badpixfind

January 12, 2017

Abstract

The **badpixfind** task searches for 'bad' pixels (whether 'hot', 'dead', 'flickering' etc.) and therefore as pixels containing no useful data. A file, containing information on the position and the type of defect of the bad pixels, is produced.

1 Instruments/Modes

Instrument	Mode
EPIC MOS	IMAGING
EPIC PN	IMAGING

2 Use

pipeline processing	yes (see comments)
interactive analysis	yes

3 Description

badpixfind searches for pixels identified as containing no useful data, and uses this information to produce an output file. This output file can then be used by the task **badpix** to append the information to the events file. A bad pixel map (or mask) can also be produced.

An important point regarding the bad pixels is that there are essentially *three separate* (though not exclusive) sets of bad pixels that must be dealt with. These are (1) the bad pixels uplinked to the satellite and eliminated on-board, (2) the bad pixels identified in the Current Calibration File (CCF) but not uplinked, and (3) the bad pixels associated with the particular observation in question. **badpixfind** searches for this third set of bad pixels, which may of course, contain within it part (or all) of the other two sets.

There are two modes to **badpixfind**. Mode 1 involves all the searching and statistical techniques and makes up the bulk of the task. Within mode 1 (`searchbadpix`='Y', [=default]), the user can search for...

1 Hot and dead pixels. The user is able to specify maximum and minimum allowed values (via the parameters `lothresh` and `hithresh`) to screen for hot and dead pixels. These threshold levels can be input (via the parameter `thresholdlabel`) in terms of number of counts, count rate, or percentages of the peak value in the dataset. In cases of supposed hot pixels, the pixel in question and its neighbours are compared with the local PSF to check whether the pixel is genuinely 'hot' or may be due to a source [see comments]. Here the user can set the `cellsize` of the box around the hot pixel to be compared with the PSF (total box width = twice `cellsize` +1). Here, known uplinked bad pixels and previously found bad pixels are removed from the PSF comparison. The user can also set the parameter `narrowerthanpsf`. This is a measure of how much more compact the pixel and its neighbours are when compared with the PSF (1 means of similar width, greater than 1 means that feature is narrower than the PSF, and hence likely a bad pixel [if `narrowerthanpsf` is set to zero, the PSF comparison is removed and all hot features, whether sources, bad pixels or whatever, can be found]). Essentially, `narrowerthanpsf` is the ratio A/B, where A is the ratio of the target (bad) pixel to the total (twice `cellsize` +1 by twice `cellsize` +1) square, and B is the equivalent for the PSF. The user may also set a `backgroundrate` (ct/s/pix) here. A negative value indicates that the mean value over the whole field is to be calculated and used. In cases where very bright, extended sources exist (and thus the mean is not too representative of the true background mean), a typical `backgroundrate` value should be given (see comments).

2 Hot and dead whole columns (channels) (`withsearchbadcolumn`='Y'). Here a column may appear bad because of a small number of individual pixels that are bad (these perhaps, depending on the threshold levels, being detected by the previously described part of the task), or it may be bad because the entire column as a whole appears bad. Via the parameter `columnsearchlabel`, one can specify whether the minimum and maximum *column* threshold values (`locolthresh` & `hicolthresh`) refer to the total counts/count rate of the column or to the median value within each column. The median choice is more sensitive to finding bad whole columns with no particularly bad pixels within them. Again, the threshold levels can be input (via the parameter `thresholdlabel`) in terms of number of counts, count rate, or percentages of the peak value in the dataset.

3 Flickering pixels. The parameter `flickertimesteps` sets the number of 'timesteps' into which the dataset can be split (if `flickertimesteps` is set to 1, no attempt is made to seach for flickering pixels). Here, each pixel is searched for significant variations. In the case of low count pixels, the distribution of counts per timestep is compared with a Poisson distribution, and a Kolmogorov-Smirnov (KS) test is applied. Pixels with a KS-statistic above the input threshold `flickerksthresh` are deemed flickering. In the high count pixel case, a chi-squared statistic about the mean counts per timestep is calculated. Pixels with a reduced chi-squared statistic above the input threshold `flickerchisqthresh` are deemed flickering. [see comments]

The whole of mode 1 above can be performed between user-defined energy bands, via the parameters `loenergythresh` and `hienergythresh` (in keV). Also, the high threshold values `hithresh` and `hicolthresh` can (in the `thresholdlabel` = COUNTS or RATE cases) refer to either a total count or countrate value or to a count or countrate value above the (either set or calculated) background (`backgroundrate`). This choice is set via the parameter `threshabovebackground`.

Within mode 2 of **badpixfind**, the user can explicitly flag invalid pixels or sets of pixels via the list parameters `rawxlist`, `rawylist`, `typelist` and `yextentlist`.

After searching for pixels which may contain double (or more) entries, and warning the user thereof, a final output file is constructed, containing for each bad pixel (or, to be precise, for each set (in RAWY-extension) of bad pixels), the position RAWX/RAWY, any extent in the Y direction, and the type of fault; 1 Hot, 2 Flickering, 3 Dead. Statistics as to the input file values are provided also by the task, before and after the individual sections of the task.

A badpixfind map may be produced via the parameter `withbadpixmap`. Here an image is formed whereby bad pixels (and if desired, surrounding neighbours [chosen via the parameter `mappixcellsize`]) are assigned a value of zero, good pixels a value of one. This map includes the pixels found by the current **badpixfind** run, and the XMM-uplinked bad pixels contained within the CCF. Furthermore, regions outside the field of view (FOV) can be set to zero in the map via the parameter `withfovmask`. This map can be used in **evselect** to select images, spectra or time series with these pixels ignored (e.g. to produce a background [source and bad pixel/column-removed] lightcurve).

4 Parameters

This section documents the parameters recognized by this task (if any).

Parameter	Mand	Type	Default	Constraints

eventset	yes	string		

input events file (either single-chip event file from **epframes** or **emevents**, or merged, calibrated multi-chip event file)

searchbadpix	no	logical	Y	Y/N

search for bad pixels?

withsearchbadcolumn	no	logical	N	Y/N

search for bad total columns?

userflagbadpix	no	logical	N	Y/N

user-flag specific (sets of) bad pixels?

thresholdlabel	no	string	rate	peak/counts/rate

Thresholds (all) choice - thresholds refer to percentage of PEAK value in the dataset, to count RATE or to pure COUNTS

ccd	no	integer	1	1-12

chip on which to search bad pixels if task is run on multi-chip event list

lothresh	no	real	0.0	≥ 0 (≤ 100 for thresh-oldlabel=peak)

Low threshold value for dead pixel search (`searchbadpix`=Y)

hithresh	no	real	0.005	≥ 0 (≤ 100 for thresh-oldlabel=peak)

High threshold value for hot pixel search (`searchbadpix`=Y)

loenergythresh	no	real	0.0	0.0-30.0)

Low energy (in keV) bound for searching

hienergythresh	no	real	30.0	0.0-30.0)

High energy (in keV) bound for searching

cellsize	no	integer	2	1-10

Cell size for PSF comparison (total width: 1 + 2×cellsize)

narrowerthanpsf	no	real	3.0	≥ 0

PSF-pixel(s) compactness comparison - 1, equal to PSF, >1, more compact, and hence likely, not a source

backgroundrate	no	real	0.00001	

Background rate (ct/s/pix) - if negative, mean over entire field assumed

threshabovebackground	no	logical	N	Y/N

determine whether `hithresh` and `hicolthresh` refer to values above the background rate (Y) or as total values (N)

columnsearchlabel	no	string	median	median/total

`withsearchbadcolumn`=Y thresholds choice - `locolthresh` & `hicolthresh` refer to TOTAL column value or MEDIAN column value

locolthresh	no	real	0.0	≥ 0 (≤ 100 for thresholdlabel=peak)

Low threshold value for dead column search (`withsearchbadcolumn`=Y)

hicolthresh	no	real	0.0015	≥ 0 (≤ 100 for thresholdlabel=peak)

High threshold value for hot column search (`withsearchbadcolumn`=Y)

flickertimesteps	no	integer	1	≥ 1 ($\leq\sim 15$ [see comments])

Number of timesteps to search for flickering pixels

flickerksthresh	no	real	0.55	0-1

K-S threshold for low count flickering pixels

flickerchisqthresh	no	real	15.0	≥ 0

Reduced Chi-sq threshold for high count flickering pixels

rawxlist	no	integer list		1-600

List of user-flagged RAWX values (`userflagbadpix`=Y)

rawylist	no	integer list		1-600

List of user-flagged RAWY values (`userflagbadpix`=Y)

typelist	no	integer list		1-5

List of user-flagged TYPE values (`userflagbadpix`=Y)

yextentlist	no	integer list		1-600

List of user-flagged YEXTENT values (`userflagbadpix`=Y) [Note: these four 'lists' must contain equal number of entries]

badpixset	no	string	badpixfind.fits	

output file from **badpixfind** containing set of bad pixels (input to **badpix**

withbadpixmap	no	logical	N	Y/N

Create bad pixel image map?

withfovmask	no	logical	N	Y/N

Mask out regions outside of the field of view (FOV) in map?

mappixcellsize	no	integer	1	≥ 0, ≤ 10

Bad cell surrounding bad pixel in map (total width: $1 + 2 \times$ `mappixcellsize`)

badpixmap	no	string	badpixmap.fits	

Bad pixel image map file name

(note: Default settings are reasonable for MOS analysis, and very conservative for pn analysis [see comments for discussion])

5 Errors

This section documents warnings and errors generated by this task (if any). Note that warnings and errors can also be generated in the SAS infrastructure libraries, in which case they would not be documented here. Refer to the index of all errors and warnings available in the HTML version of the SAS documentation.

badInput *(error)*
No, or incorrectly named extension in input events file

badInstrument *(error)*
Instrument not supported

badMode *(error)*
Mode not supported

noBadpixfindModes *(error)*
No badpixfind modes chosen

nonEqualLists *(error)*
userflagbadpix lists (rawx, rawy, type, yextent) must be of equal size

badThresholds *(error)*
low threshold values must be less than high threshold values (also, for `thresholdlabel` = PEAK, threshold values must be in range 0–100

userListValuesOutOfRange *(error)*
some `userflagbadpix` list input values are invalid

badLothresh *(error)*
Lothresh value finds too many (>10000) dead pixels - dataset is probably filled with many zeros

noGoodPixsFound *(warning)*
No good pixels found
corrective action: Continue with 'thresholds OK?' message

eventsOutOfWindow *(warning)*
Events found outside of RAWX/RAWY window - these are ignored
corrective action: Continue with warning message

eventsOutOfOrder *(warning)*
Events out of time order - flickering analysis may be in error
corrective action: Continue with warning message

insufficientData *(warning)*
insufficient data (events/timeslots) to perform flickering analysis - aborted
corrective action: Continue with warning message - no flickering analysis

findRealSources *(warning)*
Chosen narrowerthanpsf parameter is very low - real sources may be found
corrective action: Continue with warning message

badEnergyThresholds *(warning)*
hienergythresh must be greater than loenergythresh - no Energy filtering performed
corrective action: Continue with no energy filtering

lowExposure *(warning)*
Exposure time very low - spurious bad pixels may be found (low number statistics)
corrective action: Continue with warning message

6 Input Files

1. Either raw single-chip event list from **epframes** or **emevents** or final merged, calibrated, multi-chip event list. Attributes read are: `WINDOWX0` `WINDOWY0` `WINDOWDX` `WINDOWDY` `EXPOSURE`(MOS) `LIVETIME`(pn) `LIVETInn`(pn).

7 Output Files

1. File containing the RAWX/RAWY of the bad (sets of) pixels, any extent in the Y direction, and the type of fault. The extension name is `BADPIX`, and the columns are named `RAWX`, `RAWY`, `TYPE` and `YEXTENT`. This file can be accessed by **badpix**. Attributes written are: `TELESCOP INSTRUME CCDID EXP_ID OBS_ID QUADRANT CCDNODE`(MOS) `WINDOWX0 WINDOWY0 WINDOWDX WINDOWDY`.
2. Optional map of the bad pixels found. Zero values exist at the RAWX/RAWY positions of the bad pixels found in the current **badpixfind** run, the previously uplinked CCF bad pixels, and optionally, their neighbours. Values of unity are given to the good pixels. The keyword `GOODPIX` (number of good pixels in map) is written to the map header.

8 Algorithm

```
subroutine badpixfind

 * Get input file, parameters, instrument, mode, ccdid, exposure, obsid
 * set state of CAL
 * exit if no modes chosen (searchbadpix=N, userflagbadpix=N)
 * check userflagbadpix lists are of equal size
 * open output file
 * get uplinked bad pixels from CCF
 * get 'image' sizes and origin values, start times and end times
 * form images in memory from input events file, selecting energy band if
     required (in case of MOS, call pattern backprojecting and selecting
     subroutines to get true image)
 * if MOS and flicker, get timestep indices - form 'flicker' array
 * calculate and output (to screen) initial pixel statistics
 * if required, calculate and output (to screen) initial column statistics
```

```
    * call mode_1 (searchbadpix), and mode_1 column analysis
        (withbadcolumnsearch) (+ PSF comparison), if required.
        set of procedures (max/min threshold) techniques performed on image
        * if hot pixel found - check edge effects - compare with PSF, using
          only good pixels (no uplinked or found bad pixels)
          - return measure of comparison with PSF
      * if requested, perform flickering analysis (low and high count cases)
    * Obtain badpix set (for mode1) (RAWX, RAWY, Y-extent, fault type)

    * calculate and output (to screen) new pixel statistics

    * call mode_2, if required
        explicitly flag pixels as bad, the user having given RAWX, RAWY,
        Y-extent and type of fault as input lists
    * Obtain badpix set (for mode2) (RAWX, RAWY, Y-extent, fault type)

    * Check badpix sets (mode1 and mode2) for redundancy/duplication
    * calculate and output (to screen) final pixel statistics
    * Combine individual badpix sets into output file
    * Create, if required, bad pixel map (with optional FOV masking)
    * Add attributes and history

  end subroutine  badpixfind
```

9 Comments

- To quote fhelp cleansis (the ASCA equivalent routine) - *Warning: This is an analysis tool, not a program to run blindly: Clean, check, repeat.*
- As regards how much of the **badpixfind** routine can safely be incorporated into the PPS, we believe that the number of bad pixels found by **badpixfind** may depend sensitively on the input parameters (thresholds etc.). The task has been and is being developed so that an automatic running (and importantly, usage in **badpix**) can be performed. Tests on many datasets have been performed to find the most favourable parameter settings (which will be different for MOS and pn). The default settings have been set to reasonable values for MOS analysis. They are very conservative for pn analysis however, a more favourable pn parameter setting being: `hithresh`=0.0045 `withsearchbadcolumn`=Y `hicolthresh`=0.00105 `backgroundrate`=0.0001 `loenergythresh`=0.14 `hienergythresh`=10.0 `narrowerthanpsf`=1.5
- The searching algorithms within badpixfind can be performed within user-defined energy ranges. This will make any automatic usage of badpixfind a little safer, the noiser, very low energy photons being ignorable. For PN, the events are selected via their `PHA` values, for MOS, via their `ENERGYE1` values.
- It is possible for a bad pixel map to be produced (with optionally, areas outside the FOV masked out also). This is described throughout the task description (see especially the user's description and the parameters and examples section). This map can be fed into **evselect** to create 'masked' images, lightcurves etc.
- Searching for 'flickering' pixels is possible, as described in the user's description. This, as with all of the mode 1 routines, must not be used blindly - any number of effects could lead to a 'flickering' pixel (a bright source/feature passing briefly into the pixel, a genuinely variable source, CCD edge effects etc). It is recommended that a smallish number of `flickertimesteps` is used (∼5). Larger numbers lead to lots more small number statistics and sometimes (especially for MOS) very large arrays.

- It is believed that this task is inapplicable to timing and burst modes, and will not be accessed.
- In obtaining the exposure information, for the pn, the `LIVETIME` attribute is read. For MOS, the `EXPOSURE` attribute is read. In the case of merged, calibrated event lists the `LIVETInn` attribute corresponding to the appropriate chip is read for both PN and MOS.

10 Future developments

- Some procedures for mode 1 may need to be developed further.
- Work is in progress to find optimum MOS and pn parameter settings for automatic PPS usage.

11 Examples

- *badpixfind eventset=mosevents.dat* (default automatic MOS PPS setting)
- *badpixfind narrowerthanpsf=1.5 backgroundrate=0.0001 hithresh=0.0045 hicolthresh=0.00105 withsearchbadcolumn=y loenergythresh=0.14 hienergythresh=10.0 eventset=pnevents.dat* (recommended automatic PN PPS setting)
- *badpixfind eventset=mosevents.dat thresholdlabel=peak backgroundrate=-1 hithresh=90 flickertimesteps=5 flickerksthresh=0.52 flickerchisqthresh=12* (bad pixels brighter than 90% of the peak brightness are searched for, as are flickering pixels with variabilities in excess of the given flicker thresholds)
- *badpixfind eventset=mosevents.dat searchbadpix=n userflagbadpix=y rawxlist="3 6" rawylist="5 8" typelist="4 4" yextentlist="2 2" badpixset=bpxfmos.fits* (just mode 2 (explicit user flagging) is used to create two entries in output file bpxfmos.fits)
- *badpixfind eventset=pnevents.dat threshabovebackground=Y withbadpixmap=Y hithresh=0.00025 withsearchbadcolumn=y hicolthresh=0.00007 backgroundrate=-1 loenergythresh=7.0 hienergythresh=15.0 narrowerthanpsf=0.0 withfovmask=Y* (detects bad pixels/columns and also sources in a hard energy band, the thresholds set according to a calculated background level, producing a background mask [badpixmap.fits] with the FOV masked out, usable (in **evselect**) to create a background spectrum/lightcurve etc)

References