsimfmt.py and xpcustomirfs.py applications added.
- PI calculation for allx data sets improved, and bookkeeping added.
- Window contaminants correction implemented in xpsimspec.py
- Docs updated.

ixpeobssim (16.12.1) - Mon, 09 Aug 2021 15:42:35 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/166
- A bunch of facilities related to spurious modulation added, with no change in any of the standard applications.

ixpeobssim (16.12.0) - Wed, 04 Aug 2021 19:21:09 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/165
- Bug fix in handling magnetar tabular models.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/453

ixpeobssim (16.11.0) - Thu, 15 Jul 2021 14:10:25 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/163
- xpphialign.py now changing phi, q and u consistently.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/441

ixpeobssim (16.10.2) - Wed, 23 Jun 2021 18:30:23 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/159
- Draggable colorbar class added.
- Option for non-linear colorscale added in xpbinview.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/427

ixpeobssim (16.10.1) - Wed, 23 Jun 2021 18:12:51 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/161
- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/162
- Fix for the generation of the magnetar model tables.
- Added docs for the argparse odd corner with negative number in engineering format.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/438

ixpeobssim (16.10.0) - Thu, 10 Jun 2021 11:51:51 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/157
- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/158
- RA_PNT and DEC_PNT keywords used for the default ROI center in xpbin, xpselect and xpphialign
- aux.py renamed to auxiliary.py to allow interoperability with Windows.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/426
  - https://github.com/lucabaldini/ixpeobssim/issues/434

ixpeobssim (16.9.1) - Wed, 09 Jun 2021 17:53:13 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/160
- Bug fix in how the model files were handle by the pipeline xpphialign wrapper.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/353

ixpeobssim (16.9.0) - Thu, 03 Jun 2021 18:26:51 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/154
- Critical bug fix affecting the vignetting application, xpselect.py and the Chandra to IXPE conversion (please update).
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/423

ixpeobssim (16.8.5) - Thu, 03 Jun 2021 17:50:50 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/155
- notebooks folder removed.
- Added a paragraph about the regions installation on the docs.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/418
  - https://github.com/lucabaldini/ixpeobssim/issues/405

ixpeobssim (16.8.4) - Mon, 31 May 2021 08:19:33 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/153
- Normalization factor for the Galactic X-ray background changed.
- Interface to the ROSAT PSPC response matrix added.
- Unit test added.

ixpeobssim (16.8.3) - Sat, 29 May 2021 10:17:24 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/152
- A series o minor tweaks to the data format, and in particular the header keywords.
- OBJECT, RA_PNT/RA_OBJ and DEC_PNT/DEC_OBJ keywords added, and xpobssim.py equipped with a new –objname command-line switch.
- DET_ID added for the physical identification of the detector units.
- Timing keywords updated in the GTI extension.
- APID, PKTTYPE and PKTSTYPE keyords removed.
- DAQ_VER keyword removed.
- CREAT_ID keyword removed, and version written into CREATOR.
- RUN_ID and STA_ID keywords removed.
- Header keyword comments capitalized.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/419
  - https://github.com/lucabaldini/ixpeobssim/issues/422

ixpeobssim (16.8.2) - Sat, 29 May 2021 09:37:58 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/151
- New tool xpstat.py added for a quick look at the counts for a various components in a photon list.
- Docs updated.

- Minor refactoring of the energy binning base routine.
- Figure generation inhibited in a few unit tests.

ixpeobssim (16.8.1) - Fri, 28 May 2021 14:16:29 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/150
- Minor tweaks to the IRF plotting.

ixpeobssim (16.8.0) - Fri, 28 May 2021 14:12:28 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/149
- Initial implementation of the classes for the Extra-Galactic and Galactic X-ray background.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/57

ixpeobssim (16.7.1) - Tue, 25 May 2021 20:17:28 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/148
- xpsimspec.py application added to create user spectrum files to be fed into ixpesim.
- Unit test for the energy redistribution added.

ixpeobssim (16.7.0) - Mon, 24 May 2021 20:32:51 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/147
- New version of the IRF (v7) generated (but not the dafault, yet) with a non-diagonal response matrix, no 80% cut and Monte Carlo based modulation factor.
- Full machinery for processing and post-processing auxiliary files informing the response functions.
- Script to generate response functions at an arbitrary pressure added.
- New xLogNormal, xGeneralizedGaussian and xHat models added to core.modeling
- New xInterpolatedPiecewiseUnivariateSpline class added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/402
  - https://github.com/lucabaldini/ixpeobssim/issues/154

ixpeobssim (16.6.1) - Tue, 18 May 2021 12:42:22 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/146
- Emergency patch for a regression in xpselect—this was a *MAJOR* breakage, if you have checked out version 16.0.0 please UPDATE IMMEDIATELY!
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/421

ixpeobssim (16.6.0) - Thu, 13 May 2021 15:04:26 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/145
- New tool xpstripmc.py to process ixpeobssim photon lists and creating a verbatim copy without the Monte Carlo information (i.e., the MONTE_CARLO and ROI_TABLE extensions).
- –irfname option added to xpbin.py to support the analysis of files with no Monte Carlo information.
- xEventFile class modified to support photon lists with no Monte Carlo information.

- xpselect.py modified to support photon lists with no Monte Carlo information.
- Docs updated and unit tests added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/398
  - https://github.com/lucabaldini/ixpeobssim/issues/148

ixpeobssim (16.5.0) - Tue, 13 Apr 2021 16:43:27 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/139
- Major refactoring of non-celestial sources, now split out the ixpeobssim.srcmodel.roi model into ixpeobssim.srcmodel.calibsrc
- First implementation of the FCW CalC source.
- Finalization of the event list refactored in its own method, automatically called right before the event list is written to file, to provide a unique and consistent interface for filling the ancillary columns.
- xpobssim modified to allow for interleaving celestial observations with FCW CalC calibration runs.
- New xpcalib.py app added to simulate calibration runs.
- Docs updates, unit tests added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/393
  - https://github.com/lucabaldini/ixpeobssim/issues/394
  - https://github.com/lucabaldini/ixpeobssim/issues/399

ixpeobssim (16.4.1) - Tue, 13 Apr 2021 12:26:32 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/144
- Proper auxfile setup for the g21 and vela examples.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/408

ixpeobssim (16.4.0) - Tue, 13 Apr 2021 11:18:46 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/141
- Dropping pyregion altogether in favor of the astropy affiliated package regions.
- Added sky filtering with astropy regions for sky coordinates.
- Docs, requirements and unit tests updated.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/381

ixpeobssim (16.3.0) - Fri, 09 Apr 2021 16:06:33 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/142
- Changing PHE_Q and PHE_U columns to DET_Q and DET_U.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/400

ixpeobssim (16.2.0) - Fri, 09 Apr 2021 16:03:39 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/143
- Configuration region files moved from ixpeobssim/config/fits to ixpeobssim/config/reg
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/233
  - https://github.com/lucabaldini/ixpeobssim/issues/406

ixpeobssim (16.1.1) - Thu, 08 Apr 2021 08:29:04 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/140
- Fixed offset-by-one bug in the charging calculation.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/404

ixpeobssim (16.1.0) - Tue, 06 Apr 2021 13:14:03 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/138
- Major refactoring of the code for generating response function, with lots of cleanup and complete removal of the toy response functions.
- Initial support for creating response functions from the full Monte Carlo simulation, with auxiliary files for the passive conversions and the energy dispersion, as well as the ROI size distribution.
- Major refactoring of the ixpeobssim.core.hist module, with a complete cleanup of the interfaces, support for errors on unweighted and weighted histogram and for data persistence in FITS format.
- KDE smoothing added to the histogram classes.

ixpeobssim (16.0.0) - Thu, 01 Apr 2021 15:32:20 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/136
- New iteration (v6) of the response function—first one informed by the MMA and e2e calibration, and last one using the 80% cut.
- Use the MMA effective area curves from the mirror calibrations.
- Use the post-BAC best estimates of the GPD asymptotic pressures.
- Adjust the focal length to the measured value.
- Change the binning for the response functions.
- Use the measured PSF HPD
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/333
  - https://github.com/lucabaldini/ixpeobssim/issues/334
  - https://github.com/lucabaldini/ixpeobssim/issues/335
  - https://github.com/lucabaldini/ixpeobssim/issues/336
  - https://github.com/lucabaldini/ixpeobssim/issues/369
  - https://github.com/lucabaldini/ixpeobssim/issues/387

ixpeobssim (15.1.0) - Wed, 31 Mar 2021 14:09:48 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/137

- Fix for the polarization degree negative values from the magnetar table models.
- Added support for magnetar models with QED off.

ixpeobssim (15.0.0) - Tue, 23 Mar 2021 19:42:56 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/135
- NUM_CLU and LIVETIME columns added to the EVENTS extension.
- FILE_LVL keayword added to the file headers.
- Old livetime correction based on the number of discarded events replaced with the sum of event livetimes.
- A few methods related to the livetime added to the event file read interface.
- Small refactoring to avoid multiple conversions from start_date to start_met.
- pyregion import protected.
- Support for pseudo-Lv1a output added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/392

ixpeobssim (14.2.1) - Tue, 23 Mar 2021 13:48:50 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/134
- Got rid of a few deprecation warnings from matplotlib 3.3
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/373

ixpeobssim (14.2.0) - Mon, 22 Mar 2021 14:16:47 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/133
- Bug fix in the charging model with empty temporal bins.
- Bug fix in the charging model with a missing transpose.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/389

ixpeobssim (14.1.0) - Mon, 22 Mar 2021 14:12:16 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/132
- Major refactoring of the xEphemeris class
- get_phase_func() removed
- phase_function() deprecated
- t0 argument to xpphase changed to met0
- xEphemeris.rvs() implemented, and unit test added.
- Ephemeris handling fixed in srcmodel.roi
- Periodic source examples cleaned up.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/52

ixpeobssim (14.0.0) - Tue, 16 Mar 2021 11:26:05 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/131
- Added facilities to parse and operate with the magnetar models provided by Roberto and Roberto, and obsolete parsing routines removed.
- Added machinery for ixpeobssim auxiliary files.
- Example axp_1rxs_j1708.py revamped using the new functionality.
- Docs and unit tests updated.
- Some unintended fallout from merging pull request 129 cleaned up, and higher terms in the sourcse ephemeris disengaged until issue #52 is fixed.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/361

ixpeobssim (13.1.0) - Fri, 12 Mar 2021 15:15:13 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/130
- New facility added for setting XSPEC model strings.

ixpeobssim (13.0.0) - Sat, 27 Feb 2021 10:18:53 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/129
- Added the xpophase tool and xptimetophase replaced with xpphase.

ixpeobssim (12.11.0) - Mon, 15 Feb 2021 17:04:03 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/128
- New xBinarySource class, and associated infrastructure and unit tests.
- New xpphase.py (replacing the old xptimetophase.py, now removed) and xpophase.py applications.
- New configuration file and associated pipeline toy_binary.py

ixpeobssim (12.10.0) - Tue, 13 Oct 2020 13:31:28 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/127
- Added facility to save polarization map arrows as ds9 region file.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/361

ixpeobssim (12.9.0) - Tue, 13 Oct 2020 09:20:09 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/125
- Added g21_bucciantini example and analysis pipeline.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/359

ixpeobssim (12.8.0) - Mon, 12 Oct 2020 17:41:23 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/125
- New srcmodel.tdelays module and associated unit tests.

ixpeobssim (12.7.0) - Sun, 04 Oct 2020 17:53:55 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/124
- Bug fix in xFITSImageBase.center(), now using the wcs conversions.

- New wcs facilities in utils.astro, and used in binning.
- Facilities to build intensity maps for arbitrary models in srcmodel.roi.
- Unit tests added.

ixpeobssim (12.6.0) - Fri, 02 Oct 2020 15:54:42 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/98
- xEphemeris class moved out of srcmodel.roi into the new module srcmodel.ephemeris
- A bunch of facilities for binary sources added to srcmodel.ephemeris
- inverse() method implemented in the univariate base class.
- mjd_to_met() function added in `utils.time_`
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/358

ixpeobssim (12.5.1) - Thu, 01 Oct 2020 15:41:21 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/123
- New implementation of the angular separation function.

ixpeobssim (12.5.0) - Thu, 01 Oct 2020 08:31:33 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/122
- Many improvements in xFITSImageBase plotting routines, courtesy of Niccolo Bucciantini.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/355

ixpeobssim (12.4.0) - Thu, 24 Sep 2020 06:57:27 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/121
- New toy_pwn and toy_rim source examples added to aid the development of tools for the study of extended sources.
- Small tweaks to the binning module, and more zero-division-error protections added.
- evt.deconvolution module added (unused).
- xUniformAnnulus class added in srcmodel.roi
- Unit tests added.

ixpeobssim (12.3.1) - Tue, 22 Sep 2020 10:24:26 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/120
- Bug fix in xFITSImageBase.sky_bounding_box(), courtesy of Niccolo Bucciantini.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/352

ixpeobssim (12.3.0) - Sat, 19 Sep 2020 14:20:24 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/119
- Binned products can now be manipulated and saved to file.
- **Issue(s) closed:**

  - https://github.com/lucabaldini/ixpeobssim/issues/345

ixpeobssim (12.2.0) - Sat, 19 Sep 2020 13:48:51 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/118
- Added the option to pass the tick marks labels on the colorbar of the xFITSImageBase class.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/351

ixpeobssim (12.1.0) - Wed, 16 Sep 2020 06:40:30 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/117
- Smoothing out some rough edges around the XSPEC local models.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/350

ixpeobssim (12.0.0) - Wed, 09 Sep 2020 15:02:09 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/116
- This is backward-incompatible release that incorporates significant changes and refactoring in several different areas.
- New IRF (version 5) released—incorporating a small change in the format of the modulation factor.
- Formalism in Kislat et al. (2015) now consistently implemented throughout.
- Major refactoring of the binned data structures: SCUBE algorithm removed, MCUBE algorithm deprecated, and several algorithms added (PHA1QN, PHA1N, PCUBE, MDPMAP, MDPMACUBE, PMAP, PMAPCUBE),
- Several small fixes in the binning routines.
- Improved support for XSPEC, including new models and purely polarimetric fits to the normalized Q/I and U/I Stokes parameters.
- Improved support for analysis and visualization of extended sources, including maps of MDP and normalized Stokes parameters.
- Small wrapper around GRPPHA added to the pipeline.
- Documentation updated.
- Documentation pdf target fixed.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/171
  - https://github.com/lucabaldini/ixpeobssim/issues/265
  - https://github.com/lucabaldini/ixpeobssim/issues/303
  - https://github.com/lucabaldini/ixpeobssim/issues/311
  - https://github.com/lucabaldini/ixpeobssim/issues/328
  - https://github.com/lucabaldini/ixpeobssim/issues/329
  - https://github.com/lucabaldini/ixpeobssim/issues/330
  - https://github.com/lucabaldini/ixpeobssim/issues/332
  - https://github.com/lucabaldini/ixpeobssim/issues/332

    - https://github.com/lucabaldini/ixpeobssim/issues/337
    - https://github.com/lucabaldini/ixpeobssim/issues/338
    - https://github.com/lucabaldini/ixpeobssim/issues/339
    - https://github.com/lucabaldini/ixpeobssim/issues/340
    - https://github.com/lucabaldini/ixpeobssim/issues/341
    - https://github.com/lucabaldini/ixpeobssim/issues/342
    - https://github.com/lucabaldini/ixpeobssim/issues/344

ixpeobssim (11.2.1) - Thu, 20 Aug 2020 15:48:08 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/115
- Tentative fix for issue #325.
- Ephemeris for toy_periodic_source changed to trigger possible folding problems.
- Unit test added.
- **Issue(s) closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/325

ixpeobssim (11.2.0) - Thu, 20 Aug 2020 13:32:09 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/113
- New benchmark infrastructure added, with two examples.
- New xBinnedCountSpectrumSet implemented to calculate the broadband polarization, and releated facilities added.
- Docs updated.
- Command-line switch to initialize the fit parameters added back to xpxspec.py
- Added protection against wrong number of input files to xpxspec.py
- "pha1*" pattern now supported in pipeline.file_list(), and examples modified.
- Unit test streamlined.
- Resolution removed from the Gaussian model stat box.

ixpeobssim (11.1.0) - Thu, 20 Aug 2020 07:39:34 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/114
- Write and read interfaces to MDP maps implemented.
- MDPMAP algorithm added to xpbin.py
- xpbinview.py now handling MPDPMAP binned files.
- Unit test added, and toy_disk example complemented.
- Docs updated.
- **Issue(s) closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/187

ixpeobssim (11.0.0) - Tue, 18 Aug 2020 12:05:40 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/112

- Major rework of the XSPEC local models shipped with ixpeobssim, with the linpol and quadpol additions, and all parameter names changed for consistency.
- toy_linpol.py example added.
- Added facility to load the XSPEC local models programmatically.
- xpxspec and xpxspec3 merged and largely streamlined, with all the examples modified accordingly.
- pha1* tweak added to pipeline_file_list(), docs updated and examples modified.
- Docs for XSPEC support largely revised.

ixpeobssim (10.5.1) - Wed, 12 Aug 2020 15:10:02 +0200

- Ops—release notes updated.

ixpeobssim (10.5.0) - Wed, 12 Aug 2020 15:08:12 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/111
- Cen A example cleaned up.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/310

ixpeobssim (10.4.0) - Tue, 11 Aug 2020 15:12:47 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/109
- Specific versions added to requirements.txt
- Documentation updated with more details about the dependencies.
- Unit test added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/290

ixpeobssim (10.3.0) - Tue, 11 Aug 2020 12:12:26 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/110
- Dropping the imp module in Python 3.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/277

ixpeobssim (10.2.0) - Tue, 11 Aug 2020 12:05:50 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/107
- Support for scaling added in the xMDPRecord and xMDPTable classes.
- Broadband values for MDP tables now calculated dinamically at run time (this makes the bookkeeping much easier).
- eef and deadtime corrections added to xpmdp and xppimms.
- Added command-line switch to select the source in the ROI for xpmdp.
- –sourceID changed to –srcid throughout.
- xpmdp and xppimms fully refactored.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/314

  - https://github.com/lucabaldini/ixpeobssim/issues/312
  - https://github.com/lucabaldini/ixpeobssim/issues/164

ixpeobssim (10.1.0) - Tue, 11 Aug 2020 07:39:15 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/108
- Added irgen.mktab.py facility to dump all the ingredients for the MDP calculation in tabular format (support for csv and xlsx)

ixpeobssim (10.0.0) - Mon, 10 Aug 2020 16:36:32 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/106
- New version (v4) of the response function released.
- Generic asymptotic pressure for each of the DU now supported at the IRF generation stage—this includes the GPD quantum efficiency, the modulation factor and the passive conversion. Version 4 use 687 mbar for all the DUs, which is consistent with the sensitivity estimated for the Mission Integration Review.
- Combined and non-standard IRFs available in the previous iterations are now discontinued.
- xpppims and xpmdp modified to loop aver the three DUs, rather than using the combined IRFs.
- New DME density scaling (the same used in ixpesim) and pressure now measured in mbar (as opposed to atm) throughout.
- Small refactoring of the code handling the Be window contaminants, and certified list Be contaminants from manufacturer is now the default.
- Obsolete files removed and massive cleanup of the IRF documentation.
- pairwise() facility moved into the new module utils.misc.py
- Caching mechanism implemented for loading xcom data.
- A bunch of stuff factored out from irf.modf to evt.mdp.
- Weighted average facility added in `utils.math_`
- Unit tests added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/294
  - https://github.com/lucabaldini/ixpeobssim/issues/284
  - https://github.com/lucabaldini/ixpeobssim/issues/295
  - https://github.com/lucabaldini/ixpeobssim/issues/296
  - https://github.com/lucabaldini/ixpeobssim/issues/297
  - https://github.com/lucabaldini/ixpeobssim/issues/298
  - https://github.com/lucabaldini/ixpeobssim/issues/275
  - https://github.com/lucabaldini/ixpeobssim/issues/160

ixpeobssim (9.0.0) - Mon, 10 Aug 2020 16:27:25 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/103
- Dependence on aplply removed (now relying on astopy.visualizing), docs updated.
- xFITSImage class streamlined and refactored, wit all plotting functionalities moved into a base class in core.fitsio.

- Subtle bug fix (pixel offset by one) fixed in binned count maps, as well as xFITSImage random sampling—unit tests added.
- Mid-size rafactoring of the xStokesCube class, with arrow-related code moved out and rationalized.
- Stokes parameters set to zero outside the physical bounds of the underlying interpolator for Stokes sky maps and cubes.
- xpbinview added to the pipeline.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/272
  - https://github.com/lucabaldini/ixpeobssim/issues/281
  - https://github.com/lucabaldini/ixpeobssim/issues/166

ixpeobssim (8.8.3) - Fri, 07 Aug 2020 08:11:56 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/105
- Fix for skyfield 1.26.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/309

ixpeobssim (8.8.2) - Fri, 07 Aug 2020 07:40:53 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/104
- Fix for skyfield 1.26.

ixpeobssim (8.8.1) - Sat, 01 Aug 2020 15:20:02 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/102
- Major cleanup of the hist.py module.
- Unit tests improved.
- pytz dependence removed.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/293

ixpeobssim (8.8.0) - Sat, 01 Aug 2020 13:53:49 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/101
- Bug fix in xpselect when operating with ds9 region files.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/282
  - https://github.com/lucabaldini/ixpeobssim/issues/300

ixpeobssim (8.7.0) - Wed, 29 Jul 2020 14:15:29 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/99
- Major restructuring of the support for XSPEC spectral model at simulation time.
- Documentation added.
- Unit tests improved.
- **Issue(s) closed:**

  - https://github.com/lucabaldini/ixpeobssim/issues/301

ixpeobssim (8.6.2) - Sat, 25 Jul 2020 16:45:59 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/100
- Fixed a regression triggered by skyfield version 1.23+
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/302

ixpeobssim (8.6.1) - Wed, 03 Jun 2020 15:28:18 +0200

- Small facility added for packaging the docs in pdf and zipped html formats.

ixpeobssim (8.6.0) - Thu, 28 May 2020 21:25:08 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/95
- New dependence on skyfield added.
- Basic TLE interface for creating a proxy of the baseline IXPE object.
- Parametrization of the SAA and calculation of the SAA epochs added.
- Calculation of Earth occultation added.
- Calculation of the angles to the Sun and the Moon added.
- A few new facilities in the `utils.time_` module added to facilitate time conversions.
- Calculation of realistic good time intervals implemented, and all relevant pieces of code updated to reflect that (including xpobssim command-line options)
- Initial implementation of a simple visibility tool.
- startmet command-line switch changed to startdate throughout, and default observation start changed to 2022-04-21.
- startmet option removed from xpmdp and xppimms.
- Docs added.
- Unit tests added.
- A whola lotta improvements to the docs (branch revamp_docs merged).
- Bonus: fix in the time calculation for periodic sources—the start MET was previously ignored.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/232
  - https://github.com/lucabaldini/ixpeobssim/issues/288

ixpeobssim (8.5.0) - Wed, 20 May 2020 10:52:16 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/96
- Partial major rework of the documentation.

ixpeobssim (8.4.3) - Sun, 10 May 2020 08:22:24 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/94
- Fix in the documentation.

ixpeobssim (8.4.2) - Sat, 02 May 2020 07:59:17 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/93

- Confusing option in xpphialign fixed.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/285

ixpeobssim (8.4.1) - Sat, 02 May 2020 06:56:29 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/89
- Removed any reference to xPolarizationMap class in the tutorial section.
- Fixed a minor typo in source_models: chandra region of interest.
- Added some more details on the xStokesSkyMap class.
- Added in the print statement that the xppims and xpmdp are calculating the mdp at the 99 % CL.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/262
  - https://github.com/lucabaldini/ixpeobssim/issues/276

ixpeobssim (8.4.0) - Thu, 23 Apr 2020 09:29:36 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/91
- xInstrumentalBkg and xPowerLawInstrumentalBkg classes added in srcmode.roi
- New instrumental_bkg config file and example.
- New facilities for projecting GPD positions in the sky.
- Minor refactoring and cleanup.
- Documentation updated.
- Unit tests added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/264
  - https://github.com/lucabaldini/ixpeobssim/issues/278

ixpeobssim (8.3.1) - Wed, 22 Apr 2020 18:21:26 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/92
- New mechanism to parse package versions, now handling the PEP 440 specs.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/280

ixpeobssim (8.3.0) - Wed, 22 Apr 2020 16:13:38 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/90
- Allow changing the default 1–12 keV energy range for xpobssim, via the –emin and –emax command-line switches.
- Improved error message when the input spectral model is not positive-definite.
- GRS 1915+105 model completely refactored.
- Generic routine to parse energy-filtered tabular data implemented.
- New facilities to compute the integral flux and integral energy flux in srcmodel.spectrum

- Significant improvement in core.spline.scale()
- Documentation modified to reflect the changes.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/267
  - https://github.com/lucabaldini/ixpeobssim/issues/279

ixpeobssim (8.2.2) - Thu, 27 Feb 2020 11:28:33 +0100

- Extended the MAX_ENERGY value from 11 to 15 keV in the grs1915_105 config file.

ixpeobssim (8.2.1) - Mon, 10 Feb 2020 08:56:49 +0100

- Merging in pull requests https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/88
- XCOM cross sections for H and O added.

ixpeobssim (8.2.0) - Mon, 10 Feb 2020 08:51:55 +0100

- Merging in pull requests https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/87
- New infrastructure for the simulation of alpha particles.

ixpeobssim (8.1.3) - Thu, 30 Jan 2020 13:59:21 +0100

- Obsolete matplotlib parameters removed from the setup.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/270

ixpeobssim (8.1.2) - Thu, 30 Jan 2020 12:21:09 +0100

- Merging in pull requests https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/86
- Fix for CircleCI continuous integration.
- Minor fix to the code for parsing package version strings.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/263

ixpeobssim (8.1.1) - Thu, 30 Jan 2020 11:23:01 +0100

- Phony tag because I tagged 8.1.0 on a different branch.

ixpeobssim (8.1.0) - Thu, 30 Jan 2020 11:18:07 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/85
- Default value for the BACKFILE and CORRFILE in the header of the SPECTRUM extension of binned PHA1* files changed from None to ‘’
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/254

ixpeobssim (8.0.0) - Fri, 20 Dec 2019 15:40:37 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/84
- New round (v3) of response functions.
- New MMA effective area estimated from the post-CDR design
- Effect of all the contaminants in the Be window (as per the Materion data sheet) added.

- Photoelectons extracted from the window and the GEM now included in the effective area.
- Modulation factor updated based on the calibration of the DU FM2.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/259
  - https://github.com/lucabaldini/ixpeobssim/issues/258
  - https://github.com/lucabaldini/ixpeobssim/issues/257
  - https://github.com/lucabaldini/ixpeobssim/issues/200
  - https://github.com/lucabaldini/ixpeobssim/issues/256
  - https://github.com/lucabaldini/ixpeobssim/issues/255

ixpeobssim (7.1.0) - Thu, 31 Oct 2019 11:05:58 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/83
- Added facilities to evt.event to calculate the weighted average of the polarization degree for a given photon list.
- Added convenience functions to get the polarization models with the right signature (for ixpeobssim) from the Stokes sky-maps and cubes.

ixpeobssim (7.0.0) - Tue, 29 Oct 2019 09:58:38 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/81
- Bug fix in the function creating an event mask from a ds9 region file.
- Old `utils.filter_` module moved to utils.astro.
- xpselet now accepts arbitrary ds9 region files through the –regfile command-line switch (and the temporary regindex switch has been removed.)
- Added facility to retrieve the WCS information from an event file.
- MC_X and MC_Y columns added in MONTE_CARLO extension of the event file.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/181
  - https://github.com/lucabaldini/ixpeobssim/issues/234
  - https://github.com/lucabaldini/ixpeobssim/issues/240

ixpeobssim (6.5.1) - Mon, 28 Oct 2019 09:27:35 +0100

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/82
- Minor refactoring for the matplotlib color wheels.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/251
  - https://github.com/lucabaldini/ixpeobssim/issues/252

ixpeobssim (6.5.0) - Fri, 25 Oct 2019 06:39:40 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/80
- Fix in how we calculate the approximate radius of the WCS for Stokes sky maps.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/249

ixpeobssim (6.4.2) - Thu, 24 Oct 2019 15:36:34 +0200

- MSH 1552 example updated.

ixpeobssim (6.4.1) - Thu, 24 Oct 2019 12:13:26 +0200

- Bug fix in the logger.
- xpphialign added to the reference docs page.
- Small tweak to the top-level Makefile.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/250

ixpeobssim (6.4.0) - Thu, 24 Oct 2019 11:50:36 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/79
- **Layout of the repository Changed**
  - ixpeobssim/test -> tests
  - doc -> docs
  - sphinx documentation files refactored and cleaned up
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/195

ixpeobssim (6.3.0) - Thu, 24 Oct 2019 11:36:15 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/74
- xpphiradalign.py renamed to xpphialign and generalized to handle arbitrary polarization models.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/248

ixpeobssim (6.2.3) - Thu, 24 Oct 2019 05:58:50 +0200

- Some minor cleanup of old branches.

ixpeobssim (6.2.2) - Thu, 24 Oct 2019 05:48:06 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/76
- Unit test fixed.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/244

ixpeobssim (6.2.1) - Thu, 24 Oct 2019 05:10:29 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/74
- Small fix for sphinx 2x
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/246

ixpeobssim (6.2.0) - Wed, 23 Oct 2019 10:51:25 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/73
- Stokes sky cubes can now load layers from FITS map in the Q/U or polarization degree/angle space.

ixpeobssim (6.1.0) - Tue, 22 Oct 2019 14:01:04 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/71
- Bug fix in the energy interpolation for Stokes sky cubes.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/247

ixpeobssim (6.0.2) - Tue, 22 Oct 2019 11:34:58 +0200

- Updated release notes.

ixpeobssim (6.0.1) - Tue, 22 Oct 2019 11:31:48 +0200

- This time I tagged the master, but I forgot to update the release notes.

ixpeobssim (6.0.0) - Tue, 22 Oct 2019 11:27:14 +0200

(I tagged by mistake a development branch—sorry about that.)

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/71
- Major bug fix in xpphiradalign
- A few bug fixes and improvements in the new xStokesSkyMap and xStokesSkyCube classes, which should now be ready for prime time.
- Generic plotting support for polarization models in extended sources improved.
- Rotation option added when loading Stokes sky maps from polarization degree and angle.
- Cleanup of the config files for extended sources, and files renamed.
- Polarization map arrows now working on plain matplotlib figures.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/241
  - https://github.com/lucabaldini/ixpeobssim/issues/188
  - https://github.com/lucabaldini/ixpeobssim/issues/194
  - https://github.com/lucabaldini/ixpeobssim/issues/245
  - https://github.com/lucabaldini/ixpeobssim/issues/235
  - https://github.com/lucabaldini/ixpeobssim/issues/243

ixpeobssim (5.4.0) - Fri, 18 Oct 2019 16:44:37 +0200

(Mind this was tagged for logistical reasons with known issues and you are strongly advised against using this version. Please update straight to 6.0.0)

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/70
- New Tycho config file and example using the Stokes sky-cube functionality.
- Improvements to xpphiradalign: new command-line options, and fix in the pipeline wrapper and alignment algorithm refactored.
- Error message prompted upon sky coordinates outside the Stokes sky-map grids
- Logger tweaked to print messages in color.

ixpeobssim (5.3.0) - Wed, 16 Oct 2019 09:35:10 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/69

- New small tool to align the photoelectron directions radially
- Tool made available into the pipeline.

ixpeobssim (5.2.1) - Wed, 16 Oct 2019 09:18:45 +0200

- Custom argument formatter added to the argument parser to allow for more flexibility in the parser output.
- __description__ for all the applications revamped.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/239

ixpeobssim (5.2.0) - Tue, 15 Oct 2019 11:02:39 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/67
- Added an option –innerrad to select annuli in xpselect
- Added an option –invert to invert a selection in xpselect
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/237
  - https://github.com/lucabaldini/ixpeobssim/issues/238

ixpeobssim (5.1.0) - Mon, 14 Oct 2019 12:31:37 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/67
- Full refactor of all the classes dealing with polarization maps into a single xStokesSkyMap class, that is able to read in FITS files with Q/U, x/y, or polarization degree/angle, but only uses the Stokes parameters internally.
- Configuration files and examples updated.
- New xStokesSkyCube added, for 3-d interpolation in x, y and energy.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/119
  - https://github.com/lucabaldini/ixpeobssim/issues/181

ixpeobssim (5.0.0) - Wed, 09 Oct 2019 12:26:27 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/66
- GEM charging introduced in the simulation.
- GEM gain added to the output file, in the MONTE_CARLO extension.
- xpchrgcorr.py added to process event lists and correct for the charging.
- New EFLUX binning algorithm implemented.
- Pulse-height analysis code refactored (issue #224)
- Simple decorator to time functions added.
- Horrible inverse spline for the energy to channel conversion replaced with a binary search.
- Energy dispersion classes modified to keep track of the measured energy before digitization
- Various minor refactorings and fixes.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/231

    - https://github.com/lucabaldini/ixpeobssim/issues/221
    - https://github.com/lucabaldini/ixpeobssim/issues/224
    - https://github.com/lucabaldini/ixpeobssim/issues/71

ixpeobssim (4.2.1) - Tue, 01 Oct 2019 10:52:55 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/65
- Minor update to the data format (some D changed to E)
- Definition of the event-by-event Stokes parameters reverted back to the original definition (with the extra factor of 2).
- **Issue(s) closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/218
    - https://github.com/lucabaldini/ixpeobssim/issues/220

ixpeobssim (4.2.0) - Wed, 18 Sep 2019 09:45:51 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/64
- **Issue(s) closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/219

ixpeobssim (4.1.0) - Mon, 19 Aug 2019 17:39:40 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/63
- Initial implementation of a spiral dithering pattern (unit-tested but not yet used for either analysis or simulations).

ixpeobssim (4.0.0) - Thu, 04 Jul 2019 16:01:47 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/62
- Dithering pattern implemented in the simulation and fully exposed through ixpeobssim command-line switches.
- DETPHI output column is now correctly rotated with respect to PHI on a DU by DU basis.
- GPD fiducial area is now properly applied, and the DU rotation is implemented in conformance with the relevant IXPE mechanical interface control document.
- Unit tests added.
- Typo in a function name in utils.units fixed.
- **Issue(s) closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/193

ixpeobssim (3.6.0) - Fri, 28 Jun 2019 13:32:51 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/60
- This is mainly to allow disengaging the application of the vignetting for situations where the source is generated into the detector and the optics are irrelevant (e.g., the internal backgroud). But really, this is a significant refactoring of the entire srcmodel.roi module.
- Added a base class for model components representing calibration flat fields.
- Added check to make sure that the event times are sorted at the event-list filling time.
- xEventList.__len__() deprecated.
- **Issue(s) closed:**

    - https://github.com/lucabaldini/ixpeobssim/issues/185
    - https://github.com/lucabaldini/ixpeobssim/issues/214
    - https://github.com/lucabaldini/ixpeobssim/issues/215

ixpeobssim (3.5.1) - Thu, 27 Jun 2019 11:25:15 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/61
- Patch to avoid tracebacks from xFITSImage
- **Issue(s) closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/167
    - https://github.com/lucabaldini/ixpeobssim/issues/213

ixpeobssim (3.5.0) - Wed, 26 Jun 2019 12:52:22 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/58
- Added a check that the pdf is positive for all the random number generators.
- Added diagnostics for unphysical polarization values.
- **Issue(s) closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/39
    - https://github.com/lucabaldini/ixpeobssim/issues/174

ixpeobssim (3.4.0) - Wed, 26 Jun 2019 12:20:24 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/57
- This is essentially an update of the data format, to align it more with the corresponding document.
- Unrelated minor fix in utils.system.cmd(), see issue #211
- **Issue closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/175
    - https://github.com/lucabaldini/ixpeobssim/issues/206
    - https://github.com/lucabaldini/ixpeobssim/issues/207
    - https://github.com/lucabaldini/ixpeobssim/issues/211

ixpeobssim (3.3.0) - Wed, 26 Jun 2019 12:17:55 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/59
- Random seed set to a random value by default.
- **Issue(s) closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/189

ixpeobssim (3.2.0) - Sun, 23 Jun 2019 08:54:24 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/55
- Refactoring of the spectrum model, where the time-dependent spectral parameters are now handled in a sensible and consistent fashion.
- **Issue closed:**
    - https://github.com/lucabaldini/ixpeobssim/issues/202

ixpeobssim (3.1.0) - Wed, 19 Jun 2019 14:24:20 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/45
- Added a tutorial section to the documentation.

ixpeobssim (3.0.0) - Wed, 19 Jun 2019 14:17:16 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/52
- Chandra2ixpe application removed to avoid potential confusion between xpobssim and the converter (the two applications were almost identical). As a result of this change, a standard simulation or a conversion from Chandra will take place based on the type of ROI implemented in the source configuration file.
- **Issue closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/192

ixpeobssim (2.10.0) - Wed, 19 Jun 2019 12:37:11 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/56
- xpeInterstellarAbsorptionModel changed to xInterstellarAbsorptionModel
- README file updated, and all contents from the wiki removed.
- Programmatic import cleaned up to fix a warning with Python 3.
- Copyright notice in the startup message updated.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/203
  - https://github.com/lucabaldini/ixpeobssim/issues/70
  - https://github.com/lucabaldini/ixpeobssim/issues/165
  - https://github.com/lucabaldini/ixpeobssim/issues/198

ixpeobssim (2.9.0) - Tue, 18 Jun 2019 16:25:04 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/54
- This is a massive refactoring of all the spline classes, including those dealing with the generation of random numbers, the two most profound changes being a fix of the way we handle meshgrids in bivariate splines, and a change in the signature of random number generators with auxiliary variables (rv and aux swapped). Both changes involve a subtle variation of the semantics but the net result is now clean and self-consistent, and the operation was absolutely necessary before we started implementing new complex features on top of our spline/rng ecosystem. (This should be all largely transparent to end user but, needless to say, we might have introduced side effects, so watch out.)
- xSourceSpectrum class added.
- As a consequence of the spline refactoring our interbale representations of the energy dispersion and the azimuthal response generators, as well as the source and count spectra, are now rotated by 90 degrees.
- Interpolation in log space now working.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/196
  - https://github.com/lucabaldini/ixpeobssim/issues/163
  - https://github.com/lucabaldini/ixpeobssim/issues/88
  - https://github.com/lucabaldini/ixpeobssim/issues/54

  - https://github.com/lucabaldini/ixpeobssim/issues/58
  - https://github.com/lucabaldini/ixpeobssim/issues/113
  - https://github.com/lucabaldini/ixpeobssim/issues/126
  - https://github.com/lucabaldini/ixpeobssim/issues/115
  - https://github.com/lucabaldini/ixpeobssim/issues/197
  - https://github.com/lucabaldini/ixpeobssim/issues/199
  - https://github.com/lucabaldini/ixpeobssim/issues/201

ixpeobssim (2.8.2) - Mon, 03 Jun 2019 14:45:08 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/50
- Two minor changes to the data format to conform to the FITS standards ("DEC—TAN" changed to "DEC–TAN" and comment for the "DATE" field in the event lists shortened).

ixpeobssim (2.8.1) - Mon, 03 Jun 2019 14:39:44 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/49
- One unit test on the unpolarized response added (this was triggered by issue #182.)

ixpeobssim (2.8.0) - Mon, 03 Jun 2019 14:21:44 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/48
- "Bug" in the time assignment when converting Chandra observations (see issue #191—up to now we were using the Chandra event times and this, in turn,was causing issues with the CCD data). The Chandra event times are now ignored, and assigned randomly between the start and the end of the observation, instead.
- Unit tests added.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/191

ixpeobssim (2.7.0) - Mon, 03 Jun 2019 10:56:08 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/48
- Top-level Makefile changed, now it works on different devices
- New Furier_harmonics method to build pulse profiles and/or phase-dependent polarization degree function in the configuration
- New cutoff_power_low in spectrum.py
- Implemented xXspecModel and xXspecModelBuilder classes for a generic source configuration, including PYXSPEC support

ixpeobssim (2.6.1) - Tue, 28 May 2019 18:20:32 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/46
- This fixes an issue with the trigger ID column for periodic sources.
- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/142

ixpeobssim (2.6.0) - Tue, 02 Apr 2019 14:57:50 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/41
- Major revision of the distutil-related files—now we should be up and and running into user-mode.

- **Issue(s) closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/176

ixpeobssim (2.5.0) - Wed, 07 Nov 2018 07:03:51 +0100

- Merged in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/43
- Merged the polarization class with the combined polarization class.
- Removed the xpolarizationarrows class.
- Added a draw arrows method to the xBinnedStokesCube class.
- Added the vela nebula pulsar configuration and example script
- Fixed the msh1552 configuration and example script.

ixpeobssim (2.4.0) - Wed, 10 Oct 2018 09:09:23 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/40
- XFLT0001 keyword added to the SPECTRUM extension of the binned Stokes spectra.
- Added support for native spectro-polarimetric fit in XSPEC.

ixpeobssim (2.3.0) - Sun, 16 Sep 2018 22:31:51 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/37
- Stokes-spectra output now implemented in xpbin.
- New .mrf response functions in the caldb.
- evt.xspec module update to support spectro-polarimetric fitting in XSPEC.
- Option mc=True for xpbin.py in PHA1* mode correctly implemented.
- Pipeline now saving and plotting figures automatically.
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/155,
  - https://github.com/lucabaldini/ixpeobssim/issues/172

ixpeobssim (2.2.1) - Thu, 13 Sep 2018 16:50:39 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/39
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/173

ixpeobssim (2.2.0) - Wed, 12 Sep 2018 21:55:45 +0200

- Merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/38
- Added facilities to profile the memory usage.
- Avoid copying HDU lists when selecting rows within an event file, which in turn fixes a serious memory leak in xpselect, see https://bitbucket.org/ixpesw/ixpeobssim/commits/6a7b62cc55856892f751c55b7361b9531b9027a6
- Pipeline now saving and plotting figures automatically.
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/149,
  - https://github.com/lucabaldini/ixpeobssim/issues/172

ixpeobssim (2.1.1) - Mon, 03 Sep 2018 16:53:22 +0200

- Minor changes, tagging the correct branch (master).

ixpeobssim (2.1.0) - Mon, 03 Sep 2018 16:42:47 +0200

- Full rewrite of the pipeline module, which is now a plain set of function, as oposed to a class.
- Documentation of the new pipeline module added.
- All examples reviewed in light of the new pipeline framework.
- Application signatures in ixpeobssim/bin uniformed to make it easier to wrap them.
- pairwise() and pairwise_enum() methods added to binning.
- POL_ANGLE and POL_ANGLE_ERR fields for binned modulation cubes changed to POL_ANG and POL_ANG_ERR for uniformity with the rest of the framework.
- Default DU suffix for the output file name changed from 1, 2, 3, to du1, du2, du3.
- Full rewrite of the old evt.fitting module, which is now evt.xspec; like the new pipeline, the module is now a collection of functions, as opposed to a class.
- Major revamp of the crab_pulsar configuration file and associated example pipeline, with docs added in the proper section.
- Major revamp of the J1708 configuration file and associated example pipeline, with docs added in the proper section.

ixpeobssim (2.0.0) - Sun, 26 Aug 2018 15:25:59 +0200

- **Version v2 of the IXPE response functions released, with many changes, see https://github.com/lucabaldini/ixpeobssim/issues/161**
  - IRFs are now defined from 1 to 12 keV (was 15 in the previous iteration).
  - new MMA effective area parametrization, with thicker thermal shields, see https://github.com/lucabaldini/ixpeobssim/issues/152
  - effect of the aluminination of the Be window included, see https://github.com/lucabaldini/ixpeobssim/issues/151
  - transparency of the UV filter added, see https://github.com/lucabaldini/ixpeobssim/issues/159
  - GPD quantum efficiency updated with the tabulated DME density at 20 degrees C.
- Added new modulation response function in the CALDB, and associated tools to load and visualize it.
- Added toy response functions for debugging purposes.
- **Major refactoring of the code generating the response functions:**
  - xcom interface added, and all the relevant GPD characteristics are now calculated on the fly, rather than written in text files.
  - calibgen folser moved to irfgen
  - all irfgen data files reorganized into folders matching the new irfgen Python modules.
- **Major refactoring of the IRF classes:**
  - xIrfBase class created, and now all the response-file classes are inheriting from it.
- Major refactoring of the command-line parsing facilities.
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/161,

  - https://github.com/lucabaldini/ixpeobssim/issues/152,
  - https://github.com/lucabaldini/ixpeobssim/issues/153,
  - https://github.com/lucabaldini/ixpeobssim/issues/151,
  - https://github.com/lucabaldini/ixpeobssim/issues/159

ixpeobssim (1.8.1) - Fri, 24 Aug 2018 15:25:48 +0200

- ngc1068 configuration file (and pipeline) adapted to the new source etiquette.
- Source documentation tweaked.

ixpeobssim (1.8.0) - Wed, 22 Aug 2018 16:55:03 +0200

- Merged in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/
- All toy models renamed and revamped following a minimal set of guidelines.
- Pipelines added for all toy models, with the corresponding documentation linked from the main docs page.
- Section about source etiquette added in the documentation.
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/141

ixpeobssim (1.7.0) - Fri, 17 Aug 2018 14:07:57 +0200

- Merged in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/32
- Active area of the GPD readout on the focal plane implemented (events outside the detector are trimmed out).
- DU rotation implemented (each of the three DU now gets its own orientation).
- Added option to set the telescope roll angle in xpobssim and chandra2ixpe.
- binsz (now pixsize) functionality restored in xpbin.
- Default image size for xpbin in CMAP mode adjusted.
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/48

ixpeobssim (1.6.0) - Wed, 08 Aug 2018 15:10:03 +0200

- Major refactoring of the binning classes, merging in pull reques https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/31
- Sum of the relevant binning data structures now implemented for all types via the __iadd__ (as opposed to __add__) overload.
- PHASG binning algorithm renamed as PP.
- nxpix and nypix command-line switches for binned map now unified in npix (i.e., we shall be only producing square maps).
- x-axis errors supported in xScatterPlot.
- Modulation cube objects now using the internal information for visualization purposes (as opposed to fitting).
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/93,
  - https://github.com/lucabaldini/ixpeobssim/issues/120,
  - https://github.com/lucabaldini/ixpeobssim/issues/130,

  - https://github.com/lucabaldini/ixpeobssim/issues/134,
  - https://github.com/lucabaldini/ixpeobssim/issues/145

ixpeobssim (1.5.1) - Thu, 02 Aug 2018 15:48:13 +0200

- Cleanup imports, merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/29
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/140

ixpeobssim (1.5.0) - Wed, 01 Aug 2018 21:24:22 +0200

- Windows support added, merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/29

ixpeobssim (1.4.1) - Wed, 01 Aug 2018 21:05:44 +0200

- merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/22 (this only involves files in the sandbox)

ixpeobssim (1.4.0) - Wed, 01 Aug 2018 21:03:13 +0200

- Improved azimuthal response generator, merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/25
- This is closing issues https://github.com/lucabaldini/ixpeobssim/issues/49 and https://github.com/lucabaldini/ixpeobssim/issues/135

ixpeobssim (1.3.0) - Wed, 01 Aug 2018 20:46:17 +0200

- NGC 1068 model added, merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/28
- Full implementation of the harmonic addition theorem added in srcmodel/polarization.py

ixpeobssim (1.2.0) - Wed, 01 Aug 2018 20:38:54 +0200

- Added entry points for the user installation, merged in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/24
- Default output folder changed to ~/ixpeobssimdata
- xpbinviewer.py and xpirfviewer.py renamed to xpbinview.py and xpirfview.py
- Some obsolete scripts in ixpeobssim/bin moved to the sandbox
- Documentation updated.

ixpeobssim (1.1.0) - Thu, 26 Jul 2018 12:55:32 +0200

- Tycho example added, merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/26
- Emergency fix for issue https://github.com/lucabaldini/ixpeobssim/issues/137

ixpeobssim (1.0.0) - Tue, 17 Jul 2018 15:26:36 +0200

- New set of irfs available for each detector unit with 350 energy channels.
- Added support to the new caldb structure.
- All applications, test scripts and examples updated to support the new du-based irfs.
- Maps of polarization degree and angle can now be used to build the input polarization model.
- Added methods to plot and overlay arrows to the input and output counts maps.
- New ARMAP binning algorithm implemented.
- Major refactoring of Stokes cubes data format.

- add() and from_list() methods implemented in (almost) all binning classes.
- Added a flag to exclude regions in xChandraObservation class.
- Added a simulation of msh1552 using models provided by N. Bucciantini.
- Added a simulation of AXP 1RXS_J1708 using models provided by R. Turolla and R. Taverna.
- Merged in pull request: https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/23
- Added a new application to calculate the phase and append the column to the event list. Merging in pull request: https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/19
- Phase column removed from output file of xpobssim and chandra2ixpe (issue #9).
- **Fixed crab_pulsar, casa, cena examples. Merging pull requests:**
  - https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/15,
  - https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/18,
  - https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/21.
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/66,
  - https://github.com/lucabaldini/ixpeobssim/issues/67,
  - https://github.com/lucabaldini/ixpeobssim/issues/68,
  - https://github.com/lucabaldini/ixpeobssim/issues/80,
  - https://github.com/lucabaldini/ixpeobssim/issues/121,
  - https://github.com/lucabaldini/ixpeobssim/issues/125,
  - https://github.com/lucabaldini/ixpeobssim/issues/116.
- Added sandbox folder.

ixpeobssim (0.64.0) - Mon, 21 May 2018 12:27:08 +0200

- Bug fix in the simulation of periodic sources, marging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/16
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/118

ixpeobssim (0.63.1) - Fri, 27 Apr 2018 15:03:38 +0200

- __future__ imports added everywhere to support Python 2, merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/14/

ixpeobssim (0.63.0) - Sat, 14 Apr 2018 09:28:20 +0200

- Creator ID added to all the output fits files.
- chandra2ixpe unit test and pipeline interface fixed.
- Merged in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/13
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/104

ixpeobssim (0.62.1) - Sat, 14 Apr 2018 01:47:17 +0200

- Visibility changed to modulation throughout, and some docs added, merged in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/12/fix-visibility/diff
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/109

ixpeobssim (0.62.0) - Fri, 13 Apr 2018 20:03:01 +0200

- Data format updated to the new, tentative definition of the Level 2 files, following the I2T face to face meeting on April 13, 2018, mergin in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/11
- Highlights include the fact that output photon lists can now be correctly parsed by the standard tools of the community, including ds9 and ximage.
- Deadtime-related keywords included in all the relevant headers, and propagated to the PHA1 binned files for spectral fitting with xspec.
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/92,
  - https://github.com/lucabaldini/ixpeobssim/issues/8

ixpeobssim (0.61.0) - Wed, 11 Apr 2018 11:49:33 +0200

- Major refactoring of the modeling, fitting and plotting classes, merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/10
- Major restructuring of the modulation cube class.
- Refactoring of the binning module.
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/34,
  - https://github.com/lucabaldini/ixpeobssim/issues/56,
  - https://github.com/lucabaldini/ixpeobssim/issues/105,
  - https://github.com/lucabaldini/ixpeobssim/issues/106

ixpeobssim (0.60.0) - Sat, 07 Apr 2018 10:53:02 +0200

- xpselect refactoring for the new data format, merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/9
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/102
  - https://github.com/lucabaldini/ixpeobssim/issues/103

ixpeobssim (0.59.0) - Fri, 06 Apr 2018 14:01:24 +0200

- Complete refactoring of the chandra2ixpe converter, merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/8
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/20,
  - https://github.com/lucabaldini/ixpeobssim/issues/81
  - https://github.com/lucabaldini/ixpeobssim/issues/85
  - https://github.com/lucabaldini/ixpeobssim/issues/89

  - https://github.com/lucabaldini/ixpeobssim/issues/91

ixpeobssim (0.58.0) - Tue, 27 Mar 2018 23:49:02 +0200

- Tentative implementation of the new data format and major refactoring of the xpobssim internals, merging pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/7
- More unit tests for the simulation and analysis pipelines.
- **Issues closed:**
  - https://github.com/lucabaldini/ixpeobssim/issues/90
  - https://github.com/lucabaldini/ixpeobssim/issues/94
  - https://github.com/lucabaldini/ixpeobssim/issues/38

ixpeobssim (0.57.0) - Sat, 24 Mar 2018 10:24:07 +0100

- Documentation revamped and improved, merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/5
- A typo in all the header files fixe, see issue #5, merging in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/6

ixpeobssim (0.56.0) - Sun, 11 Mar 2018 18:12:06 +0100

- Now officially supporting Python 3x (and discontinuing Python 2x). Merged in pull request https://bitbucket.org/ixpesw/ixpeobssim/pull-requests/4

ixpeobssim (0.55.1) - Mon, 02 Oct 2017 16:53:25 +0200

- Minor.

ixpeobssim (0.55.0) - Mon, 02 Oct 2017 16:15:34 +0200

- A few new models added or updated (Crab, Cas A, MSH 15-52).
- Added code to generate IRFs in the new CALB format (not yet used).

ixpeobssim (0.54.1) - Tue, 01 Aug 2017 08:53:52 +0200

- More work on the documentation.

ixpeobssim (0.54.0) - Mon, 31 Jul 2017 10:11:13 +0200

- Extensive restructuring of the documentation.

ixpeobssim (0.53.0) - Sat, 29 Jul 2017 09:25:35 +0200

- Added an example script (examples/uniform_disk_stokes.py) to plot the polarization degree and angle using the stokes cube.
- Added a script (ximpol/detector/gpd.py) and associated files to cross check the calculation of the window transmission and the GPD efficiency.
- Added SCUBE algorithm to xpbin.
- Added new events binning classes to make and read the stokes cubes (SCUBE).
- Added method to use stokes parameters to build the polarization maps in example script casa_pol_map.py.
- Added xppimms to the pipeline.
- A couple of modifications introduced by mistake reverted.
- xppimms and chandra2ximpol scripts made executable.

ximpol (0.52.0) - Tue, 24 Jan 2017 12:14:17 +0100

- First implementation of xppimms for MDP calculation with source paramteres provided through command-line switches (issue #153)
- Added example script to compare polarization values found via mcube vs stokes, using a single point source.
- Added polarization fraction method to class xStokesAccumulator
- Refactoring of evt.fitting.xSpectralFitter class.
- Implemented the vignetting in the chandra2ximpol converter.
- Added a function to draw the psf circle in irf.psf.xPointSpreadFunction (closing issue #151).
- Scripts to simulate GRS1519+105 and the txt files provided by Banafsheh.
- Minor refactoring of the IRF plotting code (closing issue #150).
- Some tweaks to the xpmdp.py script (closing issue #107).
- Bug fix for issues #25 and #22 (closing them).
- Observation plan plot added in the examples folder.
- Scripts to simulate GRS1519+105 using the txt files provided by Michal.
- Added analysis pipeline to simulate Cen A using chandra-to-ximpol converter.
- Added a first implementation of a "Stokes accumulator" class (see issue #148).

ximpol (0.51.1) - Fri, 01 Jul 2016 14:04:20 +0200

- Bug fix in the check for the polarization degree (see issue #73).

ximpol (0.51.0) - Fri, 01 Jul 2016 12:00:53 +0200

- First stub at implementing splines in log space (issue #20).
- Complete PSF refactoring.
- New sets of IRFs created, and default now pointing to 'xipe_mirror-30s-f4_psf-jetx-rescaled-hew30_gpd-baseline'

ximpol (0.50.0) - Thu, 30 Jun 2016 14:56:31 +0200

- Blazar sensitivity plot added in the examples folder.
- New config file and associated pipeline added for J1708.
- Avoid reading the modulation factor in xBinnedModulationCube (using the effective factor written in the fits file instead, when it comes to converting a visibility into a polarization fraction.)
- One unit test added.
- New bivariate spline class added supporting orders greater than 1 on both axes.
- Bug fix for issue #143 (closing it).
- xUnivariateAuxGenerator now supporting spline orders greater than 1 on both axes (taking advantage of the new bivariate spline class).
- Fix for issue #73 (clsing it).

ximpol (0.49.0) - Fri, 24 Jun 2016 16:31:55 +0200

- "xipe_goal" IRFs now used by default by all the xp tools.

ximpol (0.48.0) - Wed, 22 Jun 2016 23:12:31 +0200

- Added script to make the polarization map of casa in examples

- Added the option to draw the psf circle in the count map of casa in the main casa.py example.
- Added the Cyg-X1 ascii files for the model at 40 degree inclination.
- Added checks in the univariate spline constructors to make sure that the input x-values are sorted and unique (closing issue #84.)
- binning module modified in such a way that all the relevant quantities are also calculated over the entire energy range when the energy binning is longer than 1 (i.e., if you do 2–4 keV and 4–8 keV, you get 2–8 keV for free).

ximpol (0.47.3) - Wed, 22 Jun 2016 08:56:35 +0200

- Added M87 configuration files and analysis pipeline for chandra-to-ximpol converter.
- Added configuration files and analysis pipeline example for MCG-6-30-15, ARK 120 and NGC 1365.
- Configuration file for GK Per slightly tweaked.

ximpol (0.47.2) - Sat, 18 Jun 2016 06:56:49 +0200

- Minor doc update.
- Added GK Per configuration file and analysis pipeline example.

ximpol (0.47.1) - Thu, 16 Jun 2016 16:39:40 +0200

- Slight tweak to the command-line switches for xpbin and xpselect, in order to be able to set the mc keyword argument to True from the pipeline.
- Fix for issue #105 (closing it).

ximpol (0.47.0) - Thu, 16 Jun 2016 15:34:41 +0200

- Redshift added to the simulation (issue #121).

ximpol (0.46.0) - Thu, 16 Jun 2016 15:26:42 +0200

- Interstellar absorption added to the simulation (closing issue #120).

ximpol (0.45.1) - Wed, 15 Jun 2016 15:00:50 +0200

- Minor bug fix.
- ximpol.srcmodel.polarization module added to the documentation.

ximpol (0.45.0) - Wed, 15 Jun 2016 14:46:01 +0200

- Galactic absorption column density and redhsift added to all the model components (working our way through issue #120).
- Some cleanup of the Galactic absorption code (issue #120).
- Unit test for the intertellar absorption added.
- Fix for issue #103.
- Some significant refactoring of the xpmdp code.
- xpmdp added to the pipeline (closing issue #137).
- Added a unit test comparing the xpmdp.py output with the figures from the online sensitivity calculator.

ximpol (0.44.2) - Fri, 10 Jun 2016 17:58:27 +0200

- Trivial fix.

ximpol (0.44.1) - Fri, 10 Jun 2016 17:54:30 +0200

- Installation notes updated (closing issue #131).

ximpol (0.44.0) - Fri, 10 Jun 2016 15:35:16 +0200

- Added method to the xPolarizationMap object to plot the polarization degree and angle (in addition to the x and y components).
- Added new corona models for Cyg-X1 in ascii folder
- Added new Cyg-X1 config files (one per model) in the config folder
- Added new script tp make the plot comparing the polarization fraction for Cyg-X1 for the two coronal models in examples
- Added method to run several simulations (with different seeds) and merge the output files to one single. This is in the run method for the Cyg-X1 example.
- Added script to make the map of the polarization degree and map of the sigma for the polarization degree.
- Initial import of the module and files for parametrizing the Galactic absorption (issue #120).
- Quick fix to issue #129.

ximpol (0.43.0) - Sat, 04 Jun 2016 22:13:08 +0200

- Added a script to plot MDP for the crab pulsar with the nebula background in examples.
- Added effective mu, source counts and mdp to the xBinnedModulationCube class
- Showcase updated with the fix to the PSF.
- First implementation of the chandra2ximpol converter.

ximpol (0.42.1) - Wed, 20 Apr 2016 21:11:25 +0200

- Team updated.

ximpol (0.42.0) - Wed, 20 Apr 2016 11:30:41 +0200

- xpmdp.py adapted (via brute-force) to periodic sources.
- MDP information added to the xpbin output in MCUBE mode.

ximpol (0.41.1) - Tue, 12 Apr 2016 16:54:48 +0200

- Minor fix in the MDP calculator output.

ximpol (0.41.0) - Tue, 12 Apr 2016 16:28:11 +0200

- New utils/units module added.
- Configuration file for Abell 85 (along with its configuration file) added.
- Source model string formatting improved (issue #101).

ximpol (0.40.0) - Tue, 12 Apr 2016 13:38:08 +0200

- Initial implementation of the Cyg-X1 config/example.
- First implementation of a script for the calculation of the MDP.
- Significant refactoring of the srcmodel/spectrum.py module.

ximpol (0.39.4) - Thu, 07 Apr 2016 13:59:05 +0200

- Cas A example tweaked.

ximpol (0.39.3) - Thu, 07 Apr 2016 07:00:47 +0200

- Added minimal support for log scale on the the z axis when plotting bivariate splines.
- Added doc/scripts folder (work in progress).

ximpol (0.39.2) - Tue, 22 Mar 2016 14:51:31 -0700

- Cas A movie updated.

ximpol (0.39.1) - Tue, 22 Mar 2016 13:59:13 -0700

- Added a GRB example to the gallery.
- Added the Cas A movie.

ximpol (0.39.0) - Fri, 18 Mar 2016 11:54:43 -0700

- GRB 130427 configuration file revamped.
- Bug fix in the binning module for the LOG time binning.
- Some more infrastructure in place for arbitrary source-based sampling times (issue #44).
- Added a new example for GRB 130427.

ximpol (0.38.1) - Thu, 17 Mar 2016 09:51:51 -0700

- References for the Crab pulsar example added.

ximpol (0.38.0) - Wed, 16 Mar 2016 15:48:56 -0700

- One more unit test added.
- A few tweaks and some cleanup.
- Optional scale and offset parameters added to the plot() method for the univariates splines.
- Bug fix for issue #97.
- Crab pulsar example revamped.

ximpol (0.37.1) - Tue, 15 Mar 2016 17:10:54 -0700

- Crab pulsar added to the showcase.

ximpol (0.37.0) - Tue, 15 Mar 2016 15:15:36 -0700

- Added a pipeline example for the Crab pulsar.
- Equipopulated-binning code refactored (issue #93).
- evt.select.py renamed as evt.subselect.py (issue #96).
- xpxspec refactored, with most of the code being moved to evt.fitting.py (issue #92).
- Some specific refactoring.
- Equipopulated binning refactored (issue #93).

ximpol (0.36.1) - Sat, 12 Mar 2016 07:40:17 -0800

- First complete Cas A section in the gallery (issue #80).

ximpol (0.36.0) - Sat, 12 Mar 2016 06:03:59 -0800

- Initial stub at the ximpol gallery (issue #80).
- Short version of the command-line switches removed from xpxspec, and all of them passed as keyword arguments (issue #71).
- xpxspec added to the pipeline, and a new example added.
- More tweaks to the Cas A analysis pipeline example.
- ebinning LIST mode added to xpbin.

- Significant refactoring of the xBinnedModulationCube class to allow to reuse single analysis/plotting tasks externally.
- Pretty much done with the lamp_post pipeline example.
- A few interface tweaks.
- Fix for issue #77.
- Getting started on documenting the architecture of the package.
- Model for Tycho added.
- Bug in the PSF fixed (issue #82).
- A few files renamed (removed the leading test) to prevent issues with the unit testing.

ximpol (0.35.4) - Wed, 09 Mar 2016 16:54:43 -0800

- Enforce data-type consistency in the output event files (issue #66).

ximpol (0.35.3) - Wed, 09 Mar 2016 14:01:52 -0800

- Clobber mechanism implemented at the single tool level and propagated to the pipeline.

ximpol (0.35.2) - Wed, 09 Mar 2016 10:06:32 -0800

- Smoother version of the Cas A spectral models.

ximpol (0.35.1) - Tue, 08 Mar 2016 22:01:46 -0800

- New Cas A spectral models.

ximpol (0.35.0) - Tue, 08 Mar 2016 15:31:58 -0800

- Modified configuration file for Cas A, now with separate extended components for the thermal and the non-thermal emission.
- Subtraction implemented for unidimentional splines.
- Classes for the fit of the azimuthal distributions tweaked.
- One full analysis pipeline for Cas A implemented in ximpol/examples/casa.py.
- A few obsolete files removed.

ximpol (0.34.1) - Mon, 07 Mar 2016 16:54:42 -0800

- Help formatter for xpobssim, xpselect and xpbin changed.

ximpol (0.34.0) - Mon, 07 Mar 2016 16:45:19 -0800

- Internals of xpobssim, xpselect and xpbin tweaked to be fully configurable via keyword arguments, to facilitate pipelining analyses.
- ximpol/examples folder added.
- First stup at an analysis pipeline to facilitate complex simulation/analysis chains.

ximpol (0.33.1) - Mon, 07 Mar 2016 12:11:25 -0800

- Minor changes to the lamp_post_accreting_bh source model.

ximpol (0.33.0) - Mon, 07 Mar 2016 10:59:30 -0800

- Some changes for the creation of the Cas A movie.
- Minor tweaks.

- Layout of the configuration files reorganized, with ximpol/srcmodel/config moved to ximpol and the ascii and fits folder moved as subfolders therein.
- ximpol.__init__.py and cleanup script modified accordingly.
- Lamp-Post accreting BH model (by Alberto) added.
- All soure models adapted to the new layout.
- And all the unit tests run for cross-check.

ximpol (0.32.1) - Wed, 02 Mar 2016 10:44:26 -0800

- Changed name of xpbinview.py to xpviewbin.py.
- Short options removed from xpobssim (issue #71).

ximpol (0.32.0) - Tue, 01 Mar 2016 16:45:31 -0800

- Implemented position-dependent polarization patterns based on FITS maps.
- All configuration files updated to the new interfaces.

ximpol (0.31.0) - Tue, 01 Mar 2016 15:17:37 -0800

- Replace numpy.fill() with numpy.full() when appropriate (issue #66).
- New display interface to binned files ximpol/bin/xpview.py (issue #55).
- Obsolete script ximpol/bin/xpimgview.py removed.
- Obsolete script ximpol/bin/xpevtview.py removed.
- A couple of bug fixes in the source models.

ximpol (0.30.1) - Sat, 27 Feb 2016 08:20:34 -0800

- Closed issue #63.

ximpol (0.30.0) - Sat, 27 Feb 2016 06:55:48 -0800

- A couple of command-line switches added to xpselect (issue #51).
- xpbin options propagated to the output files (issue #60).

ximpol (0.29.0) - Fri, 26 Feb 2016 18:41:42 -0800

- Source model for Cas A added.
- First xpselect implementation (issue #51).
- Subtle bug fix in the CMAP binning (issue #70).

ximpol (0.28.1) - Thu, 25 Feb 2016 16:48:43 -0800

- Updated installation instructions (issue #64).

ximpol (0.28.0) - Thu, 25 Feb 2016 15:51:26 -0800

- Phaseograms implemented in xpbin.py (issue #67).

ximpol (0.27.0) - Thu, 25 Feb 2016 15:31:53 -0800

- Work started toward the implementation of periodic sources (issue #43).
- New xEphemeris class in ximpol.srcmodel.roi.py
- New xPeriodicPointSource class in ximpol.srcmodel.roi.py
- Some significant refactoring of the spline and rand classes to allow for more flexibility.

- Major change to the source model interface—the energy spectrum and polarization degree and angle are now passed to the constructor.
- A whole bunch of obsolete stuff removed from ximpol.srcmodel.spectrum (issue #64).
- All configuration files reworked according to the new interfaces.

ximpol (0.26.0) - Tue, 23 Feb 2016 16:42:27 -0800

- FILE mode implemented for tbinalg (issue #53).

ximpol (0.25.0) - Tue, 23 Feb 2016 16:33:27 -0800

- ebinalgs FILE and EQP implemented (issue #56).

ximpol (0.24.1) - Tue, 23 Feb 2016 15:55:06 -0800

- Fixed unit tests.

ximpol (0.24.0) - Fri, 19 Feb 2016 16:14:36 -0800

- Vignetting now into the effective area tables (but not used in the simulation, yet).

ximpol (0.23.1) - Thu, 18 Feb 2016 15:03:59 -0800

- More information added to the IRF primary headers (issue #49).

ximpol (0.23.0) - Thu, 18 Feb 2016 14:56:15 -0800

- Major refactoring of ximpol/detector/xipe.py to use the new classes (issue #49).
- New optics aeff files provided by Fabio committed (but only the on-axis values used for the time being).
- XIPE baseline and goal response functions created (only the effective areas differ for the time being).

ximpol (0.22.4) - Mon, 08 Feb 2016 16:34:11 -0800

- Fix for issue #59.

ximpol (0.22.3) - Mon, 08 Feb 2016 16:25:59 -0800

- Fix for issue #58.

ximpol (0.22.2) - Mon, 08 Feb 2016 15:51:53 -0800

- Quick polarization analysis routine in place.
- Bug fix in the new code reading the IRFs.

ximpol (0.22.1) - Mon, 08 Feb 2016 15:11:38 -0800

- More refactoring of the binning classes.
- Detector, ROI and IR information propagated from the event to the binned files (issue #57).

ximpol (0.22.0) - Fri, 05 Feb 2016 13:56:10 -0800

- MCUBE mode implemented in xpbin.py

ximpol (0.21.2) - Thu, 04 Feb 2016 15:41:41 -0800

- Source model string formatting improved.
- A few minor changes.

ximpol (0.21.1) - Thu, 04 Feb 2016 14:28:43 -0800

- Committed a whole bunch of files left out by mistake.

ximpol (0.21.0) - Thu, 04 Feb 2016 14:27:20 -0800

- Major refactoring and revamp of xpevtview.py
- New class for tabulated stationary spectra.
- New configuration file for the SgrB complex.
- Spectral data for the SgrA and SgrB complexes.
- New small utility (xpsrccoords.py) to search for source coordinates.

ximpol (0.20.0) - Thu, 04 Feb 2016 10:43:26 -0800

- Gaussian disk spatial template implemented.
- A few srcmodel config files renamed.

ximpol (0.19.1) - Wed, 03 Feb 2016 16:17:09 -0800

- Updated documentation.

ximpol (0.19.0) - Wed, 03 Feb 2016 16:12:42 -0800

- Uniform disk implemented (issue #54).
- Added command-line option to use the MC Ra/Dec for xpbin.

ximpol (0.18.0) - Wed, 03 Feb 2016 15:13:52 -0800

- More work on xpbin.py (closing issues #42 and #52).

ximpol (0.17.0) - Tue, 02 Feb 2016 15:41:14 -0800

- Major refactoring of xpbin.py (issue #42).
- Minimum and maximum valid times added to the model components.
- Configuration file for a GRB added.

ximpol (0.16.1) - Tue, 26 Jan 2016 18:49:19 -0800

- Minor refactoring of the ximpol.core.fitsio module.

ximpol (0.16.0) - Tue, 26 Jan 2016 18:40:11 -0800

- Module ximpol.core.fitsio added (issue #49).
- ximpol.evt.event refactored to use the new ximpol.core.fitsio module.
- GTI list in the output event file (issue #24)
- ROI source table in the output event file (issue #45).
- IRF name added in the output event file header (issue #24).
- ROI information added in the output event file header (issue #48).

ximpol (0.15.2) - Mon, 25 Jan 2016 18:04:33 -0800

- Minor refactoring of bin/xpimgview.py

ximpol (0.15.1) - Mon, 25 Jan 2016 16:37:52 -0800

- astropy.wcs used in ximpol/srcmodel/img.py, and aplpy still used for plotting (issue #41).
- Documentation for ximpol/srcmodel/img.py added.

ximpol (0.15.0) - Mon, 25 Jan 2016 15:57:27 -0800

- srcmodel config files renamed.
- Point source in the Crab complex sample file dimmer.

- Added option to xpimgview.py to save the image to file.
- Horrible hack in the azimuthal fit to prevent the visibility from going negative (issue #34, significantly more work needed).
- Some refactoring and more documentation.
- Radius removed from the xROIModel class, and ROI model for the Crab nebula now correctly centered on the right coordinates.

ximpol (0.14.0) - Fri, 22 Jan 2016 20:54:23 -0800

- xpobbsim.py generating an output file name based on the source model (if not specified).
- Added CMAP mode to xpbin.py

ximpol (0.13.0) - Fri, 22 Jan 2016 13:58:51 -0800

- Implemented the infrastructure for multiple source in ROI

ximpol (0.12.1) - Fri, 22 Jan 2016 06:44:01 -0800

- Bug fix in srcmodel/source.py.

ximpol (0.12.0) - Thu, 21 Jan 2016 16:35:14 -0800

- First implementation of extended sources.

ximpol (0.11.1) - Wed, 20 Jan 2016 16:57:24 -0800

- Minor addition to the doc.

ximpol (0.11.0) - Wed, 20 Jan 2016 15:43:39 -0800

- load_irf moved from bin/xpobssim.py to irf/__init__.py, so that it can be reused.
- Unit test for IRF plotting added (issue #30).
- Some documentation for the IRFs added.

ximpol (0.10.1) - Tue, 19 Jan 2016 16:41:33 -0800

- More documentation and unit tests.

ximpol (0.10.0) - Tue, 19 Jan 2016 14:45:50 -0800

- Added math support in the sphinx config file.
- Major refactoring of the classes related to the modulation factor (issue #28).
- More unit tests added.
- More documentation added.

ximpol (0.9.1) - Sat, 16 Jan 2016 07:17:52 -0800

- All unit tests fixed (issue #26).

ximpol (0.9.0) - Fri, 15 Jan 2016 16:34:58 -0800

- IRFs extended ("by hand") down below 1 keV (need to do it properly, see issue #19).
- A couple of subtle bug fixes in the energy dispersion (see issues #21 and #22).
- First version that allows to recover the spectral parameters in XSPEC.

ximpol (0.8.0) - Fri, 15 Jan 2016 11:53:01 -0800

- Obsolete files removed, and some name refactoring.
- xpbin.py created.

- All figures from unit tests moved to doc/figures.
- More unit tests.
- Event times in xpobbsim sorted.
- Spectral analysis in xspec added.

ximpol (0.7.0) - Thu, 14 Jan 2016 15:15:44 -0800

- Modulation factor generator returning angles in degrees.
- Unit test for the modulation factor classes added.
- Source configuration moved out of xpobsim.py
- Folder srcmodel/config created.
- Added optimization step for the x grid in xInterpolatedBivariateSplineLinear.build_vppf() (issue #18).

ximpol (0.6.3) - Wed, 13 Jan 2016 16:16:38 -0800

- .travis.yml file tweaked to add display support for matplotlib.

ximpol (0.6.2) - Wed, 13 Jan 2016 16:11:55 -0800

- One more unit test added.

ximpol (0.6.1) - Wed, 13 Jan 2016 15:38:20 -0800

- Parameter tweak in the xEnergyDispersionMatric class.
- Added unit test for the xCountSpectrum class, with inline images.
- One unit test relaxed.

ximpol (0.6.0) - Wed, 13 Jan 2016 12:13:06 -0800

- Number of XIPE energy channels changed from 1024 to 256 and IRFs regenerated.
- Removed all the hard-coded values for the number of energy channels (issue #13).
- xEnergyDispersionMatrix now inheriting from xUnivariateAuxGenerator (i.e., it has facilities to throw random numbers.)
- Down-sampling mechanism implemented for the xEnergyDispersionMatrix class on the energy axis to streamline performance.

ximpol (0.5.0) - Tue, 12 Jan 2016 15:24:17 -0800

- A couple of bug fixes in the irf.mrf module.
- Major xpobbsim refactoring using all the new classes.

ximpol (0.4.2) - Mon, 11 Jan 2016 07:08:21 -0800

- Minor refactoring.

ximpol (0.4.1) - Sun, 10 Jan 2016 08:01:03 -0800

- Grid optimization for the spline definition implemented (issue #15).
- Small application for visualizing an event file (xpevtview.py) created, and plotting stuff moved out of xpobbsim.

ximpol (0.4.0) - Sat, 09 Jan 2016 10:17:52 -0800

- New module ximpol.core.rand created (issue #16).
- Major rework and speed up of the provisional observation simulator (event loop removed).
- New event list classe in.

- Some cleanup.

ximpol (0.3.1) - Thu, 07 Jan 2016 16:36:04 -0800

- Added PSF classes, with facility to draw random numbers.

ximpol (0.3.0) - Thu, 07 Jan 2016 13:53:07 -0800

- Added make_ppf to the spline base class.
- Some improvement in the plotting facility for the energy dispersion.
- Added unit tests for the irf classes.
- Removed the xmin and xmax arguments from the constructor of all the spline classes, since the integral() method does not understand extrapolations and having spurious values outside the array ranges was causing troubles. (Note the splines can still be extrapolates in the evaluation.)
- Added facilities for normalization, cdf and ppf in the univariate spline base class.
- xmerge() method of the base univariate spline class removed in favor of numpy.union1d()

ximpol (0.2.1) - Thu, 07 Jan 2016 06:57:12 -0800

- First full implementation of the energy dispersion.

ximpol (0.2.0) - Wed, 06 Jan 2016 15:56:38 -0800

- Refactoring of the core.spline module, and plotting functionalities added.
- Unit tests for the utils.os module added.
- Initial import of the utils.matplotlib configuration module.
- Added xEffectiveArea class to irf.arf.
- Added xModulation factor class to mrf.arf.
- bin/xpirfview.py application created (issue #7).

ximpol (0.1.2) - Tue, 05 Jan 2016 08:34:30 -0800

- Minor changes.

ximpol (0.1.1) - Tue, 05 Jan 2016 07:05:43 -0800

- Minor refactoring of the irf specifications, with the OGIP part now included in ximpol.irf.base
- Some documentation added to the irf classes.

ximpol (0.1.0) - Mon, 04 Jan 2016 16:15:30 -0800

- setup.py file added (issue #11).
- release folder renamed as tools.
- ximpol.__logging__ module moved to ximpol.utils.logging (issue #8). Note we use the trailing undescore to avoid name conflicts with the correponding module from the standard library.)
- ximpol.__utils__ module splitted into ximpol.utils.os and ximpol.utils.system (issue #8).
- Code to create the instrument response functions moved to detector.xipe.
- New spline code used when generating the response functions and old xFunction1d classes removed (issue #3).
- fileio folder removed.
- Using the astropy facilities to generate the fits headers (issue #4).

ximpol (0.0.16) - Sun, 03 Jan 2016 14:31:56 -0800

- ximpol is now linked to Travis CI, and the build output is shown and linked from the main github page.

ximpol (0.0.15) - Sat, 02 Jan 2016 07:19:39 -0800

- xChrono class moved to utils.profile. Documentation and unit tests in place.

ximpol (0.0.14) - Sat, 02 Jan 2016 06:59:19 -0800

- Minor formatting fix.

ximpol (0.0.13) - Sat, 02 Jan 2016 06:56:54 -0800

- Added a makefile for the unit tests, and some more documentation about it.

ximpol (0.0.12) - Fri, 01 Jan 2016 07:51:56 -0800

- Some more edits and additions to the documentation.
- Module core.xInterpolatedUnivariateSpline moved to core.spline.
- __package__.py removed, and content moved to ximol.__init__.py, with all imports changed accordingly (issue #10).
- Code to be executed in __main__ moved from test() to main() in all modules (since the test code will be in the form of unit tests).

ximpol (0.0.11) - Thu, 31 Dec 2015 17:19:37 -0800

- Started migrating the documentation from the github wiki to the rst sphinx files, and added more stuff.

ximpol (0.0.10) - Wed, 30 Dec 2015 07:53:08 -0800

- Bug fix in the release script (hopefully).

ximpol (0.0.9) - Wed, 30 Dec 2015 07:48:26 -0800

- Major folder restructuring to make the layout compatible with Read the Docs.
- Documentation effort started (issue #1).
- Suite of unit tests started (issue #4).
- These release notes moved to a .rst file (issue #12).
- utils.xFunction1d being replaced by core.xInterpolatedUnivariateSpline

ximpol (0.0.8) - Mon, 28 Dec 2015 06:29:54 -0800

- Added script to generate the rmf file. Still not working perfectly.
- Some folder refactoring.

ximpol (0.0.7) - Fri, 11 Dec 2015 13:33:49 -0800

- Removed the srcmodel/yaml folder and all the associated parser classes.

ximpol (0.0.6) - Fri, 11 Dec 2015 06:39:21 -0800

- Many minor changes.
- First stab at a parser for the source model.
- FITS images of some sources added, along with a small visualization script.
- Added a script that generates the header for the mrf file.
- Added a script to generate the .mrf file based on the ascii table provided.

ximpol (0.0.5) - Tue, 08 Dec 2015 11:41:24 -0800

- Small fix in the .arf XIPE file.

ximpol (0.0.4) - Tue, 08 Dec 2015 11:33:40 -0800

- Added a first stab at the effective area table definition.
- Added ascii data files for the XIPE IRFs (as in the proposal).
- Script to generate the .arf file for XIPE based on the ascii table.
- Added a general-purpose one-dimensional function class.

ximpol (0.0.3) - Fri, 04 Dec 2015 12:11:49 -0800

- Changed thge release note because I was cheating…

ximpol (0.0.2) - Fri, 04 Dec 2015 12:05:42 -0800

- Folder structure created

ximpol (0.0.1) - Fri, 04 Dec 2015 06:39:19 -0800

- Initial setup of the github repository

CHAPTER

THIRTYTWO

ABOUT IXPEOBSSIM

32.1 License

ixpeobssim is free software licenced under the GNU General Public License version 3. For more information, see the LICENSE file included in the distribution.

PYTHON MODULE INDEX

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z