embadpixfind

January 25, 2021

Abstract

Find EPIC-MOS bad pixels for one node of one EPIC-MOS CCD.

1 Instruments/Modes

Instrument	Mode
EPIC MOS	IMAGING

2 Use

pipeline processing	yes
interactive analysis	yes

3 Description

embadpixfind aims at finding bad pixels in an EPIC-MOS image in a completely automatic way, by taking advantage of the broad PSF in pixel units, which makes it impossible to mistake a source for a bad pixel.

3.1 Preparatory work

Call **emeventsproj** with `rejectbadevents`=Y to project the events file to an image. An image built by **evselect** would be accepted as well, but would not include the secondary pixels of multiple events (this may miss weak bad pixels next to a strong one).
For optimal electronic noise rejection, the events file should be built by successive calls to **emevents** (`analysepatterns`=N `flagbadpixels`=N `splitdiagonals`=N `randomizeposition=N` allow it to run much faster) and **emenergy** (`correctcti`=N `correctgain`=N `randomizeenergy`=N allow it to run much faster).

For calibration purposes, it is possible to improve the statistics (and the sensitivity to weak bright pixels or dark pixels) by stacking many images output of emeventsproj on top of one another before calling **embadpixfind**.

3.2 Basic algorithm

embadpixfind estimates the local statistical average μ in a running window around each pixel by taking the smallest of the average or the median + 1 (1 is added to take care of the case when the median is 0, the median allows to remove the effect of other bad pixels in the vicinity). Then it builds a significance map via the Li and Ma criterion (Li & Ma 1983, ApJ 272, 317):

$$S \;=\; \sqrt{2}\,\sqrt{N_{\rm on}\,ln\frac{N_{\rm on}}{\mu_{\rm tot}} + N_{\rm off}\,ln\frac{\mu}{\mu_{\rm tot}}} \tag{1}$$

where $N_{\rm on}$ is the number of counts in the current pixel, $N_{\rm off} = N_{\rm pix}\,\mu$ is the number of reference counts, $N_{\rm pix}$ is the number of pixels used to compute the local average ((2 `halfwidth2d` + 1)^2 - 1, if none of the pixels in the window has been rejected already), $N_{\rm tot} = N_{\rm on} + N_{\rm off}$ is the total number of counts in the window, and $\mu_{\rm tot} = N_{\rm tot}/(N_{\rm pix} + 1)$ is the average number of counts per pixel in the window.

This significance map is then used to locate the most promising candidate bad pixels. They are examined in turn, in decreasing order. The exact probability that the current excess is a statistical anomaly of a flat distribution is computed from the cumulative binomial law:

$$P(k \geq N_{\rm on}) \;=\; \sum_{k=N_{\rm on}}^{N_{\rm tot}} p_B(k, N_{\rm tot}, q) \;=\; I_q(N_{\rm on}, N_{\rm off} + 1) \tag{2}$$

where $q = 1/(N_{\rm pix} + 1)$ is the probability that a random count fall in the central pixel, and $I_x(a, b)$ is the incomplete beta function. This is significantly different from the probability estimated from Eq.(1) for small numbers (Eq.3 gives a larger probability). If that probability is smaller than `probathreshold`, the pixel is flagged as bright, the average is recomputed around the bad pixel ignoring it, and the loop goes on. The loop stops when the next largest excess is smaller than the significance corresponding to `probathreshold`.

3.3 Columns and rows

If `findbadsegments`=Y **embadpixfind** looks for bright rows and columns too. This is done by projecting the image along rows and columns, and applying the same algorithm as above on the resulting 1-D vectors, except the width of the 1-D window (2 `halfwidth1D` + 1) is normally chosen larger than that of the 2-D window, to improve the background determination.
If a bright row or column is found, it is analysed to look for bright segments within. If it is found that the rest of the row/column is compatible (to 10% probability) with the neighbouring rows/columns, then only the bright segments are declared as bad. The minimum length of the bright segments is set such that one expects about 1 count in that length in the normal (not bright) parts.

3.4 Intrinsic dispersion

Sometimes the distribution of the number of counts in the window around the tested pixel does not follow the Poisson law at all, but is much broader. This is particularly true for columns at low energy, because of charge transfer efficiency variations from one column to the next. This effect is of course more obvious for large count rates, in particular within bright sources.

To avoid wrongly detecting columns as bad (either dark or bright), the observed dispersion in the distribution is used to compute the significance of an excess assuming a Gaussian distribution. What is actually measured is the average absolute deviation (this is more robust than the root mean square when a few other bright pixels are present) divided by 0.8 (to recover the standard deviation when the distribution is normal). The true significance is taken to be the smallest of the Gaussian estimate and the Poisson one (from Eq.1). The cost of this security is to detect less easily groups of bad columns/rows, because the observed dispersion is large even when the tested column/row is discounted.

In addition, the `minratio` parameter avoids detecting bright pixels, rows or columns with too small contrast on observations with high statistics.

3.5 Dark features

If `finddead`=Y, **embadpixfind** looks for too dark pixels, rows and columns too. The Li and Ma formula (Eq.1) is not used for dark pixels. The Gaussian significance is computed in the same way (but on the negative side) and the cumulative binomial probability is computed from:

$$P(k \leq N_{\mathrm{on}}) \;=\; \sum_{k=0}^{N_{\mathrm{on}}} p_B(k, N_{\mathrm{tot}}, q) \;=\; I_{1-q}(N_{\mathrm{off}}, N_{\mathrm{on}}+1) \tag{3}$$

The statistics in a single observation is usually not enough to find any dark pixel, but dark rows or columns may be found. The `maxratio` parameter avoids detecting 'grey' pixels, rows or columns on observations with high statistics.

Whatever `findbright` and `finddead`, dark and bright columns and rows are always searched for together in order of decreasing significance (either positive or negative). This avoids finding spurious bright columns/rows next to very dark ones, and vice-versa.

If one of `findbright` or `finddead` is set to False, the corresponding bad pixels/columns/rows are not written to the output file.

3.6 Iteration

Each step in the detection process may affect the other steps. For example, detecting a bright row or column may facilitate detecting a moderately bright pixel next to it. For that reason, the whole process is iterated until nothing new is detected, or until the maximum number of iterations (set by the `niter` parameter) is reached.

3.7 Calibration access

If `usecal`= Y, the uplinked bright pixels and known dead pixels (within the current window) are read from the CAL and those pixels are ignored in computing the local median and the threshold.
Optionally (`includedeadpixels` parameter) one may include the dead pixels from the CAL in the output list. It is also possible (`ignoreccfbright` parameter) to ignore the bright pixels declared in the CCF and redetect them from the data. This does not apply to uplinked and dead pixels which appear dark in the data and cannot be detected in a single observation. If **embadpixfind** was called with `ignoreccfbright`=Y `includedeadpixels`=Y, **badpix** may be called with `getnewbadpix`=Y `getotherbadpix`=N to keep only the bright pixels active in the current exposure while preserving the information about dead pixels.

With `usecal`=N, the **embadpixfind** algorithm is not XMM specific at all and works on any image where the normal structure size is larger than 5 pixels.

3.8 Incremental search

The default mode is to ignore the bad pixels file on input. If `incremental`=Y, **embadpixfind** reads the bad pixels file on input, and ignores the pixels mentioned there in the search. On output the bad pixels file contains both the original bad pixels and the newly found ones.

4 Parameters

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

Parameter	Mand	Type	Default	Constraints

evimageset	yes	dataset	’ ’	none

Input image file (from emeventsproj)

badpixset	no	dataset	badpix.out	none

Output bad pixels file

incremental	no	boolean	no	yes/no

Add newly found bad pixels to the bad pixels file contents

probathreshold	no	real	1.10^{-6}	$> 0, < 1.10^{-3}$

False detection probability per pixel

halfwidth2d	no	integer	2	> 0

Half width for 2D searches (images)

findbadsegments	no	boolean	yes	yes/no

Look for bad segments of rows or columns as well

halfwidth1d	no	integer	3	> 0

Half width for 1D searches (columns/rows)

findbright	no	boolean	yes	yes/no

Look for too bright pixels, rows and columns

minratio	no	real	1.5	> 1

Minimum ratio to neighbours for bright features (when `findbright`=Y)

finddead	no	boolean	yes	yes/no

Look for too dark pixels, rows and columns

maxratio	no	real	0.5	$> 0, < 1$

Maximum ratio to neighbours for dark features (when `finddead`=Y)

niter	no	integer	10	> 0

Maximum number of iterations of the full detection process

usecal	no	boolean	yes	yes/no

Get uplinked and dead pixels from the CAL

includedeadpixels	no	boolean	no	yes/no

Include dead pixels from CAL in output list

ignoreccfbright	no	boolean	no	yes/no

Ignore the bright pixels declared in the CAL (except uplinked)

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.

getParamValues03 *(error)*
keyword incompatibility between image and bad pixels files (`incremental`=Y)

getCalBadpix10 *(warning)*
bright pixel wrongly declared as uplinked in the CCF. That pixel is treated as not uplinked.
corrective action: inform SOC (this is a CCF error)

6 Input Files

1. Projected image file (from emeventsproj) as Integer*4 array in PRIMARY. If they exist, the window keywords `WINDOWX0`, `WINDOWDX`, `WINDOWY0` and `WINDOWDY` are read to avoid underestimating the median and average at the borders in window mode.
2. File with `BADPIX` extension (if `incremental`=Y) output of **badpixfind**, **embadpixfind** or **badpix**.

7 Output Files

1. Bad pixels file (for **badpix**) as `BADPIX` extension with `RAWX`, `RAWY`, `TYPE`, `YEXTENT` and `BADFLAG` Integer*2 columns.

8 Algorithm

```
Read the parameters

Define goodPixel array, set to True

if incremental then
   Read the bad pixels already in file
```

```
      goodPixel(bad pixels) = False
   endif

   if usecal then
      Read the bad pixels in CAL
      goodPixel(uplinked and dead pixels) = False
      if not ignoreccfbright then goodPixel(bright pixels) = False
      if includedeadpixels then Write dead pixels to output list
   endif

   Read map = projected image

   Iterate
      call findAllBad(map, goodPixel, bad)
   until nothing new is found or niter is reached

   Write bad pixels file


subroutine findAllBad(map, goodPixel, bad)

! Get local average around each pixel and estimated significance
    call avMedFilter(map, goodPixel, medsmooth, badtest)
! Look for bright pixels
    call findBadPix(map, goodPixel, medsmooth, badtest, False, bad)
    if findbadsegments then
! Look for bright/dark columns
       profil = sum(map,2)
       call findBad1D(profil, bad)
! Look for bright/dark rows
       profil = sum(map,1)
       call findBad1D(profil, bad)
       Update goodPixel and medsmooth/badtest around bad columns/rows
    endif
! Look for dark pixels
    call findBadPix(map, goodPixel, medsmooth, badtest, True, bad)

end subroutine findAllBad

subroutine avMedFilter(map, goodPixel, medsmooth, badtest)

    Loop over current pixel
       Consider all good pixels in window around current pixel
       Extract average (or median+1 if smaller) into medsmooth
       Estimate dispersion from average absolute deviation divided by 0.8
       Estimate significance S1 of excess using Gaussian law
       If larger than 3, estimate significance S2 of excess using Li and Ma
       badtest = min(S1,S2)
    endloop

end subroutine avMedFilter

subroutine findBadPix(map, goodPixel, medsmooth, badtest, negative, bad)

    Loop over current bad pixel
```

```
       Find maximum in badtest (minimum if negative) down to probathreshold
       Get probability of excess using binomial law
       if probability < probathreshold  then
          add bad pixel to list
          Update goodPixel and  medsmooth/badtest around maximum
       endif
    endloop

end subroutine findBadPix

subroutine findBad1D(profil, psf1D, bad)

    Find bad columns in profil (same as findBadPix)
    Loop over bad columns
       Get expected distribution along column from its neighbours
       width = 1/(expected count rate per pixel)
       Get running integral over bins of width pixels
       While total(rest of column) larger (lower if negative) than expected
          Find maximum integral (minimum if negative)
          Remove segment of width pixels around it
       endwhile
    endloop

end subroutine findBad1D
```

9 Comments

- The algorithm to get the local average could be improved over using the local median. An algorithm similar to that used in computing the offsets in **emdiag** should be considered.

References