pyXsis

Release 0.0.1

Lia Corrales

Aug 25, 2021

CONTENTS

pyXsis = Python X-ray Spectral Interpretation System

This is a toy python code for manipulating high resolution X-ray Spectra.

This python library owes it's inspiration to the Interactive Spectral Interpretation System written by J. Houck and J. Davis (creator of S-lang), as well as long-time users and contributors M. Nowak, J. Wilms, and the group at Remeis Observatory. Thank you also to the entire Chandra HETG group at MIT for personal help throughout the years with interpreting high resolution X-ray spectra.

This library also directly utilizes code from the clarsach repo written by Daniela Huppenkothen, which reverse-engineered the response file treatment provided by the standard high energy software package, XSPEC.

CHAPTER

ONE

INSTALLATION

1.1 Install Instructions

1.1.1 Requirements

- Astropy
- Specutils
- Numpy
- Matplotlib

1.1.2 Install development version

You can download and install pyXsis from the Github repository:

```
git clone https://github.com/eblur/pyxsis.git
cd pyxsis
python setup.py install
```

1.2 Testing the Installation

To run the test notebooks in pyxsis/tests/notebooks, you will need to download the test HETG data.

First, download the tar ball from https://doi.org/10.5281/zenodo.2528474

Next, copy the downloaded file to your pyxsis folder and unpack the tar ball:

```
tar -xvf pyxsis_test_data.tar
```

You're now ready to run the notebooks!

1.3 Quick Start Guide

To open a .pha file with pyXsis, you can follow these commands:

```
import pyxsis
my_spectrum = pyxsis.io.load_chandra_hetg('my_HETG_file.pha')

import matplotlib.pyplot as plt
ax = plt.subplot(111)
pyxis.plot_counts(ax, my_spectrum, xunit='keV')
```

CHAPTER

TWO

PACKAGE BASICS

2.1 1D Spectrum Classes

This package provides several 1D spectrum classes and funcitons to support common tasks used in visualizing and interpreting X-ray spectra.

2.1.1 XraySpectrum1D

The basic functionality to load and visualize X-ray spectra is provided with the XraySpectrum1D class.

To initialize XraySpectrum1D, you must input the counts histogram data directly.

class pyxsis.xrayspectrum1d.**XraySpectrum1D**(*bin_lo*, *bin_hi*, *counts*, *exposure*, *arf=None*, *rmf=None*, ***kwargs*)

Spectrum properties specific to X-ray data

Attributes

Inherits from specutils Spectrum1D object. The following inputs are stored as additional attributes.

bin_lo [astropy.Quantity] The left edges for bin values

bin_hi [astropy.Quantity] The right edges for bin values

counts [astropy.Quantity] Counts histogram for the X-ray spectrum

exposure [astropy.Quantity] Exposure time for the dataset

arf [ARF object] Telescope response file describing the effective area as a function of photon energy.

rmf [RMF object] Telescope response file, a 2D matrix, describing the detector signal (pulse heights) distribution as a function photon energy.

rest_value [astropy.units.Quantity, default 0 Angstrom] See Spectrum1D rest_value input

apply_response(*mflux*, *exposure=None*)

Given a model flux spectrum, apply the response. In cases where the spectrum has both an auxiliary response file (ARF) and a redistribution matrix file (RMF), apply both. Otherwise, apply whatever response is in RMF.

The model flux spectrum *must* be created using the same units and bins as in the ARF (where the ARF exists)!

Inputs

mflux [astropy.units.Quantity] A list or array with the model flux values, typically with units of phot/s/cm^-2

exposure [astropy.units.Quantity (default: None)] By default, the exposure stored in the ARF will be used to compute the total counts per bin over the effective observation time. In cases where this might be incorrect (e.g. for simulated spectra where the pha file might have a different exposure value than the ARF), this keyword provides the functionality to override the default behaviour and manually set the exposure time to use.

Returns

model_counts [numpy.ndarray] The model spectrum in units of counts/bin

If no ARF file exists, it will return the model flux after applying the RMF If no RMF file exists, it will return the model flux after applying the ARF (with a warning) If no ARF and no RMF, it will return the model flux spectrum (with a warning)

assign_arf(*arf_inp*, ***kwargs*)
Assign an auxiliary response file (ARF) object to the XraySpectrum1D object

Inputs

arf_inp [string] File name for the area response file (FITS file)

Returns

Modifies the XraySpectrum1D.arf attribute

assign_rmf(*rmf_inp*)
Assign a redistribution matrix file (RMF) object to the XraySpectrum1D object

Input

rmf_inp [string] File name for the response matrix file (FITS file)

Returns

Modifies the XraySpectrum1D.rmf attribute

2.1.2 XBinSpectrum

This is a subclass of XraySpectrum1D that provides extra attributions and functions for grouping the spectrum bins.

class pyxsis.binspectrum.**XBinSpectrum**(**args*, ***kwargs*)

assign_bkg(*bkgspec*, *format='chandra_hetg'*, *colname='COUNTS'*)
Assign a background spectrum.

Inputs

bkgspec : str or XBkgSpectrum

If a string, loads background from the specified FITS file. Otherwise, stores the input in the XBinSpectrum.bkg attributes.

format : str

Specifies the format of the background file.

If 'chandra_hetg', runs *XBkgSpectrum.load_HETG*. Otherwise, runs *XBkgSpectrum.load*

colname : str

Specifies the column name of the relevant counts histogram when running *XBkgSpectrum.load*

binned_bkg(*bin_unit=None*, *use_backscale=True*)
Return the background spectrum using the same spectral binning.

Inputs

bin_unit [Astropy unit (default:None)] If not None, returns the bin edges in the desired units

use_backscale [bool] If True, returns the background scaled by the backscal value.

Returns

bin_lo [astropy.Quantity] Lower edges for the new background bins

bin_hi [astropy.Quantity] Higher edges for the new background bins

counts [astropy.Quantity] Counts for the binned background histogram

cts_err [astropy.Quantity] Error on the new background bins

binned_counts(*bin_unit=None*, *error_type='Gauss'*, *subtract_bkg=False*, *use_backscale=True*)
Returns the binned counts histogram from the noticed spectral region.

Inputs

bin_unit [Astropy unit (default:None)] If not None, returns the bin edges in the desired units

error_type [string ['Gehrels' (default) or 'Gauss']] If 'Gauss', returns cts_err = sqrt(cts). If 'Gehrels', returns cts_err = 1.0 + sqrt(cts + 0.75)

subtract_bkg [bool] If True, supply the background subtracted region spectrum (only if a background spectrum is supplied)

use_backscale [bool] If True, scales the background by the backscal value

Returns

bin_lo [astropy.Quantity] Lower edges for the new bins

bin_hi [astropy.Quantity] Higher edges for the new bins

counts [astropy.Quantity] Counts for the new bins

cts_err [astropy.Quantity] Error on the new bins

notice_all()
Resets the notice attribute to notice the entire range.

notice_range(*bmin*, *bmax*)
Define edges for spectral regions to notice. Notices regions exclusively.

```
>>> XBinSpectrum.notice(1.0*u.keV, 3.0*u.keV)
```

will notice *only* the region between 1 and 3 keV.

Inputs

bmin [astropy.Quantity] Minimum value for the notice region

bmax [astropy.Quantity] Maxium value for the notice region

Returns

Modifies the XraySpectrum1D.notice attribute.

reset_binning()
Resets the binning attribute

2.1.3 XBkgSpectrum

This is a subclass of XBinSpectrum that provides extra functionality for scaling and applying background corrections to X-ray spectrum data.

class pyxsis.binspectrum.**XBkgSpectrum**(**args*, *backscale=1.0*, ***kwargs*)

Class for reading in background spectra. This is a subclass of specutils.XraySpectrum1D.

To load from a file:

XBkgSpectrum(from_file=, format=, colname=)

from_file [str] Name of FITS file for background spectrum

format [str] Format of FITS file (Default: 'chandra_hetg' uses the 'BACKGROUND_UP' and 'BACKGROUND_DOWN' columns to read in the data)

colname [str] Name of FITS data column that contains the counts histogram of interest. (Default: 'COUNTS')

Attributes

Inherits all attributes from specutils.XraySpectrum1D

backscale [numpy.ndarry or float] Value for scaling the background count rate to the associated source area. Defaults to 1.0

binned_counts(*notice=None*, *binning=None*, *use_backscale=True*, ***kwargs*)

Inputs

notice [ndarray, dtype=bool] Defines what regions of the spectrum to notice (Default: None, uses all of the counts histogram.)

binning [ndarray] Defines the binning for the spectrum, same method as XBinSpectrum. (Default: None, does not group any of the bins)

use_backscale [bool] If True, the background will be scaled using XBkgSpectrum.backscale

Returns a binned background spectrum

(bin_lo, bin_hi, bkg_counts, bkg counts_err) : astropy Quantity arrays

2.1.4 Binning Functions

Group by number of channels

pyxsis.binspectrum.**group_channels**(*spectrum*, *n*)

Group channels in a spectrum by a constant factor, n

Inputs

spectrum [pyxsis.XBinSpectrum] Must contain *binning* attribute (ndarray)

n [int] Integer factor for binning the spectrum channels

Returns

Modifies spectrum.binning

Group by number of counts

pyxsis.binspectrum.**group_mincounts**(*spectrum*, *mc*)

Group channels in a spectrum so that there is a minimum number of counts in each bin

Inputs

spectrum [XBinSpectrum] Must contain *binning* attribute (ndarray)

mc [int] Minimum number of counts per bin

Returns

Modifies spectrum.binning

Apply a custom binning

pyxsis.binspectrum.**bin_anything**(*x*, *binning*, *notice=None*)

Group anything according to a binning array

Inputs

x [np.array] Array to bin

binning [np.array] Bin numbers for sorting

notice [bool np.array (optional)] Array values to notice

Returns

A numpy array that holds the binned results

2.2 Instrument Response Classes

Interpreting X-ray spectra requires knowledge of the instrumental response. There are two essential types of response files. Pyxsis provides a class for handling each of them.

2.2.1 Auxiliary Response File (ARF)

The Auxiliary Response File (frequenty referred to simply as **ARF**) defines the telescope efficiency as a function of energy. This is typically a 1D file with units of [cm^2 counts / photon] as a function of photon energy. See the CXC documentation page on ARFs

class pyxsis.xrayspectrum1d.**ARF**(*e_low*, *e_high*, *eff_area*, *exposure=None*, *fracexpo=1.0*)

apply_arf(*model*, *exposure=None*)

Fold the spectrum through the auxiliary response file (ARF). The ARF is a single vector encoding the effective area information about the detector. A such, applying the ARF is a simple multiplication with the input spectrum.

Parameters

model [numpy.ndarray] The model spectrum to which the arf will be applied

exposure [float, default None] Value for the exposure time. By default, *apply_arf* will use the exposure keyword from the ARF file. If this exposure time is not correct (for example when simulated spectra use a different exposure time and the ARF from a real observation), one can override the default exposure by setting the *exposure* keyword to the correct value.

Returns

s_arf [numpy.ndarray] The (model) spectrum after folding, in counts/s/channel

static read(*filename*, *block='SPECRESP'*)
Load an ARF object from FITS file.

Inputs

filename [str] Path to the FITS file

block [str] FITS file block keyword, if the spectral response is stored under an extension other than "SPECRESP"

Returns

The ARF object that is represented by the FITS file

2.2.2 Redistribution Matrix File (RMF)

The Redistribution Matrix File (frequenty referred to simply as **RMF**) describes the probability distribution of detector pulse heights that arise when a photon interacts with the detector. It is a 2D matrix with the pulse height distribution as a function of incoming photon energy. Seee the CXC documentation page on RMFs

class pyxsis.xrayspectrum1d.**RMF**(*energ_lo=None*, *energ_hi=None*, *matrix=None*, *energ_unit=None*, *offset=0.0*, *n_grp=array([], dtype=float64)*, *f_chan=array([], dtype=float64)*, *n_chan=array([], dtype=float64)*, *detchans=0*)

apply_rmf(*model*)
Fold the spectrum through the redistribution matrix.

The redistribution matrix is saved as a flattened 1-dimensional vector to save space. In reality, for each entry in the flux vector, there exists one or more sets of channels that this flux is redistributed into. The additional arrays n_grp, f_chan and n_chan store this information:

- n_group stores the number of channel groups for each energy bin
- f_chan stores the *first channel* that each channel for each channel set
- n_chan stores the number of channels in each channel set

As a result, for a given energy bin i, we need to look up the number of channel sets in n_grp for that energy bin. We then need to loop over the number of channel sets. For each channel set, we look up the first channel into which flux will be distributed as well as the number of channels in the group. We then need to also loop over the these channels and actually use the corresponding elements in the redistribution matrix to redistribute the photon flux into channels.

All of this is basically a big bookkeeping exercise in making sure to get the indices right.

Inputs

model [numpy.ndarray] The (model) spectrum to be folded

Returns

counts [numpy.ndarray] The (model) spectrum after folding, in counts/s/channel

2.3 Telescope Specific Loaders

The *pyxsis.io* submodule provides helper functions for loading files that are specific to an X-ray observatory.

2.3.1 Chandra X-ray Observatory

To load a Chandra PHA level 1 file extracted from an HETG observation:

```
from pyxsis.io import load_chandra_hetg
spectrum = load_chandra_hetg("my_heg_m1.pha")
```

If the PHA file has the ARF and RMF filenames specified in the FITS file header ('ANCRFILE' and 'RESPFILE', respectively), then they will be automatically loaded by the pyxsis ARF and RMF classes. Otherwise, no response files will be assigned and the user must specify them.

pyxsis.io.**load_chandra_hetg**(*filename*, *arf=None*, *rmf=None*)

Load Chandra HETG spectral data from a file into a spectrum object.

Inputs

filename [str] The path to the FITS file

arf [str -or- pyxsis.xrayspectrum1d.ARF] Filename for the area response file (ARF) or a pre-loaded AreaResponse object

rmf [str -or- pyxsis.xrayspectrum1d.RMF] Filename for the response matrix file (RMF) or a pre-loaded ResponseMatrix object

Returns

pyxsis XraySpectrum1D object representing the data in the input FITS file

2.4 Plotting Functions

This package provides convenience plotting functions for visualizing and interpreting X-ray spectra. It relies on the matplotlib plotting infrastructure.

All functions take a Matplotlib AxesSubplot object object as the primary argument. We recommend setting up the Axes object using the matplotlib.pyplot.subplot function. For example, to set up axes for a single plot window:

```
import matplotlib.pyplot as plt
ax = plt.subplot(111)
```

2.4.1 Plot a counts histogram

pyxsis.plot.**plot_counts**(*ax*, *spectrum*, *xunit='keV'*, *perbin=True*, *rate=False*, *plot_bkg=False*, *subtract_bkg=True*, *use_backscale=True*, *scale_factor=1.0*, ***kwargs*)

Plots the counts histogram for a 1D X-ray spectrum.

ax [matplotlib AxesSubplot object] The figure axis on which to plot.

spectrum [pyxsis XBinSpectrum object] The binnable 1D spectrum object to plot.

xunit [string (default: 'keV')] Defines the unit type to be used on the x-axis. The options are 'angs' and 'keV'.

perbin [bool (default: True)] If True, the plot y-axis will show the number of counts per bin. If False, the plot y-axis will show the number of counts per bin-width, depending on the xunit. For example, if xunit='keV', then the y-axis units will be counts/keV.

rate [bool (default: False)] If True, the plot y-axis will show the number of counts per second. If False, it will show the total number of counts.

plot_bkg [bool (default: False)] If True, the background spectrum assigned to the input spectrum will be plotted instead of the primary source spectrum.

subtract_bkg [bool (default: True)] If True, the background spectrum assigned to the input spectrum will be subtracted before plotting. If False, no background subtraction will be implemented.

use_backscale [bool (default: True)] If True, the background spectrum will be scaled by the pyxsis.XBkgSpectrum.backscale attribute. This attribute generally holds the ratio of the background extraction area to the source extraction area. If False, the raw background spectrum count rate will be used. This is helpful if you just want to view the raw background spectrum.

scale_factor [float (default: 1.0)] A normalization value to apply to the entire spectrum to be plotted. This option is provided solely for convenience, for example, in comparing spectra to each other.

kwargs are passed to the main histrogram plotting function (ax.step). This can be used to change the color, line widths, line style, and more.

2.4.2 Plot a flux spectrum

pyxsis.plot.**plot_unfold**(*ax*, *spectrum*, *xunit='keV'*, *perbin=False*, *subtract_bkg=True*, *use_backscale=True*, *scale_factor=1.0*, ***kwargs*)

Plots the flux histogram for a 1D X-ray spectrum.

ax [matplotlib AxesSubplot object] The figure axis on which to plot.

spectrum [pyxsis XBinSpectrum object] The binnable 1D spectrum object to plot.

xunit [string (default: 'keV')] Defines the unit type to be used on the x-axis. The options are 'angs' and 'keV'.

perbin [bool (default: False)] If True, the plot y-axis will show the number of counts per bin. If False, the plot y-axis will show the number of counts per bin-width, depending on the xunit. For example, if xunit='keV', then the y-axis units will be counts/keV.

subtract_bkg [bool (default: True)] If True, the background spectrum assigned to the input spectrum will be subtracted before plotting. If False, no background subtraction will be implemented.

use_backscale [bool (default: True)] If True, the background spectrum will be scaled by the pyxsis.XBkgSpectrum.backscale attribute. This attribute generally holds the ratio of the background extraction area to the source extraction area. If False, the raw background spectrum count rate will be used. This is helpful if you just want to view the raw background spectrum.

scale_factor [float (default: 1.0)] A normalization value to apply to the entire spectrum to be plotted. This option is provided solely for convenience, for example, in comparing spectra to each other.

kwargs are passed to the main histrogram plotting function (ax.step). This can be used to change the color, line widths, line style, and more.

2.5 Example Guide

All of these examples can be run with the test files located at https://doi.org/10.5281/zenodo.2528474. See the Install instructions for details.

2.5.1 Group a spectrum and plot it

This example shows how to load a spectrum with pyXsis, group the spectrum to a minimum number of counts, and plot the resulting flux spectrum.

```
import matplotlib.pyplot as plt
from pyxsis.io import load_chandra_hetg
from pyxsis import group_mincounts, plot_unfold

spec = load_chandra_hetg("tests/data/17385/heg_-1.pha")
group_mincounts(spec, 30)

ax = plt.subplot(111)
plot_unfold(ax, spec, xunit='keV')
plt.loglog()
plt.show()
```

2.5.2 Plot a summed counts histogram

Because it's nearly impossible to combine an RMF file from different observations or instrument spectra (e.g., combining the RMF from a +1 and -1 order), **we do not recommend fitting combined spectra**. You should use your preferred fitting method to fit separate spectra simultaneously. However, in low signal-to-noise spectra it may be useful to display the summed +1 and -1 orders from an HETG spectrum for visual inspection only. This example demonstrates how to load two spectra, sum their counts, create a new Spectrum object to hold the summed counts histogram, bin the spectra, and plot the resulting count rates.

WARNING: This method requires the two spectra to have the same bin edge values, which is true when comparing +1 and -1 orders from the same gratings type. No errors will arise if you attempt to use this method to combine HEG+1 and MEG+1 orders, for example, but the results will be entirely incorrect!

```
import pyxsis

# Load the HEG+1 and HEG-1 order spectra
spec_hm1 = pyxsis.io.load_chandra_hetg("tests/data/17392/heg_-1.pha")
spec_hp1 = pyxsis.io.load_chandra_hetg("tests/data/17392/heg_1.pha")

# Sum the counts histograms
summed_counts = spec_hm1.counts + spec_hp1.counts
summed_heg = pyxsis.XBinSpectrum(spec_hm1.bin_lo, spec_hm1.bin_hi,
    summed_counts, spec_hm1.exposure)

# Group the summed histogram
# and apply that same grouping to the original spectra
# (for visualization purposes)
pyxsis.group_mincounts(summed_heg, 100)
spec_hm1.binning = summed_heg.binning
spec_hp1.binning = summed_heg.binning

# Plot the count rates for each spectrum on the same axis
```

(continues on next page)

(continued from previous page)

```
ax = plt.subplot(111)
ax.set_xlim(1.7,1.9)
pyxsis.plot_counts(ax, spec_hm1, rate=True,
    color='r', label='HEG-1')
pyxsis.plot_counts(ax, spec_hp1, rate=True,
    color='b', label='HEG+1')
pyxsis.plot_counts(ax, summed_heg, rate=True,
    color='k', label='Summed')
ax.legend()
plt.show()
```

INDEX