
ximpol Documentation

Release 0.52.0

The ximpol team

Jun 12, 2017

Contents

1	Contents:	3
1.1	Showcase	3
1.2	Architectural overview	14
1.3	Quick start	18
1.4	Installation	20
1.5	Source models	22
1.6	Instrument response functions (IRFs)	22
1.7	Code development	27
1.8	System tests	30
1.9	Release notes	30
1.10	About ximpol	42
2	Indices and tables	45

ximpol is a simulation framework specifically developed for X-ray polarimetric applications, based on the [Python](#) programming language and the [SciPy](#) stack. It is not tied to any specific mission or instrument design and is meant to produce fast and yet realistic observation-simulations, given as basic inputs:

- an arbitrary source model including morphological, temporal, spectral and polarimetric information;
- the response functions of the detector under study, i.e., the effective area, the energy dispersion, the point-spread function and the modulation factor.

The format of the response files is OGIP compliant, and the framework has the capability of producing output files that can be directly fed into the standard visualization and analysis tools used by the X-ray community, including XSPEC—which make it a useful tool not only for simulating physical systems, but also to develop and test end-to-end analysis chains.

If you are wondering what this is all about, you might want to start off by taking a look at our [showcase](#).

Showcase

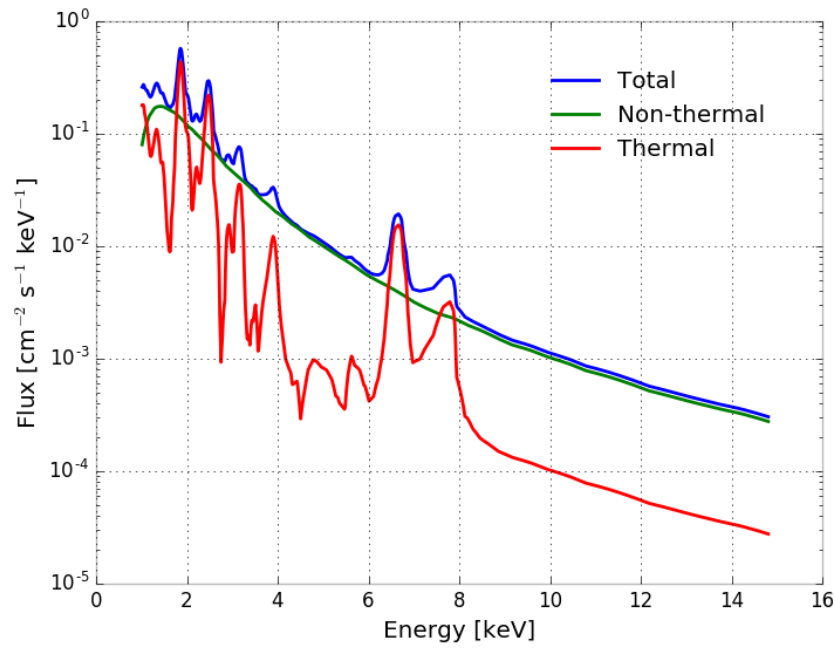
Here are a few source models and simulation outputs illustrating some of the basic ximpol capabilities.

Cas A

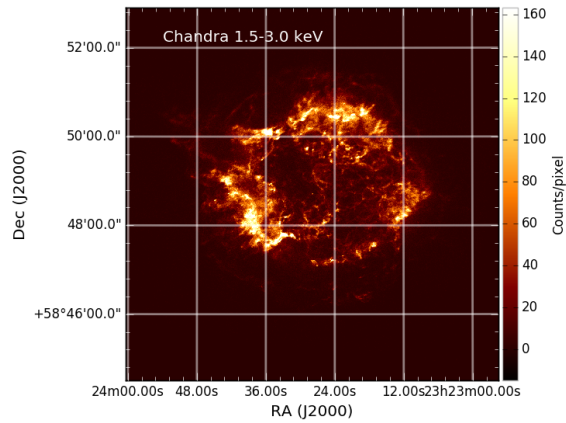
Input model

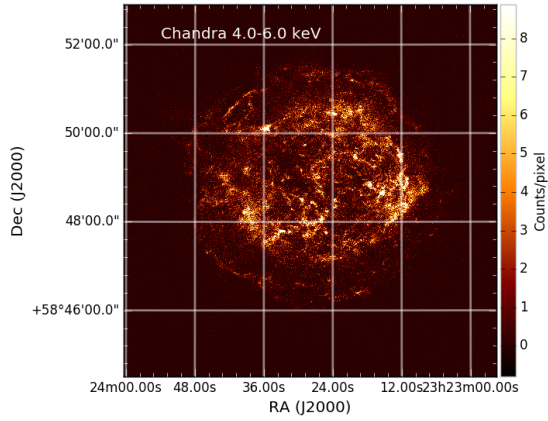
Full source model definition in [ximpol/config/casa.py](#).

The spectral model is taken (by hand) from Figure 5 of E.A. Helder and J. Vink, “*Characterizing the non-thermal emission of Cas A*”, *Astrophys. J.* 686 (2008) 1094–1102. 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, as illustrated in the figure below.

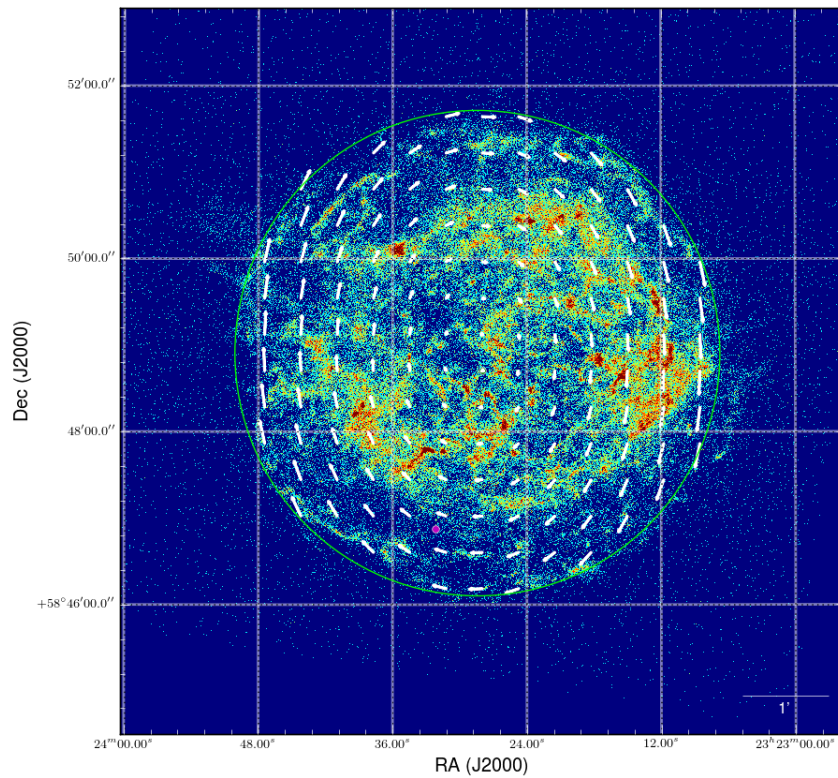


The morphology of the source is energy-dependent in a non trivial way. We start from two Chandra images (in the 1.5–3 keV and 4–6 keV energy ranges, respectively) and associate the former to the thermal spectral component and the latter to the non-thermal one (note the absence of spectral lines between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

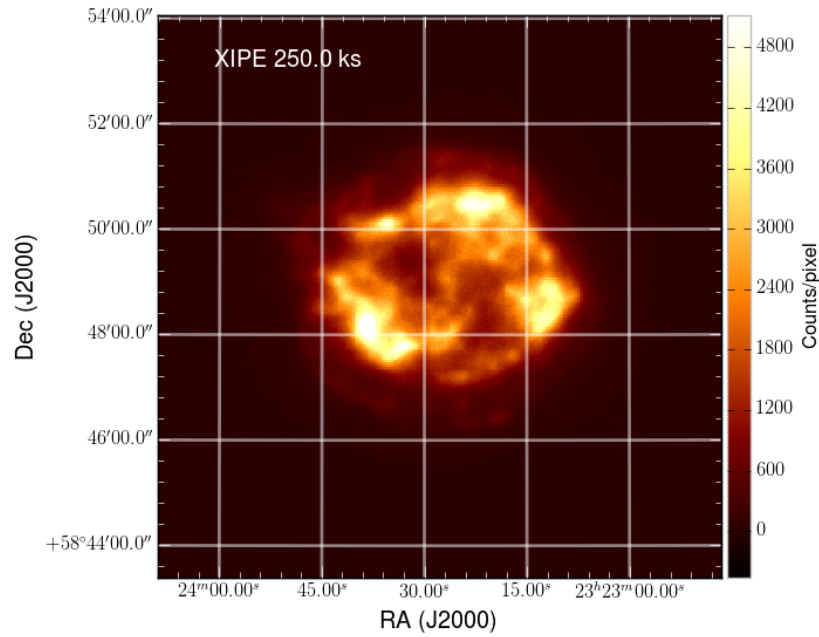


Our total model of the region of interest is therefore the superposition of two independent components, with different spectral, morphological and polarimetric properties. Crude as it is, it's a good benchmark for the observation simulator.

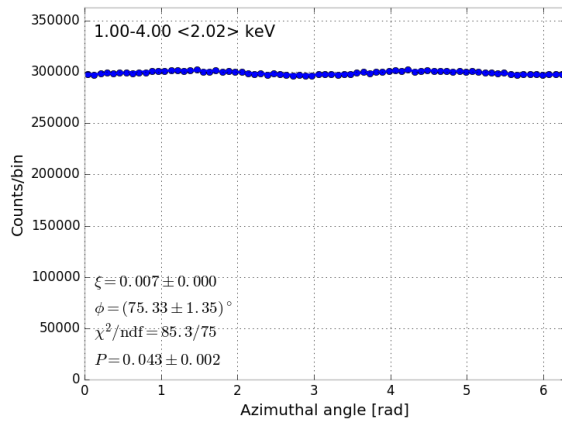
Simulation output

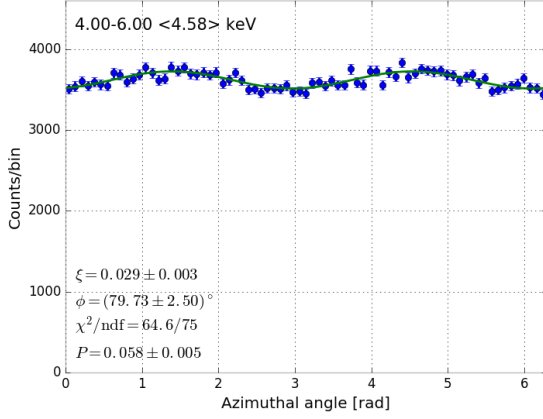
Generation/analysis pipeline in [ximpol/examples/casa.py](#).

Below is a binned count map of a 250 ks simulated XIPE observation of Cas A, based on the model described above.

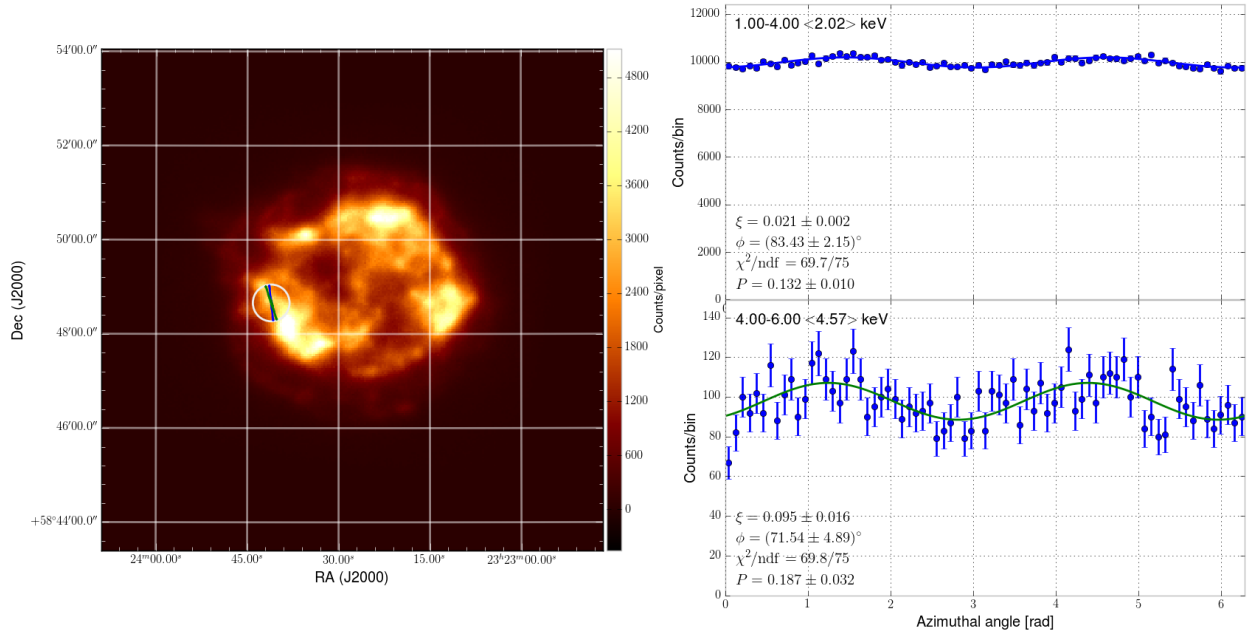


When the entire source is analyzed at once, most of the polarization averages out and even in the high-energy band, where the emission is predominantly non-thermal, the residual polarization degree resulting from the averaging of the different emission regions is of the order of 5%.

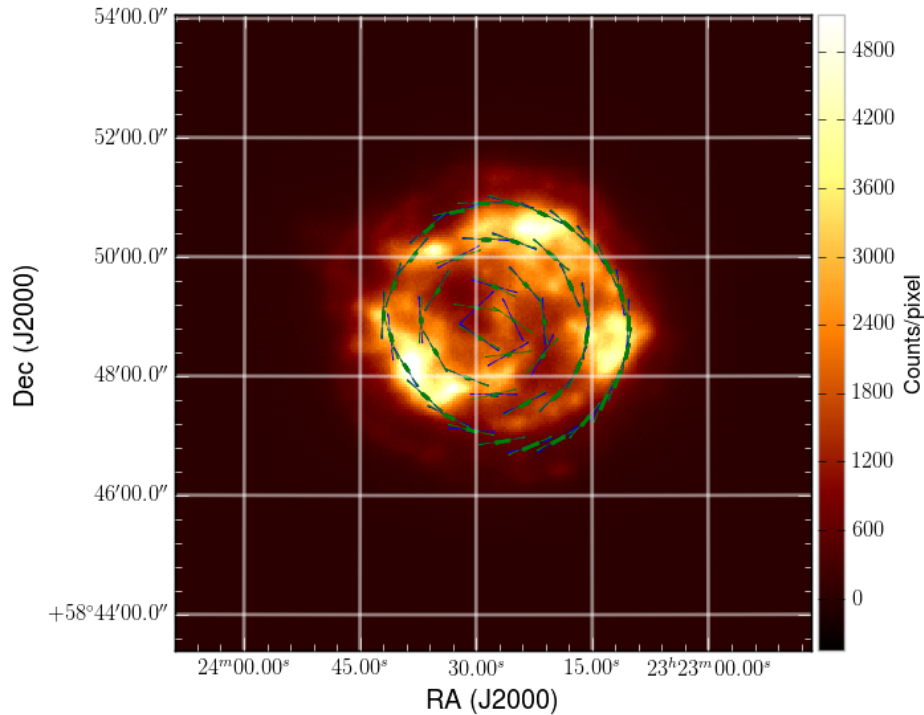




On the other hand, spatially- and energy-resolved polarimetry would in this case reveal much of the richness in the original polarization pattern. Below is an example of the azimuthal distributions in the two energy bands for the circular region of interest indicated by the white circle in the left plot. (The green and blue lines in the ROI indicate the reconstructed polarization angle.) 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, as shown in the figure below. (Note that at the center of the image the polarization is close to zero and the arrows have little meaning.)



And below is a short animation illustrating the whole thing.

The Crab pulsar

Input model

Full source model definition in `ximpol/config/crab_pulsar.py`.

The input model consists of tabulated models for the phase-resolved optical polarization angle and degree and X-ray spectral parameters. The main reference we used for the compilation is Weisskopf, M. C. et al., “*Chandra Phase-Resolved X-Ray Spectroscopy of the Crab Pulsar*”, *Astrophys. J.* 743 (2011) 139–149 and essentially all the data points come from Figure 4 of this paper. The optical polarimetry data are from Słowikowska, A. et al., “*Optical polarization of the Crab pulsar: precision measurements and comparison to the radio emission*”, *MNRAS*, 397, Issue 1 (2009) 103–123.

For any specific phase value the polarization angle and degree are energy-independent (and, in the absence of X-ray data, we just assume that they are the same as the values measured in optical) and the spectral model is a simple power law (with the normalization and spectra depending on the phase). The sinusoidal parametrization of the power-law index as a function of the pulsar phase, mutated from the reference above, is somewhat unphysical, but from our prospective is a good test of the simulation chain.

The input spatial model is simply a point source. The timing ephemeris is taken from Weisskopf, M. C. et al., “*Chandra Phase-Resolved X-Ray Spectroscopy of the Crab Pulsar*”, *Astrophys. J.* 601 (2004) 1050–1057.

Simulation output

Generation/analysis pipeline in `ximpol/examples/crab_pulsar.py`.

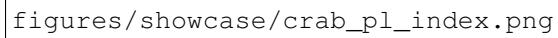
All the plots below refer to a 100 ks simulation of the Crab pulsar. (It is worth emphasizing that in this particular context we only simulate the pulsar—not the nebula. Simulating the Crab complex can surely be done within the current capabilities of the framework, but for this particular example we did not want to make the downstream analysis too complicated.)

We splitted the sample into 20 phase bins and created counts spectra (i.e., PHA1 files) and modulation cubes for each of the phase bins.

We fitted the count spectra in each phase bin with XSPEC, and the fitted parameters track reasonably well the input model. We might be seeing a slight bias in the values of the spectral index toward sistematically higher values, but overall things do look good.

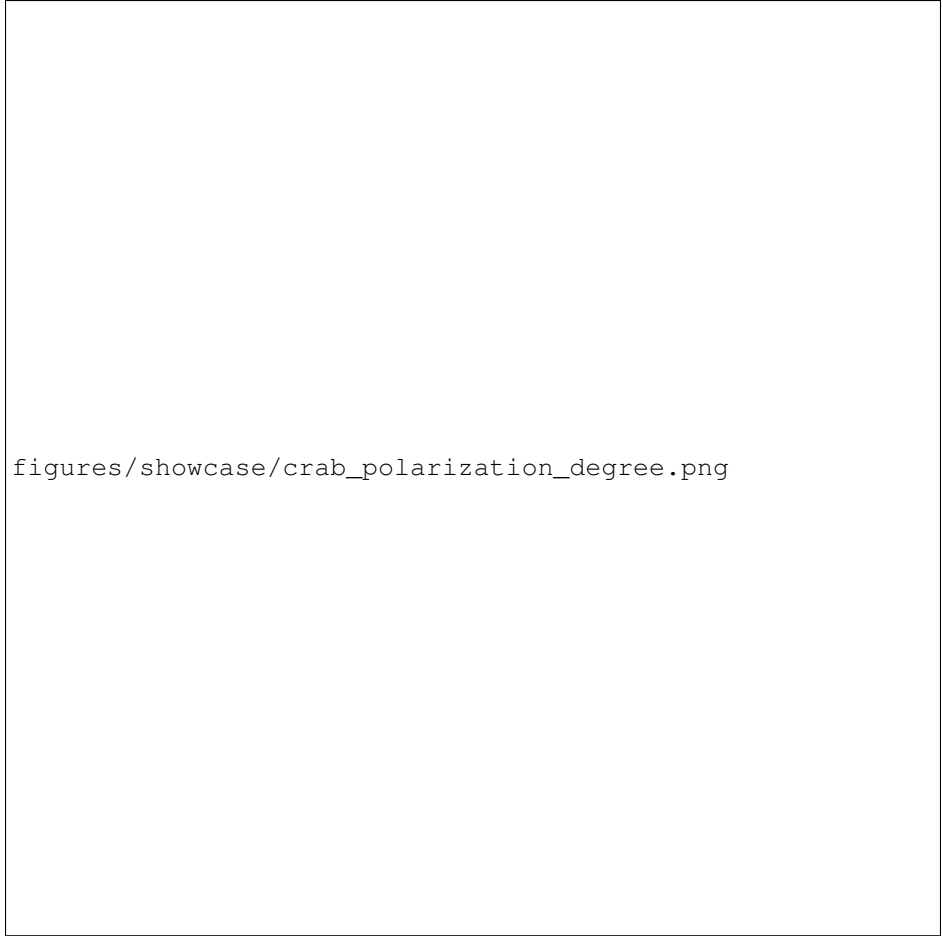


figures/showcase/crab_pl_norm.png

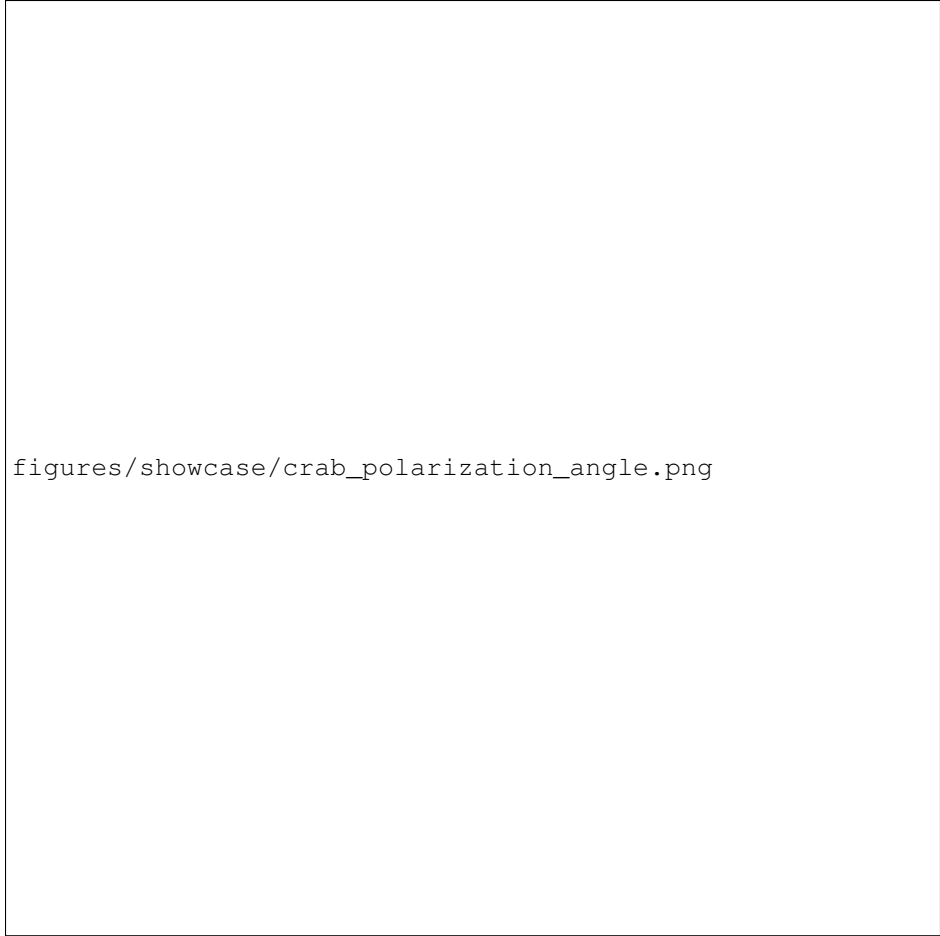


figures/showcase/crab_pl_index.png

We measure the average polarization degree and angle in each phase bin (we remind that the input polarization model is energy-independent) and, again, model and simulation agree well across all the phase values.



figures/showcase/crab_polarization_degree.png



figures/showcase/crab_polarization_angle.png

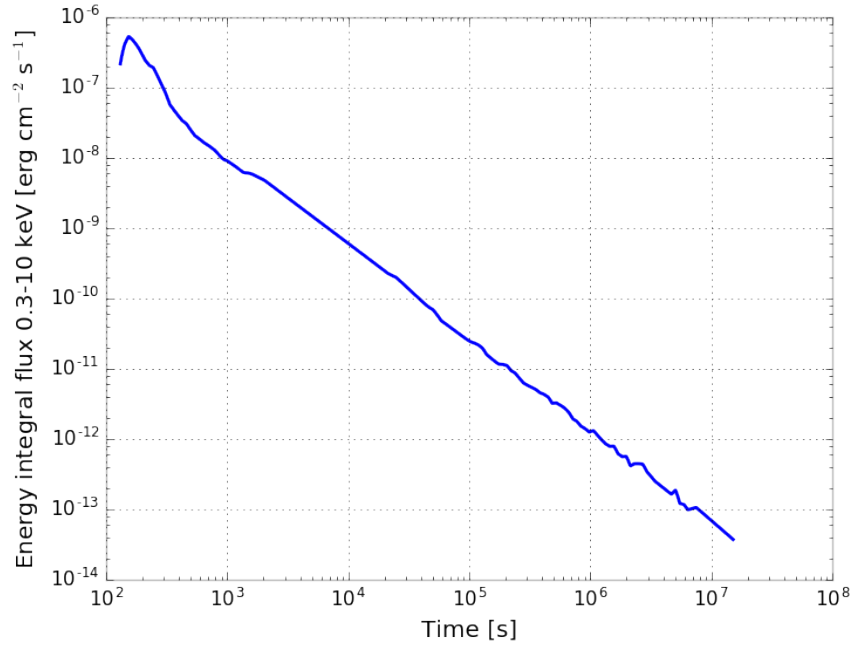
We have also simulated 100 ks of the Crab pulsar together with the nebula. Below is a short animation of the Crab complex illustrating the imaging capabilities of XIPE.

GRB 130427A

Input model

Full source model definition in `ximpol/config/grb130427_swift.py`.

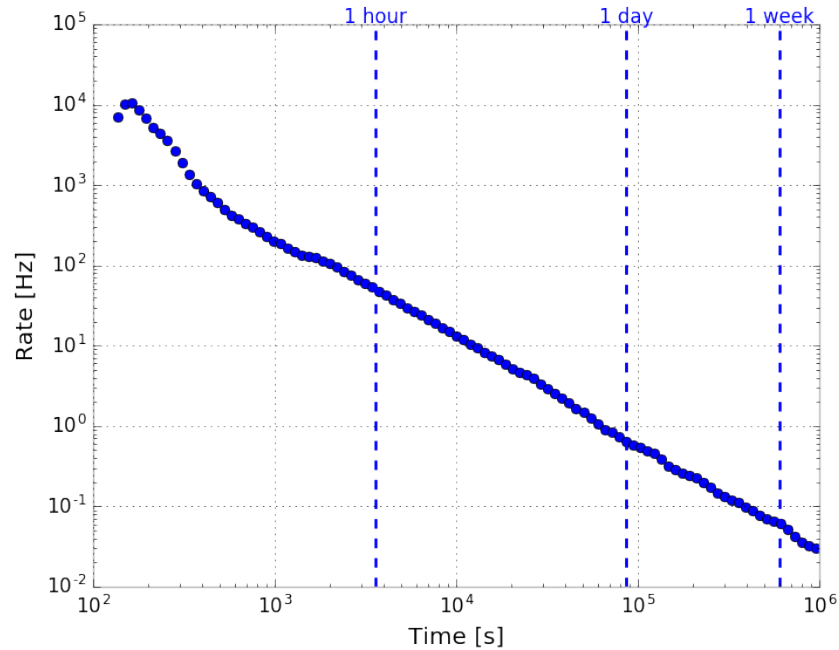
This example is meant to illustrate the simulation of a time-dependent source model. GRB 130427A (at $z = 0.34$) is one of the brightest GRBs ever observed in X-rays. The data points to build the light curve (shown below) are taken from the [Swift XRT light-curve catalog](#).



For the polarization, we made up a model where the polarization degree is decreasing with time (starting at about 40% and reaching about 10% 1 Ms after the burst) and the polarization angle is constant (see the input models in the simulation output below).

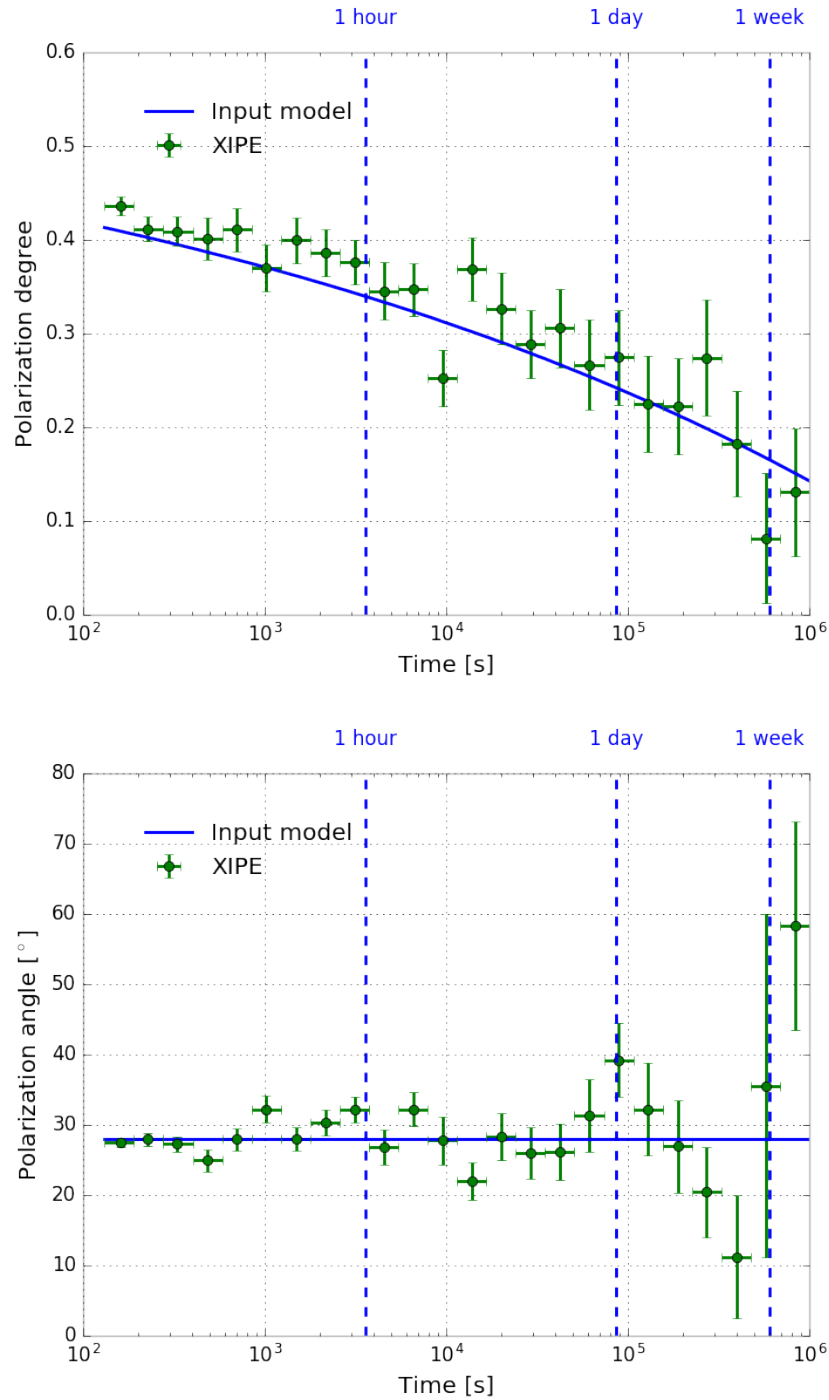
Simulation output

We simulated a 1 Ms observation of the GRB with XIPE. The plot below shows the count rate as a function of time.



We subselected the event file into non-overlapping time slices whose width is increasing logarithmically with time. Below are the reconstructed polarization degree and angle in each of the time bins, with the corresponding input

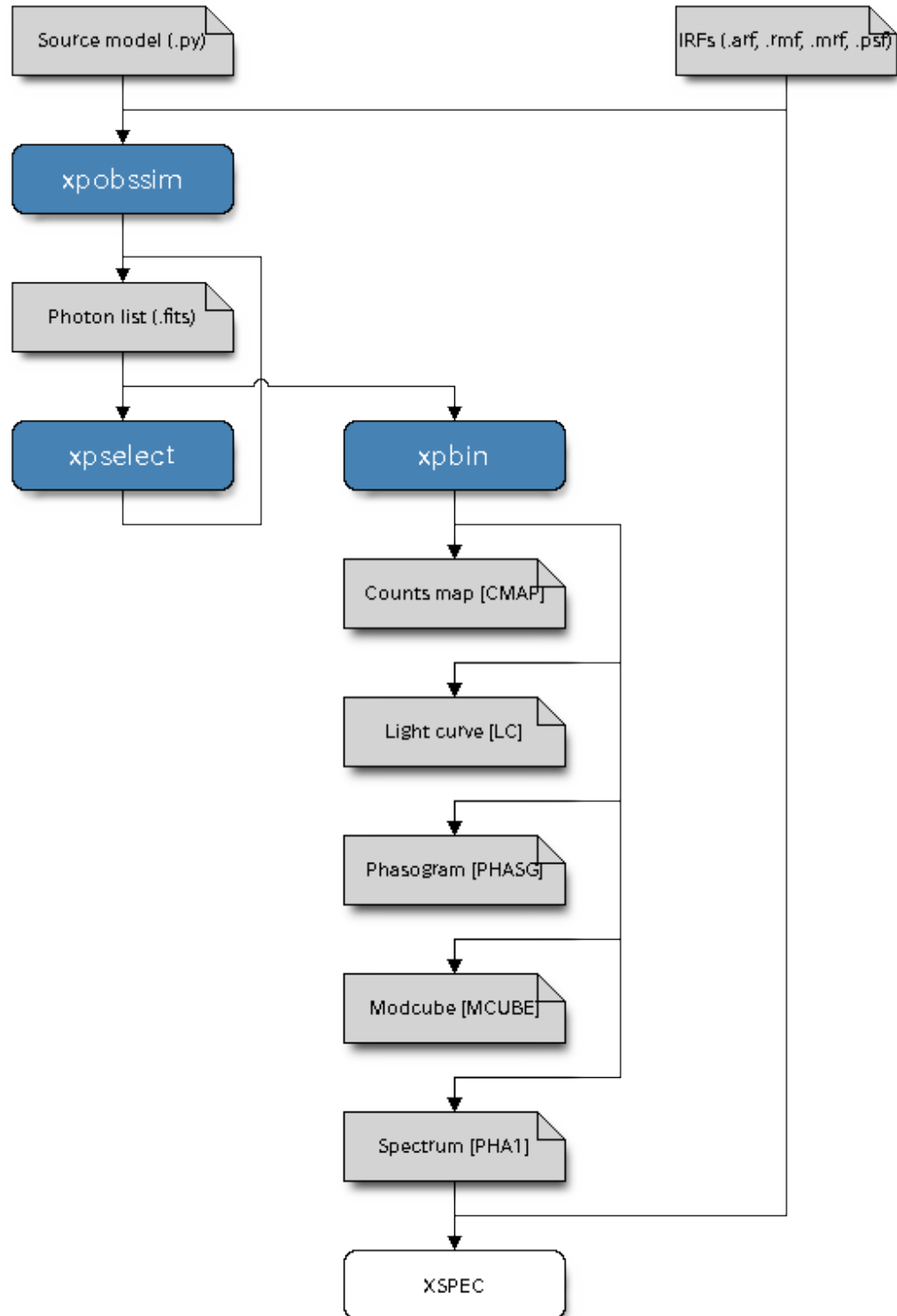
model overlaid. Most notably, if we were able to repoint the telescope to the GRB direction within a day from the burst, we would still be sensitive to a 10–20% polarization degree in an intergration time of the order of 100 ks.



Architectural overview

The vast majority of the simulation and data preparation facilities implemented in ximpol are made available through three main *executables*, as illustrated in the block diagram below:

- `xpobssim`: given a *source model* and a set of *instrument response functions*, it produces a photon list corresponding to a given observation time. The persistent incarnation of a photon list (that we call an *event file*) is binary FITS table whose format is defined in `ximpol.evt.event`.
- `xpselect`: allows to select subsamples of photons in a given event file, based on the event energy, direction, time or phase (and, really, any of the columns in the photon list), producing a new (smaller) event file.
- `xpbin`: allows to bin the data in several different flavors, producing counts maps and spectra, light curves, phasograms and *modulation cubes* (i.e., histograms of the measured azimuthal distributions in multiple energy layers).



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 ximpol simulation and analysis tools are fully configurable via command-line and the corresponding signatures are detailed here. In addition, ximpol 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 *xpselect* 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).

Implementation details

The basic flow of the simulation for a single model component is coded in `ximpol.srcmodel.roi.xModelComponentBase.rvs_event_list()`. Notably, 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.

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 $A_{\text{eff}}(E)$, we calculate the count spectrum as a function of the energy and time:

$$\mathcal{C}(E, t) = \mathcal{S}(E, t) \times A_{\text{eff}}(E) \quad [\text{s}^{-1} \text{ keV}^{-1}].$$

2. We calculate the light curve of the model component (in counts space) by integrating over the energy:

$$\mathcal{L}(t) = \int_{E_{\min}}^{E_{\max}} \mathcal{C}(E, t) dE \quad [\text{Hz}].$$

3. We calculate the total number of expected events N_{exp} by integrating the count rate over the observation time:

$$N_{\text{exp}} = \int_{t_{\min}}^{t_{\max}} \mathcal{L}(t) dt$$

and we extract the number of *observed* events N_{obs} according to a Poisson distribution with mean N_{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_{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 length N_{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 corresponding 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

$$\begin{aligned} \hat{\xi} &= \hat{P}(\hat{E}, \hat{t}, \hat{\text{RA}}, \hat{\text{Dec}}) \times \mu(\hat{E}) \\ \hat{\phi}_0 &= \hat{\alpha}(\hat{E}, \hat{t}, \hat{\text{RA}}, \hat{\text{Dec}}), \end{aligned}$$

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.

Quick start

Warning: This is now already obsolete and we should revamp it to the latest ximpol versions.

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

The main Monte Carlo simulation script is [ximpol/bin/xpobssim](#) and its signature is

```
ximpol/bin/xpobssim.py
usage: xpobssim.py [-h] [--outfile OUTFILE] --configfile CONFIGFILE
                  [--irfname IRFNAME] [--duration DURATION] [--tstart TSTART]
                  [--seed SEED] [--vignetting {True,False}]
                  [--clobber {True,False}]
```

Assuming that the current working directory is the ximpol root folder, the command

```
ximpol/bin/xpobssim.py --configfile ximpol/config/single_point_source.py --duration_
↪10000
```

should produce an event (FITS) file with a 10 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 and angle (correctly folded with all the instrument response functions: effective area, modulation factor, energy dispersion and point-spread function). The format definition for the event file is in [ximpol/evt/event.py](#).

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

```
ximpol/bin/xpbin.py ximpol/data/single_point_source.fits --algorithm MCUBE --emin 2. -
↪-emax 8. --ebins 1
```

that should produce a new FITS file (called modulation cube), containing the MDP value (at 99%) and the histogram of the azimuthal distribution of photoelectrons emission for every bin.

The binned output file can be easily visualized using the `xpviewbin` tool:

```
ximpol/bin/xpviewbin.py ximpol/data/single_point_source_mcube.fits
```

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

```
ximpol/bin/xpbin.py ximpol/data/single_point_source.fits --algorithm PHA1
```

Finally we can feed the binned file (along with the corresponding `.arf` and `.rmf` response functions) into XSPEC and recover the input parameters of our source.

```
ximpol/bin/xpxspec.py ximpol/data/single_point_source_pha1.fits
```

Note that `xpspec.py` is an example of how to use `pyXspec`, unfortunately not all of the XSPEC capabilities have been implemented in `pyXspec` (for example how to save a plot) so it is left to the user to decide whether to use XSPEC or `pyXSPEC` for the spectral analysis.

Below is the output from XSPEC on `single_point_source_phal.fits`:

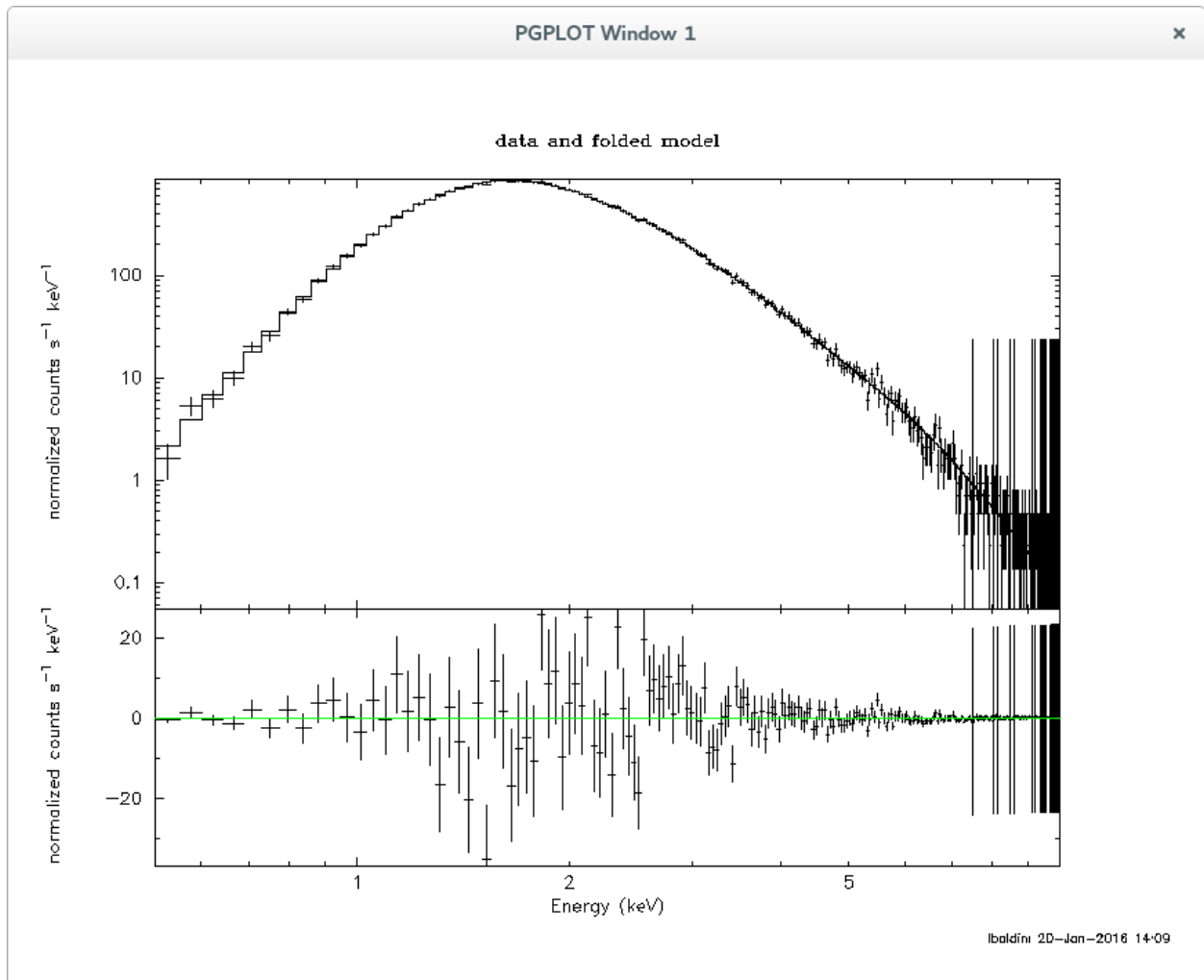
Model powerlaw<1> Source No.: 1 Active/On						
Model	Model	Component	Parameter	Unit	Value	
par	comp					
1	1	powerlaw	PhoIndex		1.99866	+/- 7.42715E-04
2	1	powerlaw	norm		9.99055	+/- 7.66357E-03

Fit statistic : Chi-Squared = 188.76 using 191 PHA bins.

Test statistic : Chi-Squared = 188.76 using 191 PHA bins.

Reduced chi-squared = 0.99875 **for** 189 degrees of freedom

Null hypothesis probability = 4.911785e-01



Installation

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 side 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).
- [APLpy](#) (optional): a Python module for plotting astronomical imaging data in FITS format.
- [pyregion](#): a Python module to parse ds9 region files
- [PyXspec](#) (optional): the Python binding for XSPEC.

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 aplpy
>>> import xspec
>>> import pyregion
```

(The last two are not strictly necessary to run a simulation, but in the long run you probably want to have them.) If any of the required packages fails to import, take a deep breath and fix the issue before you move on.

Using anaconda

Since the all the python third-party packages that we use are under active development (and for some of them there have been some non-trivial interface changes across the most recent version), you do care about the package versions that you have installed. For instance, we do need the ability to evaluate spline on multidimensional arrays that was added in SciPy 0.16 but was not there in SciPy 0.14. The bottomline is that, depending on the exact version of the packages that you have installed on your system, ximpol might or might not work correctly.

One possible OS-independent way to get a fully fledged ecosystem in which ximpol can work is to use [anaconda](#). You should be able to get up and running, in terms of the pre-requisites to run ximpol, in a matter of minutes following the instructions in there.

Note that the default anaconda installer does not come with all the packages that you need (most notably, `aplpy` is missing), but you can easily install whatever you need after the fact, e.g.,

```
pip install aplpy
pip install pyregion
```

If you are uncertain on what to do, you might want to try anaconda first, as this might be the less painful solution.

For XSPEC and the corresponding Python bindings, refer to the [HEASOFT download page](#).

More OS-specific resources

There are obviously other ways to install the prerequisites, e.g., by doing everything manually or use a package manager that your OS makes available.

Warning: We need to add specific information for different platforms (e.g., Windows and Mac) and GNU/Linux distros. Any help from anybody is appreciated!

For GNU/Linux in the Fedora flavor, for instance, you would do something like

```
dnf install numpy scipy python-matplotlib python-astropy APLpy
```

Downloading the code

The easiest (though not the best) way to get the code is by directly downloading the zip or tar file for the latest tag (or the tag you are interested in) from the [github release page](#). Unzip the archive in your favorite folder and setup the environment as detailed in the next section.

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 page about the code development.

Warning: At some point we should provide a working cross-platform installation mechanism through the setup.py file, and possibly packages/installers for the operating systems we use. Again, any help would be appreciated.

Basic environment

The only thing you have to do is to make sure that the root folder of the repository is included in the \$PYTHONPATH environmental variable. You might also want to add *ximpol/bin* to the \$PATH environmental variable, so that you have the executables off hand. Here is an example for users of the Bourne shell (sh, ash, ksh, and bash):

```
export PYTHONPATH=/data/work/xipe/ximpol:$PYTHONPATH
export PATH=/data/work/xipe/ximpol/ximpol/bin:$PATH
```

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

```
>>> import ximpol
```

without getting back an error message like this

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

again, you should be all set. If not, I am sorry to say, you really do have to fix this before moving on.

Source models

Warning: This need to be though through and documented.

While we don't have all the machinery in place to define source models, yet, we do have a fully functional (though rudimentary) configuration file that illustrate how we are planning on implementing complex model with non trivial energy- and time-dependence of the underlying parameters.

The file `srcmodel/config/stationary_point_pl.py` defines a stationary source with a power-law spectrum and energy- and time-independent polarization degree and angle, but in a way that is suggestive how we'll go about defining more complex models.

In the mid term, the basic data structures that we envisage to define arbitrary models is something along the lines of

- a `xSourceComponent` class, encapsulating all the morphological, spectral and polarization properties of a given component;
- a `xSource` class, containing a list of `xSpectralComponents`;
- a `xSource List` class, containing a list of sources (i.e., our models).

(Names might change.)

Instrument response functions (IRFs)

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

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 [here](#) for more details).

We identify four different types of response functions, listed in the following table.

IRF Type	Extension	ximpol module	ximpol class
Effective area	<i>.arf</i>	<code>ximpol.irf.arf</code>	<code>ximpol.irf.arf.xEffectiveArea</code>
Point-spread function	<i>.psf</i>	<code>ximpol.irf.psf</code>	<code>ximpol.irf.psf.xPointSpreadFunction</code>
Modulation factor	<i>.mrf</i>	<code>ximpol.irf.mrf</code>	<code>ximpol.irf.mrf.xModulationFactor</code>
Energy dispersion	<i>.rmf</i>	<code>ximpol.irf.rmf</code>	<code>ximpol.irf.rmf.xEnergyDispersion</code>

(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](#)).

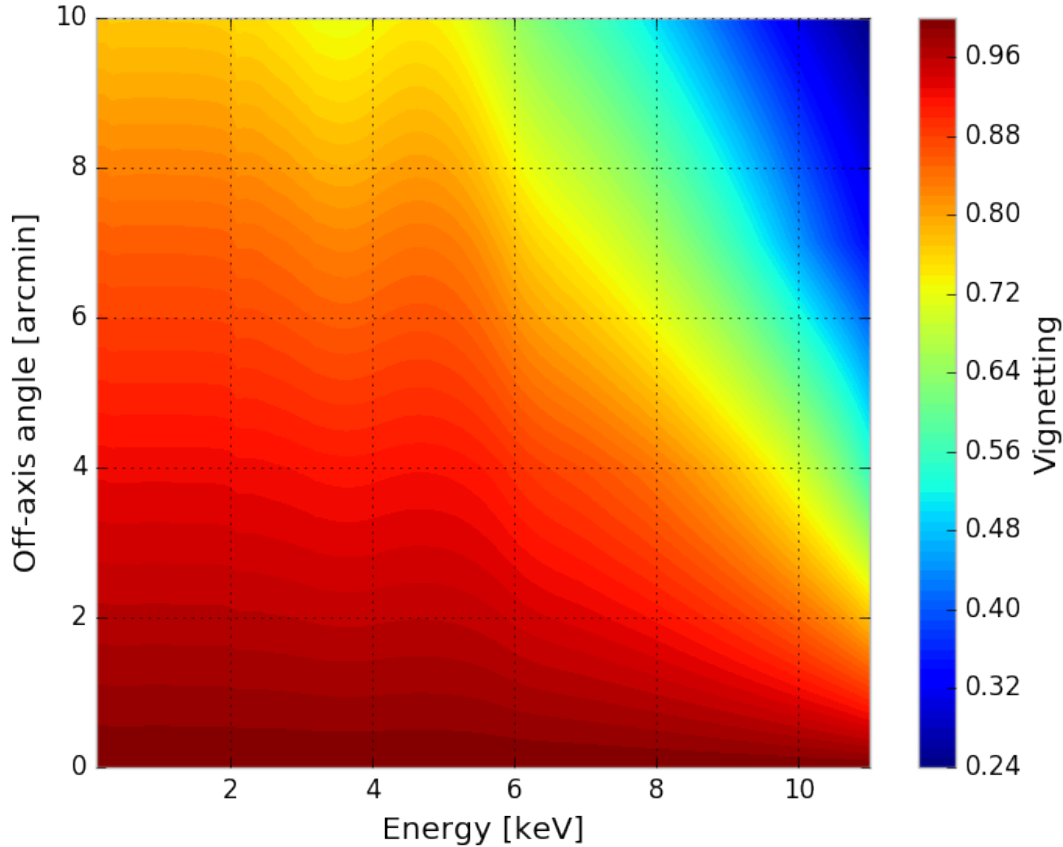
ximpol provides facilities for generating, reading displaying and using IRFs, as illustrated below.

Creating an IRF set: XIPE

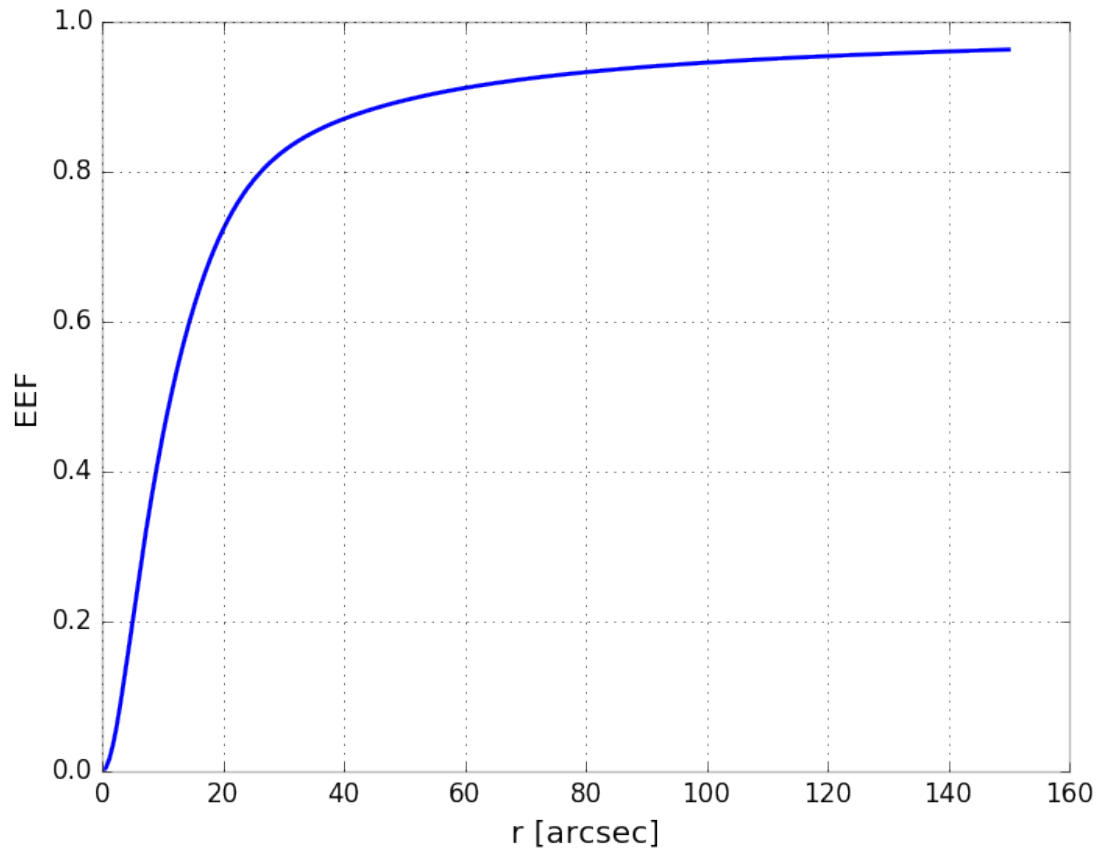
The python module `ximpol.detector.xipe.py` generates a full set of instrument response functions for the baseline XIPE configuration, and it's a sensible example of how to go about generating IRFs for arbitrary detector configurations.

The XIPE effective area (shown in the figure below) is the product of two distinct pieces:

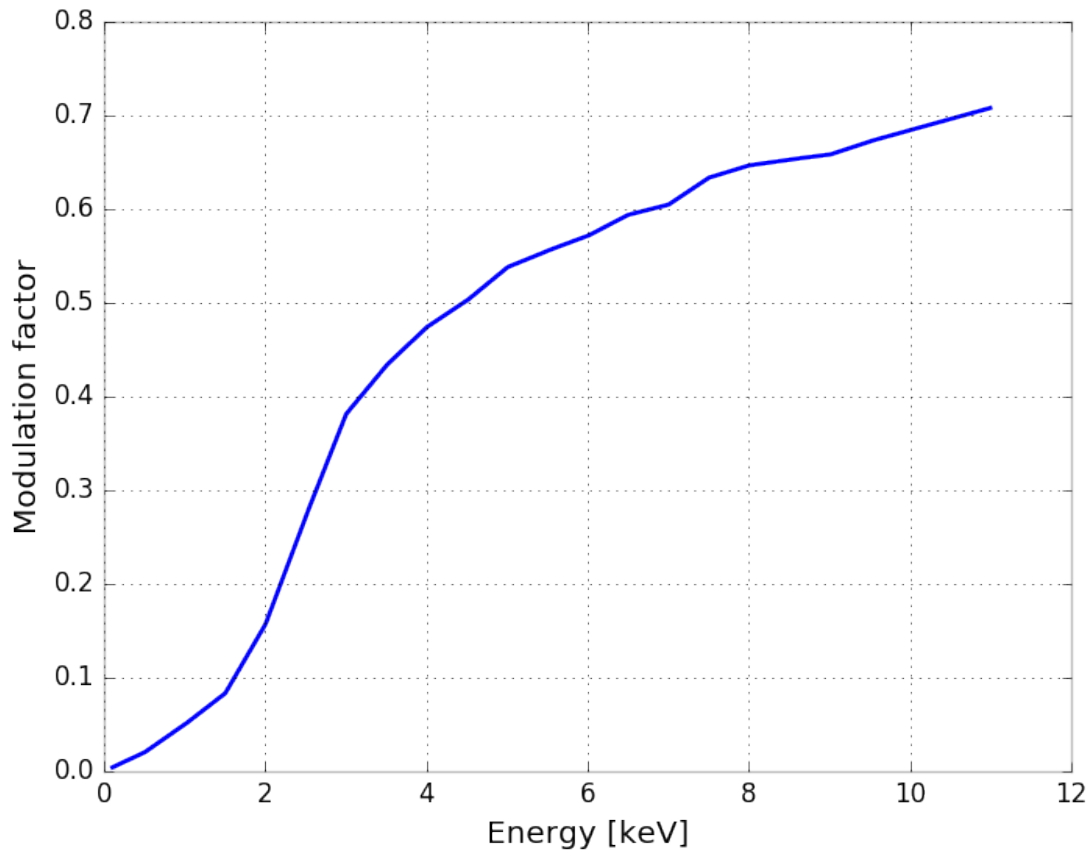
- The collecting area of the XIPE optics, available in `aeff_optics_xipe_m4_x3.asc` in ascii format. (This is the overall area for the three telescopes.)
- The quantum efficiency of the MGD in the baseline configuration (Ne/DME 80/20, 1 atm, 1 cm absorption gap), available in `eff_hedme8020_1atm_1cm_cuts80p_be50um_p_x.asc`. (Mind this includes the effect of the 50 μm Be window and that of an energy-independent cut on the quality of the event to achieve the modulation factor predicted by the Monte Carlo.)



The point-spread function (PSF) is apparently difficult to estimate accurately based on the design of the optics, as it depends substantially on the defects of the surfaces (i.e., you need some metrology on the actual mirrors). For the XIPE baseline design we start by assuming a gaussian + King profile with a HEW of 15 arcsec, with the exact same parametrization and parameters values of [Fabiani et al. \(2014\)](#). For completeness, another paper of interest is [Romano et al. \(2005\)](#). Below is the actual profile of the PSF, which should be identical to Figure 6 of Fabiani et al. (2014); the parameter values were taken from Table 2 (@ 4.51 keV).

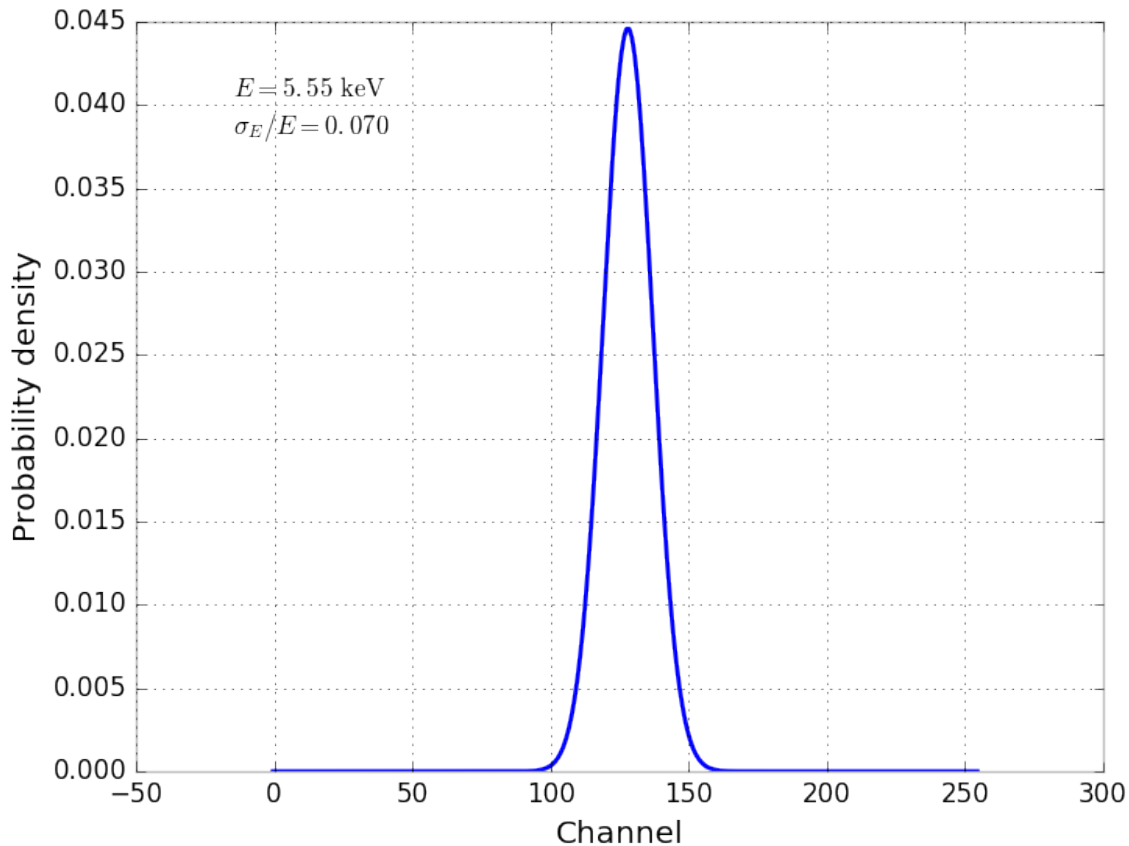


The modulation factor as a function of the energy for the relevant GPD configuration is tabulated in `modfact_hedme8020_1atm_1cm_mng.asc` and shown in the figure below.



The energy dispersion (i.e., the content of the .mrf FITS files) contains a two-dimensional table of the redistribution matrix in the PHA-true energy space (the `MATRIX` extension), and a one-dimensional table containing the correspondence between the PHA channels and the energy (the `EBOUNDS` extension).

For the time being we are using a simple gaussian parametrization whose FWHM as a function of the energy is tabulated in [eres_fwhm_hedme8020_1atm_1cm.asc](#). We're using 256 channels between 0 and 11 keV (or 0.043 keV/channel), which seems to sample the energy dispersion adequately across the entire energy range (the typical FWHM being 1 keV or 25 channels). The `MATRIX` and `EBOUNDS` are shown in the top panel of the figure below. The bottom panel shows slices (i.e., the pdf as a function of channel) of the `MATRIX` at 3.73 keV and 7.36 keV.



Loading (and using) IRFs

All the instrument response functions are stored in FITS files (living in *ximpol/irf/fits*) and have suitable interfaces to load and use them. You can load the baseline XIPE effective area by running the following:

```
>>> import os
>>> from ximpol import XIMPOL_IRF
>>> from ximpol.irf.arf import xEffectiveArea
>>> file_path = os.path.join(XIMPOL_IRF, 'fits', 'xipe_baseline.arf')
>>> aeff = xEffectiveArea(file_path)
```

The same works for the other three IRFs. Note that `XIMPOL_IRF` is a convenience variable that allows you to avoid machine-dependent absolute paths and is always pointing to *ximpol/irf*, no matter where the package is installed. There's many other such variables defined in *ximpol.__init__.py*. You can also load all four of the response functions at a time:

```
>>> from ximpol.irf import load_irfs
>>> aeff, psf, modf, edisp = load_irfs('xipe_baseline')
```

The IRFs are objects that can be evaluated at any given point—compare the outputs below with the plots at the top of the page.

```
>>> # Print the values of the effective area and the modulation factor and 3 keV
>>> print(aeff(3.))
```

```
>>> 164.870643616
>>> print(modf(3.))
>>> 0.380231711646
>>> # Print the value of the PSF at 20 arcsec
>>> print(psf(20.))
>>> 0.000131813525114
```

The energy dispersion is a somewhat more complicated object, it consists of a two-dimensional redistribution matrix and one-dimensional table containing the correspondence between the PHA channels and the energy. Below are a few examples of methods which the energy dispersion object has:

```
>>> # plot the 2-dimensional redistribution matrix
>>> edisp.matrix.plot()
>>> # plot a slice of the matrix at 3 keV
>>> edisp.matrix.slice(3.).plot()
>>> # plot the correspondence between the PHA channels and the energy
>>> edisp.ebounds.plot()
>>> # print the energy (in keV) corresponding to PHA channel 23
>>> print(edisp.ebounds(23))
>>> 1.009765625
```

Note also that the IRFs are internally represented as arrays and therefore we can also evaluate the response functions over arbitrary grids of points in one pass, e.g

```
>>> import numpy
>>> energy = numpy.linspace(1, 10, 10)
>>> print(energy)
>>> [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
>>> print(aeff(energy))
>>> [ 4.9761498 305.13298991 164.87064362 68.54330826 31.6964035
 16.27694702  7.43255687  3.34847045  1.49684757  0.62106234]
```

All of the response functions have the plot method:

```
>>> aeff.plot()
```

Most importantly, IRFs have facilities to throw random numbers according to suitable distributions to generate list of events, but this is covered in another section.

The small application *bin/xpirfview.py* provides a common interface to display the content of the IRF FITS files

```
ximpol/bin/xpirfview.py ximpol/irf/fits/xipe_baseline.arf
```

Code development

Up and running with github

[git](#) is a distributed version control system and [github](#) is the web hosting service that we use to develop and maintain ximpol.

You will need to register to github and ask [Luca Baldini](#) to be granted write-access to the repository. [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
```

Cloning the repository

(This will work on GNU/Linux and MAC. We should provide instructions for Windows as well.)

Cloning the repository is as easy as typing

```
git clone git@github.com:lucabaldini/ximpol.git
```

(mind this will create a ximpol 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 you 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 “Edit profile”, “SSH keys” (on the left-hand side “Personal setting” menu), “Add SSH key” 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 you public key in `~/.ssh/id_rsa.pub`)

Basic environment

You have to make sure that the root folder of the repository is included in the `$PYTHONPATH` environmental variable.

You might also want to add `ximpol/bin` to the `$PATH` environmental variable, so that you have the executables off hand. Here is an example:

```
export PYTHONPATH=/data/work/xipe/ximpol:$PYTHONPATH
export PATH=/data/work/xipe/ximpol/ximpol/bin:$PATH
```

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 assignment.

- 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 ximpol 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 [here on github](#).

Documenting the code

We use [sphinx](#) to generate the ximpol [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.

It won't take more than a few minutes to get acquainted to the basic rules, and the [codestyle.py](#) module, along with its fellow [modules/codestyle.rst](#) file, provide a minimal working example that, compiled with sphinx, would be rendered [like this](#).

You can compile and view the ximpol documentation locally by doing

```
cd doc
make html
htmlview _build/html/index.html
```

which is useful to make sure everything is in order when writing and documenting code.

The actual documentation for the latest build is hosted on [Read the Docs](#) at [this link](#). You don't really have to worry about, as that is being automatically re-built from scratch every time a code change is pushed to the main github repository. Cool, isn't it?

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 *test* folder; [test/test_codestyle.py](#) is the simplest possible unit test, while [test/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:

```
cd ximpol/test
make test
```

Continuous integration

The ximpol github repository is linked to the [Travis CI](#) (continuous integration) framework. The whole package is checked out every time a change is pushed to github, and the unit tests are run. The build status is shown in the README file on the main ximpol github page.

System tests

Release notes

- 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 xpbins.
- 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 parameters 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 xpmddp.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 evt.binning.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 (closing 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.)
- evt.binning.py 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 xpbins 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 redshift 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 interstellar absorption added.
- Fix for issue #103.
- Some significant refactoring of the xpmdb code.
- xpmdb added to the pipeline (closing issue #137).
- Added a unit test comparing the xpmdb.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 to 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 in evt/binning.py
- 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

- xpmddp.py adapted (via brute-force) to periodic sources.
- MDP information added to the xpbm 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**.py 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 evt/binning.py 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 `ximpol.evt.binning.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 unidimensional 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 step 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 source 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).
- xpbins 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 xpbins.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

- ebinalg 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 xpbins.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 (xpsrccords.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 xpbins.

ximpol (0.18.0) - Wed, 03 Feb 2016 15:13:52 -0800

- More work on xpbins.py (closing issues #42 and #52).

ximpol (0.17.0) - Tue, 02 Feb 2016 15:41:14 -0800

- Major refactoring of xpbins.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 xpbins.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.
- xpbins.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 xpobbsim.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 underscore to avoid name conflicts with the corresponding 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 the 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.

About ximpol

Team

- Luca Baldini (luca.baldini@pi.infn.it)
- Nicola Omodei (nicola.omodei@stanford.edu)
- Melissa Pesce-Rollins (melissa.pesce.rollins@pi.infn.it)

- Niccolo' Di Lalla (niccolo.dilalla@pi.infn.it)
- Alberto Manfreda (alberto.manfreda@pi.infn.it)
- Michela Negro (michela.negro@to.infn.it)

License

ximpol is free software licenced under the [GNU General Public License version 3](#). For more information, see the [LICENSE](#) file included in the distribution.

CHAPTER 2

Indices and tables

- `genindex`
- `modindex`
- `search`