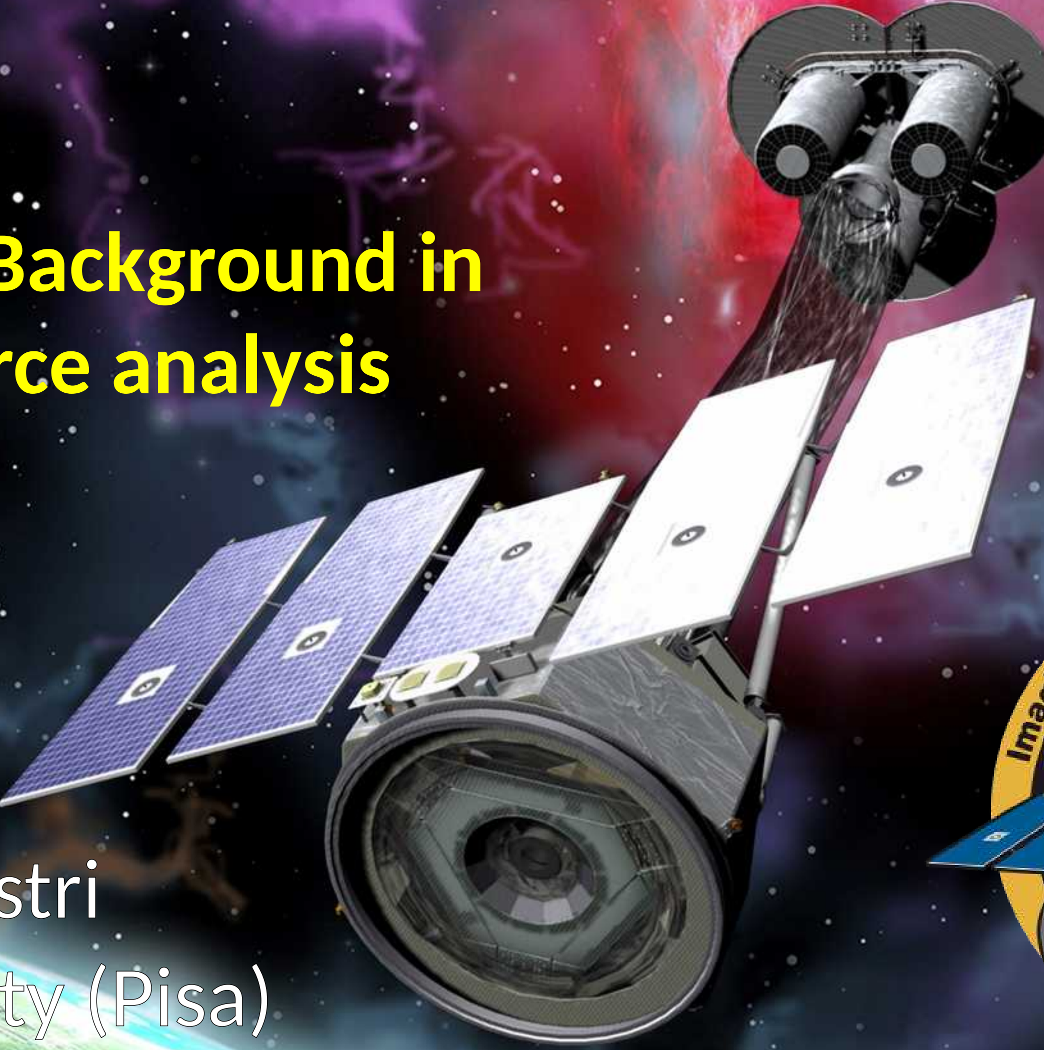
# Handling the Background in extended source analysis



Stefano Silvestri
INFN and University (Pisa)
(on behalf of the IXPE team)

# Our background components

Over the course of the first two years we identified several background components, each requiring specific care

Straightforward

- Cosmic rays triggering the detector directly
  → Tracks are morphologically distinct from X-rays, radial trend

Situational

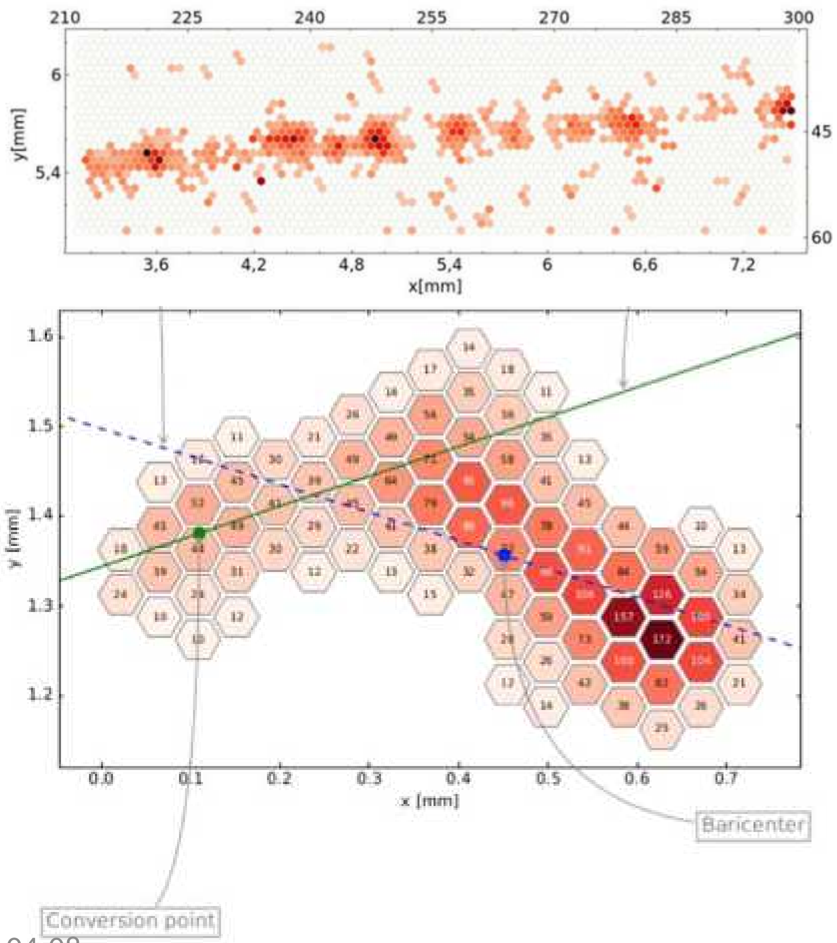
- An isotropic X-ray background of instrumental origin
  → Coordinate independent, indistinguishable from other events
- A DU-dependent time variable X-ray background of unknown origin
  → Compatible with a line at ~1.5 keV (aluminium fluorescence from solar activity?), affecting the three DUs differently

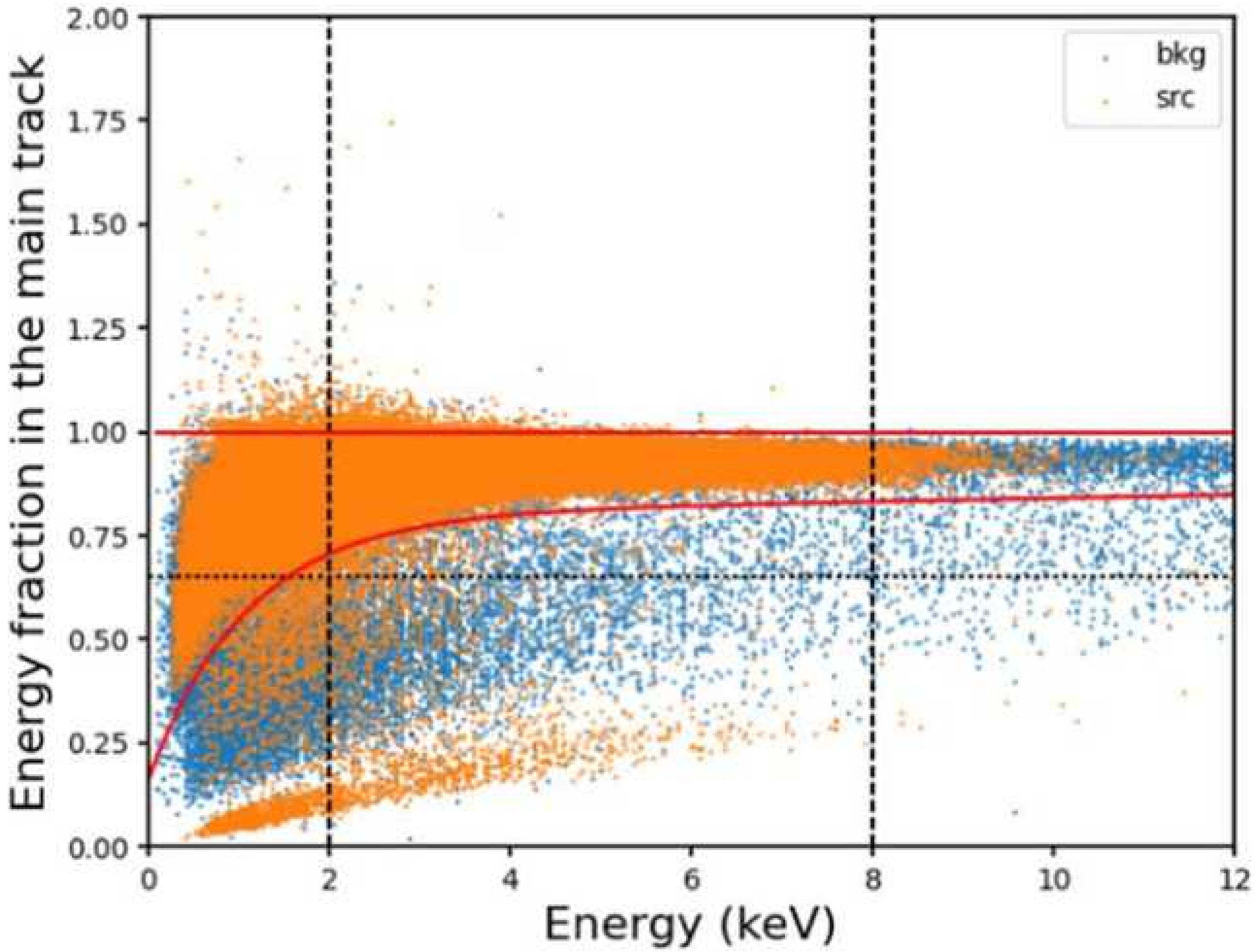
Sorcery

# Cosmic rays

# Event morphology

Morphological difference



IXPE Imaging X-Ray Polarimetry Explorer

Marshall Space Flight Center

Agenzia Spaziale Italiana

# Cosmic ray rejection

https://heasarc.gsfc.nasa.gov/docs/ixpe/analysis/

About IXPE | What's New | FAQ / Help | Related Sites | Galle

## IXPE Data Analysis

**Getting Started with IXPE Data**

The IXPE team has compiled a Quick Start Guide for users who are new to analyzing IXPE data.

**IXPE Data**

On the IXPE archive are released the processed (level 2) event files for each observation. These can be analyzed with the standard multi-mission HEASARC software tools such as ximage and xselect. These can be used to create spectra, images, and light curves for science analysis.

Note that the archive also contains raw (level 1) event files as well as housekeeping data that allows an advanced user to reprocess the data with the IXPE pipeline tools. But this is not needed for most users, since the level 2 events are already highly processed by the IXPE team.

**IXPE Software Package**

The IXPE software package has been included in HEASoft and first released with HEASoft 6.30 (March 2022) The package includes software routines that process level 1 event data with various corrections and selections to create science-ready level 2 event files. The package also includes several tools to generate related products.

The current version of the IXPE software package is distributed with the latest HEAsoft distribution, which users are encouraged to use.

In addition to the standard HEASoft tools, there is also user-contributed software which has been produced by the community that IXPE users might find helpful.

From di Marco et al. (2023): Morphological ID of the track criteria

1. number of pixels $<70+30\times E$;
2. $0.8\times[1-e^{-\frac{E+0.25}{1.1}}]+0.004\times E<$ energy fraction $<1$;
3. border pixels $<2$.

Description of the tool

- Associates the level 1 events to level 2 events
- Filters through the relevant columns (EVT_FRA, NUM_PIX, TRK_BORD)

Outputs:

- A level 2 file which is excised of cosmic ray events

Checks:

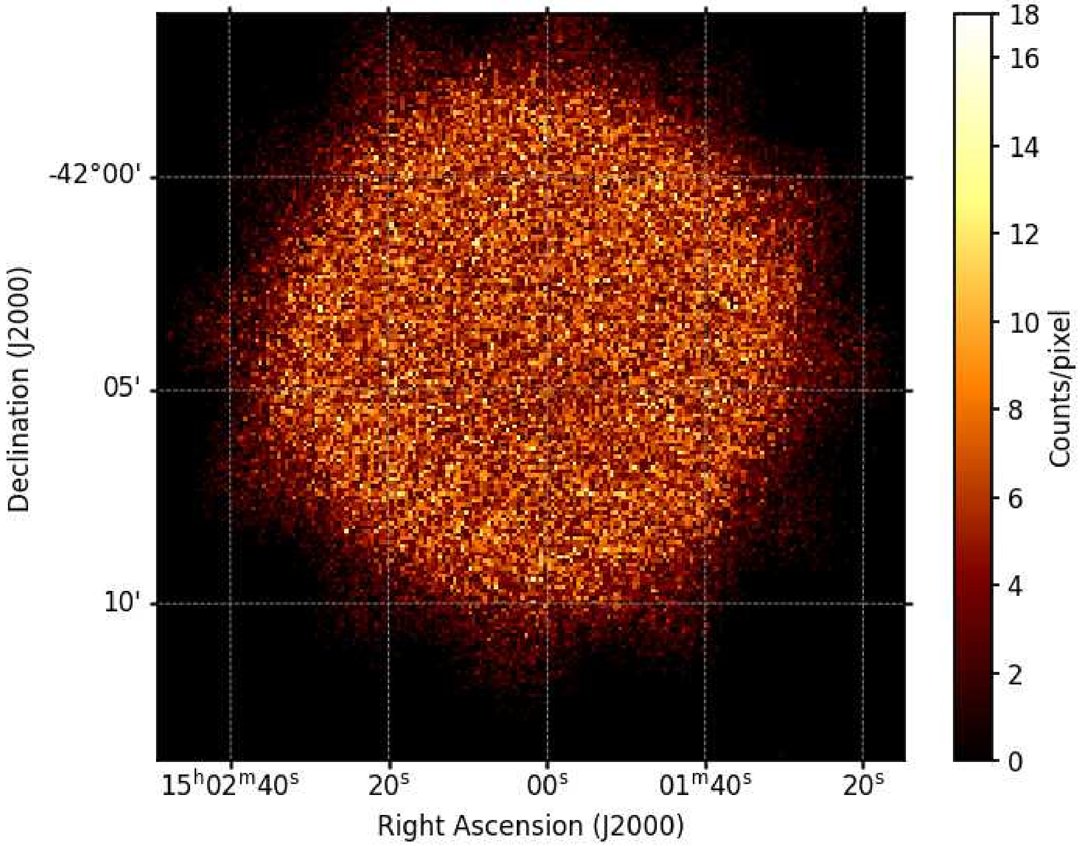
- You can check by making a map and noticing no trace of the source

- **filter_background.py** is a simple to use, stand-alone, tool to reject particle background following the prescription described in Di Marco, et al. AJ 165, 143 (2023). Given a Level 2 event list and its corresponding Level 1 file(s), the tool can be used to create a new Level 2 file that either removes events identified as background events, keeps only events identified as background events, or adds a new column that flags each event as either being background (particle) or non-background (X-ray) events. The tool is available on Github.

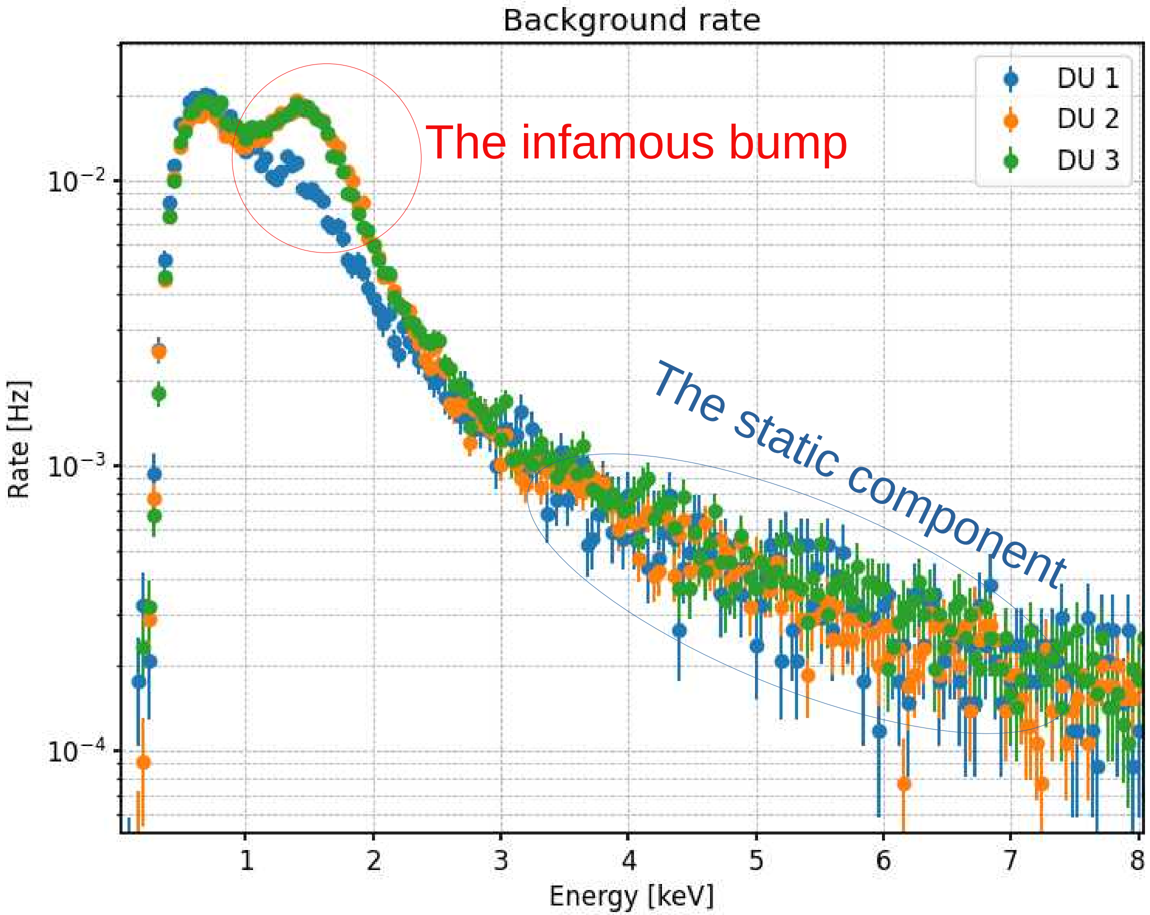
# Residual background

# Characteristics

## Plane of the sky distribution

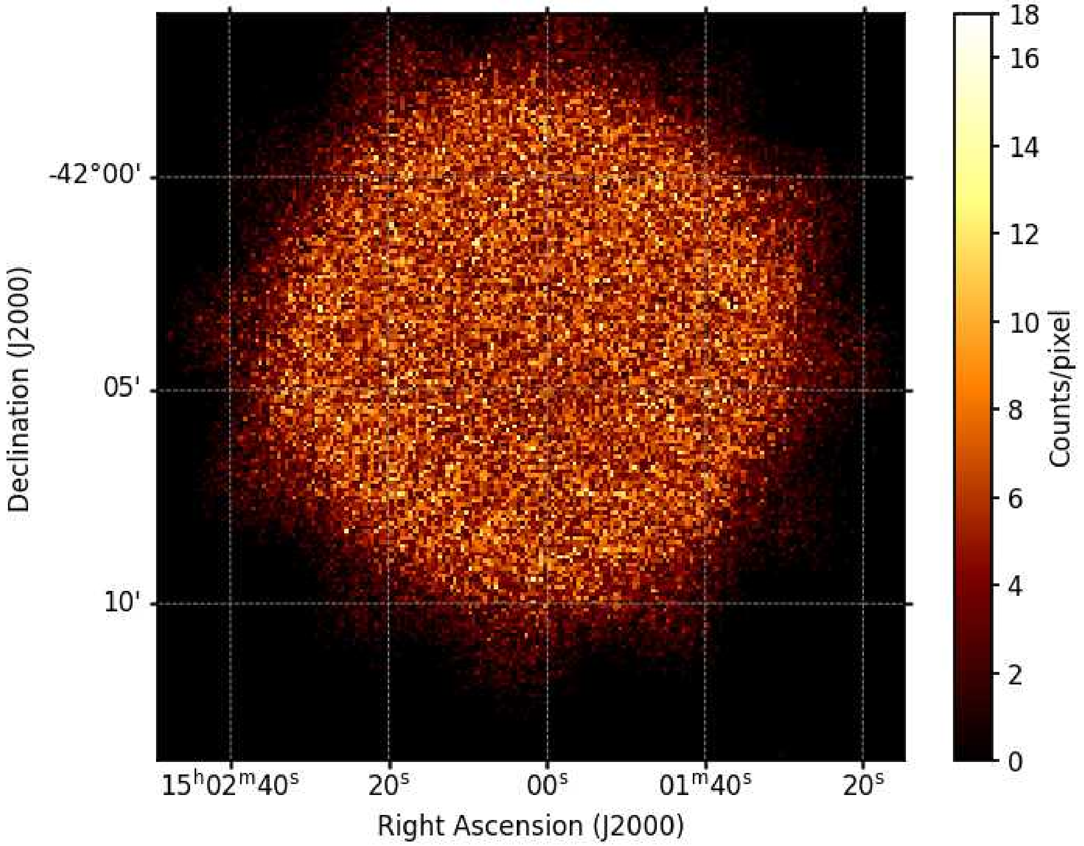



## Spectrum

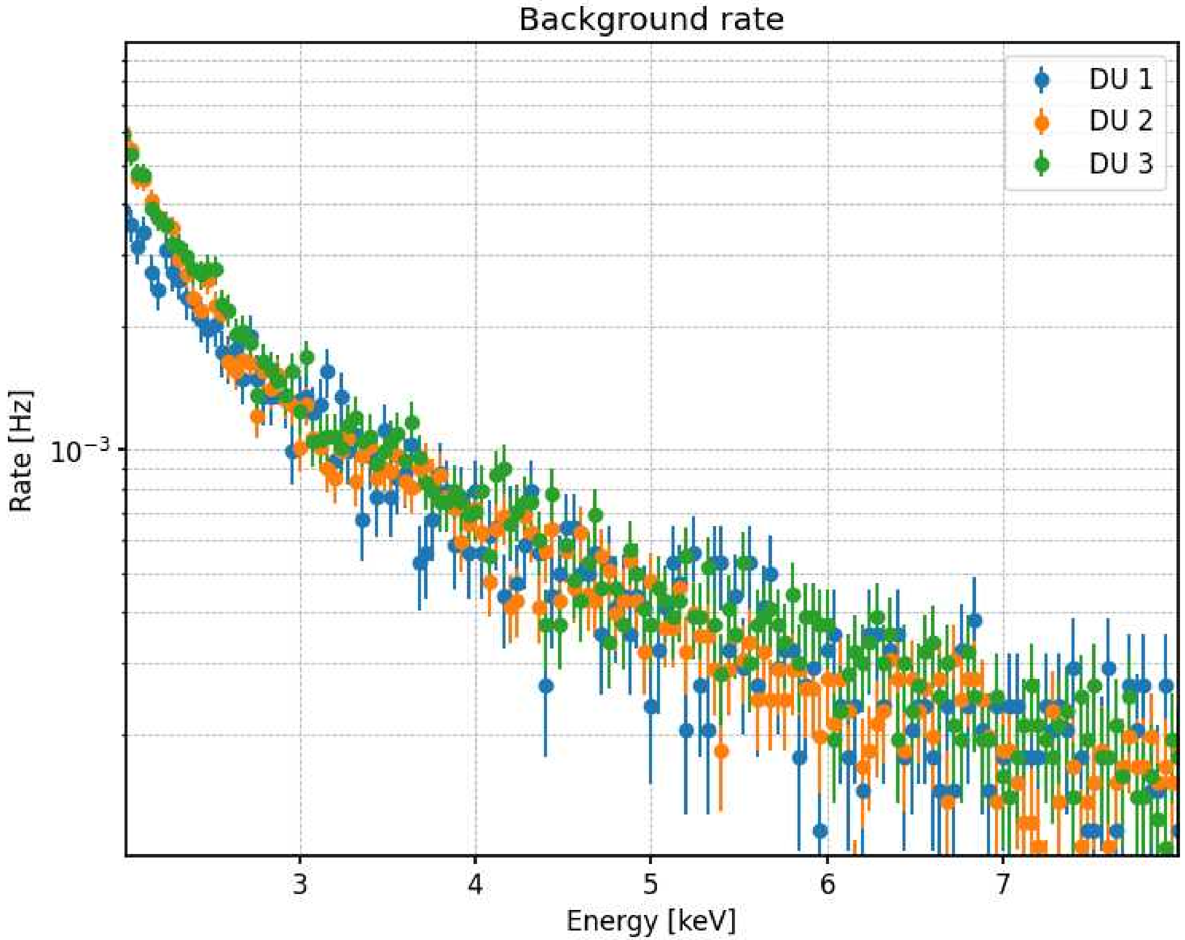


# Characteristics

## Plane of the sky distribution




## Spectrum



# Static component

No *evident* time nor space nor detector unit variability

- Estimate from an off-source region
- Estimate from another observation
- Stack other observations

- No evidence of polarization → simple dilution, we can chill

IXPE Imaging X-Ray Polarimetry Explorer

# Time-variable background

# The infamous 1.5 keV bump

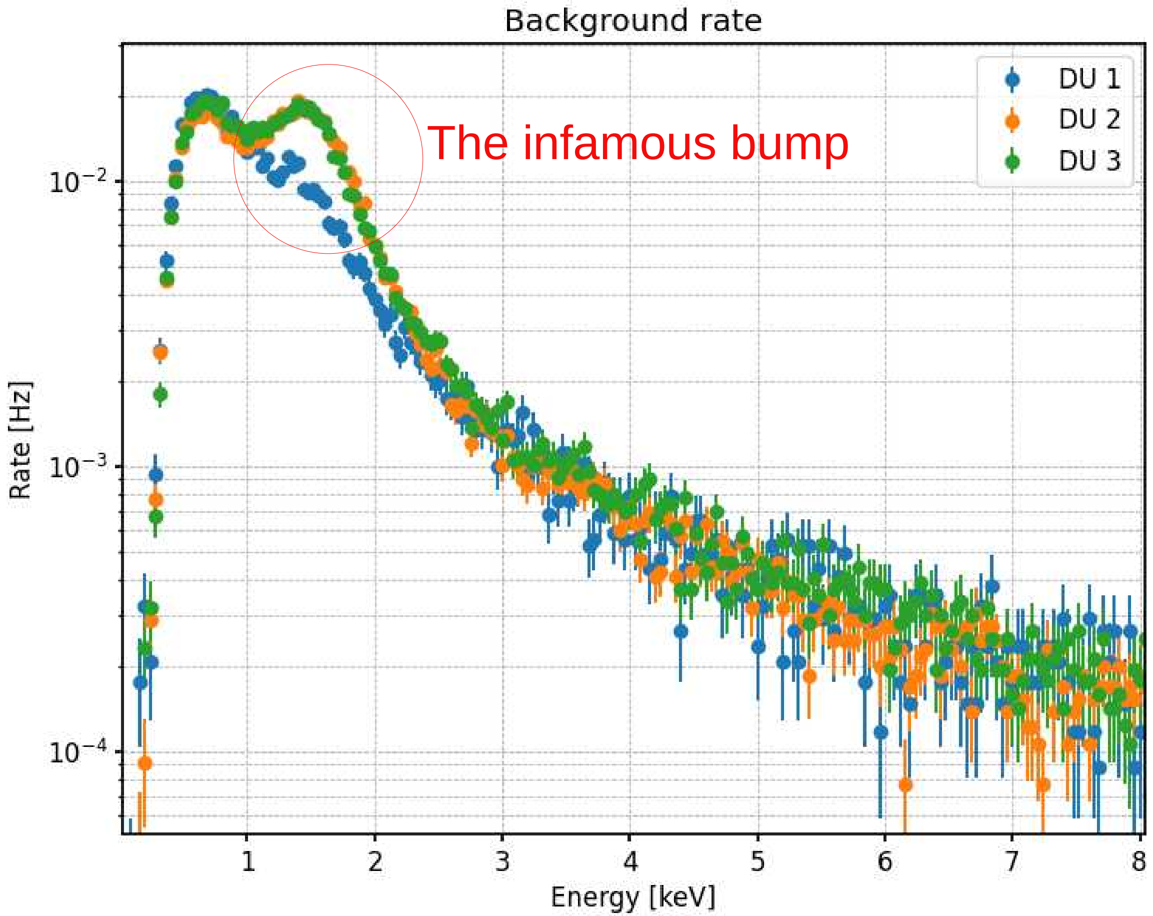
All previous strategies of characterization fail

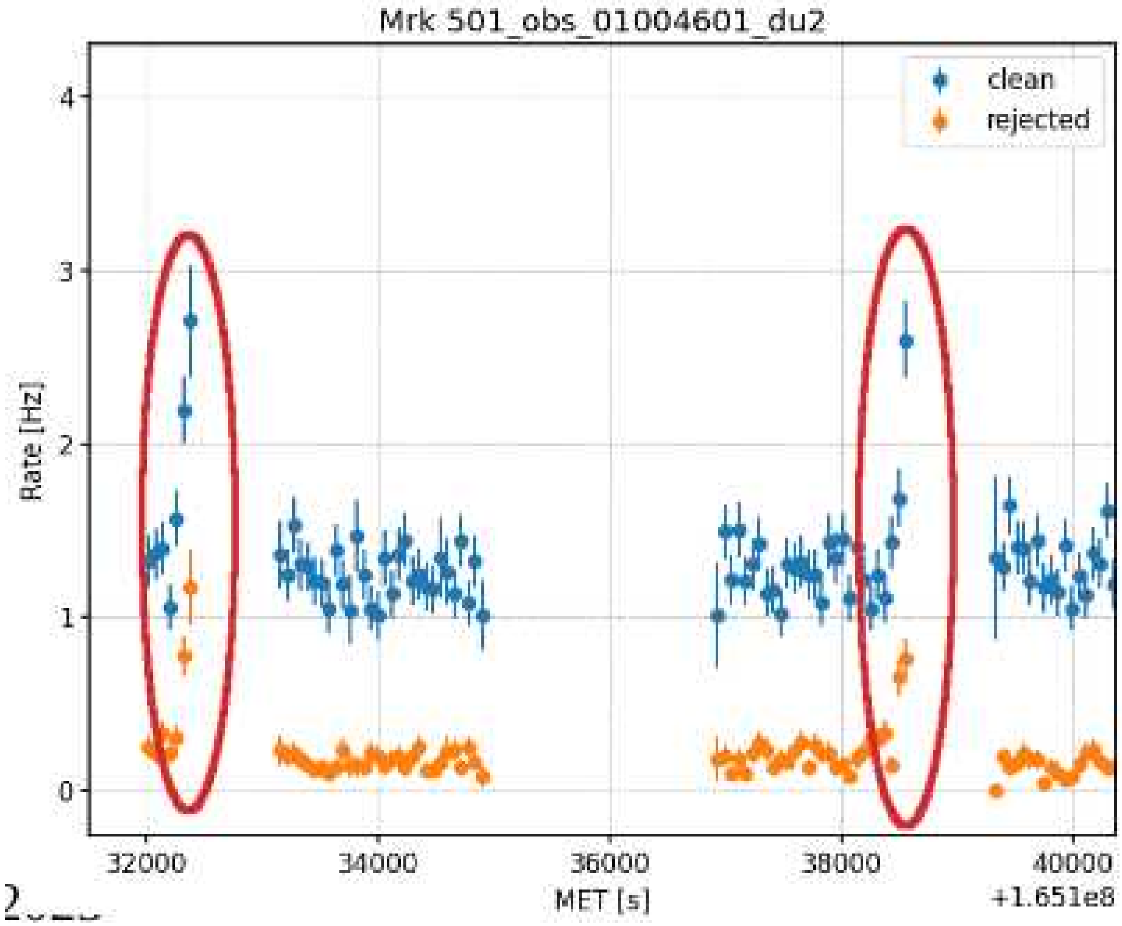
- DU-wise variability
- Short timescale variability (e.g.: “Flares”)
- Some hints of polarization → GET IT OFF ME!

A shot in the dark

- Flaring activity from the sun, directional and mildly polarized
- DU1 is more protected than DU2 and DU3, but polarization properties vary

# The infamous 1.5 keV bump




IXPE
Imaging X-Ray Polarimetry Explorer
~~Hands~~ Eyes-on session
With online Content!

# 1. Cosmic ray rejection

We'll be using obsid 02009701 for this example, we'll do a rejection and a simple sanity check which is often
The only thing that needs to be done on point sources

```
s/02009701$ time python filter_background.py event_l2/ixpe02009701_det1_evt2_v01.fit
s.gz event_l1/ixpe02009701_det1_evt1_v01.fits.gz
event_l2/ixpe02009701_det1_evt2_v01.fits.gz  opening
Filename: event_l2/ixpe02009701_det1_evt2_v01.fits.gz
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      62   ()
  1  EVENTS        1 BinTableHDU    225   272951R x 10C   ['J', 'D', '16X', '16X', '
J', 'E', 'E', 'E', 'D', 'D']
  2  GTI           1 BinTableHDU     53   571R x 2C   [D, D]
event_l1/ixpe02009701_det1_evt1_v01.fits.gz  opening
Filename: event_l1/ixpe02009701_det1_evt1_v01.fits.gz
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      53   ()
  1  EVENTS        1 BinTableHDU    206   3128989R x 39C   ['1J', '1J', '1J', '1J',
'1D', '1J', '1I', '1I', '1I', '1I', '1J', '1I', '1I', '1I', '16X', '16X', '1I', '1I'
, '1E', '1E', '1I', '1I', '1J', '1E', '1E', '1E', '1E', '1E', '1E', '1E', '1E', '1E'
, '1E', '1E', 'QI(3952)', 'E', 'E', 'E', 'E']
  2  GTI           1 BinTableHDU     54   314R x 2C   [D, D]
  3  RUN_ID        1 BinTableHDU     97   152R x 3C   [J, D, D]
START 216087649.760218 and 216088979.847248
STOP 217022036.725794 and 217021615.769332
WARNING: AstropyDeprecationWarning: The following keywords are now recognized as spe
cial column-related attributes and should be set via the Column objects: TCDLTn, TCR
PXn, TCRVLn, TCTYPn, TCUNIn. In future, these values will be dropped from manually s
pecified headers automatically and replaced with values generated based on the Colum
n objects. [astropy.io.fits.hdu.table]

real    0m48,666s
user    0m42,499s
sys     0m5,009s
```

Syntax: filter_background.py level2_file_path level1_file_path. One file (e.g.: DU) at a time

Takes a while to run, but should get you through
In seconds, minutes at most (i was running on
An external drive on my laptop)

```
ae8b86/observations/02009701$ ls event_l2/
ixpe02009701_det1_evt2_v01.fi_rej.fits
ixpe02009701_det1_evt2_v01.fits.gz
ixpe02009701_det2_evt2_v01.fi_rej.fits
ixpe02009701_det2_evt2_v01.fits.gz
ixpe02009701_det3_evt2_v01.fi_rej.fits
ixpe02009701_det3_evt2_v01.fits.gz
```

Produces _rej-tagged fits files containing a list of
X-ray only events.

# 1. Cosmic ray rejection

If you are interested on the events that have been rejected (for debugging reasons), you need to call the app with another option

```
ae8b86/observations/02009701$ time python filter_background.py eve
nt_l2/ixpe02009701_det1_evt2_v01.fits.gz event_l1/ixpe02009701_det
1_evt1_v01.fits.gz --output bkg
```

It's gonna take its time..

```
real	0m56,073s
user	0m44,678s
sys	0m6,559s
```

And you can run it for all 3 Dus. Eventually, your level 2 file folder will look like this

```
ae8b86/observations/02009701$ ls event_l2/
ixpe02009701_det1_evt2_v01.fi_bkg.fits
ixpe02009701_det1_evt2_v01.fi_rej.fits
ixpe02009701_det1_evt2_v01.fits.gz
ixpe02009701_det2_evt2_v01.fi_bkg.fits
ixpe02009701_det2_evt2_v01.fi_rej.fits
ixpe02009701_det2_evt2_v01.fits.gz
ixpe02009701_det3_evt2_v01.fi_bkg.fits
ixpe02009701_det3_evt2_v01.fi_rej.fits
ixpe02009701_det3_evt2_v01.fits.gz
```

Let's make the most basic check to see if everything went right: let's make a cmap with xpbin

```
6b511ae8b86/observations/02009701$ time xpbin.py --alg 'CMAP'
 event_l2/ixpe02009701_det?_evt2_v01.fi_bkg.fits

real	0m3,020s
user	0m2,858s
sys	0m0,519s
```

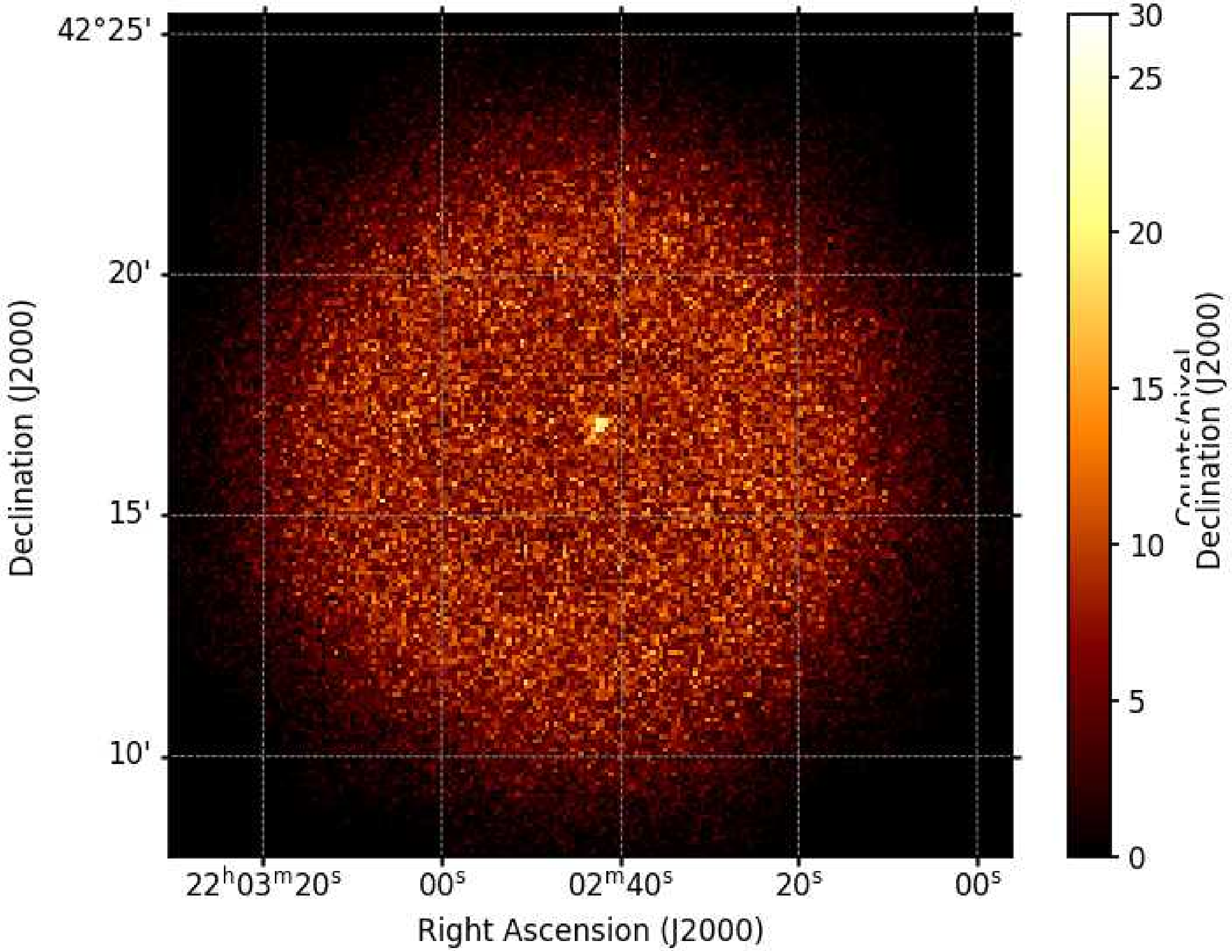
And now let's visualize it with xpbinview. To increase the signal i'm going to stack all three DUs

```
6b511ae8b86/observations/02009701$ xpbinview.py event_l2/ixpe
02009701_det?_evt2_v01.fi_bkg_cmap.fits
```

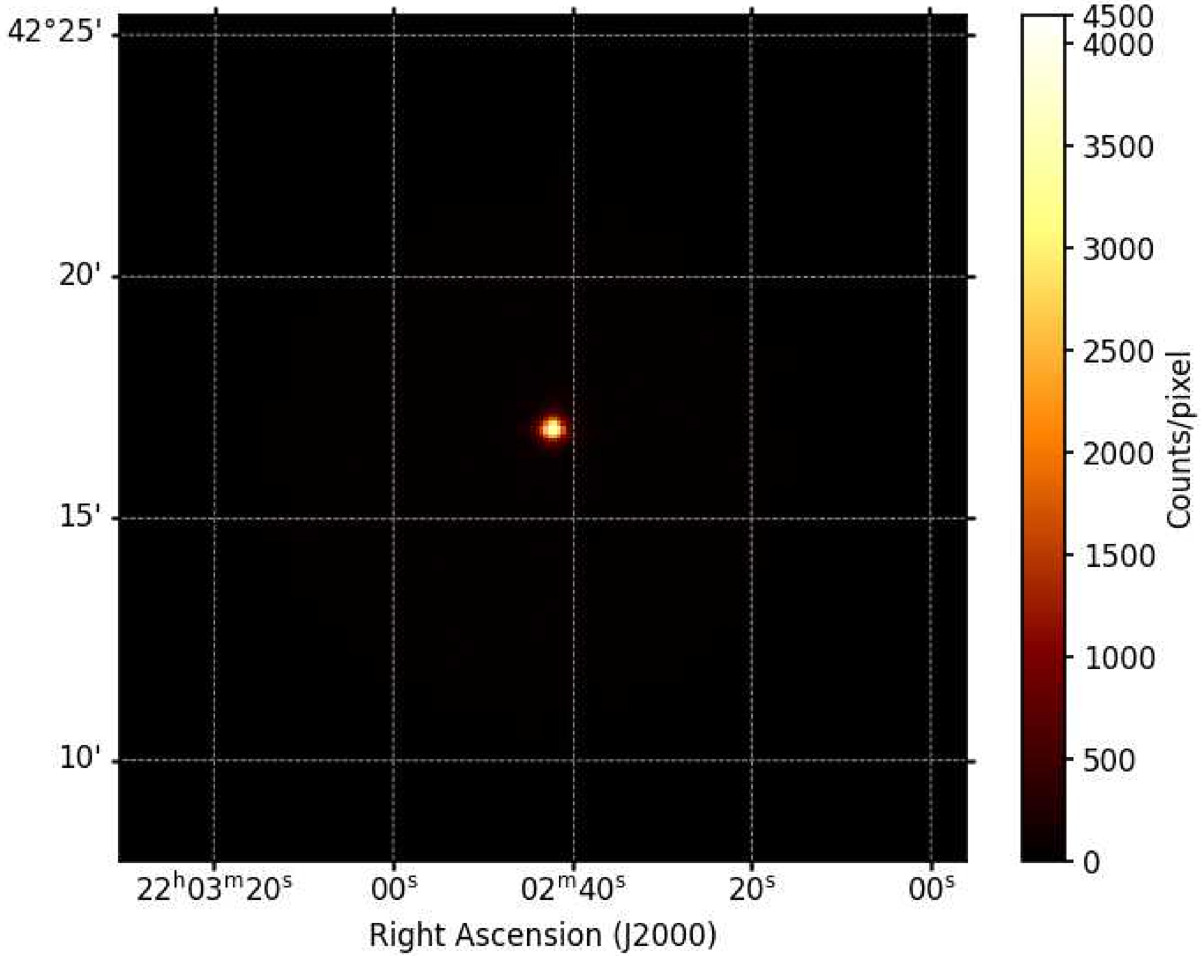
The resulting count map will pop up almost immediately. We are looking for a residual trace of the source to see wether some good events have leaked into the rejected

# 1. Cosmic ray rejection

CMAP of the rejected bkg




CMAP of the “clean” events



# 1. Cosmic ray rejection

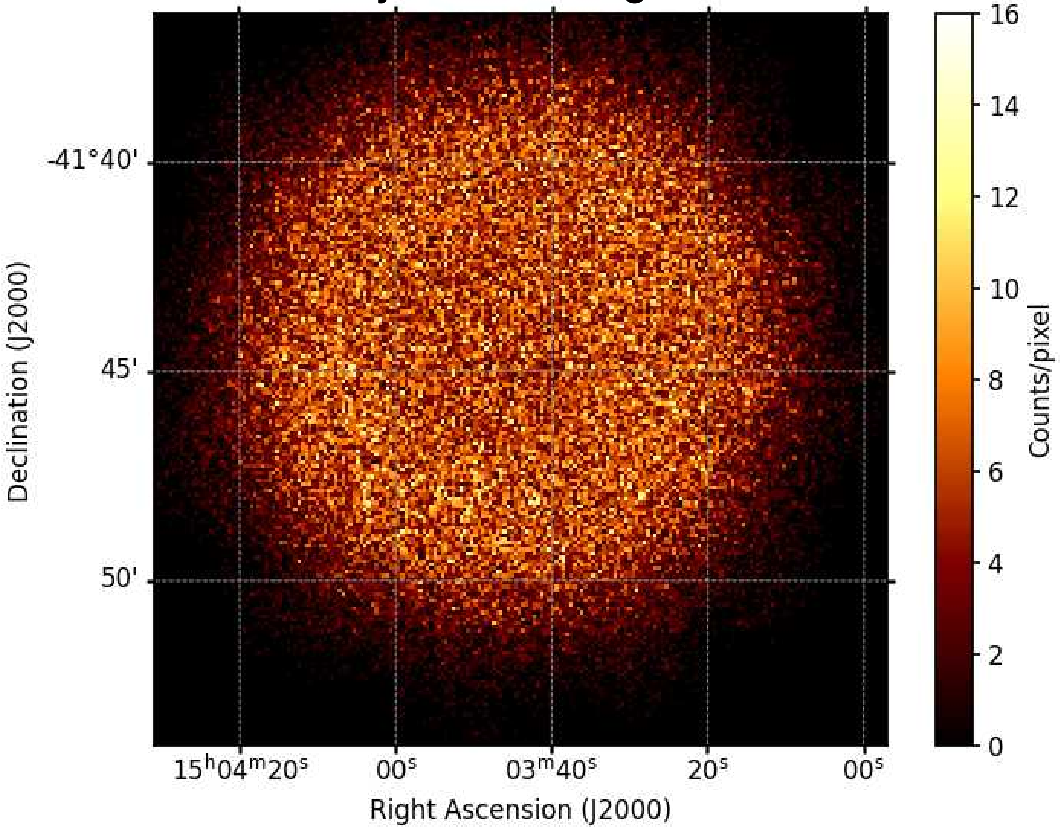
Online content:
- Di Marco's app (github)

Success! People who are interested only in point/bright sources can take a nap now...

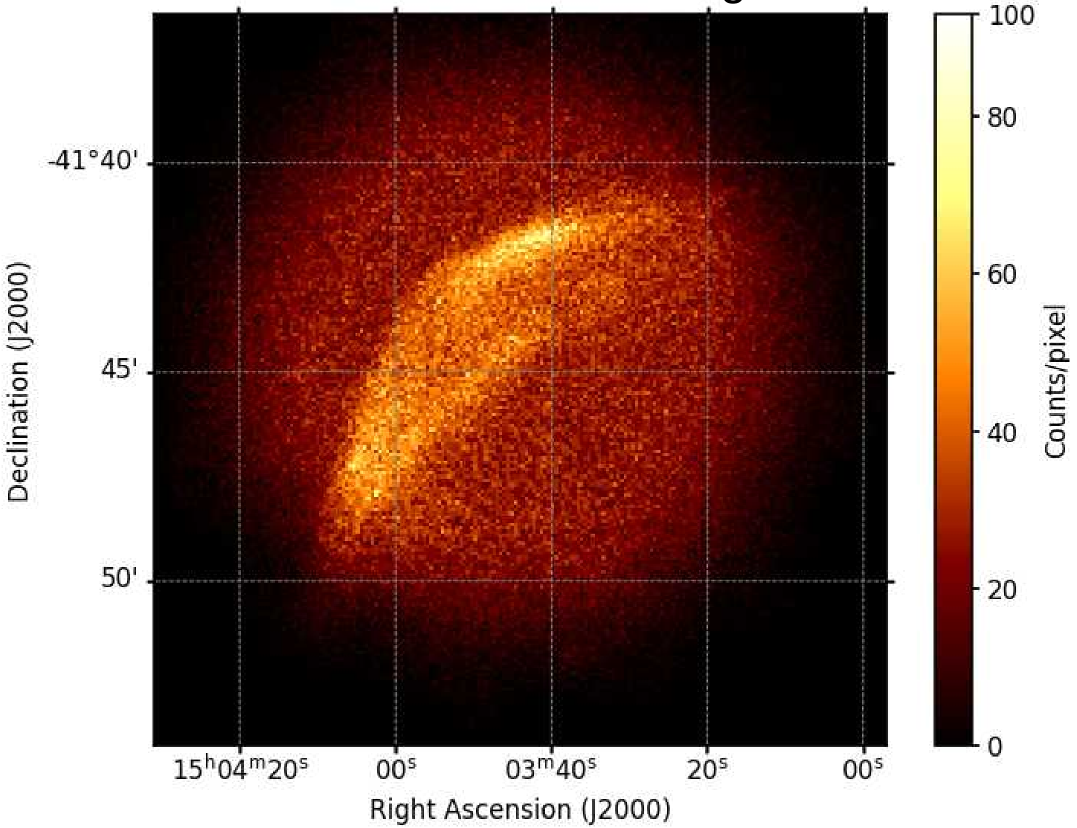
# 2. Simultaneous background extraction

Let's say that after the background rejection you still want to get rid of the residual background (but not in obsid 02009701 of course). You might be in a situation that resembles this one (obsid 02001701)

Rejected background

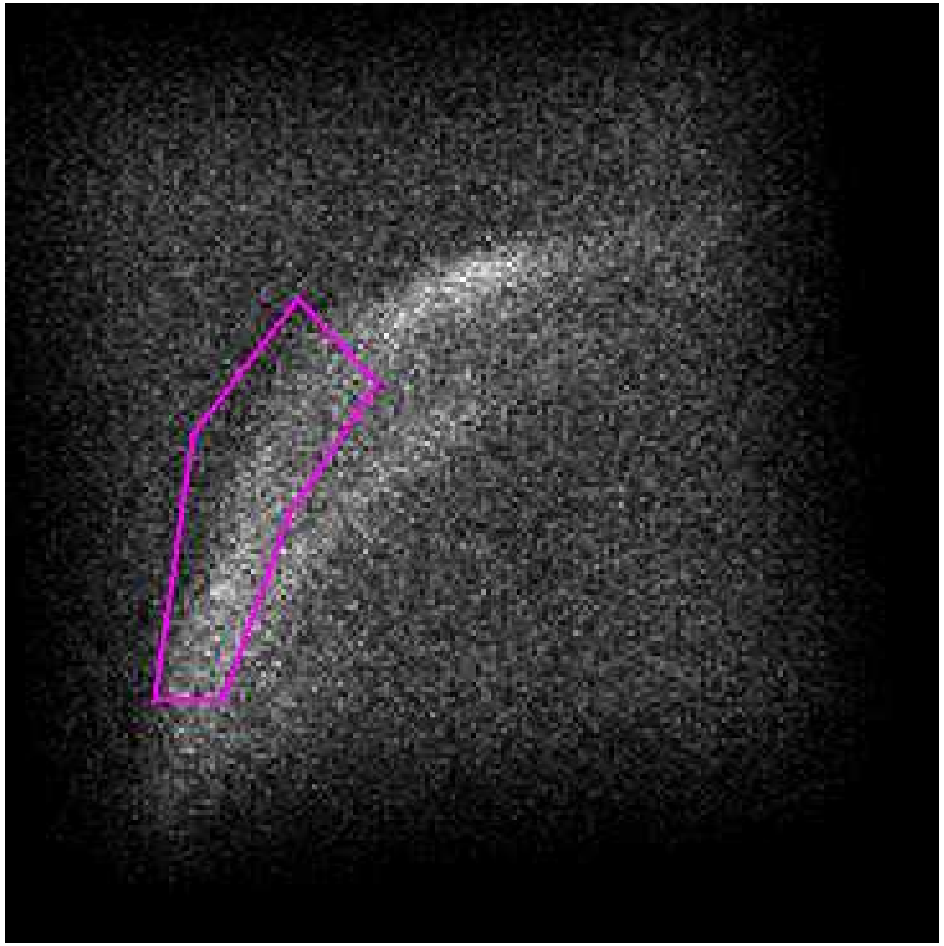



Source + residual background



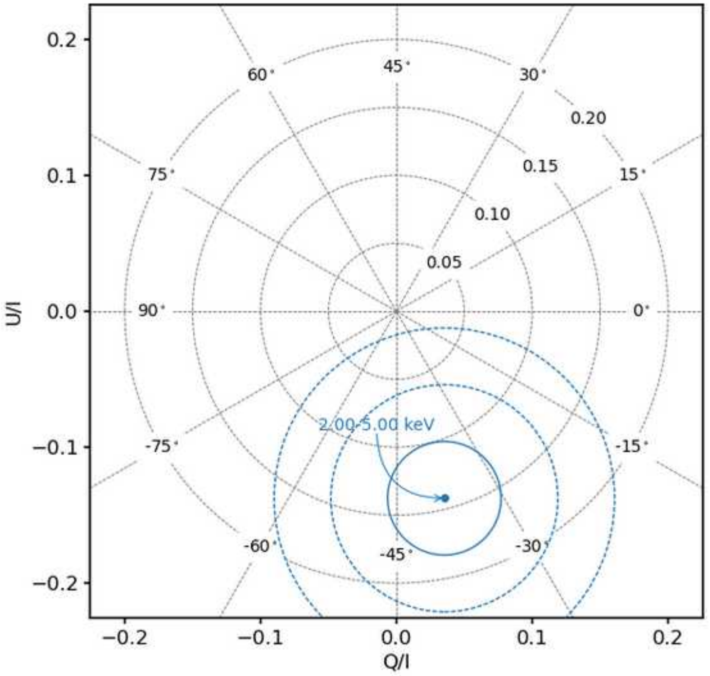
# 2. Simultaneous background extraction

You can then make a pcube of a region selected through ds9



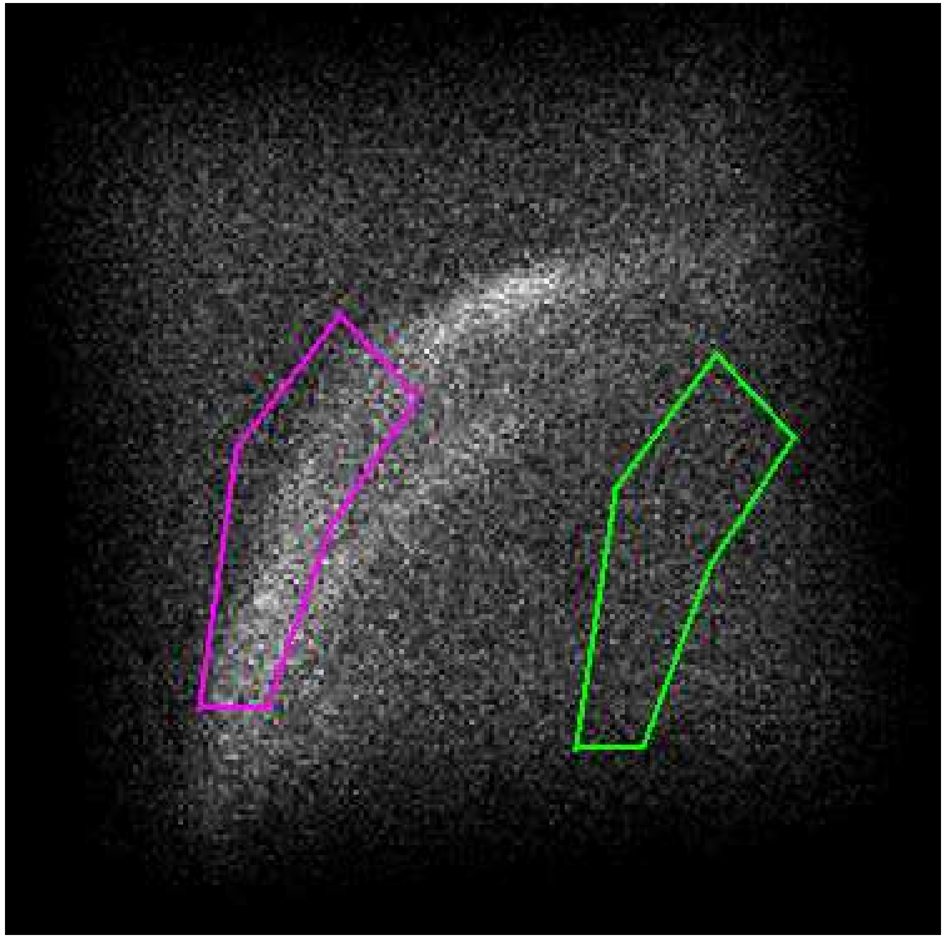
```
511ae8b86/observations/02001701/event_l2$ xpselect.py --regfile
 ../signal.reg ixpe02001701_det?_evt2_v02.fi_rej.fits
511ae8b86/observations/02001701/event_l2$ xpbin.py --alg 'PCUBE
' --emin 2. --emax 5. --ebins 1 ixpe02001701_det?_evt2_v02.fi_r
ej_select.fits --irfname ixpe:obssim:v12
```

| Quantity | 2.00--5.00 keV |
|---|---|
| E_MEAN | 3.1044778223507805 |
| COUNTS | 16542.0 |
| MU | 0.2872270385650793 |
| W2 | 52.119476318359375 |
| N_EFF | 13890.6123046875 |
| PD | 0.14200690388679504 |
| PD_ERR | 0.04178338628532177 |
| PA | -37.80359441440317 |
| PA_ERR | 8.432717153396775 |
| P_VALUE | 0.003083678212236593 |
| CONFID | 0.9969163217877635 |
| SIGNIF | 2.73874832939163 |



# 2. Simultaneous background extraction

Comparison with background only pcube



Crude estimate: about 30% is background despite rejection
this is gonna alter our polarization degree

→ Subtraction required

| Quantity | 2.00--5.00 keV |
|---|---|
| E_MEAN | 3.1044778223507805 |
| COUNTS | 16542.0 |
| MU | 0.2872270385650793 |
| W2 | 52.119476318359375 |
| N_EFF | 13890.6123046875 |
| PD | 0.14200690388679504 |
| PD_ERR | 0.04178338628532177 |
| PA | -37.80359441440317 |
| PA_ERR | 8.432717153396775 |
| P_VALUE | 0.003083678212236593 |
| CONFID | 0.9969163217877635 |
| SIGNIF | 2.73874832939163 |

| Quantity | 2.00--5.00 keV |
|---|---|
| E_MEAN | 3.2868158093823276 |
| COUNTS | 6282.0 |
| MU | 0.3028996296006631 |
| W2 | 23.86249542236328 |
| N_EFF | 5138.28857421875 |
| PD | 0.1197405532002449 |
| PD_ERR | 0.0652056359335663 |
| PA | 34.128013489925436 |
| PA_ERR | 15.605561337490874 |
| P_VALUE | 0.18439759531398617 |
| CONFID | 0.8156024046860139 |
| SIGNIF | 0.8987324453288486 |

# 2. Simultaneous background extraction

For subtraction you need to manipulate PCUBES, so we have to get to python + ixpeobssim

```
def make_pcubes(emin, emax, region_path=None, overwrite=False):
    ''' This funcion optionally cuts the data based on a DS9 region and
    created the pubes in the desired energy range.
    Is region is provided, the selected file is tagged as the region
    file name.
    '''
    if region_path is not None:
        #string manipulation to get automatic suffix
        suffix = region_path.split('/')[-1].split('.')[0]
        #xpselect equivalent using pipeline
        selected = pipeline.xpselect(*lvl2_file_list(), regfile=region_path,
                                     suffix=suffix, overwrite=overwrite)
        #xpbin equivalent using pipeline. Exploits the pipeline architecture
        #which returns the list of freshly created files paths
        pcubes_path = pipeline.xpbin(*selected, alg = 'PCUBE',
                            emin=emin, emax=emax, irfname='ixpe:obssim:v12',
                            suffix=suffix, overwrite=overwrite,
                            ebins=1)
    else:
        #Skips the xpselect step
        pcubes_path = pipeline.xpbin(*lvl2_file_list(), alg = 'PCUBE',
                            emin=emin, emax=emax, irfname='ixpe:obssim:v12',
                            ebins= 1, overwrite=overwrite)
    #Stacks all three DUs.
    return xBinnedPolarizationCube.from_file_list(pcubes_path)
```

**Pipeline**.something: like launching something.py from a terminal with few small differences:

- Returns the output file list, making it trivial to chain commands
- The arguments are comma separated instead of input with --

**xBinnedSomething**.from_file_list(file_list): creates an instance of the binned products with useful built-in methods

- plot (equivalent to xbinview.py)
- as_table() also included in xpbinview, prints out all the relevant data in a nicelt formatted way

# 2. Simultaneous background extraction

For subtraction you need to manipulate PCUBES, so we have to get to python + ixpeobssim

```python
def make_pcubes(emin, emax, region_path=None, overwrite=False):
    ''' This funcion optionally cuts the data based on a DS9 region and
    created the pubes in the desired energy range.
    Is region is provided, the selected file is tagged as the region
    file name.
    '''
    if region_path is not None:
        #string manipulation to get automatic suffix
        suffix = region_path.split('/')[-1].split('.')[0]
        #xpselect equivalent using pipeline
        selected = pipeline.xpselect(*lvl2_file_list(), regfile=region_path,
                                     suffix=suffix, overwrite=overwrite)
        #xpbin equivalent using pipeline. Exploits the pipeline architecture
        #which returns the list of freshly created files paths
        pcubes_path = pipeline.xpbin(*selected, alg = 'PCUBE',
                            emin=emin, emax=emax, irfname='ixpe:obssim:v12',
                            suffix=suffix, overwrite=overwrite,
                            ebins=1)
    else:
        #Skips the xpselect step
        pcubes_path = pipeline.xpbin(*lvl2_file_list(), alg = 'PCUBE',
                            emin=emin, emax=emax, irfname='ixpe:obssim:v12',
                            ebins= 1, overwrite=overwrite)
    #Stacks all three DUs.
    return xBinnedPolarizationCube.from_file_list(pcubes_path)
```

```python
def hands_on_example():
    ''' Runs the list of commands shown during the hands-on session
    '''
    signal_pcubes = make_pcubes(2., 5., region_path=signal_region_path)
    plt.title('Signal region before subtraction')
    signal_pcubes.plot()
    input(signal_pcubes.as_table())
    background_pcubes = make_pcubes(2., 5., region_path=bkg_region_path)
    plt.title('Background region')
    background_pcubes.plot()
    input(background_pcubes.as_table())
    signal_pcubes-=background_pcubes
    plt.title('Background-subtracted pcube')
    signal_pcubes.plot()
    print(signal_pcubes.as_table())
```

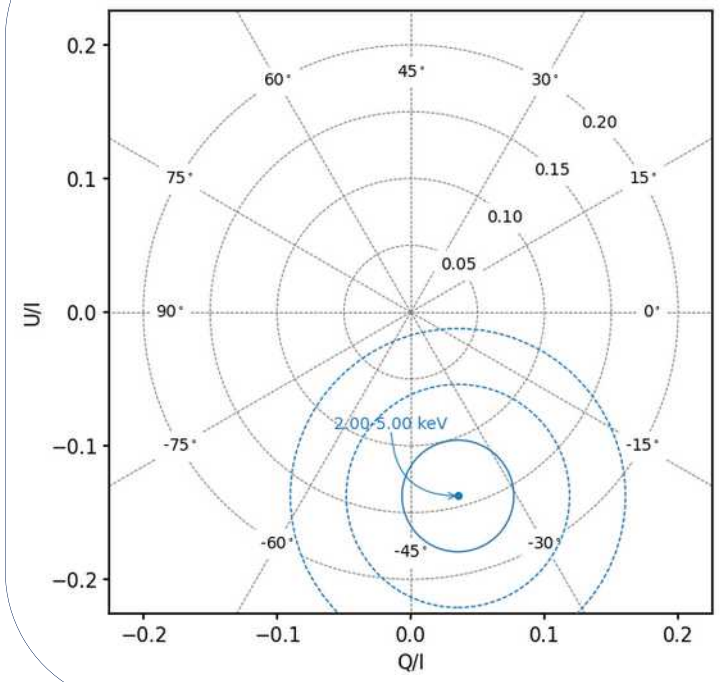
This is gonna make the pcubes, plot them and output the parameters waiting for your input before proceeding further

**Subtraction** is performed using the -=, the pcube will be updates in place

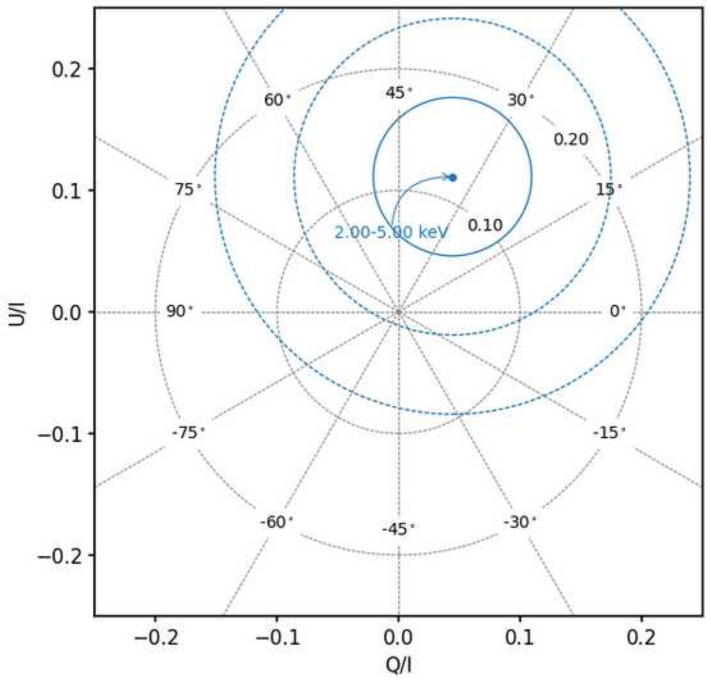
# 2. Simultaneous background extraction

For subtraction you need to manipulate PCUBES, so we have to get to python + ixpeobssim

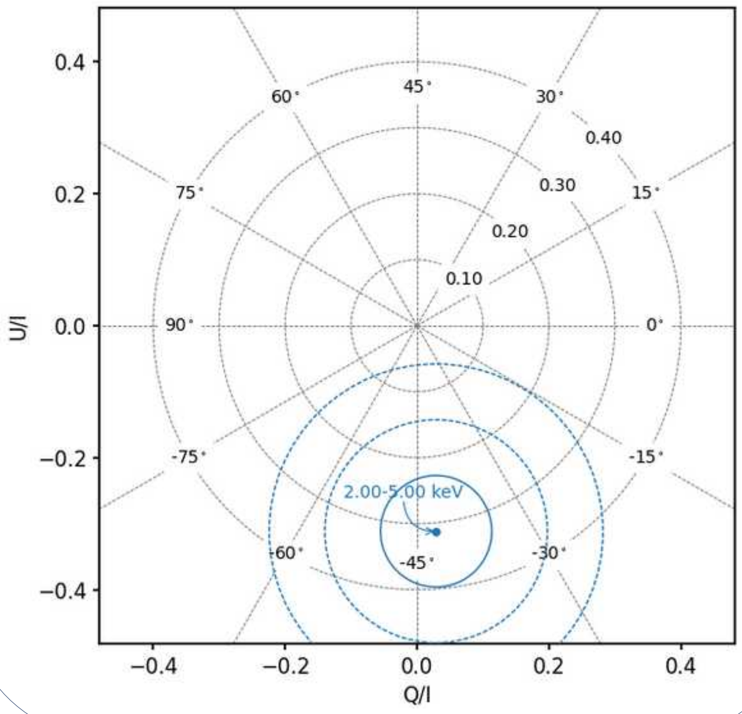
Signal region




Background region




Background-subtracted pcube



# 3. Background extraction (Lazy)

Extracting the background from the same observation is not always possible
- Extended source covering the whole FoV
- Bright source leaking into the background region by means of the PSF
- Regions you don't trust (contamination from astrophysical background, edge effects, dithering etc)

Instead of extracting the background you can simulate one from a template based on our frst test observation of SMC-X1, the simulation is run by inputing a config file as usual and by defining a duration

**Config file location**: ixpeobssim/ixpeobssim/config/instrumental_bkg_smcx1.py

**Command to run:** xpobssim.py –configfile instrumental_bkg_smcx1.py –duration 10000000

This will simulate an unpolarized background observation of 1Ms which can be used for pcube subtraction.
The config file imports the class xTemplateInstrumentalBkg, which can be used as a source model in a Simulation (e.g.: for a proposal)
**Import command:** from ixpeobssim.srcmodel.bkg import xTemplateInstrumentalBkg

We are developing a better way to extract backgrounds (e.g.: from larger statistics), but unfortuntely it's not yet super practical

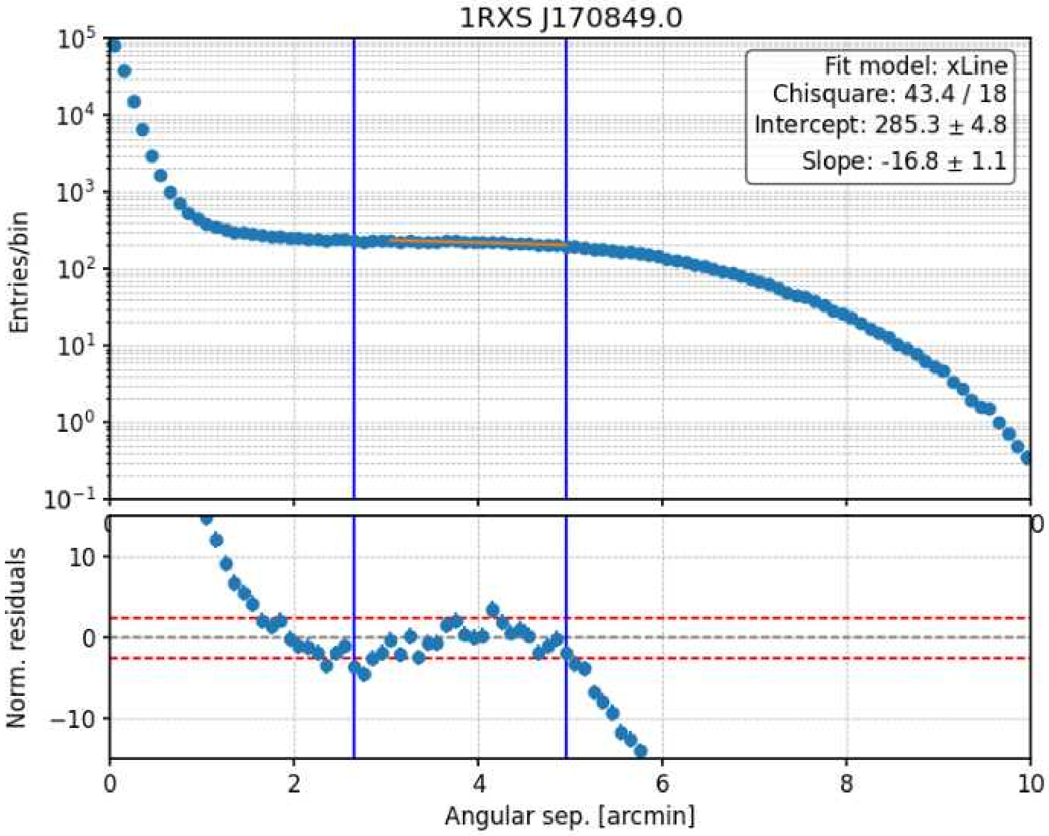
# 3. Background extraction (Other observations)

The static background component can be estimated from an arbitrary number of other observaion, increasing the statistics and the robustness of the estimate, with some caveats

- The estimate is best performed on **faint sources** (beware of the **PSF!**)
- You must be careful on getting **only static component** (so first of all check the source spectrum or light curve to see if you detect the bump or flares)
- The stacking of observations is possible even for creating a template that can be used for simulations like the SMC-X1

# 3. Background extraction (Other observations)

Remember: ixpe is dithered, so you cannot explot the full field of view for extraction. This is also documented In Di Marco et al. (2023). The best place to look for a background sample is when the radial profile plateaus




Then you need to use xpselect to select a ring around the source

```
511ae8b86/observations/02001701/event_l2$ xpselect.py --innerra
d 2.5 --rad 5 ixpe????????_det?_evt2_v0?.fi_rej.fits
```

The output file will be a ring containing pure background. The backscal keyword will be updated to reflect the real area of the ring and can be used to rescale the counts

You now have 2 options:

- Use this background “as is” and do a pcube subtraction as in the examples shown earlier
- Do this for several well-behaved observations and stack the backgrounds to create e template

# 3. Background extraction (Other observations)

The process of creating a background template is straightforward (but you need to switch to the right branch)

1) Switch to **instrumental_background branch**

```
~/IXPE/ixpeobssim$ git checkout instrumental_background
Si è passati al branch 'instrumental_background'
Il tuo branch è aggiornato rispetto a 'origin/instrumental_background'.
~/IXPE/ixpeobssim$ git pull
```

2) **Create spectra** from the background fields. You can stack as many as you want – the more the better

```
xpbin.py --alg 'PHA1' --irfname 'ixpe:obssim:v12' ixpe????????_det?_evt2_v0?.fi_rej_select.fits
```

3) Now you can create the template by simply inputting the list of pha1 files to **xpbkgtemplate**

```
xpbkgtemplate.py ixpe????????_det?_evt2_v0?.fi_rej_select_pha1.fits
```

The background template is placed ixpeobssim/srcmodel/ascii/instrumental_bkg_template.txt

4) We can now use it to **simulate a large-statistics background-only** file through xpobssim, essentially copying the structure of the config file already shipped in ixpeobssim (config/instrumental_bkg_mean.py)

# 3. Background extraction (Other observations)

The only thing that you will need to change in **instrumental_bkg_mean.py** is the file_path line

```
__model__ = file_path_to_model_name(__file__)

RA = 45.
DEC = 45.
file_path = os.path.join(IXPEOBSSIM_SRCMODEL, 'ascii', 'instrumental_bkg_mean.txt')

bkg = xTemplateInstrumentalBkg(file_path = file_path)

ROI_MODEL = xROIModel(RA, DEC, bkg)
```

Then you can create the background sample with **ixpeobssim** using this as a config file. For a Ms you need

```
xpobssim.py --configfile ixpeobssim/config/instrumental_bkg_mean.py --duration 1000000
```

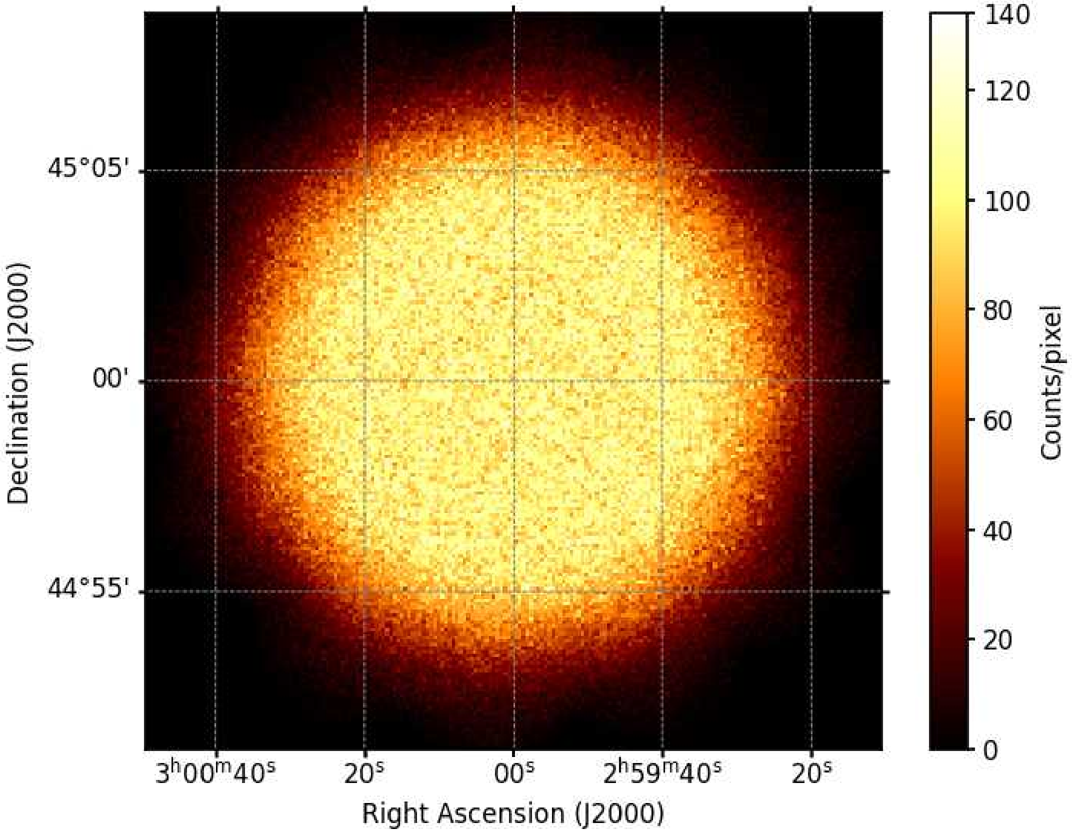
```
real    0m30,836s
user    0m35,903s
sys     0m9,787s
```

```
~/IXPE/ixpeobssim$ ls -lh ~/ixpeobssimdata/instrumental_bkg_mean_du?.fits
69M 28 mar 19.39 /home/kibuzo/ixpeobssimdata/instrumental_bkg_mean_du1.fits
69M 28 mar 19.40 /home/kibuzo/ixpeobssimdata/instrumental_bkg_mean_du2.fits
69M 28 mar 19.40 /home/kibuzo/ixpeobssimdata/instrumental_bkg_mean_du3.fits
```

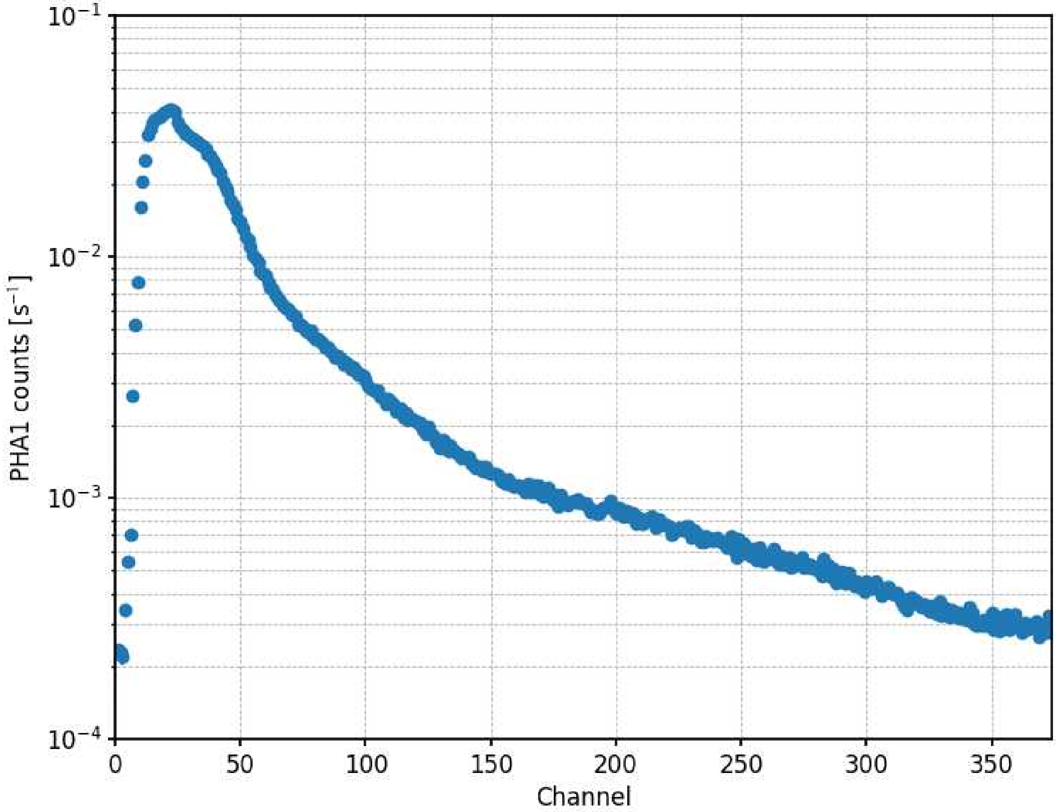
# 3. Background extraction (Other observations)

Let’s check this background using the tools that we learned to use

CMAP




PHA1



# 3. Background extraction (Other observations)

Online content:
- Di Marco's app (github)
- pcube subtraction example
- Data folder

Success! You can now do two things

- Create and subtract PCUBES
- Add xTemplateInstrumentalBkg with the right path as an ixpeobssim simulation component when making a proposal (this is more accurate then PIMMS and more sensitive to additional components)

You can get creative with the bump and background extraction but keep in mind that xTemplateInstrumentalBkg inherits from its simplified version the lack of support for background polarization,

# 4. Deflaring

The template is not **polarized** but I said that the **1.5 keV bump** *might*

- **The safest way is to just get rid of it, and make it so under-threshold that its polarization is diluted to insignificant.**
- If you want to characterize it you can avoid subtracting it but it’s some work that you are gonna do on your own
- *Hints: Model the static component and the bump from a large sample of observations and fit them in xspec, then add the model of the bkg to your overall source fit.*

# 4. Deflaring

IXPE Imaging X-Ray Polarimetry Explorer

Create a rate histogram and **recover the time intervals in which you can detect the flares** (sudden in increase)

Caveats: The rate is events/livetime, need to create two different histograms with the same binning

```python
def build_rate_hist(bin_size=60, lvl2_file_path = lvl2_file_path,
                    lvl1_file_list=lvl1_file_list()):
    ''' Creates a rate (e.g. events/livetime) histogram in time bins
    '''
    evt_friend = xEventFileFriend(lvl2_file_path, lvl1_file_list)
    l2_time = evt_friend.l1value('TIME')
    t_bins = make_tgrid(bin_size, evt_friend.start_met(), evt_friend.stop_met())
    livetime_hist = build_livetime_hist(evt_friend, t_bins)
    rate_hist = xHistogram1d(t_bins, xlabel='MET [s]', ylabel='Rate [Hz]')
    rate_hist.fill(l2_time)
    rate = rate_hist.content / livetime_hist.content
    rate_err = numpy.sqrt(rate_hist.content) / livetime_hist.content
    rate_hist.set_content(rate, rate_hist.entries, rate_err)
    return rate_hist
```

```python
def make_tgrid(bin_size, t0, t1):
    '''
    '''
    nbins = numpy.int((t1-t0)//bin_size)
    return numpy.linspace(t0, t1, nbins)


def build_livetime_hist(evt_friend, t_bins):
    '''
    '''
    l1_time = evt_friend.l1value('TIME', all_events=True)
    livetime = evt_friend.l1value('LIVETIME', all_events=True)
    lvt_hist = xHistogram1d(t_bins, xlabel='MET [s]', ylabel='Livetime [s]')
    return lvt_hist.fill(l1_time, weights = livetime / 1.e6)
```

Upon detecting sharp deviated rate histogram bins you can **define time intervals to reject**

Now, you will need to **update the LIVETIME** as well

# 4. Deflaring

```python
def deflare_observation(quantile=0.975, overwrite=True):
    '''
    '''
    evt_friend = xEventFileFriend(lvl2_file_path, lvl1_file_list())
    evt = xEventFile(lvl2_file_path)
    time = evt_friend.l2value('TIME')
    l1_time = evt_friend.l1value('TIME', all_events=True)
    bti = get_bti(quantile=quantile)
    filter = []
    filter_l1 = []
    for j in range (len(bti[0])):
        filter.append(numpy.logical_or((time<bti[0][j]), (time>bti[1][j])))
        filter_l1.append(numpy.logical_or((l1_time<bti[0][j]),
                                          (l1_time>bti[1][j])))
    time_mask = numpy.logical_and.reduce(filter)
    l1_mask = numpy.logical_and.reduce(filter_l1)
    #Filter the l2 events
    evt.filter(time_mask)
    #Update the LIVETIME using l1
    lt_array = evt_friend.l1value('LIVETIME', all_events=True)
    tot_lt = lt_array[l1_mask].sum() / 1.e6
    input (tot_lt)
    evt.primary_header['LIVETIME'] = tot_lt
    #Write the new fits file with the gticorr suffix
    new_name = os.path.splitext(lvl2_file_path)[0] + '_gticorr.fits'
    evt.write(new_name, overwrite=True)
```

Finally, **update the LIVETIME**. This is the trickiest part

- First, we get the BTIs* from the quantile clipping
- Now for each BTI we create a boolean mask of what was before and after

*Digression: they are bad because contaminated by a huge background, but have nothing to do with the negation of GTIs as intended in IXPE

# 4. Deflaring

```python
def deflare_observation(quantile=0.975, overwrite=True):
    '''
    '''
    evt_friend = xEventFileFriend(lvl2_file_path, lvl1_file_list())
    evt = xEventFile(lvl2_file_path)
    time = evt_friend.l2value('TIME')
    l1_time = evt_friend.l1value('TIME', all_events=True)
    bti = get_bti(quantile=quantile)
    filter = []
    filter_l1 = []
    for j in range (len(bti[0])):
        filter.append(numpy.logical_or((time<bti[0][j]), (time>bti[1][j])))
        filter_l1.append(numpy.logical_or((l1_time<bti[0][j]),
                                          (l1_time>bti[1][j])))
    time_mask = numpy.logical_and.reduce(filter)
    l1_mask = numpy.logical_and.reduce(filter_l1)
    #Filter the l2 events
    evt.filter(time_mask)
    #Update the LIVETIME using l1
    lt_array = evt_friend.l1value('LIVETIME', all_events=True)
    tot_lt = lt_array[l1_mask].sum() / 1.e6
    input (tot_lt)
    evt.primary_header['LIVETIME'] = tot_lt
    #Write the new fits file with the gticorr suffix
    new_name = os.path.splitext(lvl2_file_path)[0] + '_gticorr.fits'
    evt.write(new_name, overwrite=True)
```

Finally, update the LIVETIME. This is the trickiest part

- First, we get the BTIs from the quantile clipping
- Now for each BTI we create a boolean mask of what was before and after
- We make the AND of all those

*Digression: they are bad because contaminated by a huge background, but have nothing to do with the negation of GTIs as intended in IXPE

# 4. Deflaring

```
def deflare_observation(quantile=0.975, overwrite=True):
    '''
    '''
    evt_friend = xEventFileFriend(lvl2_file_path, lvl1_file_list())
    evt = xEventFile(lvl2_file_path)
    time = evt_friend.l2value('TIME')
    l1_time = evt_friend.l1value('TIME', all_events=True)
    bti = get_bti(quantile=quantile)
    filter = []
    filter_l1 = []
    for j in range (len(bti[0])):
        filter.append(numpy.logical_or((time<bti[0][j]), (time>bti[1][j])))
        filter_l1.append(numpy.logical_or((l1_time<bti[0][j]),
                                          (l1_time>bti[1][j])))
    time_mask = numpy.logical_and.reduce(filter)
    l1_mask = numpy.logical_and.reduce(filter_l1)
    #Filter the l2 events
    evt.filter(time_mask)
    #Update the LIVETIME using l1
    lt_array = evt_friend.l1value('LIVETIME', all_events=True)
    tot_lt = lt_array[l1_mask].sum() / 1.e6
    input (tot_lt)
    evt.primary_header['LIVETIME'] = tot_lt
    #Write the new fits file with the gticorr suffix
    new_name = os.path.splitext(lvl2_file_path)[0] + '_gticorr.fits'
    evt.write(new_name, overwrite=True)
```

Finally, update the LIVETIME. This is the trickiest part

- First, we get the BTIs from the quantile clipping
- Now for each BTI we create a boolean mask of what was before and after
- We make the AND of all of those
- We filter the event file with the final boolean mask

*Digression: they are bad because contaminated by a huge background, but have nothing to do with the negation of GTIs as intended in IXPE

# 4. Deflaring

```python
def deflare_observation(quantile=0.975, overwrite=True):
    '''
    '''
    evt_friend = xEventFileFriend(lvl2_file_path, lvl1_file_list())
    evt = xEventFile(lvl2_file_path)
    time = evt_friend.l2value('TIME')
    l1_time = evt_friend.l1value('TIME', all_events=True)
    bti = get_bti(quantile=quantile)
    filter = []
    filter_l1 = []
    for j in range (len(bti[0])):
        filter.append(numpy.logical_or((time<bti[0][j]), (time>bti[1][j])))
        filter_l1.append(numpy.logical_or((l1_time<bti[0][j]),
                                          (l1_time>bti[1][j])))
    time_mask = numpy.logical_and.reduce(filter)
    l1_mask = numpy.logical_and.reduce(filter_l1)
    #Filter the l2 events
    evt.filter(time_mask)
    #Update the LIVETIME using l1
    lt_array = evt_friend.l1value('LIVETIME', all_events=True)
    tot_lt = lt_array[l1_mask].sum() / 1.e6
    input (tot_lt)
    evt.primary_header['LIVETIME'] = tot_lt
    #Write the new fits file with the gticorr suffix
    new_name = os.path.splitext(lvl2_file_path)[0] + '_gticorr.fits'
    evt.write(new_name, overwrite=True)
```

Finally, update the LIVETIME. This is the trickiest part

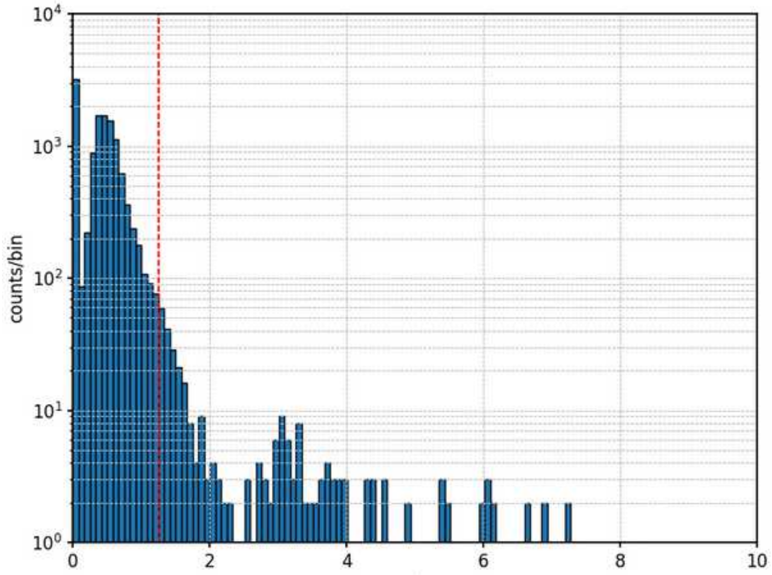
- First, we get the BTIs from the quantile clipping
- Now for each BTI we create a boolean mask of what was before and after
- We make the AND of all of those
- We filter the event file with the final boolean mask
- We **update the livetime using all_events=True**, this is because summing the livetime on the events that appear only on the level2 file wouls underestimate it due to exclusion of events coming from other cuts (e.g.: fiducial area trim)

*Digression: they are bad because contaminated by a huge background, but have nothing to do with the negation of GTIs as intended in IXPE

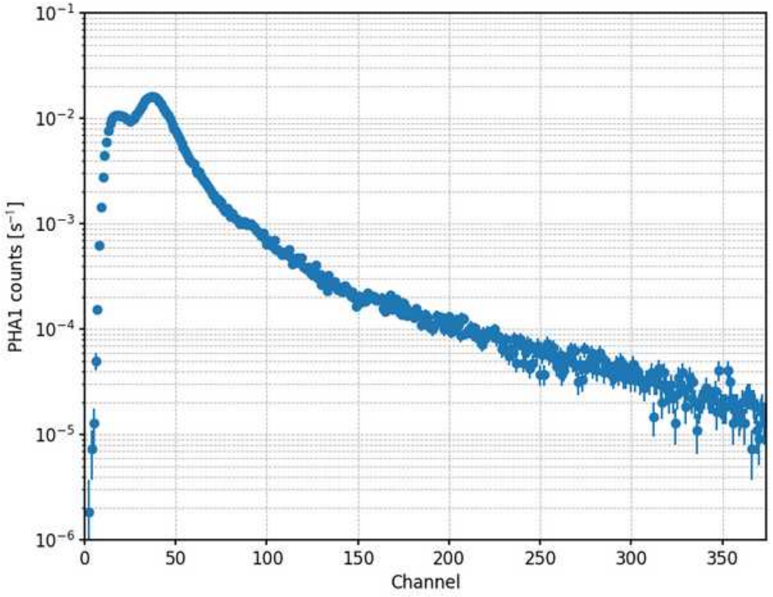
# 4. Deflaring

Example: deflaring du2 for obsid 02001599

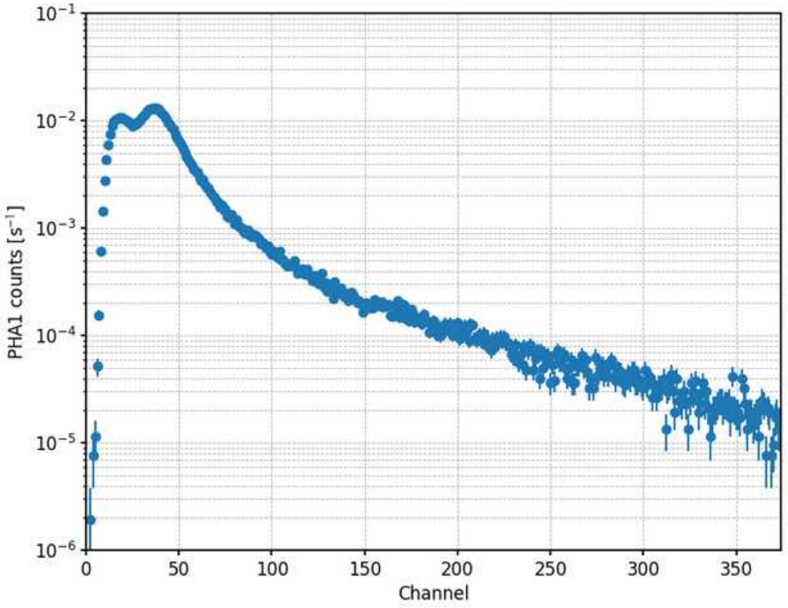
Rate distribution (histogram of build_ratehist().content) And 0.975 quantile for clipping (log scale)




Spectrum before clipping



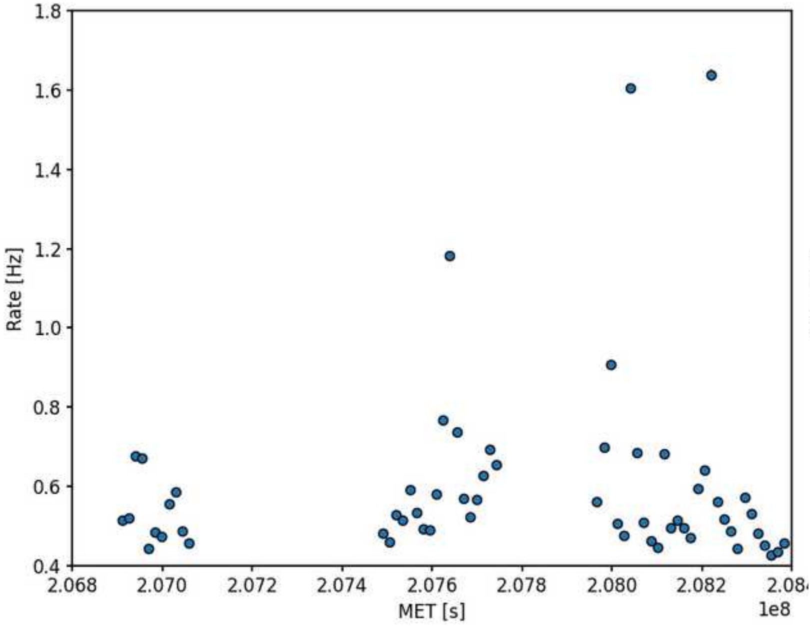

Spectrum after clipping



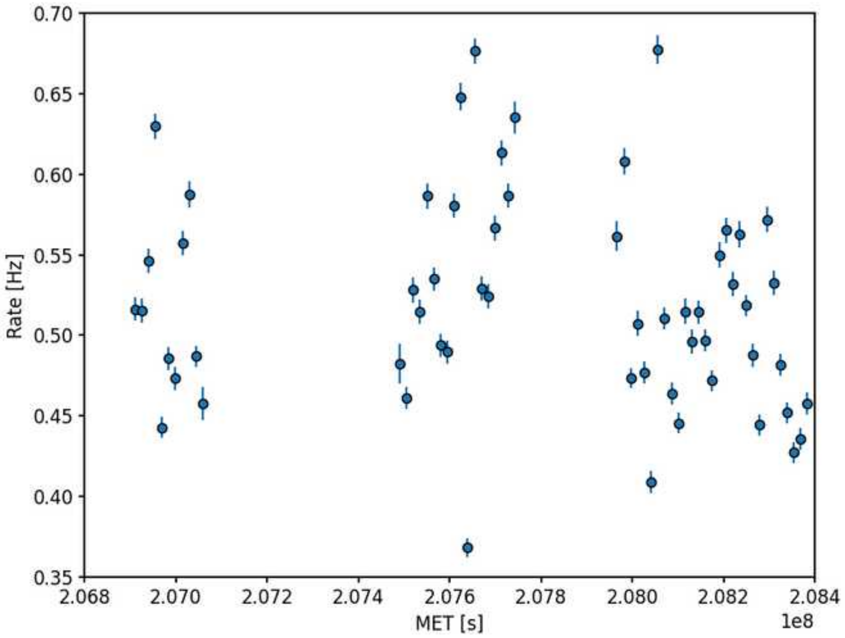
# 4. Deflaring

Example: deflaring du2 for obsid 02001599

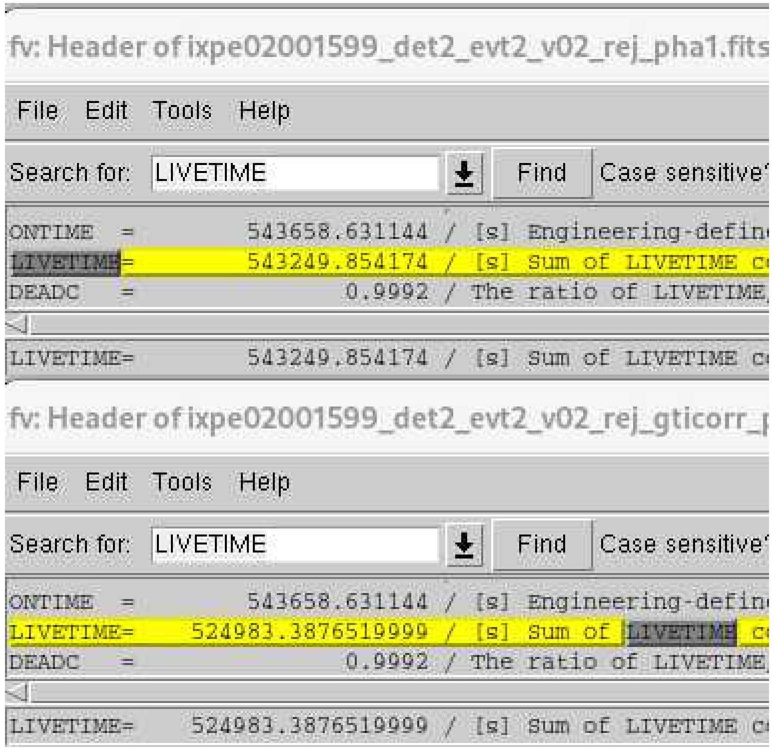
Light curve before clipping




Light curve after clipping




Livetime: before and after



# 4. Deflaring

Online content:
- Di Marco's app (github)
- pcube subtraction example
- deflaring example
- Data folder

Questions? Observations?
- mailto:stefano.silvestri@pi.infn.it

You survived! Recommended exercise:

Remove flares from du2 or 3 of some extended source (e.g.: 02006799) after rejecting the background.

- You can play around with the provided scripts to see what happens to the spectra or to the light curve with different rejection threshold
- Make histograms to check the distribution of the rate bins, check what changes with different bin size.
- What happens to polarization with diferent quantile cliippings?
- What is your best background template for subtraction after rejection and deflaring?