The Swift UVOT Grism Documentation

Release 0.9.6

Paul Kuin

March 15, 2013

CONTENTS

CHAPTER

ONE

USERS GUIDE

The grisms of the UVOT instrument provide a dispersed image for a region of about 17'x17' on the Sky. This is a description of the Python software written at UCL/MSSL to help reduce the data. The software extends the earlier Ftool version as distributed in the Heasoft Swift instrument software. This software implements all the recent calibrations, including a correction for coincidence-loss (also known as pile-up) in the photon-counting detectors. It therefore produces a superior product to that from the "uvotimgrism" Ftool.

The calibration files are bundled with the source code for the software (see *Fetching the data*) and you can grab a harcopy of this document PDF .

A simple call can in many cases generate a quick spectrum. The uvotpy module `uvotgetspec` provides the function `getSpec()` which only needs the grism data files, including attitude (auxil) files, target position in decimal degrees, and extension with the desired spectral data.

1.1 Getting started

1.1.1 Installing UVOTPY

You may already have Python installed on your system. The current version of *UVOTPY* was written for Python 2.7, though it is expected to work for Python versions 2.5 and 2.6 as well. However, *UVOTPY* has not been adapted for Python 3.x. You can check the Python version by doing from the command line:

```
python --version
```

Other Python software is required. In particular, Numpy (version >= 1.1), Scipy (version >= 0.10.1), matplotlib (version >= 1.0), STScI Python (version >= 2.12), and (optionally) ipthon (version >= 0.11) is recommended for interactive sessions. Lately, Astropy has been absorbing some packages, like *Pyfits* and is planned to replace some dependencies in the future.

If you do not have the required software on your computer, and want to install everything in one go, there are several options. For "UVOTPY" development I used mainly the STScI-Python package on Mac OSX 10.6. A quite popular choice is the binary Enthought Python Distribution which is free for academic use (see their Licence).

Not only Python packages are required. The software requires a working, configured Heasoft Swift distribution (version > 6.11), and a recent version of the Swift calibration database (CALDB).

1.1.2 Fetching the data

If you have all those installed and configured, the "UVOTPY" package can be downloaded:

```
http://www.mssl.ucl.ac.uk/~npmk/Grism/uvotpy/uvotpy-latest.tar.gz
```

Next you install it with:

```
> tar xzf uvotpy-latest.tar.gz
> cd uvotpy-x.x.x
> sudo python setup.py uvotpy
```

Next thing to do is to set up the environment. I discuss that for *unix*-like systems only.

1.1.3 Setting up the environment

The package installs a number of calibration and other template files which it needs to find. That is done through defining in the environment the variable *UVOTPY*. Say your python packages live in */usr/stsci/pyssg/Python-2.7.3/lib/python2.7/site-packages*, and you have *uvotpy* installed there. Then, from the command line, use

```
setenv UVOTPY <your-python-site-packages-dir>/uvotpy
```

to define the required environment variable.

1.1.4 Running from the command line

After installation of *uvotpy* and setting up the environment, the program *uvotgrism* can be called to extract spectra.

an example would be:

```
uvotgrism --ra=254.7129625 --dec=34.3148667 --obsid=00055900056 \
--dir=VGRISM/00055900056/uvot/image
```

There is also a spectrum for the UV grism, but the slit fails to automatically find the right spectrum location. The program is then forced to prompt as follows:

```
uvotgrism --ra=254.7129625 --dec=34.3148667 --obsid=00055900056 \
--anchor_offset=No --dir=UVGRISM/00055900056/uvot/image
```

when prompted enter “1” to position the spectrum correctly.

The data for these examples is included in the ‘test’ directory of the uvotpy tar file. To save space a set without lenticular images has been used. It is in *$UVOTPY/test* after installation. You must make a copy to run, since the program will write into the data directory or fail if it cannot. It also writes the spectrum files to the current directory.

1.2 Getting a quick spectrum with default parameter settings

There are three ways to run the program. The preferred method for me (the developer) is while using an ipython shell and the how-to was briefly shown earlier as *Running from the command line*. The second method is as a command-line call, while the third method would be to run within a Python shell.

1.2.1 Preparations

Before starting the spectral extraction, the data for your observation(s) needs to be available on your system. Besides the image files, the spacecraft attitude file and its directory should be available. Although some options were built-in to the program for other arrangements, the only known way that works has the same directory and file structure as that distributed by the Swift archive:

```
<obsid>/
<obsid>/auxil/sw<obsid>pat.fits
<obsid>/auxil/sw<obsid>uat.fits
<obsid>/auxil/sw<obsid>sat.fits
<obsid>/uvot/image/sw<obsid>ugu_rw.img (optional)
<obsid>/uvot/image/sw<obsid>ugu_dt.img (required)
<obsid>/uvot/image/sw<obsid>ugu_sk.img (example for UV GRISM, *ugv_sk.img for V GRISM)
<obsid>/uvot/image/sw<obsid>uw1_sk.img (if present)
<obsid>/uvot/image/sw<obsid>uw2_sk.img (if present)
```

The `<obsid>` is typically of the form `"00032323001"`, and is one of the required inputs needed. During the ground processing at NASA-GSFC, the initial attitude file `sw<obsid>sat.fits` may be updated to `sw<obsid>uat.fits`, or `sw<obsid>pat.fits` based on further analysis.

The second most important item needed is the source position in **RA, Dec (J2000)** decimal degrees (not in sexagesimal).

Finally, the `FITS` extension of the spectrum image (`<obsid>/uvot/image/` file) is needed; when not given it defaults to 1.

The analysis uses the `detector image *_dt.img` files which have been *distortion* corrected. This means that the same correction was applied to the grism image as was found for the lenticular filter images, but additional distortions remain in the grism spectra.

IMPORTANT: Internet access is needed so that the USNO-B1 on-line catalog can be queried from the servers at SAO.

A quick inspection of the `sky images *_sk.img` is useful to make sure the spectrum exposure was correct. There are several other pitfalls, like overlapping orders which can be diagnosed by using DS9 in combination with a DSS image. That is discussed elsewhere in *Basics of UVOT Grism analysis*.

1.2.2 Extracting with IPython

The *ipython* shell is more versatile and user-friendly in interactive sessions, with command line finish and recall, for example. A session can be started from the `<obsid>/uvot/image/` directory, or use *cd ''<obsid>/uvot/image/'* ' from within the ipython shell to go to the directory. Use the *–pylab* argument to load matplotlib for the session, and define the required parameters:

```
ipython --pylab
from uvotpy.uvotgetspec import getSpec
ra, dec = 123.45678, +65.789012
obsid = "00032109002"
ext = 2
```

Note that we imported the *getSpec* program from the *uvotpy.uvotgetspec* module of the *uvotpy* package. We can now do a simple call to the program. Assuming the obsid and position are all correct:

```
getSpec(ra,dec,obsid,ext, fit_second=False, clobber=False)
```

I have set here one more parameter “fit_second” to “false”, since this can cause problems when the second order completely overlaps the first order, i.e., is not offset in the cross-dispersion direction.

Three graphs will appear.

- The first one to quickly view the extraction is positioned approximately right, and crosses are plotted at about 200A offsets at the estimated location of the spectrum.
- In the second graph the extracted raw data in the form as total counts, i.e., (no background subtraction, and not as rates), aligned with the image section obtained after rotating to the dispersion). Also, a *very* rough guess of

the second order counts is made based on those in the first order. It tends to over-estimate, sometimes as much as a factor of two.

- In the third graph, the flux calibrated spectrum.

It should be noted that the flagging of bad areas is at the moment *pretty bad*, and needs to be looked at again.

The spectra will be written to a file, while also a background file is made for use with XSPEC. However, the RMF file is not produced automatically, since it involves a very slow piece of code. The RMF is useful, since the spectral resolution gets worse to the red, and is especially affecting wavelengths above 400nm in the UV Grism spectra.

1.3 Basics of UVOT Grism analysis

The main reason for writing this section is to help you navigate around some of the pitfalls that I have encountered when analysing the UVOT Grism spectra; some of which are not always obvious.

1.3.1 Use DS9 with the grism sky image and DSS

Even for one used to see many grism images, locating the source spectrum can be difficult. The automated processing lines up the peaks of the zeroth orders of the sources in the image with the `WCS`. If everything goes perfect, we can load the grism image and `DSS` image in `DS9`, *blink* them, and the zeroth orders and `DSS` sources line up nicely. If they don't, and there are close-by orders that make interpretation too confusing, see *Problem with positions in the grism image*.

Blinking the `DSS` and grism sky image while zooming in on the first order will allow a quick determination of contaminating zeroth orders that fall on the first order spectrum. Also, bright sources in the dispersion plane, behind the source can easily be a problem especially in the UV part of the spectrum where the target photon sensitivity is low.

1.3.2 Problem with positions in the grism image

First of all, we need to be sure that we have the header of the sky image in the best possible form. That is, check the header keyword `ASPCOR` is set to `"graspcorr"`. From the terminal command line, assuming the present working directory is the `<obsid>/uvot/image` directory and we have a UV Grism image:

```
$ftlist sw<obsid>ugu_sk.img hk include=aspcorr
```

The value should be printed for each extension in the `FITS` file. If not set, the `uvotgraspcorr` script should be ran, though `uvotgraspcorr` fails in a few percent of cases if it cannot reach the catalog server; for some fields with not enough sources of the right brightness in the field; and for some images with large galaxies. It is very difficult to say off-hand how to change the internal parameters for the program to ensure success. This usually means that the keywords accounting for the distortion of the zeroth order positions are not added to the `WCSS` header keywords. The distortions are increasing to the edges of the detector, where the problem is worst.

If the zeroth orders are clearly offset from the expected positions when blinking the grism sky image with the corresponding DSS image, the header WCS keywords in the grism image sky file can be edited "by hand" to line up the grism zeroth order coordinates. It is smartest to do that for the region close to the target source location, as that will clearly minimize the errors in the analysis.

Note that the grism extraction is *not* dependent on the aspect correction when the grism image was taken paired with an image in one of the lenticular filters.

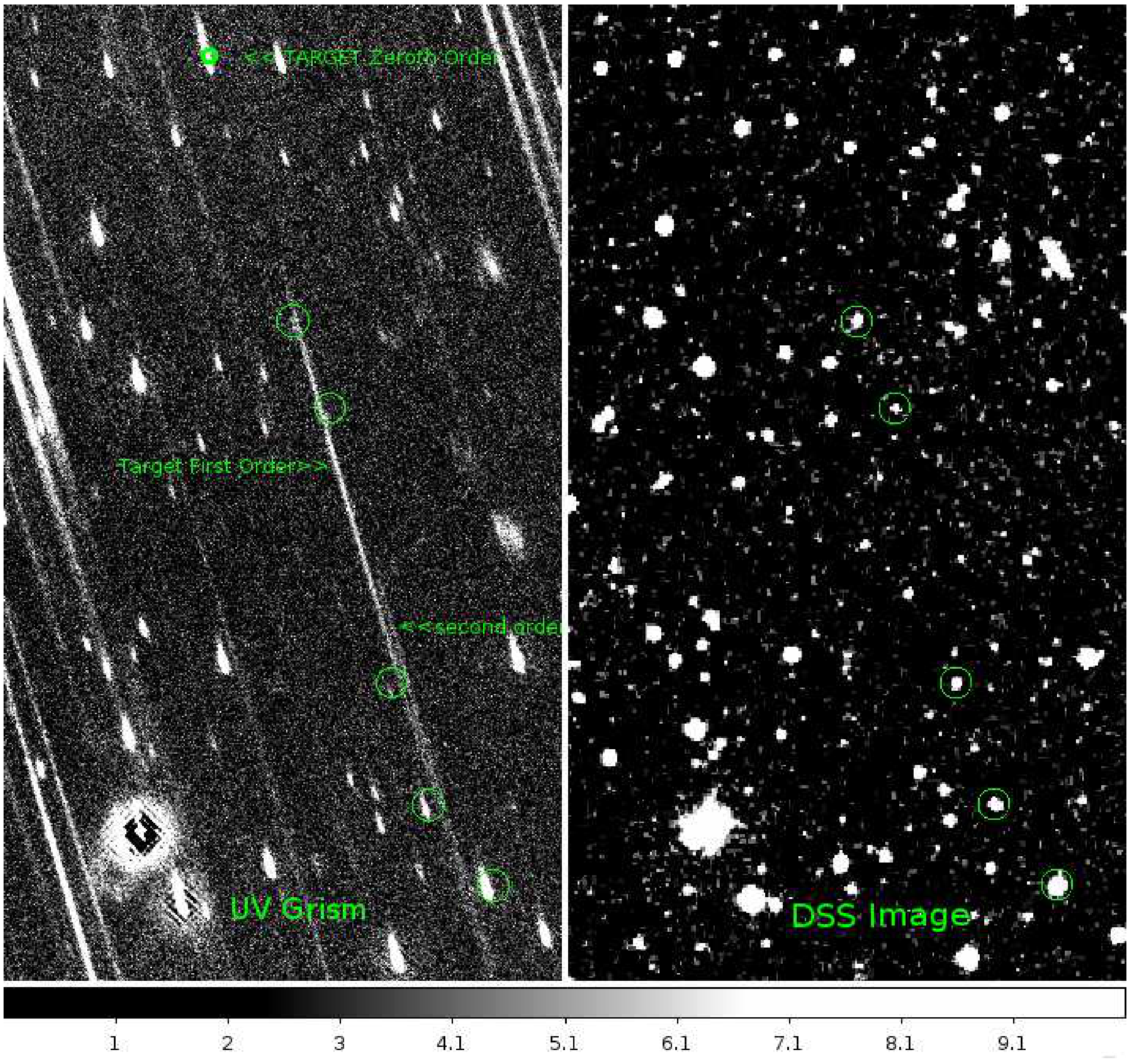

Figure 1.1: The background zeroth orders that lie close to the first order have been marked.

1.4 Lexicon

Explanation of terms:

Term	Explantion
Lenticular filter	A flat filter. One of uvw2, uvm2, uvw1, u,b,v,wh
V Grism	Grism with peak response in the optical (420nm)
UV Grism	Grism optimised in the UV (260nm)

1.5 API Documentation

To get the options to use with the command-line program *uvotgrism*, use

```
uvotgrism --help
```

NOTE: A bug was found when preparing the test data. When in a data directory there are present both a detector image file for the *uv-grism* and one for the *v-grism*, the uv-grism is always picked by the program. An option to force the program is not currently available. Copy the observation tree starting with the directory name that is the *obsid* and delete the uv-grism file in one instance, which will provide a solution.

1.5.1 Notes the version 0.9.7

General notes to *uvotgrism*

This code is designed for processing Swift UVOT Grism images

The ultimate goal is to process the spectra in the image, to do quality control and source identification. Because the UVOT is a photon counting detector, the error handling must keep track of the errors which require in principle the total exposure, the background exposure, and the exposure time. The data quality for each pixel needs to be flagged also (e.g., data dropouts, scattered light rings, halo around bright features). In summed images, bad data needs to be removed, and the background is also affected. Therefore an exposure map is needed for summing. Summing the extracted spectra requires a careful check for the correct alignment of wavelengths. The coincidence loss is managed in a rather ad-doc manner, while the count rate errors follow from the observed binomial nature of the detector, e.g., Kuin & Rosen (2008,MNRAS, 383,383) - though that prescription was for point sources. Modifications appropriate for spectra are not yet in place.

The program development and calibration is still ongoing (version 0.9.7, March 2013).

This program extracts the spectra, applies the wavelength calibration file to find anchor position and wavelength dispersion. The flux calibration has been done at the default position for both v and uv grisms. In the *uv grism* the flux calibration (effective area) was determined at a handful of selected offset anchor positions with extensions to nearby positions on the detector using a model calculation adjusted to fit the calibration observations best. The accuracy of the flux, neglecting coincidence loss, is of order 5% in the centre and with 10% at other locations on the detector. Coincidence loss corrections for bright sources can be in error by 10% or more, but look to be within 5% for most sources. Some nice graphs showing coincidence-loss corrected spectra are here.

For the *V grism*, the spectrum is flux calibrated using the Swift UVOT CALDB flux calibration which was made for observations at the default position (accurate to about 20%). No coincidence-loss correction was made for that calibration.

In the nominal grism mode (wheelpos = 200, 1000) the response varies by about 5% from centre to about 200 pixels from the edge. In the clocked grism modes (wheelpos = 160, and 955) the response has a strong drop when the spectrum falls in the upper left corner. For the rest of the image the response varies less than ~20% in the clocked modes. Further calibration is ongoing for the *v grism*.

Before using this code, it is recommended to reprocess the raw image using the *mod-8 correction*. Use a `CALDB` later than May 2010 for an improved distortion correction. The grism detector image should be attitude corrected using uvotgraspcorr. Check the header keyword `ASPCORR="GRASPCORR"`.

Find out the **RA** and **DEC** position in decimal degrees from the USNO-B1 catalog for your object since the UVOT aspect corrections also use the USNO-B1 positions, so you will avoid a systematic error in positions which will translate to a shift in the wavelengths derived.

You need to set the environment variable `UVOTPY` to point to the directory with the `UVOTPY` code and new calibration files. See :ref:setting-up-environment.

The input data files must be decompressed before running the program. The program tries to do this itself, but does not recompress the files again after the program completed.

The program was developed running in *iPython*, and it is suggested to run interactively in *iPython* rather then run as it a script.

Second order extraction and calibration are only treated very roughly at this time. Zeroth orders are only minimally identified. Third order location is approximate.

You probably won't want to dive into all the program internals, but some idea of what is there may help sometimes.

Main function

`getSpec` : main call for spectral data extraction

Other uvotpy functions

- curved_extraction: set curvature of orders, set quality flags, get spectral data
- extractSpecImg : get sub image
- findBackground : get background
- get_components : extract first, second, third order components
- getCal : get the (wave) calibration files
- predict_second_order : use first order to predict second order (very rough)
- coi_func : return an interpolating function(wave) for the coincidence loss of a spectrum [experimental]

Specialized functions are in modules

- uvotgetspec: repository of main functions
- uvotio: writes file output
- uvotio.rate2flux : convert count rate to flux
- uvotplot : plot routines
- uvotmisc : miscellaneous routines

The program assumes all data files are available in either the working directory, or the directory structure complies with the Swift project standard and is run from the `<obsid>/uvot/images` directory, while the attitude file is available in the `<obsid>/aux` directory.

The flux-calibrated 1st order spectrum is available in the second extension of the output file.

Release Notes

Version 0.9.7 March 2013

by Paul Kuin (npkuin at gmail.com|n.kuin at ucl.ac.uk)

March 2013: Flux calibration for the uv-grism

The new calibration file has multiple extensions. The first extension is the original flux calibration, which did not correct for the coincidence loss and which is to be used with the "Ftool" **uvotimgrism**. Then follow extensions with the measured effective areas at various positions on the detector. These have been corrected for coincidence-loss during the calibration. The number depends to some extent on how much the effective area varies with position. Finally, a normalised flux from a rescaled model is included that can be used to extrapolate from one of the effective areas at a nearby position. A new routine in uvotio reads the effective area and model scaling to offset if present.

A start has been made with using Sphinx (http://sphinx.org) for documentation on the web. This includes changing the inline documentation of the software. Keep an eye on http://www.mssl.ucl.ac.uk/~npmk/Grism/uvotpy.

October 2012: Initial extension of flux calibration uv-grism

The software has been updated to support a working coi-correction, though not the final one, and use the new flux calibration where available. The calibration file closest to the position on the detector image will be used. More calibration files will be supplied when they become available. The comparison of the new calibration and coi-loss correction with calibration spectra can be seen on my web site.

The package has now been created using Python distutils.

March 2012: An experimental coincidence loss correction

The background in the grism is quite high and in itself experiences about 3% coi-loss. The extended nature of the background has been cause for concern, but tests reported in Breeveld et al. (2010) did in fact show that the correction method from Poole et al. (2008) works well also in that case. However, the case for an extended linear feature, like a spectrum has not been studied. The symmetries are different, and it is not a priori clear how to best generalize the method from Poole et al. But with estimated coi-loss peaking at 3% for a 16th mag WD and estimated 7% for a 14th mag WD, with maximum coi-loss close to 50% for a 12th mag WD, it is clear that a solution is desirable as part of a flux calibration that is meaningfull. The WD are probably the worst case, since they are bright around 3000A where the effective area (and the count rate) peaks

March 2012: discovery of variable sensitivity over face of uv clocked detector image

Over most of the detector, the sensitivity only changes by a few percent. So it was a bit of a shock to see that in the clocked uv grism images, the detector sensitivity drops quite a lot in the upper left corner of the detector. More so, since this area had been selected to place spectra to completely avoid the zeroth orders. This is under investigation. We are using the Zemax optical model to get a quantitative, though approximate, measure of this effect.

April 2012: predicting the uv grism exposure

For some faint objects, there is often uvot photometry in the uv filters available. It is of a given magnitude, but what does that mean for the uv grism. I made a tool that will make a rough estimate of the exposure time needed to get a certain signal-to-noise, assuming a certain background and given a magnitude of the object as observed in a uvot filter. There is loads of uncertainty when the source is faint, and the background over the UV clocked grism varies. It defaults to 0.16c/s/pix(across the spectrum) but a value of 0.05 may happen depending of where the (uv part of) the spectrum lies on the detector. So for now, I put in the most reasonable value. If the source is getting faint, the exposure time will climb through the roof. Experience must tell how good it is and how to use it.

Altermatively a spectrum can be put in a will give a magnitude in the lenticular filters. The uvotphot.py file will be needed.

April 2012: estimate of the zeroth order effective area

The uv grism efficiency in the zeroth first and second order estimate were needed for calculating the flux sensitivity over the detector using the Zemax optical model. A rough approximation of the zeroth order effective area was determined in the course of that work. Details have been written in a report.

April 2012: determining the effective area in the uv detector

Legal Notice

This software is under development. Software is released for testing only. User of the software agreed that no responsibility for any losses of any kind resulting out of use of this software are attributable to the author, the MSSL, UCL, or contracting organizations.

This software may be freely distributed and used for scientific research provided this ReadMe notice is included and the author is invited to coauthor papers using this early version of the software.

Testing

The software was tested with the following installation:

(1) an existing installation of the STScI python, including the Scipy software, for Python version 2.7.3 A package can be downloaded from STScI. Note that you also will need to install `gfortran` (link at the bottom of the STScI Python page) to make it work. If you have `pyraf` installed, it may already be there.

(2) Installation of the `Heasoft Ftools/SWIFT` software, release not earlier than version 6.10. Download from http://heasarc.gsfc.nasa.gov/docs/software/lheasoft/ .

(3) Installation of the `HEASARC CALDB` for Swift, version later than Oct 13,2011. Download the `CALDB` for Swift from: http://heasarc.gsfc.nasa.gov/docs/heasarc/caldb/caldb_supported_missions.html

(4) Installation of Mink's `WCSTOOLS`, in particular *scat*. On unix/linux type system the command *which scat* should return the location. If it does not return anything, install: http://tdc-www.harvard.edu/software/wcstools/wcstools-3.8.4.tar.gz and remove or rename the program cphead that it installs, since it clashes with *cphead* in `Heasoft` as used by the *uvotproduct ftool*.

(5) Installation of *CDSCLIENT* which provides the *'sesame'* name resolver. [optional] http://cdsarc.u-strasbg.fr/doc/cdsclient.html

(6) setup of the environment: ("$HOME/.cshrc" for *csh* users or "$HOME/.bashrc" for *bash* or *sh* users)

- add the directory with this software to the PYTHONPATH environment variable.
- set UVOTPY to the directory with this software
- add the WCSTOOLS/bin and cdsclient directories to your PATH

1.5.2 uvotgetspec

Functions

`clipmask()`

`uvotpy.uvotgetspec.`**`clipmask`**(*f*, *sigclip=2.5*, *fpos=False*)

Provides mask to clip bad data.

Parameters **f** : 2D array

kwargs : dict

optional arguments

- **sigclip** : float

  clip data at *sigma* standard deviations above the mean

- **fpos** : bool

  if True, clip negative values

Returns **mask** : 2D array, boolean

Array of same size as image, true where within sigclip standard deviations of mean.

Notes

By default infinities are clipped.

The mask is iterated until it converges. So the effect of outliers on the standard deviation is nil. This also means that sigma needs to be chosen large enough or the standard deviation will not be a good measure of the real noise in the mean.

`coi_func()`

`uvotpy.uvotgetspec.`**`coi_func`**(*pixno, wave, countrate, bkgrate, sig1coef=[3.2], option=2, fudgespec=1.0, coi_length=26, frametime=0.0110302, background=False, sigma1_limits=[2.6, 4.0], trackwidth=2.5, ccc=[-1.5, 1.5, -1.5, 1.5, -1.5, 1.5, 0.995], ccb=[0.72, -0.72, 0.995], ca=[0, 0, 3.2], cb=[0, 0, 3.2], debug=False, chatter=1*)

Compute the coincidence loss correction factor to the (net) count rate as a function of wavelength EXPERIMENTAL

Parameters **pixno** : array-like

pixel number with origen at anchor

wave : array-like

wavelength in A

countrate : array-like

input count net rate must be aperture corrected

bkgrate : array-like

background rate for trackwidth

kwargs : dict

- **sig1coef** : list

  polynomial coefficients

- **frametime** : float

  CCD frame time in seconds

- **trackwidth** : float

  width of the extraction in standard deviations of the profile matched across the spectrum

- **option** : int

  . option = 1 : classic coi-loss, but spectrum is box like 10x32 pix across spectrum

  . option = 2 [bkg classic coi-loss, total (spectrum*poly+bkg*poly) with] polynomial corrections for extended coi-loss. classical limit for ccc= [0,0,1] ; ccb[0,0,1]

- **background** : bool

  if the background is *True* an interpolated function for the coi correction factor in the background count rate is returned

  if the background is *False* an interpolated function for the coi correction factor in the net target count rate is returned

Returns **coi_func** : scipy.interpolate.interpolate.interp1d

if **background** is *True* an interpolated function for the coi correction factor in the background count rate while if **background** is *False* an interpolated function for the coi correction factor in the net target count rate is returned

Notes

defaults to the background coincidence loss equivalent to an area of 315 sub-pixels (about pi*5"^2 on the detector)

Also see the discussion of coincidence loss in Breeveld et al. (2010). Her correction for high background + high source rate was used as inspiration.

- 2012-03-21 NPMK initial version
- 2012-07-04 NPMK added into option 1 the white-correction for high background (photometry) from Alice Breeveld (2010)
- 2012-07-24 NPMK modified source area to be same as background area
- 2012-07-24 NPMK start modifications extended coi-model
- **2012-09-25 NPMK simplify. Add extended coi-loss as polynomial using classic coi as approx.** coefficients TBD.
- **2012-10-10 NPMK temporary option 3 to address consistent approach with Breeveld et al. and** the coi-work on point sources. Basically, it is not a reduction in the background but a lack of accounting for additional losses in the main peaks (due to surrounding high background levels stealing counts from source). Option 2 has now been optimized to work. Basically, there are multiple practical solutions to the problem, the third option will be more in line with the theoretical model for coincidence loss in the UVOT.

findBackground()

`uvotpy.uvotgetspec.findBackground`(*extimg, background_lower=[None, None], background_upper=[None, None], yloc_spectrum=100, smo1=None, smo2=None, chatter=2*)

Extract the background from the image slice containing the spectrum.

Parameters **extimg** : 2D array

image containing spectrum. Dispersion approximately along x-axis.

background_lower : list

distance in pixels from *yloc_spectrum* of the limits of the lower background region.

background_upper : list

distance in pixels from *yloc_spectrum* of the limits of the upper background region.

yloc_spectrum : int

pixel *Y* location of spectrum

smo1 : float

smoothing parameter passed to smoothing spline fitting routine. *None* for default.

smo2 : float

smoothing parameter passed to smoothing spline fitting routine. *None* for default.

chatter : int

verbosity

Returns **bg** : float

mean background

bg1, bg2 : 1D arrays

bg1 = lower background; bg2 = upper background

bgsig : float

standard deviation of background

bgimg : 2D array

image of the background constructed from bg1 and/or bg2

bg_limits_used : list, length 4

limits used for the background in the following order: lower background, upper background

(bg1_good, bg1_dis, bg1_dis_good, bg2_good, bg2_dis, bg2_dis_good, bgimg_lin) : tuple

various other background measures

Notes

Global parameter

- **background_method** : {'boxcar','splinefit'}

The two background images can be computed 2 ways:

1. 'splinefit': sigma clip image, then fit a smoothing spline to each row, then average in y for each background region
2. 'boxcar': select the background from the smoothed image created by method 1 below.

extimg is the image containing the spectrum in the 1-axis centered in 0-axis *ank* is the position of the anchor in the image

I create two background images:

1. split the image strip into 40 portions in x, so that the background variation is small compute the mean sigma clip (3 sigma) each area to to the local mean replace out-of-image pixels with mean of whole image (2-sigma clipped) smooth with a boxcar by the smoothing factor
2. compute the background in two regions upper and lower linearly interpolate in Y between the two regions to create a background image

bg1 = lower background; bg2 = upper background

smo1, smo2 allow one to relax the smoothing factor in computing the smoothing spline fit

`findInputAngle()`

`uvotpy.uvotgetspec.`**`findInputAngle`**`(`*RA*, *DEC*, *filestub*, *ext*, *wheelpos=200*, *lfilter='uvw1'*, *lfilter_ext=None*, *lfilt2=None*, *lfilt2_ext=None*, *method=None*, *attfile=None*, *graspcorr='no'*, *indir='./'*, *chatter=2*`)`

Find the angles along the X,Y axis for the target distance from the bore sight.

Parameters **RA,DEC** : float

sky position, epoch J2000, decimal degrees

filestub : str

part of filename consisting of "sw"+'obsid'

ext : int

number of the extension

kwargs : dict

- **wheelpos** : int, {160,200,955,1000}

  grism filter selected in filter wheel

- **lfilter**, **lfilt2** : str, {'uvw2','uvm2','uvw1','u','b','v'}

  lenticular filter name before and after grism exposure

- **lfilter_ext**, **lfilt2_ext** : int

  lenticular filter extension before and after grism exposure

- **method** : str, {'grism_only'}

  if set to *grism_only*, create a temporary header to compute the target input angles, otherwise use the lenticular file image.

- **attfile** : str, path

  full path+filename of attitude file

- **graspcorr** : str

  if *yes* then try to update the header using a call to the *graspcorr ftool*.

- **indir** : str, path

  data directory path

- **chatter** : int

  verbosity

Returns **anker_as** : array

offset (DX,DY) in arcsec in DET coordinate system of the source from the boresight needs to be converted to input rays by applying transform.

anker_field : array

offset(theta,phi) in degrees from the axis for the input field coordinates for the zemax model lookup

tstart : float

start time exposure (swift time in seconds)

Notes

Provided a combined observation is available in a lenticular filter and a grism (i.e., they were aquired in the the same observation,) this routine determines the input angles from the boresight. Assumed is that the grism and lenticular filter image have the same extension.

If not a lenticular filter image was taken just before or after the grism exposure, the input angles are determined from the grism aspect only. Before running this, run uvotgrapcorr on the grism image when there is no lenticular filter to get a better aspect solution.

`getCalData()`

`uvotpy.uvotgetspec.`**`getCalData`**(*Xphi*, *Yphi*, *wheelpos*, *date*, *chatter=3*, *mode='bilinear'*, *kx=1*, *ky=1*, *s=0*, *calfile=None*, *caldir=None*)

Retrieve the calibration data for the anchor and dispersion (wavelengths).

Parameters **Xphi, Yphi** : float

input angles in degrees, from, e.g., *findInputAngle*.

wheelpos : int, {160,200,955,1000}

filter wheel position selects grism

date : swifttime in seconds

obsolete - not used

kwargs : dict

optional arguments

- **calfile** : str

  calibration file name

- **caldir** : str

  path of directory calibration files

- **mode** : str

  interpolation method. Use 'bilinear' only.
- **kx**, **ky** : int, {1,2,3} order of interpolation. Use linear interpolation only.
- **s** : float smoothing factor, use s=0.
- **chatter** : int verbosity

Returns **anker, anker2** : list

coordinate of anchor in first order.

C_1, C_2 : :

dispersion in first and second order.

theta : float

find angle of dispersion on detector as 180-theta.

data : FITS_rec

the wavecal data table

Notes

Given the input angle Xphi, Yphi in deg., the filterwheel position, and the date the spectrum was taken (in swift seconds), this gets the calibration data.

The boresight must be set to the one used in deriving the calibration.

getSpec()

`uvotpy.uvotgetspec.`**`getSpec`**(*RA, DEC, obsid, ext, indir='./', wr_outfile=True, outfile=None, calfile=None, fluxcalfile=None, coicalfile=None, offsetlimit=None, anchor_offset=None, anchor_position=[None, None], background_lower=[None, None], background_upper=[None, None], fixed_angle=None, spextwidth=13, curved='update', fit_second=False, predict2nd=True, skip_field_src=False, optimal_extraction=False, get_curve=False, fit_sigmas=True, get_sigma_poly=False, lfilt1=None, lfilt1_ext=None, lfilt2=None, lfilt2_ext=None, interactive=True, sumimage=None, plot_img=True, plot_raw=True, plot_spec=True, zoom=True, highlight=False, clobber=False, chatter=1*)

Makes all the necessary calls to reduce the data.

Parameters **ra, dec** : float

The Sky position (J2000) in **decimal degrees**

obsid : str

The observation ID number as a **String**. Typically that is something like "00032331001" and should be part of your grism filename which is something like "sw00032331001ugu_dt.img"

ext : int

number of the extension to process

kwargs : dict

optional keyword arguments, possible values are:

- **fit_second** : bool

  fit the second order. Off since it sometimes causes problems when the orders overlap completely. Useful for spectra in top part detector

- **background_lower** : list

  instead of default background list offset from spectrum as list of two numbers, like [20, 40]. Distance relative to spectrum

- **background_upper** : list

  instead of default background list offset from spectrum as list of two numbers, like [20, 40]. Distance relative to spectrum

- **offsetlimit** : int

  when strong spectrum lies next to target, feed offsetlimit a value. can be positive or negative. Try small numbers like -1, or 3, etc.

- **zoom** : bool

  when False, the whole extracted region is displayed, including zeroth order when present.

- **clobber** : bool

  When True, overwrite earlier output (see also outfile)

- **sumimage** : str

  Name summed image generated using `sum_Extimage()`, will extract spectrum from summed image.

- **wr_outfile** : bool

  If False, no output file is written

- **outfile** : path, str

  Name of output file, other than automatically generated.

- **calfile** : path, str

  calibration file name

- **fluxcalfile** : path, str

  flux calibration file name or "CALDB"

- **predict2nd** : bool

  predict the second order flux from the first. Overestimates in centre a lot.

- **skip_field_src** : bool

  if True do not locate zeroth order positions. Can be used if absence internet connection or USNO-B1 server causes problems.

- **optimal_extraction** : bool, obsolete

  Do not use.Better results with other implementation.

- **get_curve** : bool

  option to supply the curvature coefficients of all orders by hand. implemented but not tested.

- **fit_sigmas** : bool

  fit the sigma of trackwidths if True (not implemented, always on)

- **get_sigma_poly** : bool

  option to supply the polynomial for the sigma (not implemented)

- **lfilt1**, **lfilt2** : str

  name if the lenticular filter before and after the grism exposure (now supplied by fileinfo())

- **lfilt1_ext**, **lfilt2_ext** : int

  extension of the lenticular filter (now supplied by fileinfo())

- **plot_img** : bool

  plot the first figure with the det image

- **plot_raw** : bool

  plot the raw spectrum data

- **plot_spec** : bool

  plot the flux spectrum

- **highlight** : bool

  add contours to the plots to highlight contrasts

- **chatter** : int

  verbosity of program

Returns **None, or compound data (Y0, Y1, Y2, Y3, Y4) which are explained in the code.** :

Notes

Quick Start *getSpec(ra,dec,obsid, ext,)* should produce plots and output files

Which directory?

The program needs to be started from the CORRECT data directory. The attitude file [e.g., “sw<OBSID>pat.fits”]is needed! A link or copy of the attitude file needs to be present in the directory or ”../../auxil/” directory as well.

Global parameters

These parameters can be reset, e.g., during a (i)python session, before calling getSpec.

- **trackwidth** : float width spectral extraction in units of sigma. The default is trackwidth = 2.5 The alternative default is trackwidth = 1.0 which gives better results for weak sources, or spectra with nearby contamination. However, the flux calibration and coincidence-loss correction give currently inconsistent results. When using trackwidth=1.0, rescale the flux to match trackwidth=2.5 which value was used for flux calibration and coincidence-loss correction.
- **write_RMF** : bool set to False since the process is extremely time consuming.

- **give_result** : bool set to False since a call to getSpec with this set will return all the intermediate results. See returns

When the extraction slit is set to be straight `curved="straight"` it cuts off the UV part of the spectrum for spectra located in the top left and bottom right of the image.

`get_radec()`

`uvotpy.uvotgetspec.`**`get_radec`**(*file='radec.usno'*, *objectid=None*, *chatter=0*)

Read the decimal ra,dec from a file or look it up using the objectid name from CDS

Parameters **file: str, optional** :

path, filename of ascii file with just the ra, dec position in decimal degrees

objectid : str, optional

name of object that is recognized by the CDS Sesame service if not supplied a file name is required

chatter : int

verbosity

Returns **ra,dec** : float

Position (epoch J2000) in decimal degrees

`makeXspecInput()`

`uvotpy.uvotgetspec.`**`makeXspecInput`**(*lamdasp*, *countrate*, *error*, *lamda_response=None*, *chatter=1*)

Convert the count rate spectrum per pixel into a spectrum on the given bins of the response function.

Parameters **lamdasp** : array

wavelengths spectrum

countrate : array

count rates at wavelengths

error : array

errors at wavelengths

kwargs : dict

- **lamda_response** : array

  the wavelength for the response bins

- **chatter** : int

  verbosity

Returns **lambda** : array

wavelengths of the bins

countrate : array

count rate in the bins

error : array

errors in the bins

Notes

errors are summed as sqrt(sum (errors**2))

`plan_obs_using_mags()`

`uvotpy.uvotgetspec.`**`plan_obs_using_mags`**(*S2N=3.0*, *lentifilter=None*, *mag=None*, *bkgrate=0.16*, *coi=False*, *obsfile=None*, *grism='uv'*)

Tool to compute the grism exposure time needed to get a certain S/N in the filterband given observed magnitude in lentifilter.

Parameters **S2N** : float

signal to noise desired

lentifilter : str, {'uvw2','uvm2','uvw1','u','b','v'}, optional if *obsfile* given

lenticular filter in which a magnitude is available

mag : float, optional if *obsfile* given

measured magnitude in *lentifilter*.

bkgrate : float

the count rate in the background. This parameter determines for weak spectra to a large extent what exposure time is required.

coi : bool

apply coincidence-loss correction ? *not yet implemented*

obsfile : path, str, optional if *lentifilter*,'mag' given

ascii filename with two columns wave, flux or a fits file with the spectrum in the second extension

grism : str, {'uv'}

Returns **An estimate of the required exposure time is printed** :

Notes

Lentifilter should be one of: uvw2, uvm2, uvw1, u, b, v

Assumed source is faint - no coi (can later add coi)

The exposure time will ramp up quickly once the target gets too faint. The background in the clocked uv grism varies and can be lower, like 0.06 depnding of where the spectrum is put on the detector. So we could update this program at some point with the uv clocked background variation in it. Typically background values are below 0.1 c/s/arcsec. Higher backgrounds are found in crowded fields.

If obsfile is given, then calculate the magnitudes using the spectrum from the obsfile

TO DO: V grism placeholder

- 16 April 2012, initial version, Paul Kuin

`spec_curvature()`

`uvotpy.uvotgetspec.`**`spec_curvature`**(*wheelpos*, *anchor*, *order=1*)

Find the coefficients of the polynomial for the curvature.

Parameters **wheelpos** : int, {160,200,955,1000}

grism filter position in filter wheel

anchor : list, array

anchor position in detector coordinates (pixels)

order : int

the desired spectral order

Returns **Provides the polynomial coefficients for y(x).** :

Notes

The curvature is defined with argument the pixel coordinate in the dispersion direction with reference to the the anchor coordinates in det-img coordinates. The polynomial returns the offset normal to the dispersion.

•2011-03-07 Paul Kuin, initial version

•2011-08-02 fixed nominal coefficients order=1

`spectrumpixshift()`

`uvotpy.uvotgetspec.`**`spectrumpixshift`**(*w1*, *spec1*, *w2*, *spec2*, *wmin=None*, *wmax=None*, *spectrum=False*, *delwav=False*, *chatter=0*)

Accurately determine relative wavelength/pixel shift between 2 spectra.

Parameters **w1,spec1, w2,spec2** : array-like

wavelength, spectrum pairs

kwargs : dict

- **wmin,wmax**: float

  limits to region to use

- **spectrum** : bool

  resample 2nd spectra and return second spectrum shifted

- **delwav** : bool
- **chatter** : int

  verbosity

Returns **k** : int

shift in pixels. option spectrum *False*, for option delwav *False*

delwav : float

shift in angstroms. For option spectrum *False*, option delwav *True*

k, (w1,s2) : int, tuple

pixel shift, tuple of wave, flux for second spectrum shifted and resampled on wavelength first spectrum for option spectrum *True*

Notes

k ~ 1/6 pixel [option: resample 2nd spectra]

`sum_Extimage()`

`uvotpy.uvotgetspec.`**`sum_Extimage`**(*pha_file_list, sum_file_name='extracted_image_sum.fit', mode='create', ankerlist=None, plotimage=True, correlate=True, correlate_wavewindow=[None, None], figno=20, shiftlist=[], clobber=False, chatter=1*)

This routine will create/update/read a summed extracted image.

Parameters pha_file_list : list

list of PHA filenames written by calls of *getSpec*

kwargs : dict

- **sum_file_name** : str

  file name for sum

- **mode** : str, {'create','read'}

  when 'create' make the sum file; when 'read' read the sum file

- **ankerlist** : list, optional

  list of anchor positions

- **plotimage** : bool, optional

  make a plot of the image

- **correlate** : bool, optional

  try to determine shifts by correlating the image

- **correlate_wavewindow** : list

  when correlate *True* then use only the part of the spectrum within [wavemin, wavemax]

- **figno** : int, optional

  figure number to use

- **shiftlist** : list, optional

  list of shifts to apply

- **clobber** : bool

  write over existing file

- **chatter** : int

  verbosity

Returns **When ‘option=read‘ the following are returned:** :

- sumimg : 2D array

summed image

- expmap : 2D array

exposure map for each pixel in summed image

- exposure : float

exposure time (maximum)

- wheelpos : int

grism wheel position

- C_1, C_2 : list

dispersion coefficients

- dist12 : float

distance in pixels between the first and second order anchors

- anker : list

anchor position in summed image

- coefficients : tuple

(coef0,coef1,coef2,coef3,sig0coef,sig1coef,sig2coef,sig3coef) curvature and sigma coefficients for the summed image

- hdr : fits header

Notes

The anchor point, by default, will be at point [100,500]

mode = ‘create’ <make new sum file>, ‘read’ <read sum file>

The anchor position in the pha_file will need to be passed via ankerlist or be given as keyword ANKXIMG, ANKYIMG in the header of the PHA file (it is).

when correlate_wavewindow = [none,none] nothing is done = [2300,4000] wavelength range where to do cross correlation on flux to generate corrections to ankx

shiftlist = [None, 0, -2, None] can be used to force the shifts (in pix) of the given number in the list of spectra (here assumed to be four. List length must equal pha_file_list length.

Example:

phafiles = [’sw00032150003ugu_1_pha.fits’,’sw00032150003ugu_2_pha.fits’, ‘sw00032150003ugu_3_pha.fits’, ‘sw00032150003ugu_4_pha.fits’, ‘sw00032150003ugu_5_pha.fits’, ‘sw00032150003ugu_6_pha.fits’, ‘sw00032150003ugu_7_pha.fits’, ‘sw00032150003ugu_8_pha.fits’, ‘sw00032150003ugu_9_pha.fits’, ‘sw00032150003ugu_10_pha.fits’, ‘sw00032150003ugu_11_pha.fits’, ‘sw00032150003ugu_12_pha.fits’, ‘sw00032150003ugu_13_pha.fits’]

uvotgetspec.sumimage(phafiles, mode=’create’,chatter=1,clobber=True)

Paul Kuin 2011 (MSSL/UCL)

`sum_PHAspectra()`

`uvotpy.uvotgetspec.`**`sum_PHAspectra`**(*phafiles*, *wave_shifts=[]*, *exclude_wave=[]*, *ignore_flags=True*, *interactive=True*, *outfile=None*, *figno=[14]*, *ylim=[]*, *chatter=1*, *clobber=True*)

Read a list of phafiles. Sum the spectra after applying optional wave_shifts.

Parameters **phafiles** : list

list of filenames

wave_shifts : list

list of shifts to add to the wavelength scale; same length as phafiles

exclude_wave : list

list of lists of exclude regions; same length as pha files; one list per file

ignore_flags : bool

do not automatically convert flagged sections of spectrum to exclude_wave regions

interactive : bool

if False, the program will only use the given wave_shifts, and exclude_regions

outfile : str

name for output file. If "None" then an array of data for debug is returned

ylim : list

force limits of Y-axis figure

figno : int, or list

numbers for figures or (if only one) the start number of figures

Returns **debug information when 'outfile=None'.** :

Notes

Two figures are shown, one with flux for all spectra after shifts, one with broad sum of counts in a region which includes the spectrum, unscaled, not even by exposure.

1.5.3 uvotio

Functions

`SpecResp()`

`uvotpy.uvotio.`**`SpecResp`**(*wheelpos*, *spectralorder*, *arf1=None*, *arf2=None*)

Returns spectral response function

Parameters **wheelpos** : int, {160,200,955,1000}

spectralorder : int, {1,2}

Returns **An interpolating function for the spectral response** :

as a function of wavelength (A) :

Notes

Use *readFluxCalFile()* in case the new calibration file is present

XYSpecResp()

`uvotpy.uvotio.`**`XYSpecResp`**(*wheelpos=None, spectralorder=1, anker=[1129, 1022], test=None, chatter=0*)

the spectral response based on the position of the anchor of the spectrum. Depends on the grism mode via 'wheelpos' and the spectral order.

Parameters **wheelpos** : int

kwargs : dict

- **spectralorder** [int] order of the spectrum
- **anker** [list] position in detector coordinates (pixels)
- **test** [any] if not None then get the response at the boresight

Returns **An interpolating function for the spectral response** :

based on the position (Xank,Yank) of the anchor of the spectrum. :

Depends on the grism mode via 'wheelpos' and the spectral order. :

Notes

Will be superseded by *readFluxCalFile*

fileinfo()

`uvotpy.uvotio.`**`fileinfo`**(*filestub*, *ext*, *lfilt1=None*, *directory='./'*, *chatter=0*, *twait=40.0*)

finds files for spectrum, matching attitude and lenticular images uncompresses gzipped files if found

Parameters **filestub** : str

the base of the file name, with the Swift project convention, consisting of "sw" + the OBSID, i.e., "sw00032301001"

ext : int

the number of the extension of the grism file to match

kwargs : dict

- **lfilt1** : str, optional

  name of first lenticular filter used. Must be one of 'uvw2', 'uvm2','uvw1','u','b','v','wh'

- **directory** : path, str

  path for directory. This is the directory with the grism file.

- **chatter** : int

verbosity

- **twait** : float

  The maximum time allowed between begin and end of matched exposures of grism-lenticular filter, for each match.

Returns **specfile, attfile: str** :

filename of spectrum, the attitude file name.

lfilt1, lfilt2 : str

lenticular filter file name before the grism exposure or None, and the file name of the lenticular filter following the grism

lfilt1_ext,lfilt2_ext : int

extension number for each lenticular filter matching exposure

`getZmxFlux()`

`uvotpy.uvotio.`**`getZmxFlux`**(*x*, *y*, *model*, *ip=1*)

Interpolate model to get normalized flux.

Parameters **x, y** : float

anchor coordinate x,y to find an interpolated solution to the model

model : fits structure

binary table extension (header + data) fields are wave, xpix, ypix, flux

ip : int

The order of the interpolation (1=linear, 2=quadratic, 3=cubic)

Returns **flux interpolated at (x,y) in (xpix, ypix) as function of wave** :

`kev2angstrom()`

`uvotpy.uvotio.`**`kev2angstrom`**(*E*, *unit='keV'*)

conversion of units

Returns **The photon wavelength in angstroms** :

`rate2flux()`

`uvotpy.uvotio.`**`rate2flux`**(*wave, rate, wheelpos, bkgrate=None, pixno=None, sig1coef=[3.2], sigma1_limits=[2.6, 4.0], arf1=None, arf2=None, spectralorder=1, trackwidth=1.0, anker=None, test=None, respfunc=False, swifttime=None, option=2, fudgespec=1.0, frametime=0.0110302, debug=False, chatter=1*)

Convert net count rate to flux

WARNING: dependent on the parameters passed, the old CALDB (<=2012) OR the new flux calibration will be used. Since 10SEP2012 the coi-factor is included in the calculation of the flux and the effective area. A coi-correction is still made when using the old CALDB which will be inconsistent to that calculated in the writeSpectrum() which makes the output file.

many of the parameters are needed to calculate the coi-factor

Parameters **wave** : float ndarray

wavelength in A

rate : float ndarray

net count rate spectrum, aperture corrected

wheelpos : int

filter wheel position

bkgrate : ndarray

background rate for spectrum (2.5 sigma half-width)

pixno : ndarray

pixel coordinate (zero = anchor; + increasing wavelengths)

sig1coef : list

polynomial coefficient for spectrum halfwidth

sigma1_limits : list

sigma will be truncated at the limits

arf1 : path or "CALDB", optional

when present: use this effective area for the first order

arf2 : path or "CALDB, optional

when present: use this effective area for the second order only one of arf1 or arf2 can be present in a call

spectralorder : int

the spectral order of the spectrum, usually =1

trackwidth : float

width of the spectral extraction used in units of sigma

anker : list

anchor detector coordinate positions (pix) as a 2-element numpy array

frametime : float

the frame time for the image is required for the coi-correction

swifttime : int

swift time of observation in seconds for calculating the sensitivity loss

option : int

when =2 sets the method for the coi-correction [development]

fudgespec : float

possible correction for the coi-correction [development]

debug : bool

for development

chatter : int

verbosity (0..5)

respfunc : bool

return the response function (used by writeSpectrum())

Returns **flux, coi-corrected** :

Notes

2013-05-05 NPMKuin - adding support for new flux calibration files; new kwarg

`readFluxCalFile()`

`uvotpy.uvotio.`**`readFluxCalFile`**(*wheelpos*, *anchor=None*, *option='default'*, *spectralorder=1*)

Read the new flux calibration file, or return None.

Parameters **wheelpos** : int, required

the position of the filterwheel

kwargs: dict :

- **anchor** : list, optional coordinate of the anchor
- **option** : str option for output selection:

  option=="default" + anchor==None: old flux calibration option=="default" + anchor : nearest flux calibration + model extrapolation option=="nearest" : return nearest flux calibration option=="model" : model

- **spectralorder** [int] spectral order (1, or 2)

Returns **None if not (yet) supported** :

option == 'model' returns the fits HDU of the model :

option == 'nearest' :

returns the fits HDU of the nearest calfile

option == 'default' and anchor == None: :

returns the fits HDU of the nearest calfile

option == 'default' and anchor position given (in detector :

pix coordinates) returns the fits HDU and an interpolating function fnorm(wave in A)for the flux correction

Notes

2013-05-05 NPMKuin

`sensitivityCorrection()`

`uvotpy.uvotio.`**`sensitivityCorrection`**(*swifttime*, *sens_rate=0.01*)

give the sensitivity correction factor to divide the rate/flux by

Parameters **swifttime** : float

time of observation since 2005-01-01 00:00:00 in seconds, usually TSTART

sens_rate : float

the yearly percentage loss in sensitivity

Notes

A 1%/year decay rate since 2005-01-01 has been assumed and the length of the mean Gregorian year was used

`updateResponseMatrix()`

`uvotpy.uvotio.`**`updateResponseMatrix`**(*rmffile*, *C_1*, *clobber=True*, *lsffile='zemaxlsf'*, *chatter=0*)

modify the response matrix lineprofiles using the zemax model prediction from zemaxlsf.fit In addition the zemax profile is broadened by the instrumental broadening of 2.7 pixels.

Parameters **rmffile** : path, str

The rmffile is updated by default

C_1: ndarray :

The dispersion C_1 is used to convert pixels to angstroms.

kwargs : dict

- *lsffile* : path

  The lsffile is in the $UVOTPY/calfiles directory
- *clobber* : bool

  overwrite output.
- *chatter* : int

  verbosity

Returns **writes RMF file** :

Notes

The same algorithm was implemented in the write_rmf_file() routine which does not need the input rmf file produced by the "Ftool" *rmfgen*.

`writeSpectrum()`

`uvotpy.uvotio.`**`writeSpectrum`**(*ra*, *dec*, *filestub*, *ext*, *Y*, *fileoutstub=None*, *arf1=None*, *arf2=None*, *fit_second=True*, *write_rmffile=True*, *history=None*, *chatter=1*, *clobber=False*)

Write a standard UVOT output file - Curved extraction only, not optimal extraction.

Parameters **ra,dec** : float, float

position in decimal degrees

filestub : str

"sw" + obsid

ext : int

extension number

Y : tuple

compound variable with spectral data from uvotgetspec

Returns **Writes the output file only.** :

Notes

Output file composition

Main header: RA DEC extracted source; anchor point, input angles, ank_c, angle, offset (distance anchor to spectrum) in arcsec

The first extension is named SPECTRUM (future: 'FIRST_ORDER_PHA_SPECTRUM') and contains the standard input for XSPEC Includes processing HISTORY records from *getSpec()*

The second extension is named CALSPEC (future: 'FIRST_ORDER_NET_SPECTRUM') contains the standard input for IDL/PYTHON with pixelno(relative to anchor), wave(nm), net_count_rate(s-1 pix-1), left_background_rate(s-1 pix-1), right_background_rate (s-1 pix-1), flux(erg cm-2 s-1 A-1), flux_err(erg cm-2 s-1 A-1), quality flag, aper1corr second order: wave2, net2rate, bg2rate, flux2, flux2err, qual2, aper2corr and coincidence-loss factor as applied to the flux listed.

The third extension named 'SPECTRUM_IMAGE' contains the image of the total spectrum

A fourth extension may exist for spectra from summed images and contains the **exposure map**

history

- 2011-12-10 NPM Kuin
- 2012-03-05 fixed error in calculation bg1rate(a.k.a. bg_r) and bg2rate (or bg_l)
- 2012-09-14 added coi correction
- 2013-03-06 edited header

`write_response_arf()`

`uvotpy.uvotio.`**`write_response_arf`**(*wheelpos*, *spectralorder*, *wave*, *specresp*, *specresp_err=None*, *anker=None*, *dxy_anker=None*, *fileversion='999'*, *clobber=False*)

create an ARF file

Parameters **wheelpos** : int, {160,200,955,1000}

spectralorder: int, {1,2} :

wave: ndarray :

wavelengths in Angstrom

specresp: ndarray :

effective area (EA) in cm^2 for each wave

specresp_err: ndarray :

1-sigma EA error (random + systematic)

anker: list, ndarray[2] :

2-element array with position in det coordinates of EA

dxy_anker: list,ndarray[2] :

EA determined for box [anker[0]+/-dxy_anker[0], anker[1]+/-dxy_anker[1]]

fileversion: str :

version for this EA response ARF file.

Returns **the new effective area file with file name something like:** :

'swugu0160_ax1100ay1100_dx150dy150_o1_20041120v001.arf'

Notes

- Modified 15-SEP-2012 by Paul Kuin.

With only wheelpos, spectralorder, wave, specresp input, the output file conforms to the HEASARC approved response file. The additional keywords and error column have not been approved as of 15 September 2012.

- Modified 13 Feb 2013 by Paul Kuin

Added futher keyword COIAWARE to discriminate between the old and new effective areas and changed comments after keywords to be more descriptive.

- Modified 5 March 2013 by Paul Kuin

header edited

`spec_curvature()`

`write_rmf_file()`

`uvotpy.uvotio.`**`write_rmf_file`**(*rmffile*, *wave*, *wheelpos*, *spectralorder*, *disp*, *arf1=None*, *arf2=None*, *chatter=1*, *clobber=False*)

Write the RMF file

Parameters **rmffile** : path, str

file name output file

wave [ndarray] wavelengths of the bins

spectralorder [int] 1 or 2

disp [ndarray] dispersion coefficients

- **arf1** : path, str first order ancillary response file
- **arf2** :path, str second order ancillary response file
- **chatter** : int verbosity
- **clobber** : bool if true overwrite output file if it already exists

Returns **Writes the RMF file** :

Notes

The routine is very slow, but uses the best known PSF which varies with wavelength.

1.5.4 uvotmisc

Functions

`UT2swift()`

`uvotpy.uvotmisc.`**`UT2swift`**(*year*, *month*, *day*, *hour*, *minute*, *second*, *millisecond*, *chatter=0*)

Convert time in UT to swift time in seconds.

Parameters **year** : int

e.g., 2012

month : str

e.g., 'JAN'

day : int

e.g., 21

hour : int

minute : int

second : int

millisecond : int

Returns **swifttime** : float

in seconds (see Heasarc for more conversions)

`get_keyword_from_history()`

`uvotpy.uvotmisc.`**`get_keyword_from_history`**(*hist*, *key*)

Utility to get the keyword from the history list.

Parameters **hist** : list

key : str

Returns **value belonging to key or 'None'.** :

Notes

The history records are written while processing getSpec() and added to the FITS header of the output file.

These can be read from the header by just getting *all* the history records.

`rdList()`

`uvotpy.uvotmisc.`**`rdList`** (*file*, *symb=' '*, *chatter=0*, *line1=None*, *line2=None*, *skip='#'*)

Put data in list: chatter>4 gives detailed output restrict lines in file with line1, line2 skip lines with the skip char in first position

Parameters **file** : str

file name ascii table

symb : str

character used to split out the columns

line1,line2 : int

sub-select records[line1:line2]

chatter : int

Returns **table** : ndarray

a table of values

Notes

The table must have equal length columns and the same number of fields on each row/record.

Use rdTab to read a table with numerical only data

`rdTab()`

`uvotpy.uvotmisc.`**`rdTab`** (*file*, *symb=' '*, *commentsymb='#'*, *get_comments=False*)

RdTab will read in a table of numerical values provided every record has the same number of fields. Comment lines start by default with a hash mark, but that can be changed by passing another symbol in commentsymb comments in data records are not supported.

Parameters **file** : str

file name ascii table

symb : str

character used to separate the columns

commentsymb : str

character used in first position of line for comments

get_comments : bool

if True, return comments only

Returns **table** : ndarray

a table of values

Notes

The table must have equal length columns with only numbers.

Use rdList to read a table with character data

NPMK (MSSL) 2010

`swtime2JD()`

`uvotpy.uvotmisc.`**`swtime2JD`**`(`*TSTART*`)`

Time converter to JD from swift time

Returns **JD** : float

Julian Date

MJD : float

Modified Julian Date

gregorian : str

normal date and time

outdate : datetime

python datetime object

Notes

example (input TSTART as a string) for 2001-01-01T00:00:00.000

TSTART=0.000 MJD=51910.00000000 JD=2451910.5

`uniq()`

`uvotpy.uvotmisc.`**`uniq`**`(`*list*`)`

preserves order

`uvotrotvec()`

`uvotpy.uvotmisc.`**`uvotrotvec`**`(`*X*, *Y*, *theta*`)`

rotate vectors X, Y over angle theta (deg) with origen [0,0]

Parameters **X, Y** : arrays

coordinates

theta : float

angle in degrees

Returns **rx,ry** : arrays

rotated coordinates

1.5.5 uvotplot

Functions

`Ellipse()`

`uvotpy.uvotplot.`**`Ellipse`**(*(x, y)*, *(rx, ry)*, *angle=0.0*, *resolution=200*, ***kwargs*)

plot an ellipse using an N-sided polygon

Parameters **(x,y)** : float

centre ellipse

(rx,ry) : float

half axis ellipse

angle : float

angle in units of degrees

resolution : int

determines number of points to use

and additional kwargs for pyplot.plot() :

`binplot()`

`uvotpy.uvotplot.`**`binplot`**(**args*, ***kwargs*)

Bin up the arrays with the keyword bin=<number> Same parameters as used by plot (pyplot)

`contourpk()`

`uvotpy.uvotplot.`**`contourpk`**(*x*, *y*, *f*, *levels=None*, *xb=None*, *xe=None*, *yb=None*, *ye=None*, *s=60*, *kx=1*, *ky=1*, *dolabels=True*, ***kwargs*)

Make a contour plot with 1-D array inputs for X, Y and F. This is a wrapper to convert lists of points (X,Y,Z) in 2-D arrays, then calls contour()

Parameters **X, Y: ndarrays[:], 1D on a 2D plane** :

coordinates X, Y

Z : ndarray[:], 1D function on X,Y

`maskEllipse()`

`uvotpy.uvotplot.`**`maskEllipse`**(*maskimg*, *x*, *y*, *a*, *b*, *theta*, *test=0*, *chatter=1*)

update a mask excluding ellipse region

Parameters **maskimg** : ndarray, 2D, bool

boolean array to aplly mask to (i.e., numpy.ones(array([200,400]),dtype=bool))

x,y : int, float

ellipse center coordinate x,y

a,b : float

ellipse major axis a; minor axis b;

theta : float

rotation angle theta counterclockwise in deg.

Returns **maskimg with all pixels inside the ellipse are set to False** :

plot_ellipsoid_regions()

`uvotpy.uvotplot.`**`plot_ellipsoid_regions`**(*Xim*, *Yim*, *Xa*, *Yb*, *Thet*, *b2mag*, *matched*, *ondetector*, *img_pivot*, *img_pivot_ori*, *img_size*, *limitMag*, *img_angle=0.0*, *lmap=False*, *makeplot=True*, *color='k'*, *annulusmag=13.0*, *chatter=1*)

This routine is to plot ellipsoid regions on the grism image/graph, which may be a rotated, cropped part of the detector image

Returns **None or boolean map image, plots an ellipse on the current figure** :

waveAccPlot()

`uvotpy.uvotplot.`**`waveAccPlot`**(*wave_obs*, *pix_obs*, *wave_zmx*, *pix_zmx*, *disp_coef*, *obsid=None*, *acc=None*, *order=None*, *wheelpos=200*, *figureno=1*, *legloc=[1, 2]*)

Plots of the accuracy of the wavelength solution from zemax compared to the observed wavelengths.

Parameters **wave_obs, pix_obs** : ndarray

observed wavelengths points (green circles)

wave_zmx ,pix_zmx : ndarray

calculated zemax points (or the interpolated solution (red crosses)

disp_coef : ndarray

dispersion coefficients

disp_coef : list

coefficients in reverse order: if p is of length N, this the polynomial

is as follows for coeff named p:

y(x) = p[0]*(x**N-1) + p[1]*(x**N-2) + ... + p[N-2]*x + p[N-1]

kwargs : dict

- **acc** : accuracy in wavelength
- **order** : order of polynomial disp_coef (default len(coef))
- **obsid** : if given, append to title

Notes

Figure description

x-axis : pix - pixel number referenced to [260nm in first order]

Top panel only

y-axis: lambda - lambda_linear

linear term in the dispersion a linear term is fit to the wavelengths

$lambda_{lin}$ = coef[0]+coef[1]*pix

Bottom panel only

y-axis: residuals

wave_obs, pix_obs - wave(pix_obs) (green circles) wave_zmx, pix_zmx - wave(pix_zmx) (red crosses)

1.5.6 uvotwcs

Functions

makewcshdr()

`uvotpy.uvotwcs.`**`makewcshdr`**(*RA*, *DEC*, *filestub*, *ext*, *attfile*, *datadir='./'*, *teldef=None*, *wheelpos=200*, *randnr=None*, *graspcorr='no'*, *chatter=1*)

make the header of a lenticular filter for a grism source with known sky position (RA, DEC) to use in uvotapplywcs returns the name of the file

Parameters **RA,DEC** : float

target sky position in decimal degrees

filestub : str

identifying part of filename being "sw"+'obsid'

ext : int

extension of fits file to process

attfile : str

attitude file name

datadir : str

path, directory of data files

teldef : str

filename *teldef* file for epoch of anchor calibration

wheelpos : int, {160,200,955,1000}

filterwheel position for grism

randnr : float

random number

graspcorr: str :

if 'no' do not process the file with the *graspcorr ftool*

chatter : int

verbosity

Returns **creates a fake sky file with appropriate header to run 'findInputAngle'** :

Notes

need to update the tstart and tstop of the primary header (not a showstopper)

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

INDEX