EXOD

EPIC-pn XMM-Newton Outburst Detector

Institute	Université de Toulouse; UPS-OMP; IRAP, Toulouse, France; CNRS Anton Pannekoek Institute for Astronomy, University of Amsterdam ASTRON, Netherlands Institute for Radio Astronomy
Author	Inés Pastor Marazuela (2019) – ines.pastor.marazuela@gmail.com
Contact	Natalie Webb – natalie.webb@irap.omp.eu
Version	1.0, January 13, 2020
Source code	https://github.com/InesPM/EXOD

Abstract

The purpose of EXOD is revealing unnoticed variability into XMM-Newton observations.
The technique we use counts the amount of photons received during time windows slicing an observation and then compares those counts.
EXOD comes into two sub-programmes: a first one calculating the variability and detecting the variable areas for a given observation cleaned from high background; a second one rendering an image of the computed variability.
This document provides both a user and technical documentation on EXOD.

Contents

1 Introduction

XMM-Newton's automatic pipeline provides results from variability tests that are applied to sources where the number of counts is > 500 in total (pn, MOS1 and MOS2) (Watson et al. 2009). However, faint sources presenting rapid variations might not emit enough photons and could go unnoticed by the pipeline.
3XMM-DR8, the last version of the catalogue, is the largest catalogue of X-ray sources detected using a single X-ray observatory to date, containing 10 242 observations (Rosen et al. 2016). This large amount of observations could contain many of these sources that would provide us with useful data to constrain known phenomena or possibly hide interesting sources that have never been observed before.

The main objective of this algorithm is to detect fast transients among XMM-Newton's EPIC-pn observations, covering timescales from less than a second to a few minutes. These timescales mainly include gamma-ray bursts at high redshift, type-I X-ray bursts in distant galaxies and the X-ray counterpart to FRBs.

EXOD is a set of Python scripts consisting of two main programmes:

1. A detector that computes the variability in the whole field of view of an EPIC-pn observation from the filtered events file. The detector has two levels of detection:
   (a) The computation of the variability of each pixel of the detector by binning the observation into time windows.
   (b) The detection of variable sources with the sliding boxes technique.
2. A renderer that will produce an output image of the variability of the observation in the World Coordinate System (WCS).

2 Requirements

2.1 Hardware requirements

Depending on the way the programme is used, one should have at least 32 GB of RAM to run it.

2.2 Software requirements

EXOD is a set of Python scripts, therefore, Python3 is needed. The programme had been tested with Python v3.6, on Linux. The external libraries required for the detector are the following:

- `argparse`
- `astropy`
- `numpy`
- `multiprocessing`

The external libraries required for the renderer are the following:

- `argparse`
- `astropy`
- `pylab`
- `numpy`
- `scipy`
- `matplotlib`

2.3 Input data requirements

The algorithm uses the EPIC-pn Imaging Mode Event List (PIEVLI) performed in the *full frame mode* or *extended full frame mode* as well as the Good Time Interval (GTI) file containing the periods that have not been polluted by soft proton flares from the Sun.

3 Using EXOD

EXOD comes into two sub-programmes: a detector performing variability calculation and detection; and a renderer generating images illustrating the detection output. The following section explains the use each of these two scripts, enumerates the parameters offered to tweak the behaviour of the programme and enlightens the content of their outputs.
The source code can be found in the *EXOD* folder in `https://framagit.org/InesPM/EXOD`.

3.1 Detector

`detector.py` is the main variability detection programme. It calls the functions defined in other python scripts. It returns the number of counts per pixel and per time window, the variability of each pixel, the time windows that were used as well as the detected variable areas and variable sources.
The detector is run with the following command:

```
python3 detector.py <events_file> <gti_file> <output_folder> [options...]
```

The arguments that must be given are described below:

`events_file`	Path to the filtered events file.
`gti_file`	Path to the GTI file.
`output_folder`	Path to the folder where the output files will be generated.

The optional parameters are the following:

Table 1: Detector parameters

Parameter `--parameter`	`-par`	Accepted values	Default value	Function
`--observation`	`-obs`	OBSID		Observation ID
`--box-size`	`-bs`	[2..64]	5	Length in pixels of the detection box. Limited by the width of EPIC-pn's CCD cameras.
`--detection-level`	`-dl`	$\mathbb{R}^+$	10.0	Level above which an area is considered as being variable.
`--time-window`	`-tw`	$\mathbb{R}^+$	100.0	Duration of the time bins to count the events. It will give the minimal timescale of the variability. Minimal value given by the time resolution of the instrument (73.3 ms) and maximal value limited by the duration of the observation.
`--good-time-ratio`	`-gtr`	[0;1]	1.0	Critical (good time)/TW above which the time window will be taken into account.
`--max-threads-allowed`	`-mta`	$\mathbb{N}^*$	12	The maximal number of cores where the programme is allowed to run in parallel during the variability computation. It should be at most the maximal number of threads your computer can run simultaneously.
`--output-log`	`-ol`	string		Name of the general output file.

After each optional parameter, the desired value (`<val>`) must be given as `--parameter <val>` or the shorter version `-par <val>`

3.2 Renderer

The `renderer.py` script produces an image of the variability of the observation in the world coordinate system (right ascension and declination). This is very useful to visualize the variability

and compare it to the original observation.
The renderer is run with the following command:

```
python3 renderer.py <folder> <events_file>
```

The arguments that must be provided are the following:

`folder`	Path to the variability files where the image will be produced.
`events_file`	Path to the events file to obtain the necessary information from the header to extract the coordinates of the observation.

The optional parameters are the following:

Paruameter `--parameter`	`-par`	Accepted values	Function
`--observation`	`-obs`	OBSID	Observation ID
`--box-size`	`-bs`	[2..64]	Same than for `detector.py`. This is used for the title of the plot.
`--detection-level`	`-dl`	$\mathbb{R}^+$	Same.
`--time-window`	`-tw`	$\mathbb{R}^+$	Same.

3.3 Libraries

In this section I present the complimentary scripts that are used for the implementation of EXOD. These scripts are not runnable, they are used as libraries for both the renderer and the detector.

- `file_names.py`
  This script defines the name of the output files that will be stored in `<output_folder>`.

- `file_utils.py`
  It contains functions complimentary to the variability computation: opening and closing files and geometrical transformations to define the coordinates of the variable sources. The functions defined in the script are the following:

  - `open_files` Function opening files and writing their legend.
  - `close_files` Function closing all files.
  - `read_from_file` Function returning the content of a file.
  - `read_tws_from_file` Reads the list of time windows from its file.
  - `Source` (class) Datastructure providing easy storage for detected sources.
  - `read_sources_from_file` Reads the source from their file.
  - `size` Function defining the size of the image built by the renderer.
  - `rotation` Function rotating the output image of the renderer to match the observation angle.
  - `position_sources` Function giving the position (in raw pixel units) of the sources in the detector from the position in a ccd.

- `fits_extractor.py`
  This script extracts the data from the filtered events file and the GTI file with the following functions:

  - `extraction_photons` Function extracting the events list ordered by arrival time from its FITS events file.
  - `extraction_info` Function extracting some useful information from the header of the events file.
  - `extraction_deleted_periods` Function extracting the list of periods removed by high background filtering from the GTI file.

- `variability_utils.py`
  The functions that compute the variability and detect the variable areas are defined in this script and are the following:

  - `variability_computation` Function implementing the variability calculation.
  - `box_computations` Function summing the variability values of a box.
  - `__add_to_detected_areas` Function summing the variability values into a box.
  - `variable_areas_detection` Function detecting variable areas into a variability matrix.

3.4 Output files

Figure 1 presents the output of the renderer: the variability chart of the analysed observation. The X-axis represents the Right Ascension (RA) and the Y-axis represents the Declination (Dec) of the field of view. The color code corresponds to the variability of each pixel and the green circle the position of the detected variable source.

Table 2: Files generated by EXOD

File	Contents	Structure
`detected_variable_areas.csv`	Contains the list of the pixels for each area detected as variable.	Number of the source; CCD number; RAWX on CCD; RAWY on CCD
`detected_variable_sources.csv`	Contains the list of the gravity centre of all areas detected as variable.	Number of the source; CCD number; RAWX on CCD; RAWY on CCD; Right Ascension; Declination
`time_windows_file.csv`	Contains the list of all time windows used for variability calculation.	Number of the time window; Starting time of the time window
`variability_file.csv`	Contains the variability computed for each pixel.	Each line contains a floating number; the file is produced as: write variability for line_RAWX in ccd for value_RAWY in line_RAWX
`events_count_per_px_per_tw.csv`	Contains the list of events count for each time window for all pixels.	Each line contains a floating number; the file is produced as: write list of counts for line_RAWX in ccd for value_RAWY in line_RAWX
`log.txt`	Contains informations about the execution of the detector.	
`variability.pdf`	Image showing variability of the pixels. The darker colors represent a lower variability and the lighter colors represent a higher variability with a logarithmic colorscale.	
`sources.pdf`	Image showing the variability and the location of the detected sources, represented by a green circle.	

4 The algorithm

4.1 Variability computation

The steps to compute the variability of the filtered events are described below.

1. The time and pixel arrival of every detected event of the observation are extracted from the events file.

2. Photons detected in a 3×3 square around the central pixel are added together to ensure that only real sources are detected and to increase the signal-to-noise ratio.

3. The counted events are stored in a three-dimensional matrix that contains the number of photons that have been detected per pixel and per time window.

   - The first dimension is the RAWX pixel of the EPIC-pn camera that detected the photon.
   - The second dimension is the RAWY pixel of the camera that detected the photon.
   - The third dimension is the time window which stores the arrival time of the photon.

   This is the equivalent of creating a lightcurve for every single photon of the detector.

4. The good time intervals are extracted from the GTI file.

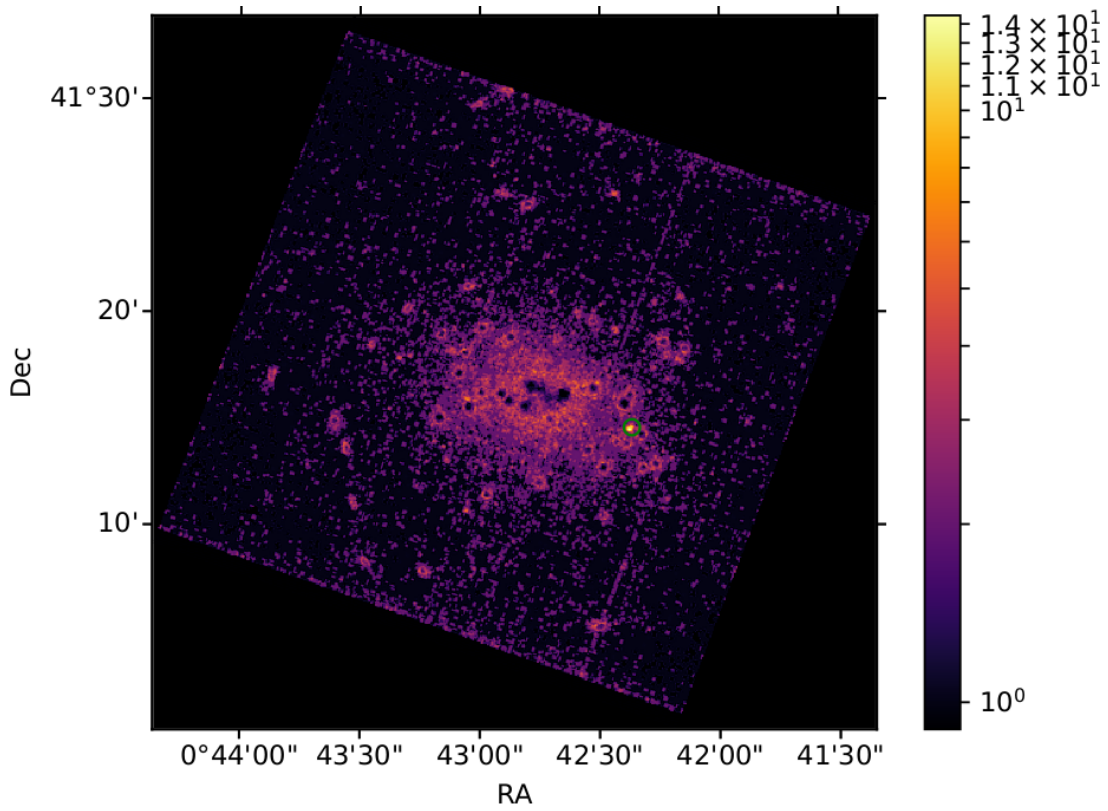

Figure 1: `sources.pdf` of the observation 0381_0112570101 computed with the default values.

5. The good time ratio, defined as the time belonging to the GTI (good time) of the considered time window divided by the duration of the time window, is computed for each time window.

6. The number of counts per time window will be divided by the good time ratio. Only those time windows with the good time ratio above a critical value of the good time ratio will be considered. This will normalize the number of photons that have been detected during a time window that has been shortened by the bad time periods.

7. The variability $\mathcal{V}$ of each pixel is given by:

$$\mathcal{V} = \begin{cases} max(\mathcal{C}_{max} - \tilde{\mathcal{C}}, |\mathcal{C}_{min} - \tilde{\mathcal{C}}|)/\tilde{\mathcal{C}} & \text{if } \tilde{\mathcal{C}} \neq 0 \\ \mathcal{C}_{max} & \text{if } \tilde{\mathcal{C}} = 0 \end{cases}$$

   Where $\mathcal{C}$ are the counts per pixel per time window, $\mathcal{C}_{max}$ and $\mathcal{C}_{min}$ are respectively the maximal and minimal number of counts of the considered pixel and $\tilde{\mathcal{C}}$ is the median number of counts over the time windows for the considered pixel.
   The expression $\mathcal{C}_{max} - \tilde{\mathcal{C}}$ targets sources presenting outbursts, while $|\mathcal{C}_{min} - \tilde{\mathcal{C}}|$ points to those sources with a period of lower flux. Considering the maximum between the two allows the detection of a wider variety of phenomena.
   The division by the median $\tilde{\mathcal{C}}$ will give the variability relative to the flux.

The variability computation algorithm can be visualised in Figure 2.

4.2 Variable sources detection

The variable areas are detected with the sliding boxes technique. A box of size $|b|^2$ pixels will move through all of the pixels of the observation.

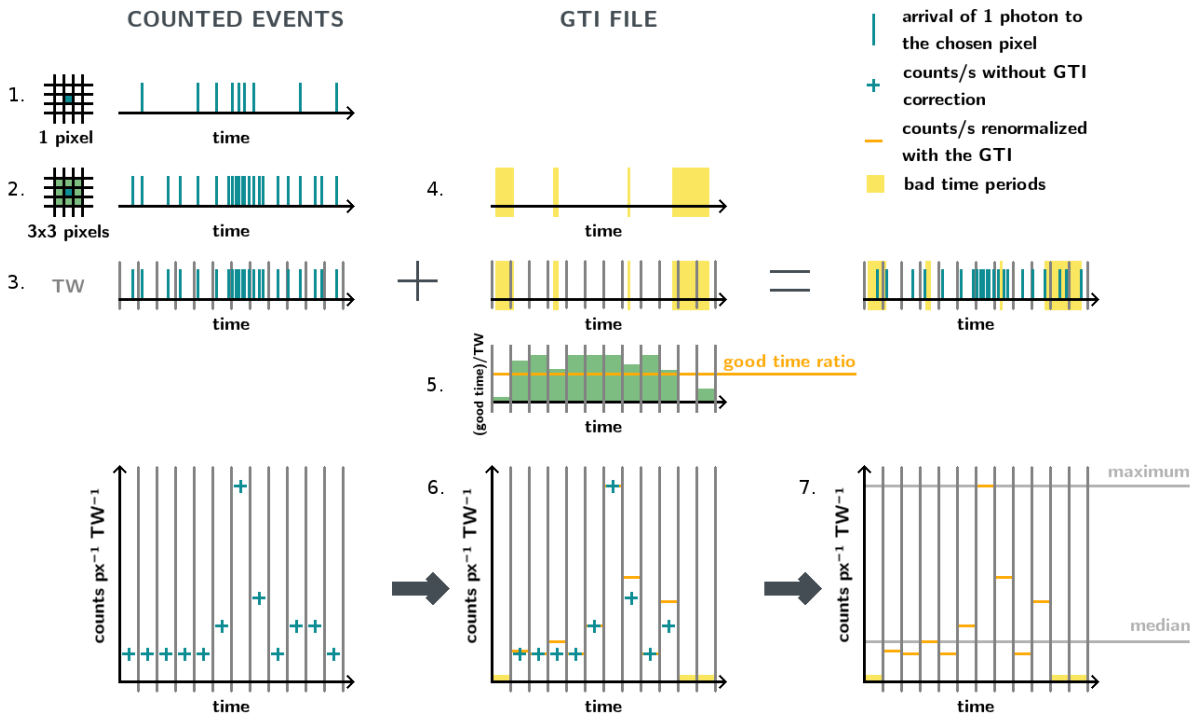

Figure 2: Diagram of the variability computation with the different stages of the algorithm.

1. The median of the variability of the pixels of the detector, $\tilde{\mathcal{V}}$, is computed.

2. Once the position of the box is defined, the variability of each pixel in the box is summed.

3. When the variability of the box $\mathcal{V}_{box}$ is above a chosen threshold, the pixels of the box are considered as a variable area. The value of this threshold is given by the following expression:

$$\mathcal{V}_{box} > DL \times |b|^2 \times \tilde{\mathcal{V}}$$

   Where DL is the detection level, b the length of the box in pixels and $\tilde{\mathcal{V}}$ the median value of the variability.

4. Once this is done, the position of the detection box is shifted by one pixel and the process is repeated.

5. When two consecutive boxes are variable, the pixels of both boxes are joined into a single variable area.

The variable sources are located at the barycenter of the variable areas. The position of the (X,Y) coordinates will be the mean value of the position of the pixels belonging to the variable areas.

5 Data analysis

EXOD is also provided with a set of scripts that allow one to perform the whole data analysis of the observations. These scripts include the following steps:

- Download the required files
- Filter the events files
- Apply EXOD to a set of observations
- Automatically generate the lightcurves of the detected sources and compute the χ^2 and Kolmogorov-Smirnov probabilities of constancy, P(χ^2) and P(KS) respectively.

Additionally to the EXOD python scripts, it makes use of XMM-Newton's Science Analysis System (SAS) version 16.1.0 (SOC 2018)[1]. We used the Xronos sub-package[2] of HEASOFT version 6.22.1 (Blackburn 1995).

5.1 Analysing a set of observations

The *bash* folder contains some scripts that can be used to perform the whole variability analysis of one or a whole set of observations.

- `download_observation.sh`
  This script downloads the files required for the variability computation and lightcurve generation. More information about the data download can be found in the XMM-Newton Science Archive[3]

- `filtering.sh`
  This script filters the PIEVLI file in an energy range between 0.5 and 12 keV, and it removes high background rates, bad pixels and PATTERN>4. This is done by calling some SAS commands. The path to where the SAS is installed is defined in the `preliminaries` function and might need to be changed.
  The default rate for the creation of the GTI file is 0.5 counts s^{-1}, but this can be modified with the optional `<RATE>` argument.
  The script is run with the following command:

  ```
  bash filtering.sh <OBSID> [FOLDER] [RATE]
  ```

  Where `<OBSID>` is the observation identifier, `[FOLDER]` is the path to where the observation is stored (the default is /mnt/data/Ines/data), and `[RATE]` is the threshold rate for the GTI generation, with 0.5 as default value. If a non numerical value or no value is given, the rate of the observation will be plotted so that the user can choose a GTI rate.
  This function generates a clean events file `$FOLDER/$OBSID/PN_clean.fits` and a GTI file `$FOLDER/$OBSID/PN_gti.fits`.

- `lightcurve.sh`
  This script generates lightcurves automatically from the position of the variable source detected with `detector.py`, and it computes P(χ^2) and P(KS). It uses a set of SAS and FTOOLS Xronos tasks. The lightcurves are plotted with `lcurve.py`, described below.
  The script is run with the following command:

[1] https://xmm-tools.cosmos.esa.int/external/xmm_user_support/documentation/sas_usg/USG/
[2] https://heasarc.gsfc.nasa.gov/ftools/xronos.html
[3] http://nxsa.esac.esa.int/nxsa-web/#aio

```
bash lightcurve.sh <OBSID> [options]
```

<OBSID> is the observation identifier, and the options are described in the table below.

Parameter		Default	Function
--source-id	-id	1	Source identifier, given in the first column of detected_variable_sources.csv.
--folder	-f	/mnt/data/Ines/data	Path to the folder containing the observations.
--scripts	-s	/mnt/data/Ines/EXOD	Path to the folder where the scripts are contained.
--detection-level	-dl	8	Detection level.
--time-window	-tw	100	Time window.
--good-time-ratio	-gtr	1.0	Good time ratio.
--box-size	-bs	3	Box size.

This script produces fits files with the .lc extension for the source, the backgrouns and the correlated lightcurves. The extraction regions are stored in the *region.txt file, and the lightcurve is stored as a pdf file. All of the outputs contain the name of the source. The probability of constancy of the source is stored in an output file $FOLDER/sources_variability_<DL>_<TW>

path to the observation, SCRIPTS is the path to the bash scripts, ID is the identifier of the variable source (integer) that was given by the detector. <DL> and <TW> are the same as before, and <output_log> is the path to the file where P(χ^2) and P(KS) will be written.

- manual_lightcurve.sh
  This script also generates a lightcurve and computes P(χ^2) and P(KS), but in this case the position and radius of the source and the background must be provided. This script should be used when the automatic extraction of the lightcurve becomes tricky, for instance with the presence of a nearby extended or bright source or in a crowded Field of View (FoV). The script is launched as follows:

```
bash manual_lightcurve.sh <OBSID> [options]
```

The inputs are the same as those described in lightcurve.sh.
Once the script is launched, the renderer.py output and PN_clean.fits will be displayed respectively with a pdf viewer and SAOImage DS9 (Joye & Mandel 2003). The position of the detected source will be displayed as a terminal output. This allows the user to select the desired extraction region on DS9 and give the source position in pixel units as a terminal input. Table 3 gives the commands that are printed on the terminal.

Table 3: Commands that are printed on the terminal when lc-sp.sh is launched.

Command	Input	Comments
Proceed? [y/n]	y or n	If the answer is y, the lightcurve analysis will continue. Otherwise, the script will be exited.
Source position [X]	X	X coordinate of the center of the source in pixel units.
Source position [Y]	Y	Y coordinate of the center of the source in pixel units.
Background position [X]	X	X coordinate of the center of the background in pixel units.
Background position [Y]	Y	Y coordinate of the center of the background in pixel units.
Radius [R]	R	Radius of the source and background extraction region in pixel units.

- lcurve.py
  This python script is called from both lightcurve.sh and manual_lightcurve.sh in order to generate a plot of the extracted lightcurve. The script is used as follows:

```
python3 lcurve.py <parameters>
```

The parameters are the following:

Parameter	Default	Function
`-src`	Mandatory	Path to the source's lightcurve fits file. Mandatory.
`-bgd`	Mandatory	Path to the background's lightcurve fits file. Mandatory.
`-gti`	Mandatory	Path to the GTI fits file. Mandatory.
`-dtnb`	100	Sampling time of the lightcurve in seconds.
`-outdir`	.	Path to the output directory.
`-name`	"Source"	Name of the source.
`-text1`	""	Text to add under the title.
`-obs`	""	Observation number.
`-id`	""	Id. of the variable source.

- `exod.sh`
  This script performs the whole variability analysis of the observations contained in a given folder. To launch this script, the PIEVLI files are required. The EPIC FITS global background timeseries (FBKTSR) is necessary for the automatic lightcurve generation. The script is run with the following command:

```
bash exod.sh [options]
```

The optional parameters are the following:

Parameter		Default	Function
`--folder`	`-f`	/mnt/data/Ines/data	Path to the folder containing the observations.
`--scripts`	`-s`	/mnt/data/Ines/EXOD	Path to the folder where the scripts are contained.
`--detection-level`	`-dl`	8	Detection level.
`--time-window`	`-tw`	100	Time window.
`--good-time-ratio`	`-gtr`	1.0	Good time ratio.
`--box-size`	`-bs`	3	Box size.
`--cpus`	`-cpus`	12	Number of CPUs to be used in parallel. The default `-mta` argument for `detector.py` is 1, so each CPU will analyse one observation at a time.

This script calls the following scripts: `filtering.sh`, `lightcurve.sh`, `parallel.sh`, `detector.py` and `renderer.py`.
It creates a series of output files in the `--folder` directory. The most important are the following:

  - `sources_variability_<DL>_<TW>_<GTR>_<BS>` : text file containing the name and observation of the detected sources as well as their P(χ^2) and P(KS).
  - `variability_observations_<DL>_<TW>_<GTR>_<BS>.pdf` : pdf containing the `sources.pdf` of each observation.
  - `lightcurves_<DL>_<TW>_<GTR>_<BS>.pdf` : pdf containing the lightcurves of all the detected sources.

- `parallel.sh`
  This script is used to run processes in parallel. The processes are read line by line from the file where they have been written. The script is used as follows:

```
bash parallel.sh <file> <CPUS>
```

`<file>` is the file containing the commands and `<CPUS>` the number of CPUs that will be used in parallel.

6 Tutorial

In this section, we give an example of how to use EXOD, including downloading the files, filtering them, computing the variability and generating the lightcurves automatically. We give the command lines that should be launched from the terminal.

6.1 Preliminaries

The EXOD repository has to be cloned to the machine where the variability analysis will be performed.
The first thing after cloning the repository from github is to modify the paths to the HEADAS, SAS and CCF installations. In the `filtering.sh` and `lightcurve.sh` scripts, they are given by

```
export SAS_CCFPATH=/home/ines/astrosoft/xmmsas_20180620_1732/ccf/
export HEADAS=/home/ines/astrosoft/heasoft-6.25/x86_64-pc-linux-gnu-libc2.27
. /home/ines/astrosoft/xmmsas_20180620_1732/setsas.sh
```

Unless you have the exact same version of HEADAS and SAS installed in the exact same path `/home/ines/Downloads`, which is very unlikely, you should change it in order to get the scripts to work. I also recommend to change the default folders defined in `exod.sh`, `lightcurve.sh` and `filtering.sh`. The default values are the following:

```
FOLDER=/home/ines/data
SCRIPTS=/home/ines/EXOD/scripts
```

You can change them to whatever default values you want to have. However, this is not necessary since they can be reset when launching the script with the `--folder` and `--scripts` options respectively.

6.2 One observation, automatic

We are first going to set some useful variables that we will use in multiple ocasions. For this tutorial, I chose to analyse the observation 0652250701 as an example.

```
obs=0652250701
```

We now proceed to the download of the observation, that can take from a few seconds to ~ 2 min, depending on the observation and your machine.

```
bash $SCRIPTS/download_observation.sh $FOLDER $obs
```

After the download, we filter the observation. In this example, the threshold rate for the GTI generation will be the default (0.5).

```
bash $SCRIPTS/filtering.sh -f $FOLDER -o $obs -s $SCRIPTS
```

This should generate the following files in the `$FOLDER/$obs` folder: `PN_clean.fits`, `PN_gti.fits`, `PN_rate.fits`, `ccf.cif`.
With these files, we are now ready to compute the variability of our observation! In this step we apply `detector.py` and `renderer.py` to our observation. See Table 1 for an explanation of the parameters used.

```
python3 -W"ignore" $SCRIPTS/detector.py $FOLDER/$obs/PN_clean.fits \
        $FOLDER/$obs/PN_gti.fits $FOLDER/$obs/8_100_1.0_3 -obs $obs -bs 3 -dl 8 \
        -tw 100 -gtr 1.0 -mta 12 -ol $FOLDER/$obs/variable_sources_8_100_1.0_3
python3 -W"ignore" $SCRIPTS/renderer.py $FOLDER/$obs/8_100_1.0_3 \
        $FOLDER/$obs/PN_clean.fits -obs $obs -tw 100 -dl 8 -bs 3
```

Since in this example I used 12 CPUs, the variability was computed in $\sim$ 80 s. This should generate the files listed in Table 2 in the `$FOLDER/8_100_1.0_3` folder. Excitingly, in Fig. 3 we can see that one variable source was detected. We will now proceed to the automatic extraction

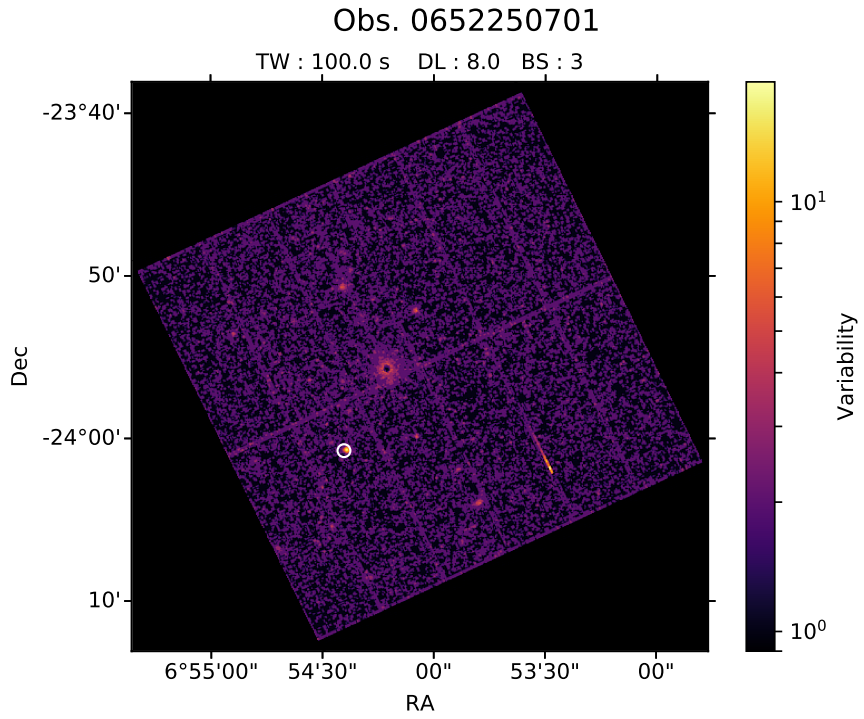

Figure 3: `sources.pdf` of the observation 0652250701 computed with the values of the tutorial. One variable source is detected at the position of the white circle.

of the lightcurve that was detected.

```
bash $SCRIPTS/lightcurve.sh -f $FOLDER -s $SCRIPTS -o $obs -dl 8 -tw 100 -gtr 1.0 \
     -bs 3 -id 1
```

This should be done in $\sim$ 70 s. With `src=J065423-240000` in this case, the files `$src_lc_0.0734_bgd.lc`, `$src_lc_0.0734_src.lc`, `$src_lc_100_bgd.lc`, `$src_lc_100_src.lc`, `$src_lccorr_100.lc`, `$src_region.txt` and `$src_lc_100.pdf` should have been generated in the `$FOLDER/$obs/lcurve_100` folder. The pdf of this lightcurve is shown in Fig. 4, and we can see that the source is showing an outburst during the observation.

6.3 Group of observations

In this section, we give an example of how to apply EXOD to a group of observations. The observations presented here were randomly chosen. We first define our variables; an array containing the OBSID of the observations we want to analyse, and the path to the observations and the scripts. Then, we apply `download_observation.sh` in a for loop.

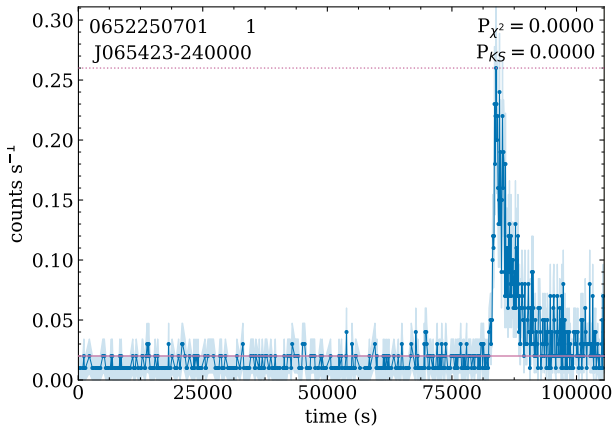

Figure 4: Lightcurve of the detected source in observation 0652250701 plotted with `lcurve.py`. The lightcurve is shown in blue with shaded segions representing the error bars. The solid pink line shows the median number of counts s^{-1}, and the dashed pink line shows the maximal number of counts s^{-1}.

```
observations=(0673420101 0677650132 0690500101 0740570101)
SCRIPTS=/mnt/data/Ines/EXOD
FOLDER=/mnt/data/Ines/data/guide

for obs in ${observations[@]}; do
  bash $SCRIPTS/download_observation.sh $FOLDER $obs
done
```

If there is a high number of observations to be analysed (> 4), we encourage to use the `parallel.sh` script to speed up the process. We can replace the previous for loop by the following, where 4 is the number of CPUs we are using. This number should be less or equal to the number of observations that are being analysed.

```
for obs in ${observations[@]}; do
  echo "bash $SCRIPTS/download_observation.sh $FOLDER $obs" >> \
       $FOLDER/process_download
done
bash $SCRIPTS/parallel.sh $FOLDER/process_download 4
```

Once we have downloaded the files, we can carry the full analysis, including filtering the observations, computing the variability, rendering the output and generating the lightcurves of the detected sources with the `exod.sh` script. The script will generate a pdf with the variability of all the observations and another one with the lightcurves of the detected variable sources.

```
bash $SCRIPTS/exod.sh -f $FOLDER -s $SCRIPTS -dl 8 -tw 100 -gtr 1.0 -bs 3 -cpus 4
```

WARNING: there might be some issues on the pdf generation when applying `exod.sh` on more than ~ 500 observations, since the size of the pdfs than ghostscript can handle is limited.
This command should produce the following outputs:
In `sources_variability_8_100_1.0_3` we can see that three sources were detected, one in obs. 0673420101 and two in obs. 0740570101. P(χ^2) and P(KS) = 0.0000 for all the three sources.

```
Observation Source DL TW P_chisq P_KS
0673420101 1 J172554-643841 8 100 0.0000 0.0000
0740570101 1 J054103-093653 8 100 0.0000 0.0000
0740570101 2 J054030-093552 8 100 0.0000 0.0000
```

The output of the renderer, `variability_observations_8_100_1.0_3.pdf`, is shown in Fig. 5, and the lightcurves of the detected sources `lightcurves_8_100_1.0_3.pdf` are plotted in Fig. 6.

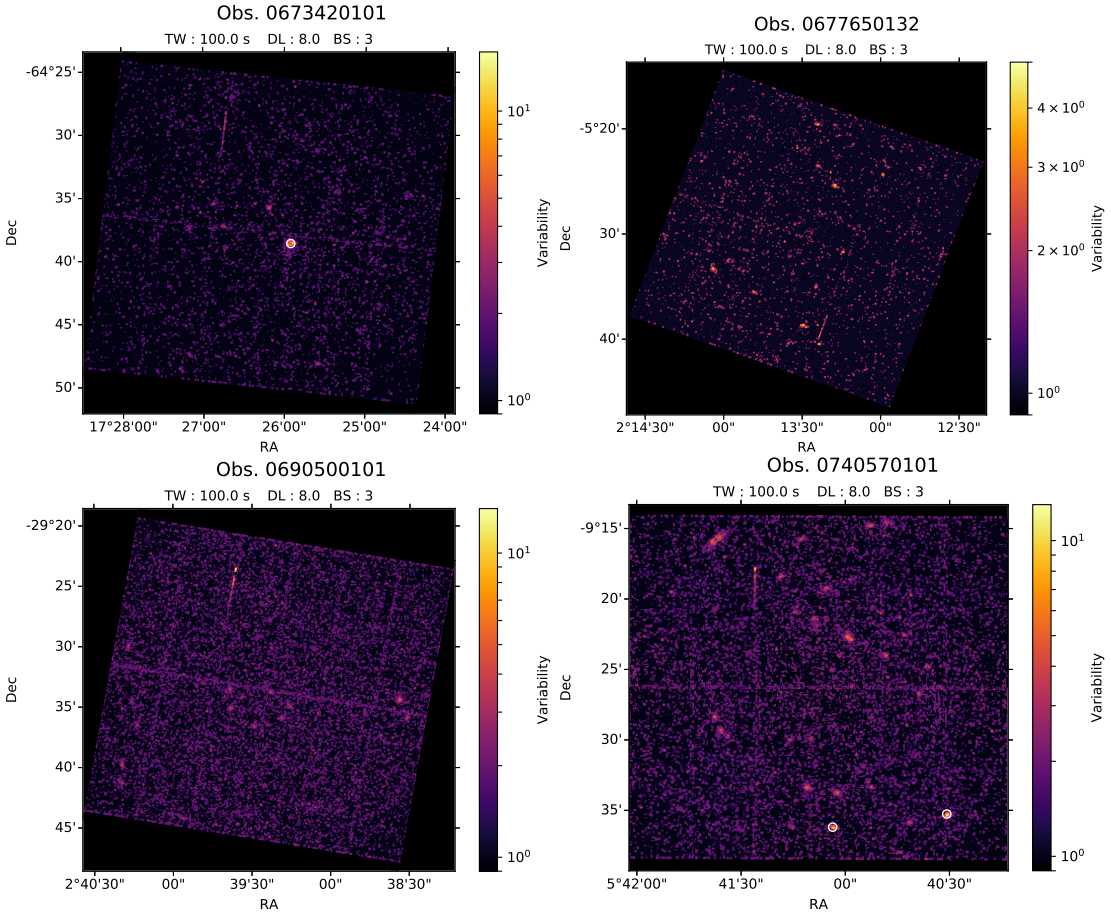

Figure 5: `sources.pdf` of the observations analysed in the example of a group of observations. One source has been detected in the top left observation, and two in the bottom right observation.

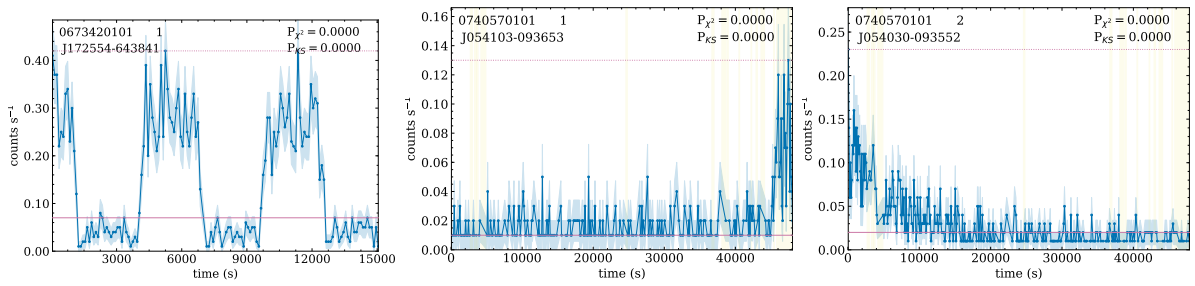

Figure 6: Lightcurves of the detected sources. The lightcurve is shown in blue with shaded segions representing the error bars. The solid pink line shows the median number of counts s^{-1}, and the dashed pink line shows the maximal number of counts s^{-1}. The light, yellow shaded regions represent the bad time intervals.

6.4 FITS output (under development)

I am developing a version of EXOD in which the variability output is on a FITS format. The scripts can be found on the fits_output branch[4].
This gets simultaneously rid of the coordinates problem. The goal is to deprecate the csv output. The use of fits_output is similar to the code in the master branch, but the detector has a few exta arguments and the renderer is ran from the detector.

```
obs=0652250701
SCRIPTS=/mnt/data/Ines/EXOD_fits
FOLDER=/mnt/data/Ines/data

bash $SCRIPTS/download_observation.sh $FOLDER $obs
bash $SCRIPTS/filtering.sh -f $FOLDER -o $obs

python3 -W"ignore" detector.py -path $FOLDER/$obs --render --ds9
```

The `-render` option will run the renderer, while the `-ds9` argument will open the variability fits file with ds9. Type `python $SCRIPTS/detector.py -h` to get the parameters list and default values.

[4] https://github.com/InesPM/EXOD/tree/fits_output

7 Future work

Some aspects of the algorithm remain to be refined. In this section I list some of the aspects that I have noticed but have not had the time to change:

7.1 Detection modes

Currently, EXOD can only be applied to EPIC-pn observations performed in the *full frame mode* and *extended full frame mode*. These two modes are those with the largest field of view (30'×30'), allowing the detection of more off-axis variable sources. The *full frame mode* has a time resolution of 73.3 ms, while the *extended full frame mode* has a time resolution of 199.2 ms (Strüder et al. 2001). The remaining observation modes have a smaller field of view but a higher time resolution, and it could be advantageous to apply EXOD to those modes in order to detect variability on the shortest timescales.
To do so, the algorithm needs some modifications regarding the size of the detector and the CCDs that are considered for each detection mode.
The algorithm could also be adapted to detect variability in the MOS1 and MOS2 detectors.

7.2 Coordinates of the detected sources

The coordinates of the sources detected as being variable in the World Coordinate System (WCS) is determined with the Python package `astropy.wcs.WCS`. The header keywords of the filtered events files do not correspond to the ones that the WCS package expects. These expected header keywords are changed in the programme (both the detector and the renderer), so that the observation is plotted with the correct coordinates.
However, the coordinates of the off-axis sources are not correctly determined. As the position of the sources is given by a rotation and a translation, there might be an off-axis deformation that is not taken into account by these transformations.
It could be useful to reproduce the algorithms that are used to plot fits files with ds9[5].

The class `Source` defined in the script `file_utils.py` could be improved by including the geometrical transformations that are performed with the functions `size`, `rotation` and `position_sources`, so that the coordinates are directly determined from the detection pixel and the information in the header of the filtered events file.

renderer The renderer was tested with Python 3.4.3. There seems to be a problem with the renderer when using Python 3.7.1; the position of the pixels is not correctly reproduced.

7.3 Properties of the detected sources

Although EXOD is able to detect variable sources, it also detects other types of sources or artifacts that might be present in an observation. This makes the visual inspection of the detected sources necessary if a lightcurve wants to be generated, to determine the exact coordinates of the source and whether it is a real source or not. These cases are presented below and they can be visualised in Figure 7.

When a bright source is variable on a timescale different from the one of the chosen time window, the central position of the source does not present a high variability value. However, its extended Point Spread Function (PSF) might be detected (even more than once) as a variable

[5] `http://ds9.si.edu/site/Home.html`

source.

Another case where a visual inspection of the detected source is necessary is when an extended source such as a galaxy cluster (typically non variable) presents a pixel above the detection level. If it is the case that a source within the galaxy cluster is presenting an outburst at the moment of the observation, the lightcurve analysis would not be possible to do, because the source would be contaminated by all of the surrounding sources.

It is also possible to detect one single variable source when two independent sources are close enough to have overlapping PSFs in the detector. In this case, the position of the source that the algorithm detects can be located between the two sources. The lightcurve of the two objects has to be generated individually.

The last instance of sources that need to be visually inspected before the generation of the lightcurve are the "bad pixels". Some pixels of an observation can have a high noise in the low energy (0.2 – 0.5 keV) range. These regions of the detector are usually detected as variable sources, although they present a square shape instead of the typical circular or elliptical shape of the PSF. To avoid detecting them, this range of energies must be filtered from the events files.

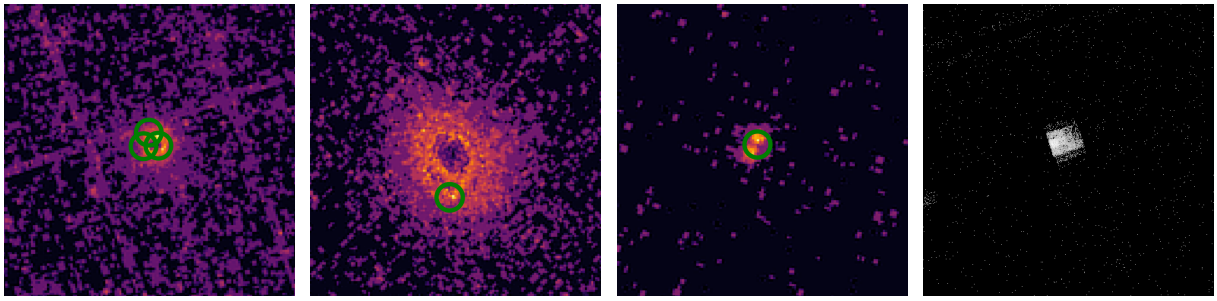

Figure 7: From left to right: **1.** bet CMa, a variable star of beta Cep type. The algorithm detected three sources on its PSF. **2.** ACO 1918, a cluster of Galaxies. One off-center source is detected. **3.** Two nearby sources, 61 Cyg A, a variable star of BY Dra type, and 61 Cyg B, a flare star, are detected as a single variable source by the algorithm. **4.** Typical shape of a bad pixel in the events file detected as a variable source.

In the future, it could be useful to take into account the shape and size of the variable area in order to avoid detecting these sources.

References

Blackburn, J. K. 1995, in , 367

Joye, W. A. & Mandel, E. 2003, in , 489

Rosen, S. R., Webb, N. A., Watson, M. G., et al. 2016, Astronomy and Astrophysics, 590, A1

SOC, E. X.-N. 2018

Strüder, L., Briel, U., Dennerl, K., et al. 2001, Astronomy and Astrophysics, 365, L18

Watson, M. G., Schröder, A. C., Fyfe, D., et al. 2009, Astronomy and Astrophysics, 493, 339