XMM Extract

08 August 2012

Contents

Change log

15/07/2012 - Reflect updates to codes:

ERR_B output added to EPIC_LOAD_LC

T_CLIP_START/END added to EPIC_LOAD_LC; removed T_CLIP from LOAD_EVENTS

T0 changed to match pn GTI (where pn is used) in MAKE_EPIC_LC

Now use one ‘scaling.txt’ index file, not three.

Replaced QUIET keyword with CHATTER to control on-screen output

18/07/2012 – Updates & corrections throughout, added “flux-flux” example

19/07/2012 – Added figures 1-6

25/07/2012 – Reflect updates to code

01/08/2012 – Added discussion of GTI_CHECK and data quality.

Description

A set of IDL tools for the extraction of time series ‘products’ from *XMM-Newton* EPIC data.

Main features

These tools are designed to simplify, speed-up and allow automation of time series extraction from *XMM-Newton* data. It provides an alternative to using the SAS or FTOOLS for extracting time series from each camera, separately for source and background and then subtracting them, and repeating this for many observations (e.g. where a source has been observed over several revolutions). The key features are:

1. extract binned time series (light curves) from *XMM-Newton* EPIC event file(s)
2. User specified energy (`PI`) range(s) and `PATTERN` ranges
3. Automatically subtract scaled background
4. Automatically perform GTI correction for drop-outs etc.
5. combine pn, M1, M2 in any order if requested
6. provide errors on output (including exposure and background corrections, etc.)
7. Provide background data so that end-user can perform their own check for background flares etc.
8. merge multiple observations into a single data array, with corresponding index array describing the output structure

With one command you can extract from *XMM-Newton* EPIC data a fully background-subtracted, GTI drop-out corrected, binned time series with user-defined time binning over multiple energy channels, and over multiple observations. The end result is packaged as a single array that can be used by a variety of other ‘down-stream’ analysis routines to produce ‘scientific product’ such as power spectra, cross spectra, rms spectra, flux-flux plots, etc.

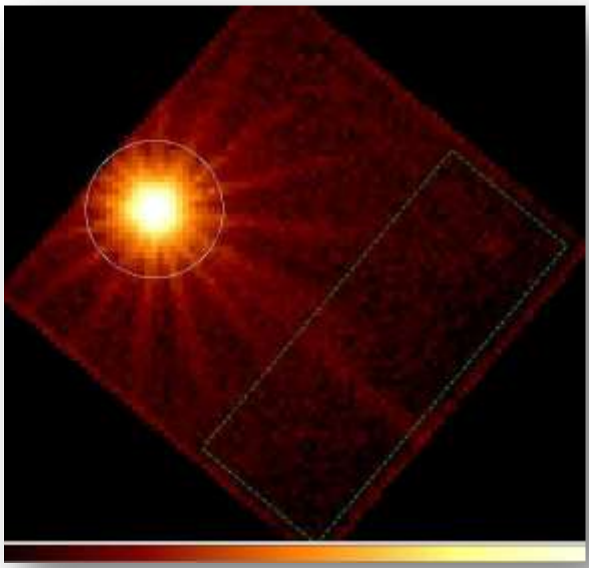
Preparing the data

Event files

The IDL extract routines assume you have performed some basic processing of the event files (e.g. from ODFs), and that you have filtered the data into separate source and background event files per camera, per observation.

To do this first produce pipeline processed event files using the standard SAS processing tools, e.g. `epchain/epproc, emchain/emproc`. You should have a single event file per camera (pn, M1, M2), per observation (assuming all three EPIC cameras provided data). Then, using e.g. `xmmselect` or directly with `evselect`, apply your standard criteria to remove "bad" events[1], and then apply a region selection to produce separate event files for the source and background regions. For example, for the source one might use a 20-40 arcsec radius circle centred on the source, and a slightly off-target (non-overlapping) box for the background. Using each region in turn to produce one filtered event file for source events, and one for background events (per camera, per observation) reduces the sizes of the event files. The two event files that are left contain only the (normally) good source and background events.

Fig 1: example EPIC pn (small window mode) image showing source and background selection regions used to define the two event list files.



The user need not perform any additional filtering (e.g. for `PATTERN`, `PI`, or `TIME`) at this stage. When analysing data spanning multiple observations (e.g. repeat exposures of the same target) it is best to reproduce the same source and background regions across observations.

The output event files should be named using the following system (this is important):

```
<obsid>_<inst>_<src/bkg>.ds
```

[1] For the pn this might be `(FLAG == 0)` and for the MOS `(#XMMEA_EM)`.

This name will be unique for each observation, camera, source/background event file.

- `<obsid>` is a unique name per observation, .e.g. 'rev1721' or 'mysource' If analysing multiple observations these names should be unique to each observation
- `<inst>` should be either 'pn', 'M1', 'M2' depending on the camera (case sensitive);
- `<src/bkg>` should be 'src' or 'bkg' for source and background.

The event files for data taken during revolution 1721 would be named

```
rev1721_pn_src.ds
rev1721_pn_bkg.ds
rev1721_M1_src.ds
rev1721_M1_bkg.ds
rev1721_M2_src.ds
rev1721_M2_bkg.ds
```

Typically one will have six such event files per observation (3 cameras, each with source and background). These filtered event files for all observations need to be placed in a single directory (this is one reason why each observation needs a unique `<obsid>` identifier).

The index file

In addition to this the user must create an ASCII (plain text) file called `scaling.txt` and place it in the same directory as the event files. This specifies the observation names `<obsid>` for each camera the source and background region BACKSCAL values. For example:

```
obs         pn_src   pn_bkg   M1_src   M1_bkg  M2_src   M2_bkg
rev1721   1975222  7452800  1532800   832000  1542400   816000
rev1722   2001628  7436800  1532800   820800  1503976   832000
```

There is a single header row, then one row per observation. Each row contains seven columns listing for each observation: the `<obsid>`, the source and the background BACKSCAL values (as obtained from e.g. spectral extraction). It is this file that is used to define the names of the event files (column 1) and the source/background scaling factor (columns 2-7). If you are analysing 6 observations of the same source there will be 7 rows in the file, one header then 6 rows for the observations.

These BACKSCAL values are not found in the filtered event files, but are produced by running the SAS meta-task `backscale` on spectral products. The simplest way to obtain these is to extract source and background spectral products using the appropriate selection regions. Then you can generate an ARF file using the SAS task `arfgen` and set the `setbackscale=true` parameter. Or directly run `backscale` on the spectral products, then extract the BACKSCAL values from the updated FITS header information of the spectral files using e.g. the FTOOL `fkeyprint`:

```
fkeyprint infile=<spec_file> keynam="BACKSCAL"
                outfile=tmp clobber=yes
```

Using `backscale` like this is more accurate than simply using the geometrical area (e.g. in det pixels) of the extraction regions because it accounts for hot/dead/edge pixels of the instrument and specific observation.

BACKSCAL = (geometrical area) - (CCD gaps) – (bad pixels)

(NB: I am not aware of any other simple way to obtain the BACKSCAL values.)

If there are no useful events from one camera (e.g. because it was not working, in an unsuitable mode etc.) then set the BACKSCAL values to 0 in the `scaling.txt` file.

A first run

Assuming you have copied the necessary IDL files to a suitable directory, and have IDL installed, you can produce a simple light curve like so:

```
IDL> rootpath = '/data/49/sav2/xmm/ngc4051/events/'
IDL> x = EPIC_LOAD_LC(ROOTPATH=rootpath)
```

The main function is called `EPIC_LOAD_LC`, and the main input required for this is a keyword called `ROOTPATH` that must be set equal to the directory in which your data are located (data event files and the `scaling.txt` file)[2]. If you do not explicitly specify the `ROOTPATH` keyword it will default to the current working directory. If no index file (`scaling.txt`) can be found you get get an error message and the routine will stop.

As it runs it will produce a short table summarising the contents of each observation

```
-- EPIC_LOAD_LC processing 5 observations using PATTERNs: 0- 4
-- Name  Inst.   Duration Start time    Src rate Bkg rate Intp frac Intp no Scale
rev1721 pn M1 M2 45094.51 357733776.868 13.23628 0.46272 0.00065  18 0.26503 1.84231   1.89020
rev1722 pn M1 M2 44103.26 357906180.483 20.68543 0.84560 0.00063  17 0.26915 1.86745   1.80766
rev1724 pn M1 M2 27618.25 358250922.905 25.15384 0.98329 0.01479 251 0.26848 1.89769   1.85036
```

The first line reminds the user that only event PATTERNS 0-4 are being used. Then each row of the table gives the following information:

(1) observation name (i.e. `<obsid>`)

(2) a list of which cameras are being used (e.g. ‘pn’ for pn only)

(3) the duration (in sec) of the exposure

(4) the start time (in spacecraft seconds)

(5-6) average (net) source and scaled background count rates

(7-8) the fraction and number of time bins that required filling by interpolation (e.g. missing data)

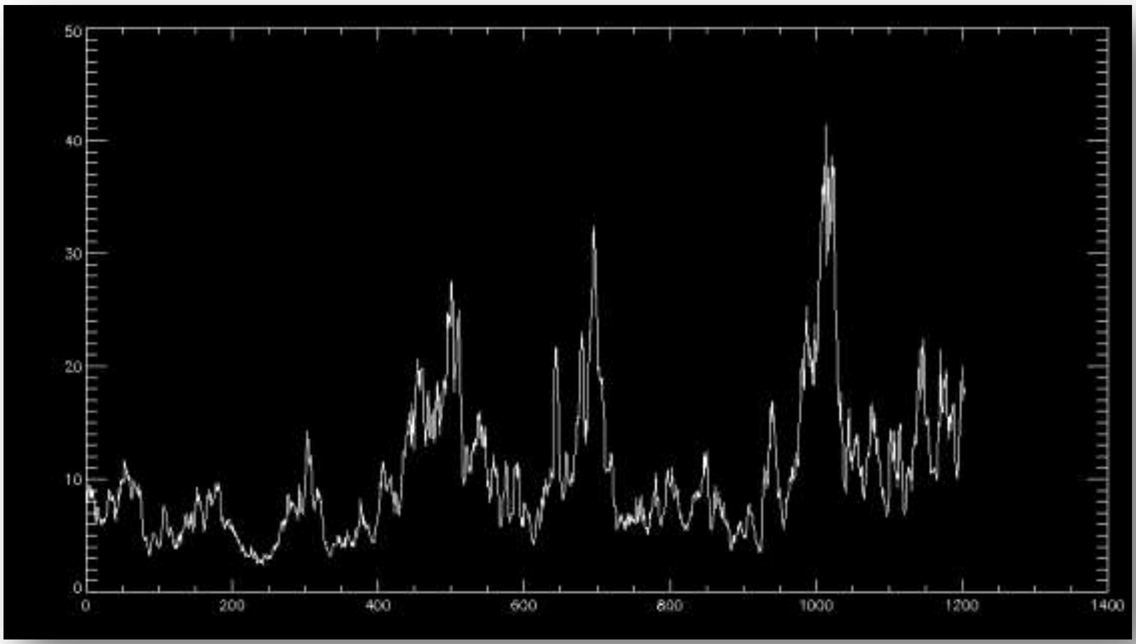
(9-11) the source/background scaling factors (0.0 if camera not used).

The main output is an array of dimensions `[1, N_t]` called `x`, where `N_t` is the number of time bins in the data. By default the routine will extract event energies 0.2-10 keV (`PI in [200, 10000]`), single and double event patterns (`PATTERN in [0, 4]`) and time bins of width `dt=100`s. Also, by default data from just the pn event files are used. Note that the output array `x` is a two-dimensional array, not a one dimensional vector, this is to allow for multiple simultaneous time series (in different energy bands, see below).

Check the output, e.g.

```
IDL> HELP, x
```

[2] IDL is not case-sensitive, but to aid reading of code I use upper case characters to denote IDL commands and functions, and their keywords. The results should be the same if you use lower case characters throughout when using IDL.



```
IDL> PLOT, x[0,*]
```

Note that IDL starts counting from 0, so the first element in an array is always 0, and a vector of length N runs from elements 0,…,N-1. The plot might look something like this

Fig 2: an example time series.

More control over what it does

You can specify more input parameters:

```
IDL> rootpath = '/data/49/sav2/xmm/ngc4051/events/'
IDL> x = EPIC_LOAD_LC(ROOTPATH=rootpath, N_CHAN=5, $
                      DT=25.0D, INST_MASK=[1,0,0])
```

The optional keywords provide more control of the output:

```
DT            - (float) time bin size (seconds)
ROOTPATH      - (string) path to the event and index files
N_CHAN        - (integer) Number of (log spaced) frequency bands
OBS_LIST      - (vector) list of observations to process
INST_MASK     - (vector) use/ignore pn, M1, M2 cameras
PATT_LIM      - (array) set lower and upper PATTERN ranges
PI_LIST       - (array) list of PI energy intervals to use
CHATTER       - (integer) more or less feedback to screen?
T_CLIP_START  - (float) length of time (sec) to 'clip' from the
T_CLIP_END    - (float)  …start/end of (the first) GTI
MAX_GAP       - (float) maximum acceptable period of bad data
MINF          - (float) minimum fraction exposure to allow
```

On a Windows platform the ROOTPATH might look more like this

```
IDL> rootpath = "C:\Users\Simon\Documents\work\ngc4051_data\events\"
```

The keyword `DT` sets the time bin size, and defaults to 100s, this should be defined in double precision using a 'D' after the value, e.g. `DT=25.0D`.

The keyword `INST_MASK` should be a three-element vector indicating which cameras to combine, where a value of 1 means include and 0 means exclude the camera, in the order [pn, M1, M2]. So `INST_MASK = [1,0,0]` means include pn only (the default value). A value `INST_MASK = [1,1,1]` means include pn+M1+M2.

You can extract from a single energy range (e.g. the broad 0.2-10 keV) range, or specify a narrow energy range, or multiple energy ranges. There are two ways to specify the choice of energy ranges. The first is using `N_CHAN`. Set this equal to a positive integer (1,2,3,…) to obtain `N_CHAN` energy bands 'logarithmically' spaced over 0.2-10 keV. By default N_CHAN=1 and you get the full 0.2-10 keV band. Setting `N_CHAN=5` will provide an array of dimension `[5, N_t]` that contains 5 exactly simultaneous time series corresponding to `PI` channels

```
    200-434, 435-954, 955-2089, 2090-4569, 4570-10000
```

These are very nearly equally sized on a log[energy] scale[3]. The logarithmically spaced energy channels are actually generated by a separate function `ENERGY_BANDS.PRO` that can be called independently to check its output

```
 IDL> PRINT, ENERGY_BANDS(5)
```

The other way to specify the choice of energy bands is to use the `PI_LIST` keyword to provide an array listing the lower/upper limits of the PI ranges you require, e.g. for 0.2-0.5, 1.0-3.0 and 5-10 keV use:

```
 IDL> pi_list = [[200,500], [1000,3000], [5000,10000]]
```

then run EPIC_LOAD_LC specifying the `PI_LIST`

```
IDL> x = EPIC_LOAD_LC(ROOTPATH=rootpath, DT=25.0D, $
                       INST_MASK=[1,0,0], PI_LIST=pi_list, $

                       CHAN_LIST=chan_list)
```

The output variable `CHAN_LIST` will return the exact PI ranges for you to check. (The `PI_LIST` keyword takes precedence over the `N_CHAN` keyword.)

```
 IDL> HELP, x
 IDL> PRINT, chan_list
```

[3] Note that the lower boundary of each band has been rounded to a multiple of 5, since the PI information in EPIC pn event files is given in multiples of 5.

This will produce 3-band output over the energies 0.2-0.5, 1-3 and 5-10 keV. Note the output `x` is always a two dimensional array of dimension `[N_CHAN, N_t]` (not a one dimensional vector) even if you extract using only one energy band (`N_CHAN=1`). Examine the first few time bins to see the data:

```
IDL> PRINT, x[*, 0:10]
```

If the index file (`scaling.txt`) lists many observations but you only wish to examine a subset of these you can use the `OBS_LIST` keyword. For example, to process only the observations listed as 2nd and 3rd in the index file set `OBS_LIST=[2,3]`. By default all observations listed in the index file will be processed. (Note: the first observation is number 1, not 0.)

You can request more on-screen information on each event file by setting `CHATTER=1` or 2. This will provide more detailed information on each event file used in the processing. You may request no event file information on screen by setting `CHATTER=-1`

Getting more useful information out

The routine will also provide more output data that may be useful for later data analysis. For example:

```
IDL> x = EPIC_LOAD_LC(ROOTPATH=rootpath, N_CHAN=2, DT=25.0D, $
         DATA_T=data_t, SEG_LIST=seg_list, T_START=t_start, $
         DATA_B=data_b, CHAN_LIST=chan_list, OBS_name=obs_name, $
         INST_MASK=[1,0,0], OBS_LIST=[1,2,3], ERROR=dx)
IDL> PLOT, data_t, x[0,*]
IDL> OPLOT, data_t, x[1,*], COLOR=100
```

The outputs include:

```
DATA_T       - (vector) start time of each output bin (sec)
ERROR        - (array) error on each output count rate
SEG_LIST     - (array) [N_OBS, 2] array listing first and last
               element numbers of the combined time series
T_START      - (vector) list of the start times of each observation
DATA_B       - (array) background count rate in each
               time/energy bin [N_chan, N_t array]
0ERR_B       - (array) errors on the background rate
CHAN_LIST    - (array) the list of PI channel ranges
OBS_NAME     - (vector) list of the observation names
GAP_LIST     - (array) [N_gap, 2] array listing gap indices
```

You can then plot the data with the correct (relative) times, e.g. for the 2nd energy band:

```
    IDL> PLOT, data_t, x[1,*]
```

and over-plot the simultaneous background time series:

```
IDL> OPLOT, data_t, data_b[1,*], COLOR=100
```

The output array `SEG_LIST` is used to keep track of which data in the concatenated array come from which observations. For example, if combining three observations, it might look like this:

```
IDL> PRINT, seg_list

       0          449          889
     448          888         1198
```

Data from the first observation is stored in elements `[*, 0:448]` of the output time series arrays; the second observation is stored in `[*, 449:888]` etc. Data from many observations can be concatenated into a single array, and `SEG_LIST` used to separate them if needed.

The bin times (e.g. `DATA_T`) have been zeroed such 0 sec corresponds to the observation start time. The start time of each observation in satellite seconds is stored in the double precision array `T_START`. This corresponds to the start of the first 'good time interval' as listed in the pn event file GTI table. (If only MOS data are used then the MOS GTI tables are used.) The reason this is subtracted is that the start time is usually a large number (e.g. 357733776.95931375) and one often loses precision if the times are large but the time bins are small (e.g. time steps of ~1s or less). The simplest way to recover the original times is by adding back the start time of the first observation:

```
IDL> original_times = data_t + t_start[0]

IDL> PRINT, original_times[0:10], FORMAT='(F20.10)'
```

The `format` keyword is needed to ensure the numbers are printed with enough digits, since the time can be a large number, usually ~10^8, and the step size can be small.

The output array may contain periods of "repaired" data, i.e. with reduced exposure. If any of these is longer than a user-specified length (`MAX_GAP`), within each separate observation, then the start and stop indices are recorded in the `GAP_LIST` output array. By default `MAX_GAP` = 100 (seconds).

```
IDL> PRINT, gap_list

  37002        37261       110254
  37174        37296       110306
```

This indicates that the interval between time bins 37002 and 37174 is "bad data". Patches of bad data shorter than `MAX_GAP` are ignored (not listed). The `GAP_LIST` array can be used later to ignore patches of bad data if necessary.

More on how it works

Structure of the routines

The following gives a very rough outline of the routines involved in XMM Extract and their dependencies.

`epic_load_lc.pro`	- top-level routine, combines data over multiple observations
`energy_bands.pro`	- compute N_CHAN log-spaced energy bands
`read_table.pro`	- read ASCII table listing event list filenames
`make_epic_lc.pro`	- per observation, combine data from cameras, subtract background, correct GTI losses
`load_events.pro`	- per event file (i.e. each observation, camera, source/background) read the data, keywords, and GTI table from FITS file
`gti_fix.pro`	- correct for 'missing' exposure using GTI table
`gti_check.pro`	- Check for patches of 'bad' data in a multi-observation time series

Each of these can be run as a stand-alone function to perform particular tasks. E.g. `LOAD_EVENTS` can be used to load the contents on single EPIC event file and header information. But the main routine `EPIC_LOAD_LC` will manage calls to these other routines for you.

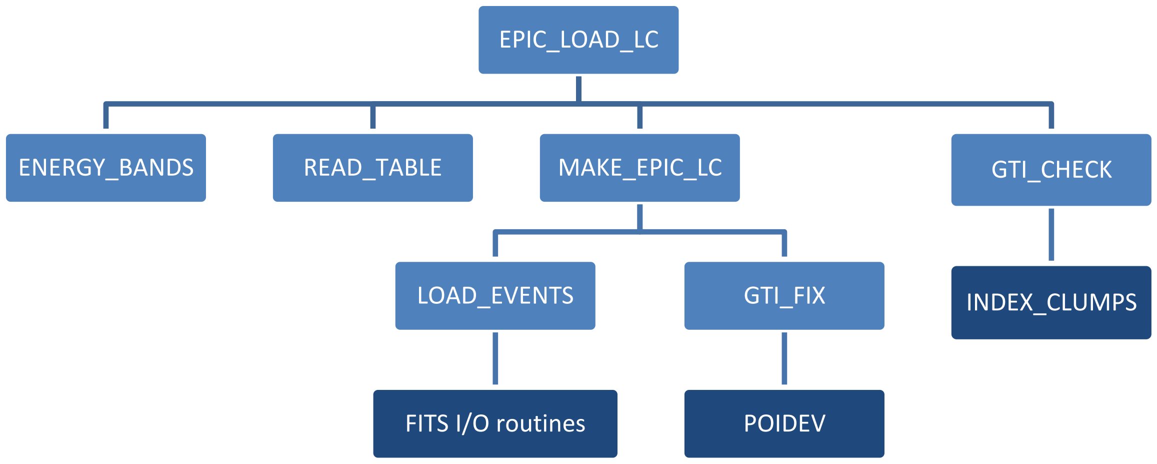
Ancillary tasks that are useful to further use of the output are

`epic_segment.pro`	- used to tidy-up the observations into multiples of a fixed segment size
`cross_spectrum.pro`	- cross-spectral analysis
`regroup.pro`	- needed by `cross_spectrum.pro`
`write_table.pro`	- write ASCII tables
`xspec_out.pro`	- export files for conversion to XSPEC format
`excess_variance.pro`	- compute multi-band excess variances

These routines are all available through www.star.le.ac.uk/~sav2/idl.html

At the start of each IDL procedure/function (plain text inside each of the `*.pro` files) is more detailed information about the inputs, outputs and workings of the routines.

The routine dependencies can be illustrated with a flow diagram



The low-level routine that does the hard work is `load_events.pro`, this uses the FITS I/O routines from the IDL Astronomy Library (http://idlastro.gsfc.nasa.gov/fitsio.html) to access the FITS event files. These routines are shown in dark blue (above) and also need to be installed in your IDL library or working directory:

```
readfits.pro
tbget.pro
sxpar.pro
fxposit.pro
poidev.pro
fits_info.pro
index_clumps.pro
```

These are routines that IDL Extract explicitly calls. The FITS I/O routines in turn require subroutines available from the IDL Astronomy Library.

```
fxmove.pro
fxpar.pro
gettok.pro
mrd_hread.pro
mrd_skip.pro
tbinfo.pro
valid_num.pro
strn.pro
tag_exist.pro
```

You will need to copy all these IDL procedures/functions to a suitable directory. This can be either your working directory or, better yet, your local IDL procedure library (e.g. your `~/idl/` directory).

Event arithmetic: background subtraction and camera addition

For each camera the event file is loaded and the time and PI channel of each event stored. Then, for each set of PI values we select the events within this range and use the `HISTOGRAM` function to produce a binned time series (this step is performed by `MAKE_EPIC_LC`). The time series are stored as double precision arrays. We repeat this for the background event file, and subtract the scaled background (in counts/bin) from the source to get the net (background subtracted) counts/bin.

The time series are generated over exactly identical time bins. This is achieved in `MAKE_EPIC_LC` by recording the start and stop times of the first exposure (usually the pn), and computing the time series as histograms of the count arrival times within this time range. This ensures the bin times from source and background and the different cameras are synchronised properly before combining. It is usually the case that the three cameras start and end at slightly different times, therefore once all the time series for a given observation have been produced the first and last time bins are removed where any (requested) camera is not fully exposed.

The scaling factor is supplied (to `MAKE_EPIC_LC` via the keyword `SCALE`, obtained from the index files). If any of these are set to zero then we ignore that camera. These scaling factors are the ratios of source/background good area (e.g. taken from the `BACKSCAL` keywords of suitable FITS files).

For each energy band (range of PI values) the net counts/bin is calculated as follows:

```
net = sum_{i=0}^{2} SRC[i] - SCALE[i] * BKG[i]
```

where `i=0,1,2` represent the three EPIC cameras (pn, M1, M2). The error on the count/bin comes from:

```
err^2 = sum_{i=0}^{2} SRC[i] + SCALE[i]^2 * BKG[i]
```

Since the variance on the sum of Poisson distributed variables is just the sum of their expectations. (The `SCALE^2` weighting comes from the scaling of the background regions.) And what's returned is the count rate `net/dt` and error `sqrt(err2)/dt`.

Event pattern selection

The `PATT_LIM` keyword is passed along to the routine `LOAD_EVENTS`. The `PATT_LIM` input can be used to specify the lower and upper event pattern ranges that are to be collected. Single pixel events are PATTERN=0, double pixel events are 1-4, triples are 5-8 and quadruples are 9-12. By default `EPIC_LOAD_LC` will use singles and doubles only (PATTERNS in [0,4]) specified by `PATT_LIM=[0,4].` But to use e.g. singles-quads set `PATT_LIM=[0,12]`, which uses only events for which PATTERN is in the range 0-12. To use only single events set `PATT_LIM=[0,0]`.

Exposure loss correction

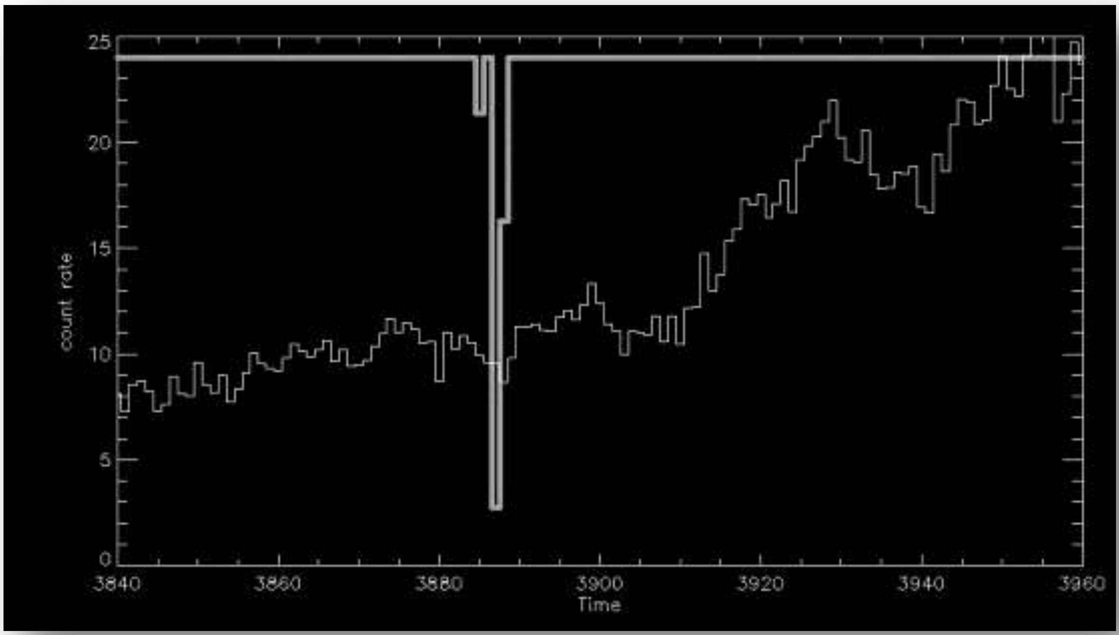
The output time series have also been corrected for missing/incomplete time bins. This is a GTI (good time interval) correction and is performed by the function `GTI_FIX`. The event files carry with them a GTI table that is read by the `LOAD_EVENTS` routine. The GTI table lists the start and stop times (in satellite seconds) of each good time interval, outside of these intervals is 'bad time' where the cameras were not observing or data was lost. The GTI information is used to construct a 'bad time interval' (BTI) list, and any time bins that lie fully or partially in a BTI will require some correction. For each time bin the fraction of good exposure is recorded in the array FRAC.

There are two ways to perform the GTI correction for each time bin:

(i) if `FRAC` is <1.0 but above some pre-defined threshold (`MINF <= FRAC < 1.0`) then the counts/bin are rescaled by a factor `1/FRAC`.

(ii) if `FRAC` is below the pre-defined threshold (`MINF > FRAC`) then the data are replaced by linear interpolation between the nearest two good data points on each side

By default `MINF` is set to 0.3, i.e. if a given time bin contains <30% good exposure then replace the data by linear interpolation between the nearest good data. Method (i) is applied first, to correct for partially missing data, then method (ii) is applied to the remainder of the data.



In order that the corrected time series are not too smooth over gaps where method (ii) was used the interpolated data are replaced by a random value from a Poisson distribution[4] whose mean is set equal to the interpolated count/bin value. This also means the error calculation remains valid even for interpolated data, and the number of counts remains an integer.

This GTI correction is applied separately to each event file (each camera, source/background) before background subtraction, so that even for interpolated (bad) time bins the Poisson error calculation is valid.

Fig 3: a section of an example time series (thin histogram) and the fractional exposure per bin (thick histogram, rescaled to make use of the vertical axis range). There are a few bins with <100% exposure that have been "exposure corrected". The result is an unbroken time series.

There is one more step applied to ensure even exposure through the data. It is often the case that the pn 'good' exposure finishes before the final stop time listed in the GTI table of the corresponding event file (this may be because the camera shutter closes prematurely). This is difficult to predict based on the event information. As a precautionary measure all pn exposures have the first `T_CLIP_START` seconds and final `T_CLIP_END` seconds removed. The keywords can be set when calling `LOAD_EPIC_LC`. By default `T_CLIP_START = 10` and `T_CLIP_STOP = 100` (which is slightly conservative but has proven successful in trials. This step is applied by `MAKE_EPIC_LC`. The loss of 100s from an exposure (some faction of which may be bad or unexposed) is a small price to pay to ensure data quality.

[4] This means that two apparently identical runs of `LOAD_EPIC_LC` will result in very slightly different count rates for a few bins, those that required interpolation. The keyword SEED is available to reset the random number generator for debugging purposes.

Further data quality checks

By default the data array returned by EPIC_LOAD_LC includes all the data from each exposure, down to the limits set by the `T_CLIP_START/END` keywords. In other words, there is very little additional quality filtering applied.

However, one can use the other outputs to check data quality and apply further filtering. For example, the FRAC output array gives the fractional exposure for each bin, which can be used to identify periods of low exposure (which will have been filled-in by interpolation). And the background time series can be used to find periods of high background.

The last stage of `EPIC_LOAD_LC` does a very basic quality check. It passes the `FRAC` (fractional exposure) array to the `GTI_CHECK` function. This scans for continuous periods of low fractional exposure within each observation, and returns a list of the start and stop times (array indices) of the 'bad time intervals' (BTIs). These respect the observation breaks, so a BTI never extends between observations. By default it only records BTIs that are continuous and longer than 100s, any shorter BTIs are ignored. And by default it searches for periods when FRAC is continuously lower than `MINF`, i.e. the value below which data are replaced by interpolation from surrounding good data.

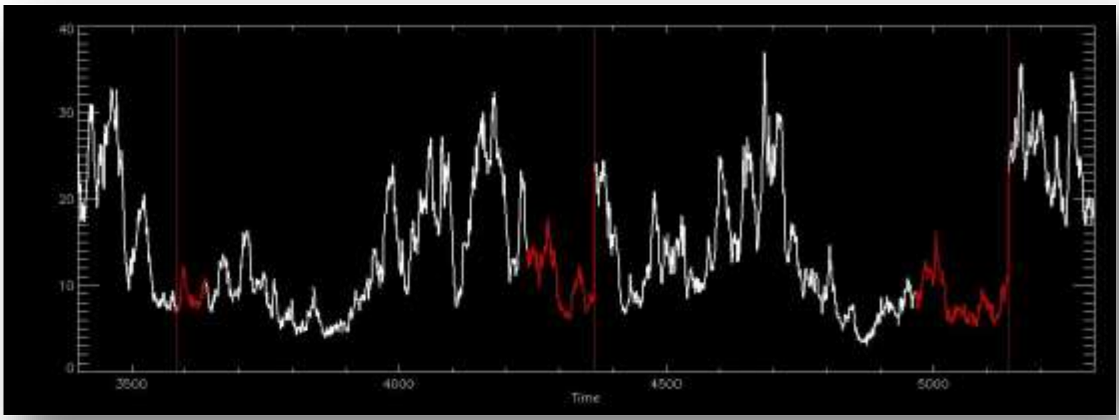


Fig 4: example of 'segmenting' using `EPIC_SEGMENT`. This shows a subset of a long time series produced by concatenating several shorter time series from distinct observations. Each time series is then broken into continuous segments of 200 points. Segments do not cross between observations, and they do not include extended periods of bad data. The red data show the intervals that were ignored in order to leave the 'good' (white) data which is in blocks that are an integer multiple of 200 in length.

The output of `GTI_CHECK` is an array listing the BTIs, structured like the SEG_LIST array. This can be used in 'downstream' analysis to ensure periods of 'bad' data are treated properly or ignored.

Using the output in ‘downstream’ analysis routines

Flux-flux analysis

Here’s an example of how to perform some simple spectral variability analysis using flux-flux analysis. First, let’s produce time series in two bands, and we shall limit ourselves to just a single observation using the `OBS_LIST` keyword. We will use all three cameras (by setting the `INST_MASK` keyword), define the two energy ranges as 0.2-1 keV and 2-10 keV (using the `PI_LIST` keyword) and set the time bin size to 100s (using the `DT` keyword).

```
  ; define the parameters for data extraction
  obs_list = [8]
  pi_list = [[200, 1000],[2000, 10000]]
  rootpath = "C:\Users\Simon\Documents\work\ngc4051_data\events\"
  dt = 100.0D

  ; extract the 2-band time series (from pn+M1+M2)
  z = EPIC_LOAD_LC(PI_LIST=pi_list, ROOTPATH=rootpath, $
                   DT=dt, OBS_LIST=obs_list, CHATTER=0, $
                   INST_MASK=[1,1,1], ERROR=dz)
```

Notice here we also used the `CHATTER` keyword. This defines the amount of information printed to screen, by default this is zero. More information is printed if the value if 1 or 2 (maximum), and less is printed if this is set to -1. Now we have the two-band data stored in the `z` array, with errors in the `dz` array. The first time series (0.2-1 keV) is stored in the first ‘column’, z[0, *]; the second time series (2-10 keV) is stored in the second ‘column’, z[1, *]. We can make a simple plot of flux vs. flux:

```
; plot flux-flux data (with errors)
  PLOT, z[0,*], z[1, *], PSYM=1, $
        XTITLE="Flux (0.2-1 keV)", $
        YTITLE="Flux (2-10 keV)"
  PLOT_ERR, z[0,*], z[1,*], dz[1,*], DX1=dz[1,*]
```

The next step is to fit a linear model to these data, which can be done using the IDL function `LINFIT`. Note that this is a simple un-weighted least squares fit, i.e. assuming uniform errors on the data and minimising the residuals between model prediction and the hard band fluxes.

```
; fit with a linear model y = m*x + c
  result = LINFIT(z[0,*], z[1,*], SIGMA=sigma)
  PRINT, result, sigma

; overlay the model on the plot
  x_mod = [0, 100]
  y_mod = x_mod * result[1] + result[0]
  OPLOT, x_mod, y_mod, COLOR=140, THICK=5
```

The result is a plot with a fitted model, the parameters of which (gradient and intercept) are stored in the RESULTS array. It would be straightforward to generalise this procedure to multiple energy bands.

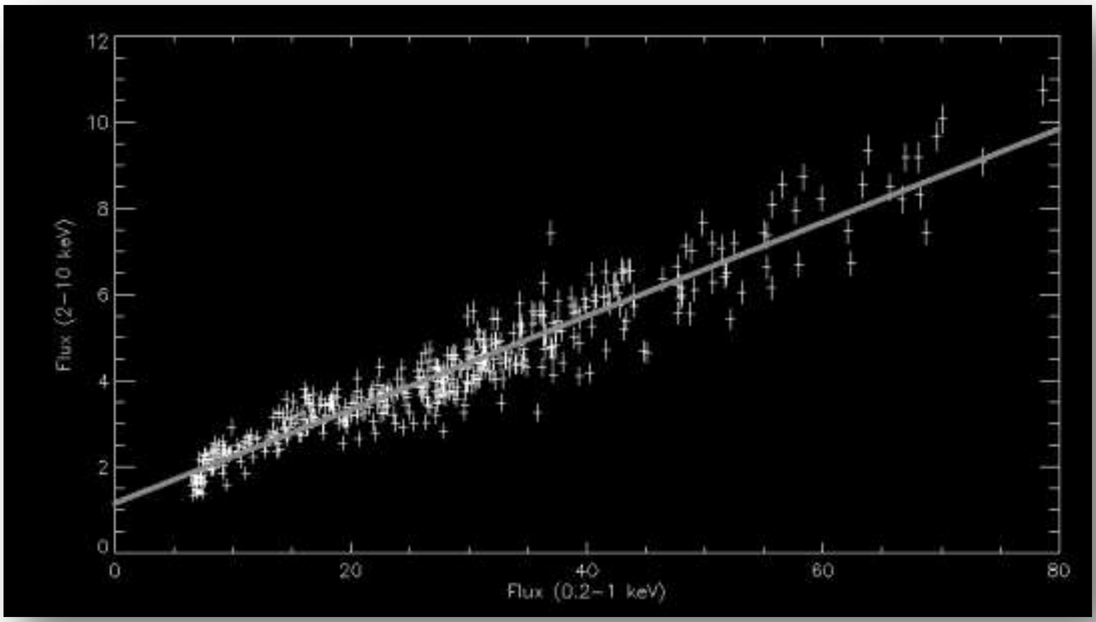


Fig 4: example flux-flux plot with fitted linear model

The rms spectrum

Here's a worked example of how to compute the mean and rms spectrum for a single observation

```
  rootpath = "C:\Users\Simon\Documents\work\ngc4051_data\events\"
  dt = 100.0D
  obs_list = [1]
  z = EPIC_LOAD_LC(ROOTPATH=rootpath, DT=dt, N_chan=25, $
                   ERROR=dz, OBS_LIST=obs_list, CHAN_LIST=energies)

; compute the excess variance for each energy band
  xs = EXCESS_VARIANCE(z, dz, DIMENSION=2, MEAN=meanx)

; convert PI channel to keV units, and mid (geometric) mid-bin point
  energies = energies / 1000.0
  energy = SQRT(energies[0,*] * energies[1,*])

; rms is sqrt of excess variance
  rms = SQRT(xs)

; plot the rms vs. energy with nice axes and labels
  PLOT, energy, rms, /XLOG, /YLOG, PSYM=1, $
        XRANGE=[0.16, 12.0], YRANGE=[3.0E-3, 1.0], $
        /XSTYLE, /YSTYLE, CHARSIZE=1.5, $
        XTITLE="Energy (keV)", YTITLE="rms and mean spectra"

; use the PLOT_HIST function to join using a histogram
  PLOT_HIST, rms, x0=energies[0,*], x1=energies[1,*], /NOPLOT

; overlay the mean spectrum
  PLOT_HIST, meanx, x0=energies[0,*], x1=energies[1,*], $
             color=150, /NOPLOT
```

In this example we set the input keywords of `LOAD_EPIC_LC` such that the output is an array containing 25 time series spanning energies 0.2-10 keV, roughly equally spaced in log(energy). Each light curve has time resolution of `DT=100` seconds, uses only pn data (the default setting for `INST_MASK`), and uses events from only the first observation listed in the index file. As output we record the array of time series, the corresponding error array, and the lower/upper PI values for the 25 energy bands (in the `CHAN_LIST` array).

We then compute the excess variance for each of the 25 time series, which can be done with a single call to the `EXCESS_VARIANCE` function, as long as we specify with `DIMENSION` of the array we wish to compute variances over (i.e. which is the time dimension). We then take the square root to give the rms (in ct/s units), and plot this using the `PLOT_HIST` procedure, along with the mean spectrum (simply the mean count rate for each of the 25 time series).

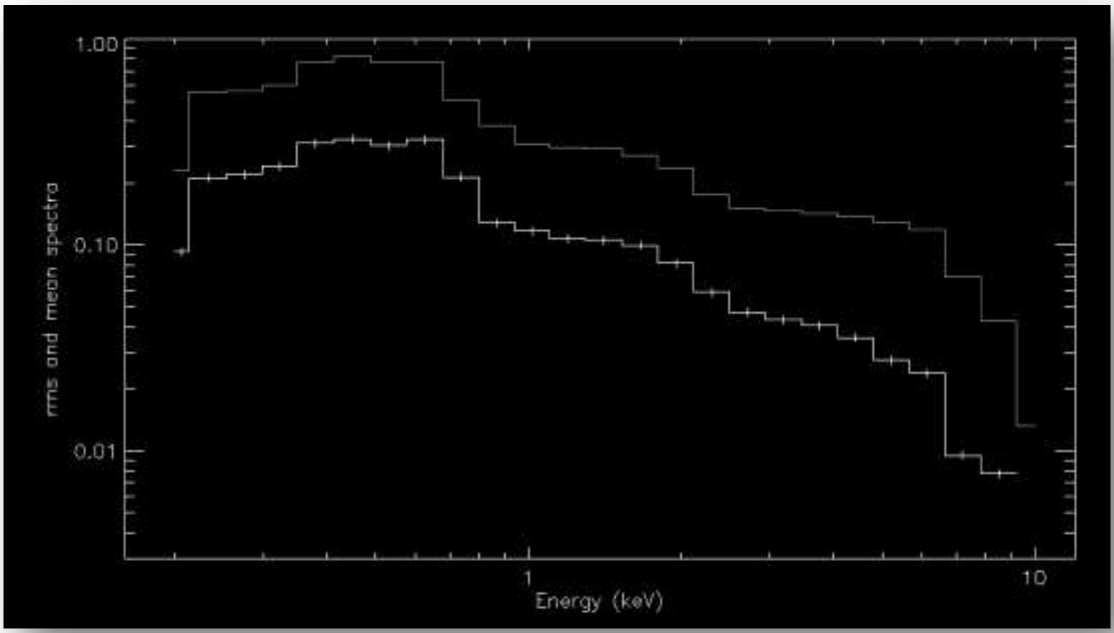


Fig 5: the rms spectrum computed using 25 energy bands from a single observation (white) along with the mean spectrum (grey), plotted in ct/s units on a log-log scale.

Cross-spectrum

Here's a worked example using EPIC_LOAD_LC to produce a 2-band light curve over multiple observations, and then using these as input to CROSS_SPECTRUM to generate cross-spectral products. The first step is to extract the time series

```
; define energy bands, data directory, time bin size
  pi_list = [[400, 700],[2000, 5000]]
  rootpath = "C:\Users\Simon\Documents\work\ngc4051_data\events\"
  dt = 5.0D

; extract the EPIC time series (from pn only)
  x = EPIC_LOAD_LC(PI_LIST=pi_list, ROOTPATH=rootpath, $
                   ERROR=error, DT=dt, DATA_T=time, $
                   SEG_LIST=seg_list, T_START=t_start, $
                   DATA_B=bkg, CHAN_LIST=chan_list, $
                   OBS_NAME=obs_name, INST_MASK=[1,0,0], $
                   OBS_LIST=INDGEN(8)+1, GAP_LIST=gap_list)

; plot the two time series for visual quality check
  HELP, x, error
  PLOT, time, x[0,*]
  OPLOT, time, x[1,*], color=120
```

In this case there are multiple observations, so the output SEG_LIST contains the start/stop indices of each observation. These observations will in general have different lengths, but for Fourier analysis we prefer to work in segments of equal length. So the next step of the analysis is to 'trim' the data array so that the time series are neat multiples of some N_SEG length. In this case we'll work with 10 ks segments.

```
; trim segments to N_SEG multiples in length
; using EPIC_SEGMENT
  seg_len = 10000.0
  n_seg = ROUND(seg_len/dt)
  x_seg = EPIC_SEGMENT(x, SEG_LIST=seg_list, N_SEG=n_seg, $
                         GAP_LIST=gap_list, MEANS=means, INDX=indx)
  HELP, x_seg
```

Now any routine that divides the time series into segments of length N_SEG should work fine, irrespective of how many distinct observations were combined. Note that we pass the GAP_LIST output of EPIC_LOAD_LC as input to EPIC_SEGMENT. This ensures that 'bad time intervals' are skipped over when splitting the data into segments of length N_SEG.
We can also 'trim' the error array in exactly the same way by making use of the INDX output:

```
  err_seg = error[*, indx]
  HELP, err_seg
```

The next step is to compute the cross-spectrum. We only supply the data and error arrays (X and ERROR), the time bin size DT, and the segment length (N_SEG). The data and error arrays are then broken into segments of length N_SEG points, and no segment crosses between observations (this is the purpose of using EPIC_SEGMENT).

```
; extract vectors for 'soft' and 'hard' band time series
  xx = REFORM(x_seg[0, *])
  yy = REFORM(x_seg[1, *])

; Compute the cross spectrum
  c = CROSS_SPECTRUM(XX, YY, DT=dt, N_SEG=n_seg, /POIS, $
                     TLAG=tlag, LAG_ERR=lag_err, /RMS, $
                     F_C=f, BINF=-1.3, F_L=f_l, F_U=f_u)

; plot the time delay spectrum
  xrange = [1e-4, 1e-2]
  yrange = [-200, 200]
  PLOT, f, tlag, PSYM=10, /XLOG, /XSTYLE, $
        YTITLE="time delay (sec)", $
        /YSTYLE, XTITLE="Frequency (Hz)", $
        XRANGE=xrange, YRANGE=yrange, $
        CHARSIZE=1.5
  PLOT_ERR, f, tlag, lag_err
  OPLOT, xrange, [0,0], LINESTYLE=2
```

Note: to plot errors we used the PLOT_ERR.PRO routine (also available at the above web site).
In the above example we did not make explicit use of the error array extracted by EPIC_LOAD_LC. Instead, we use the /POIS keyword and assumed the time series were Poissonian (i.e. the 'error' on each count rate just comes from the usual sqrt[counts] rule). If there is non-negligible background this will not be strictly true. In this case we can supply the error arrays (as two time series) to the CROSS_SPECTRUM function.

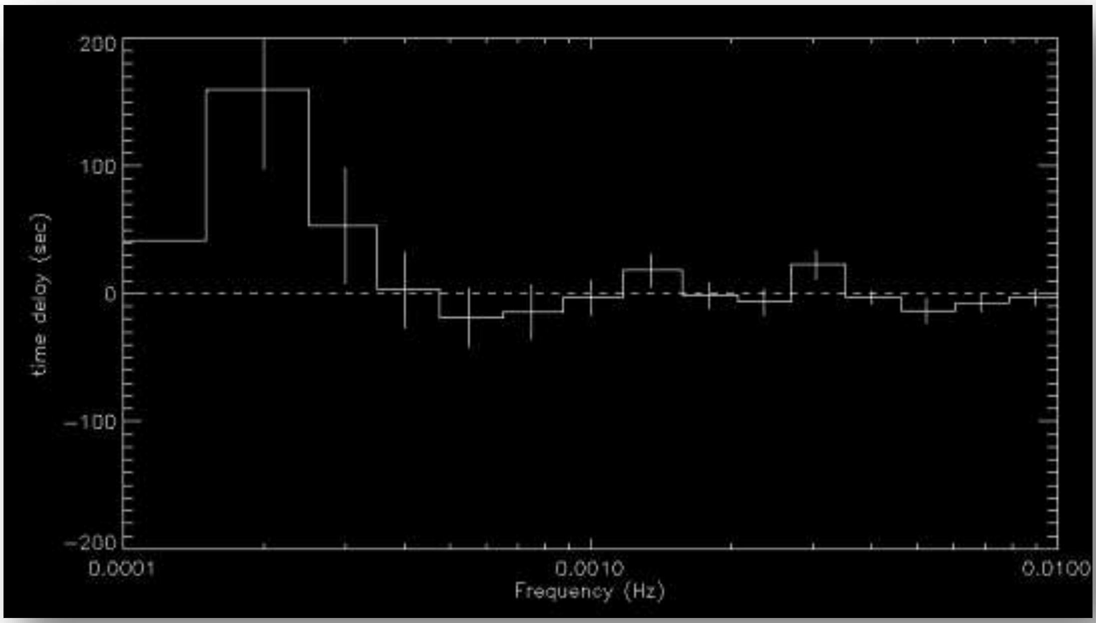


Fig 6: the time delay spectrum, derived from the cross-spectrum phase delays using the CROSS_SPECTRUM function. This compares the 0.2-1 keV and 3-10 keV bands.

Here's a full list of the inputs to `CROSS_SPECTRUM`

```
X       - (float array) first time series, evenly sampled
Y       - (float array) second time series, same sampling
DX      - (float array) 'errors' on series 1 data
DY      - (float array) 'errors' on series 2 data
DT      - (float) sampling period (same for x and y)
N_SEG   - (integer) number of points in each segment
POIS    - (logical) if set then data are Poisson
BINF    - (float) frequency binning factor
CTR     - (logical) indicates data are in ct/bin (cf. ct/s)
MINBIN  - (logical) minimum points per bin (for log-freq rebinning)
RMS     - (logical) whether to use 1/mean
                   normalisation on FTs
```

and here's a list of all the outputs

```
C       - (complex array) Complex cross-spectrum of two series
COH     - (float array) coherence of two series
CO_ERR  - (float array) error on coherence
PHASE   - (float array) phase lag (-pi, +pi)
PH_ERR  - (float array) error on phase lag
TLAG    - (float array) time lag
LAG_ERR - (float array) error on time lag
COVAR   - (float array) covariance spectrum
```

```
COV_ERR - (float array) error on covariance
F_C     - (float array) frequency (centre of bin)
F_L     - (float array) lower bound of frequency bin
F_U     - (float array) upper bound of frequency bin
DF      - (float) resolution of spectrum
P_1     - (float array) auto-spectrum of series 1 (x)
P_2     - (float array) auto-spectrum of series 2 (y)
P1ERR   - (float array) errors on spectrum of series 1
P2ERR   - (float array) errors on spectrum of series 2
BIN_N   - (int array) number of periodogram points in each bin
```

The degree of frequency averaging is specified with the `BINF` input keyword. Using `BINF` > 1 the routine performs even frequency binning, averaging together `BINF` Fourier frequencies. With `BINF` < -1 the routine uses logarithmic frequency binning, with each bin spanning a factor of |`BINF`|. E.g. with `BINF = -1.2` each bin spans $f_j - 1.2f_j$. The `MINBIN` keyword can be used to ensure each bin contains at least this many Fourier frequencies, i.e. to set a limit to how small a bin can be.

If the `POIS` keyword is set the data are assumed to be Poisson data in counts/sec units with `DT` in units of seconds, and computed the noise on the time series assuming the usual sqrt[counts] rule of counting statistics. The coherence and its error are then calculated using equation 8 of Vaughan & Nowak (1997) which applies a Poisson noise correction. If the 'errors' (`DX, DY`) are supplied for the time series then the noise terms in the coherence correction are derived from these instead.

The covariance is simply defined as the product of the coherence between bands X and Y, and the power spectrum of band X, i.e. it represents the power in band Y that is covariant with that of X. Actually, here we output the sqrt[covariance] since this will be in 'linear' units.

With tighter quality control

The above analysis can be repeated with the additional step over applying some data quality checks. Before running EPIC_SEGMENT we can inspect a plot of the background to source ratio:

```
; quality check: inspect the background/source ratio
N_obs = N_ELEMENTS(seg_list[*,0])
bs = REFORM(x[0,*]/bkg[0,*])
PLOT, 1.D/bs, YRANGE=[0,1]
FOR i=1, N_obs-1 DO $
 OPLOT, [seg_list[i,0], seg_list[i,0]], [0, 5], color=160, thick=4
```

After looking at this maybe we decide that the interval 28000:29500 has unacceptably high background. We can create an additional array listing the new BTI and add this to the existing list

```
gap_list2 = REFORM([28000,29500], 1, 2)
gap_list = [gap_list, gap_list2]
```

Then, upon running EPIC_SEGMENT this new BTI will be excised from the 'trimmed' array.

Loading products into XSPEC

The output can quickly be converted to an XSPEC compatible file. The output vectors `F_L` and `F_U` list the lower and upper edges of each frequency bin, so can be used to define an XSPEC-style data table. The routine `XSPEC_OUT` will format this correctly for you

```
    IDL> chan_list=[[f_l],[f_u]]
    IDL> XSPEC_OUT, tlag, lag_err, CHAN_LIST=chan_list, $
                 OTYPE=1, FILENAME="spec.txt"
```

This simply writes an ASCII (plain text) file with four columns of data:

lower frequency of bin, upper frequency of bin, lag*bin_width, lag error*bin_width.

The reason for multiplying the data by frequency bin width is that XSPEC automatically divides by the bin width upon loading a dataset (it assumes the data are counts per bin, but works on counts/keV, so divides by bin width in keV units).

The FTOOL `flx2xsp` can then be used to convert this ASCII file into two FITS files that can be loaded directly into XSPEC. One file is the (lag) spectrum data and the other is a response matrix file. (The response is a dummy - simply a diagonal matrix in this case – but is needed for XSPEC to run.)

```
> flx2xsp infile="spec.txt" phafil="spec.pha" rspfil="spec.rsp"
```

The load the file `spec.pha` into XSPEC

Outstanding issues

These procedures/functions have been tested using IDL 8.1 (Linux and Windows) on several *XMM-Newton* datasets, but no doubt problems remain. In particular, the following are corrections/additions that I intended to implement in the near future:

- The 'covariance' spectrum produced by `cross_spectrum.pro` does not use the correct error formulation. At the moment the fractional error on the covariance is scaled and applied to the covariance, which will be only approximately correct.

- The main task `EPIC_LOAD_LC` will produce background time series (if requested). But as yet there is no explicit use of this to check for background flares, etc. It is assumed the end user will examine the background and check this. An additional task that takes in the output from `EPIC_LOAD_LC` and performs some analysis of the background to produce diagnostic information and/or automated background filtering would be useful.

Feedback

If you have any comments or questions about the routines or documentation please send them by email to simon.vaughan@leicester.ac.uk