CZTI calibration data processing

Varun Bhalerao, Nilkanth Vagshette

Document version 4, April 16, 2015

Contents

History

This document describes the steps used in analysis of CZTI ground calibration data, and is intended to be a quickstart manual for intermediate / advanced users.

Version 1 Initial document (Varun).

Version 2 Incremental updates after CZTI meeting (10 March 2015)

Version 3 Added some sections, first complete draft

Version 4 First circulation–ready version

Part 1
Overview

This document is a summary of the steps used in processing the ground calibration data for the Cadmium Zinc Telluride Imager (CZTI) on board the *Astrosat* satellite. This is not a complete manual for all the software used in this process, and is not meant to be a guide for beginners. Instead, this document is a quick reference point for users who are familiar with the overall scheme, and are interested in specifics of commands.

We start with a overview of the steps in this part. In Part 2 we discuss data reduction, viz. conversion of the raw binary data to `FITS` files. Part 3 describes conversion of those files into several health and monitoring data products, as well as creation of calibration products. Part 4 provides consolidated information about location and nature of key data products. Finally, Part 5 gives short descriptions of some codes that were not discussed in detail elsewhere in this document.

Commands and filenames are listed in `typewriter text` in this document. Code snippets or long commands are denoted in code blocks:

```
sample_command infilename outfilename
```

Data and code locations are specified in all sections, marked in blue:

All data are located at `/data2/cztidata/` *on the* `rohini` *server.*

1.1 Background

The Cadmium Zinc Telluride Imager (CZTI) is a coded aperture mask imaging instrument on board *Astrosat*. It consists of four identical quadrants, each with sixteen CZT modules (detectors). Each of these 64 detectors is a 39.06 mm wide square divided into a 16×16 array of pixels. The central 14 pixels are 2.46 mm wide, while the side pixels are only 2.31 mm.

These detectors underwent extensive ground calibration and testing for characterizing their spectral and imaging response. Calibration data were primarily acquired in TIFR[1] and VSSC[2]. The TIFR data were used for shortlisting modules (detectors) to populate six test quadrants. These modules were then tested further at VSSC to select the final 64 flight modules. Flight modules were calibrated using various radioactive sources (^{241}`Am`, ^{133}`Ba`, ^{109}`Cd`, ^{57}`Co`) at multiple temperatures. They were also subjected to thermal cycling, and compared in a before/after performance analysis. In this document, we restrict ourselves to the data reduction and analysis of these final 64 modules. The results of these calibrations are documented elsewhere.

1.2 Data flow

Raw data is saved in a binary format in the laboratory. This data is uploaded to the `rohini` server at IUCAA through the `garuda` network, or through another `NKN`[3] conduit. The raw data is copied to two places: a working directory, and a backup directory. At the time of this writing, final calibration backups are stored in `/data2/cztidata/vssc_tar_files` on `rohini`. Redundant backups of the raw as well as processed data are also performed on a separate NAS server at `/czti_backup/archive` and `/czti_backup/vssc_raw_files` respectively. A separate document, "*Astrosat / CZTI lab data structure*" (or `filestructure.pdf`) describes the convention adopted for file and directory names. For now, it is important to note that file and directory names contain a timestamp. All data processing and analysis leaves the base name of the file untouched, so that it is easy to locate the parent data set for any data product. Attempts are made to keep relevant metadata within data products wherever feasible (eg. Appendix B).

Each contiguous data acquisition without changing operation conditions (temperature, radioactive source, etc) is called a "*run*". The data set from each run is saved in a separate

[1] Tata Institute of Fundamental Research, `http://web.tifr.res.in/~daa/XRay.html`
[2] Vikram Sarabhai Space Center, `http://www.vssc.gov.in/internet/`
[3] National Knowledge Network

directory. The binary data consists of two types of files: LBT (Low Bitrate Telemmetry) and HBT (High Bitrate Telemetry). These data are first processed by a set of c codes and bash scripts and converted into a standard FITS format (see Part 2, data reduction). The FITS files created in this step are event files (.evt), similar to those used by ftools. These scripts also create various auxillary files, including the .dph (Detector Pulse Histogram) files used for imaging. The data are then processed by a suite of IDL scripts and programs (see Part 3, data analysis). These programs create some machine-readable products (like lists of dead and noisy pixels, fits file with spectra of all pixels in a quadrant, etc) and some human-readable products (PDF plots of module spectra, count maps, etc.). Wherever feasible, key human-readable files are compiled into a reports and saved in another directory. After running the usual scripts, a typical directory structure is as follows:

```
[czti@rohini /data2/cztidata/vssc/FQ0/20140116_FQ0_Am3_20C_4hrs]$ ls -1
alphaveto - Event files for the veto spectrum, and
-           allspec.fits.gz files with pixelwise spectra for the full quadrant
events    - Main data in event format (.evt)
log       - Logs of test setup and automated processing
plots     - Plots produced by various analysis codes
proc      - Intermediate products produced by analysis codes
raw       - Raw HBT and LBT files saved in laboratory
reports   - Concise collections of plots produced in batch processing
sci       - Auxiliary files produced by data reduction software
```

Calibration data are located at /data2/cztidata/vssc *on the* rohini *server.*

All codes are under version control on a SVN *server running on* rohini*. Within the IUCAA network, they can be checked out from* svn://192.168.11.35/czti*.*

Part 2
Data reduction

Data reduction is streamlined into a single script, `proc_raw2evt`. In this part, we first introduce the commands required for processing (§2.1). In practice, the user simply has to `cd` into the appropriate directory and run the shell script as described in §2.2.

2.1 Basic codes

Conversion of raw data to event files is handled by two programs: `rawtolevel1` and `scitoevt`. `rawtolevel1` takes the `HBT` and `LBT` files as input, along with "command" and "comment" files. The latter two files are not present in typical lab data, and dummy files are used instead. The code produces a science file (`.sci`), a housekeeping file (`.hk`), a time calibration table file (`.tct`) and updates a log file. In short, this code takes the raw data stream and separates the various functional components from it into separate files. Then, `scitoevt` is run to convert the separated data frames into fits files. This code creates event files (`.evt`) for the CZT X-ray data as well as the veto spectra. It also adds headers with Module IDs for all modules present.

> `c` *codes for data reduction are under version control at* `$SVN/trunk/code/level1`.

2.2 proc_raw2evt

To simplify processing, these codes are wrapped into a bash script called `proc_raw2evt`. This script utilizes the standard filenames and directory structures to call both `rawtolevel1` and `scitoevt`. This script creates output directories as required. The syntax for running this script is simply:

```
proc_raw2evt
```

The script is to be executed in the base directory for any data run. It expects that raw data (`.HBT` and `.LBT` files) is present in `./raw`, and configuration files are present in `./log`. In particular, the script checks that the `./log` directory contains exactly one `testrecords.log` file (Appendix A), and exactly one file ending in `.config.ini`. If these conditions are not met, the script aborts with an appropriate message saved in `./log/proc_raw2evt.log`. The script also logs exact commands spawned and their output in the same log file. Another important task for the script is to give read and write permissions to the `czti` group, which is necessary for `cztuser` on the `algol` server to further analyze the data.

> *The* `bash` *scripts for data reduction are under version control at* `$SVN/trunk/code/autoproc`.

Part 3
Data analysis

The fits files created in data reduction are further analyzed with IDL codes. Typical data analysis for a single quadrant for a single run is done by an IDL command basicproducts. This code internally calls various codes in order to create a allspec.fits.gz file with pixel wise spectra, and various plots and text files with count maps, module and pixel spectra, noisy and dead pixel flagging, pixel-wise count rate plots etc. For processing datasets with multiple quadrant data, similar functionality is provided by the quadprod command. The splitproc code attempted to automatically separate data sets, but was not used in practice due to some issues. Use of that code is now deprecated. In some cases, the operator manually copied the raw data into multiple locations (for example, in the FQ0/ and FQ2 directories) and then processed it with basicproducts.

The IDL codes for data reduction are under version control at $SVN/trunk/code/analysis.

The details of running both codes follow:

3.1 basicproducts

The basicproducts command is simple:

```
basicproducts, '/path/to/directory', fitsext=fitsext
```

Let us consider data in /data2/cztidata/vssc/FQ0/20140116_FQ0_Am3_20C_4hrs as an example. As the folder name suggests, this has data for FQ0: Flight Quadrant 0. By convention, this data is stored in fits extension 1. *In general, data for Quadrant N is stored in fits extension N+1.* After going to the directory, the user should start IDL and give the command,

```
basicproducts, '/data2/cztidata/vssc/FQ0/20140116_FQ0_Am3_20C_4hrs', fitsext=1
```

First, the code examines the testrecords.log file for determining which detectors are to be analyzed from which files. This is useful for cases where users do not care about data for all modules. In testrecords.log, files may labeled as 'ALL' or 'BG' in order to process all detectors. Then, basicproducts runs the following codes in order:

1. alphaveto to convert the event file to pixel-wise spectra. This file is stored as a fits image rather than a binary table, and can be read significantly faster by other codes.
2. countmap to create text files that list the counts in each pixel of each module. It also creates PDF plots showing physical distribution and histogram of counts in the module.
3. countrate to generate plots showing the count rate of each pixel in the module as a function of time
4. noisydead to create text files listing dead and noisy pixels. It also creates a plot showing location of the dead and noisy pixels.
5. modspec to generate the spectrum of the entire module. *Note that no gain-offset correction is applied at this stage!* Both: a text file containing the spectrum and a PDF plot of the spectrum are created.
6. pixspec to create a PDF showing plots of pixel-wise spectra, 16 pixels per page.

All the plots are saved in the plots/ subdirectory, while all text files are saved in the proc/ subdirectory. The base name of the input raw file is retained, and extensions like .counts00.pdf or .spec13.txt are added. The text files have comments at the top which describe the necessary metadata regarding input files and any free parameters used in the code (Appendix B). Comment lines begin with a #. After all this processing is complete for all modules, basicproducts spawns a shell command to combine related PDFs into a single PDF in the reports/ subdirectory. A separate report is produced for each file which has 'ALL' or 'BG' as its testrecords.log entry.

All the support codes called by `basicproducts` *are under version control at* `$SVN/trunk/code/analysis`.

3.2 quadprod

Most of calibration data sets consist of combined data for multiple quadrants. However, each quadrant usually had a different radioactive source, so that data analysis should still be done on a per-quadrant basis. The usual command thus excludes the fits extension:

```
quadprod, '/path/to/directory'
```

This functionality is provided by `quadprod`, short for `quad`rant `prod`ucts. Functionally, `quadprod` is similar to `basicproducts` — it runs several standard processing steps *for each quadrant* in the data file. First, the code checks for the presence of a `./log/testrecords.log` file, although contents of that file are not used in further processing. A short logfile is created at `./log/quadprod.log`. The following codes are then run for each of the four quadrants:

1. `alphaveto` to convert the event file to pixel-wise spectra. This file is stored as a fits image rather than a binary table, and can be read significantly faster by other codes.
2. `countmap` to create text files that list the counts in each pixel of each module. It also creates PDF plots showing physical distribution and histogram of counts in the module.
3. `modspec` to generate the spectrum of the entire module. *Note that no gain-offset correction is applied at this stage!* Both: a text file containing the spectrum and a PDF plot of the spectrum are created.
4. `procpix` to fit a line to each pixel spectrum, log the best fit parameters, and plot pixel-wise spectra with line fit overlaid.

Unlike `basicproducts`, `quadprod` does not run `countrate` and `noisydead`. This decision was taken as four-quadrant data were usually obtained as shorter integrations and noisier operating conditions. Manual inspection of count rate plots was infeasible due to high data volume, and nosiy/dead flags were unreliable due to the operating conditions.

The other difference is that `pixspec` is replaced by the more advanced `procpix` program. Here, `quadprod` tries to parse the input filename to figure out if the source is one of ^{241}`Am`, ^{133}`Ba`, ^{109}`Cd`, or ^{57}`Co`. If none of the source names matches, then `quadprod` instructs `procpix` to search for the strongest line[4] above channel 800 ($\sim$ 39 keV), with $\sigma \approx 45$ channels ($\approx$ 2.2 keV, FWHM $\approx$ 5.2 keV).

The user can override these defaults. For example, the command to search for lines above channel 400, with $\sigma \approx 30$ channels is:

```
quadprod, '/path/to/directory', linepos= -400, linesig=30
```

If linepos is positive (eg `linepos=1200`), then `procpix` searches for a line centered at that channel.

As with `basicproducts`, all the plots are saved in the `plots/` subdirectory, while all text files are saved in the `proc/` subdirectory. The base name of the input raw file is retained, and extensions like `.Q0.counts00.pdf` or `.Q3.spec13.txt` are added. Note that the extensions now contain a quadrant ID too. The text files have comments at the top which describe the necessary metadata regarding input files and any free parameters used in the code. Comment lines begin with a `#`. After all this processing is complete for all modules, `quadprod` spawns a shell command to combine related PDFs into a quadrant-wise PDFs in the `reports/` subdirectory.

The support codes called by `quadprod` *are under version control at* `$SVN/trunk/code/analysis`.

[4] Specifically, the code looks for the channel with maximum counts.

Part 4
Data products

In this section, we discuss more advanced data products, usually those that are made by combining data from several runs.

4.1 Dead-noisy lists

For each individual run, a list of dead and noisy pixels is created by `basicproducts` (Section 3.1). Dead pixels are defined as pixels with zero counts, and noisy pixels are 5-σ outliers to the distribution of counts in pixels. These lists calculated for individual runs were all collated and compared to produce master lists. The processing is done using two `IDL` codes, `dead_list` and `noisy_list`, as described below.

4.1A Dead lists

Dead pixel lists are collated by running `dead_list` as follows:

```
dead_list, dirfile='filelist.txt', fm='FM2'
```

Here, `dirfile` is simply a list of directories (runs) to be processed. Each line in `dirfile` should be the top level directory of a run (Section 1.2), with subdirectories like `log` and `proc` where the dead and noisy pixel lists reside.

Master lists of dead pixels are saved at `/data2/cztidata/vssc/analysis/dead_lists` *on the* `rohini` *server.*

The output file lists all pixels which were dead in every single test of that module.

4.1B Noisy lists

Noisy pixel lists can be generated in a similar format by running `noisy_list.pro`[5]. However, this process was superseded by the `noisy_props` code. `nosiy_props` accepts contextual inputs like quadrant name and temperature to locate all files satisfying those conditions. If the temperature is not specified, `noisy_props` collates data from all temperatures. In addition, wildcard-like data types can be specified to narrow down the data selection: for example, specifying `dtype='hr'` will select only folders like `20140118_FQ0_Cd_20C_4hrs` but not `20140118_FQ0_Cd_20C_stabilization` or `20140118_FQ0_Cd_20C_warmup`. These keywords eliminate the need for first creating a text file listing the directories to be processed.

Master lists of noisy pixels are saved on the `rohini` *server at* `/data2/cztidata/vssc/analysis/noisystats/codebased/allquad` *and* `/data2/cztidata/vssc/analysis/noisystats/codebased/FQ*`.

`noisy_props` generates several subdirectories *in the directory which it is run.* Each temperature and quadrant combination gets a subdirectory of the type `./FQ0/15C` or `./FQ2/alltemp`. Each subdirectory contain various files:

```
alltime_noisy.txt       : List of pixels which were noisy in all files
frequently_noisy.txt    : Pixels noisy > 75% of the time
occassionally_noisy.txt : Noisy in 25% - 75% cases
often_noisy.txt         : Noisy in < 25% cases
once_noisy.txt          : Noisy only once
histogram.txt           : Summary of number of noisy pixels in all modules
noisystats_??.pdf       : Plot of how often individual pixels were noisy
noisystats_??.txt       : List of how often individual pixels were noisy
```

[5] Results from this processing are saved on the `rohini` server in the directory `/data2/cztidata/vssc/analysis/noisystats/codebased/all_time_noisy`. Note that these are superseded by `noisy_props` data products.

Detailed help can be obtained by typing "noisy_props, /help". A typical noisy_props command is of the form.

```
noisy_props,basedir='/data2/cztidata/vssc/',quadname='FQ0',temp='20C'
```

4.2 Flickering pixels

Flickering pixels were identified by visual inspection of diagnostic plots produced by the IDL program countrate. Machine–readable lists of these flickering pixels have the following format:

```
#Module no, no of noisy pixels, pixel list
#Q0:
00,5,0,31,64,143,176
01,0
02,1,16
03,1,14
04,3,1,2,16
05,4,48,80,143,208
06,0
07,1,239
08,1,128
09,7,64,94,126,141,205,207,223
10,1,227
11,2,175,241
12,5,79,173,189,206,244
13,14,9,30,34,36,50,61,63,155,171,224,228,235,251,252
14,7,5,111,156,171,186,187,250
15,12,1,30,46,47,62,63,93,169,171,190,237,242

#Q1:
00,3,49,55,185
01,0
```

For example, the line for FQ0 Module 2 is 02,1,16, which says that only 1 pixel is flickering, and that is pixel number 16. On the other hand, the line 01,0 means that module 1 did not have any flickering pixels.

Two lists of flickering pixels, before and after the thermovac tests are saved on the rohini *server at* /data2/cztidata/vssc/thermovac/stability/flickering_list_20140611.txt *and* flickering_list_20150116.txt.

These files were used in stability comparisons with plotrates.pro located in the same directory, and referred to in Section 4.6.

4.3 Spectroscopically bad pixels

Secondly, pixel-wise spectra were visually inspected for a large number of data sets, and any peculiarities were noted. These include noise at low energies, missing lines, etc. Detailed reports of this inspection are saved in PDF reports with names of the form FQ1_vispix.pdf.

Detailed reports of visual pixel inspection are saved on the rohini *server at* /data2/cztidata/vssc/analysis/vispix/FQn/FQn_vispix.pdf. *Machine–readable lists of pixel quality are saved with files named of the form* /data2/cztidata/vssc/analysis/vispix/FQ1/15C/FQ1_mod02_15C.vispix.txt.

A sample machine–readable pixel report file, FQ1_mod02_15C.vispix.txt, is reproduced here:

```
# visual inspection list
# FQ1 Det02 15C
sometimes_weird
sometimes_sup_peak
```

```
sometimes_bad_en_resol 12 112
sometimes_low_en_noise 14
sometimes_noisy
weird 11
sup_peak
bad_en_resol
low_en_noise 0 1 8 9 15
noisy
```

The same directories also contain PDF maps showing the characteristics of each pixel at each temperature. We caution that some characteristics of pixels may not be reproducible even at the same temperature.

4.4 Gain-offset calculation

Gain and offset calculations are done in two steps. In the first step, we calculate an approximate value of gain and offset with the `gainoffset` code, using ^{241}Am (59.54 keV) and ^{57}Co (122.06 keV) lines. These values are then refined by fitting to all usable data sets with the `linearity`.

4.4A gainoffset

The `gainoffset` code finds the strongest peak above a certain channel number in ^{241}Am and ^{57}Co data, fits Gaussian line profiles (with a `fitline`–like procedure, see Section 5.3) and calculates gain and offset from these peaks. Gain and offset relate energy to channels by Equation 1:

$$\text{Energy} = \text{Channels} \times \text{Gain} + \text{Offset} \tag{1}$$

A typical `gainoffset` command is:

```
gainoffset, detid=2, $
fitsfile1="file1-Am-allspec.fits.gz", line1=59.54, chcut1=-700, lwidth1=40.9, $
fitsfile2="file2-Co-allspec.fits.gz", line2=122.06, chcut2=-1600, lwidth2=45.6, $
gainfile="gainfile-gainoffset.txt", plotgain="plotgain-gainoffset.pdf"
```

where the code will look for a $line1 = 59.54$ keV (^{241}Am) line above `|chcut1|`, viz above channel 700. It expects the nominal σ of the line to be 40.9 channels — this value is used to select the fitting region as well as the starting guess for the fit. Similar parameters are specified for the ^{57}Co file. Outputs are saved as a text file in `gainfile-gainoffset.txt`, with plots saved to `plotgain-gainoffset.pdf`.

In practice, a user will usually want to run gain–offset calculations for all temperature data for an entire quadrant in one shot. This is facilitated by the `IDL` routine `goAmCo`. To process FQ3 data, the command is:

```
goAmCo, quad=3
```

This routine outputs data products in the default location given below:

First pass calculations of gain and offset are saved in `/data2/cztidata/vssc/analysis/go_AmCo` *on the* `rohini` *server.*
NOTE: most users should not need to access these files, see Section 4.4B about `linearity` ***calculations.***

4.4B linearity

Calibration data has been acquired for several sources: ^{241}Am (59.54 keV), ^{57}Co (122.06 keV, 136.47 keV), ^{109}Cd (21.0 keV, 88.06 keV) and ^{133}Ba (30.97 keV, 81.0 keV). We use the two–point gain and offset values calculated in Section 4.4A as starting point to fit a straight line through all known lines in all available data sets for every module at every temperature. Apart

from calculating refined gain and offset values, `linearity` produces several useful by–products including plots of fits for each pixel, residuals for each pixel, maximum non-linearities for each pixel, etc. Since several input files are used, `linearity` requires a filelist in a text file, specifying the following parameters:

```
# This file is called filelist_sample.txt
# filename                 module_num    energy(kev) 1sigma_linewidth(channels)
file_am.Q2.allspec.fits.gz       06          59.56          60.0
file_co.Q1.allspec.fits.gz       05          122.0          120.0
file_co.Q1.allspec.fits.gz       05          136.0          130.0
```

Note that module number is required on each line as some modules were relocated during final assembly. In the example above, we are calculating data for a module which was mounted in Quadrant 2 at position 6 while acquiring ^{241}Am data, but moved to FQ1 position 5 while acquiring ^{57}Co data. `linearity` checks module IDs wherever available[6] to ensure that data for the same module are being used throughout the calculation. A minimal call to linearity would be of the form,

```
linearity, /allspec, filelist='filelist_sample.txt', $
    gainfile='twopoint_gainoffset.txt', outgainfile='new_gainoffset.txt'
```

`linearity` has several other options, which can be found by typing "`linearity, /help`". This code carries out extensive computation and can take a while to run. This can be a serious handicap for processing data of all 64 flight modules at six temperatures each. This task is simplified by the wrapper routine, `linearity_run`. This routine runs `linearity` calculations for a given quadrant at all temperatures. Using this code needs that `filelist`s for all modules are already present at predefined default locations. A typical call is of the form,

```
linearity_run, quadrant=1
```

Final values of gain and offset are saved in `/data2/cztidata/vssc/analysis/gainoffset` *on the* `rohini` *server.*

4.5 Gain–offset corrected spectra

A useful data product for any further analysis is gain–offset corrected spectra. The `IDL` code `modspec`[7] can be used to generate such spectra on demand. `modspec` has several useful options like applying pixel–wise gain–offset corrections, selecting the pixels with best energy resolution etc. A typical data product needed for further analysis is a spectrum for each {Module, Source, Temperature} combination. As these are intended for spectroscopic analyses rather than imaging, we select only the best 90% of the pixels in each module[8] for making a combined spectrum. Such spectra were produced for all test sources (^{241}Am, ^{133}Ba, ^{109}Cd and ^{57}Co) by using the `go_corr_mod.pro` routine in `$SVN/trunk/code/analysis`. The spectra are saved as text files with comment lines (`#`) containing metadata, followed by two space–separated columns listing energy and counts.

Text files and plots containing gain–offset corrected spectra for calibration data are saved at `/data2/cztidata/vssc/analysis/go_corr_spec` *on the* `rohini` *server. It contains subdirectories of the form* `FQ?/??C/`*, containing individual spectral files with names like* `FQ0_mod01_ID05051_00C_Cd.go_corr_spec.txt`.

[6] For instance, all `.allspec.fits` files have module IDs in their headers.
[7] Some details about `modspec` are given in Section 5.8.
[8] We define the best pixels as ones having the sharpest ^{241}Am lines at that temperature.

4.6 Module stability tests

We examined pre– and post–thermovac data for all quadrants to look for change in the number of dead/noisy pixels, and found a marginal increase in their numbers. We also compared module–level spectra in these two data sets, and found that the spectral characteristics do not vary significantly (Figure 1). Note that more detailed comparisons were performed using `perfcom`, discussed in detail in Section 5.7.

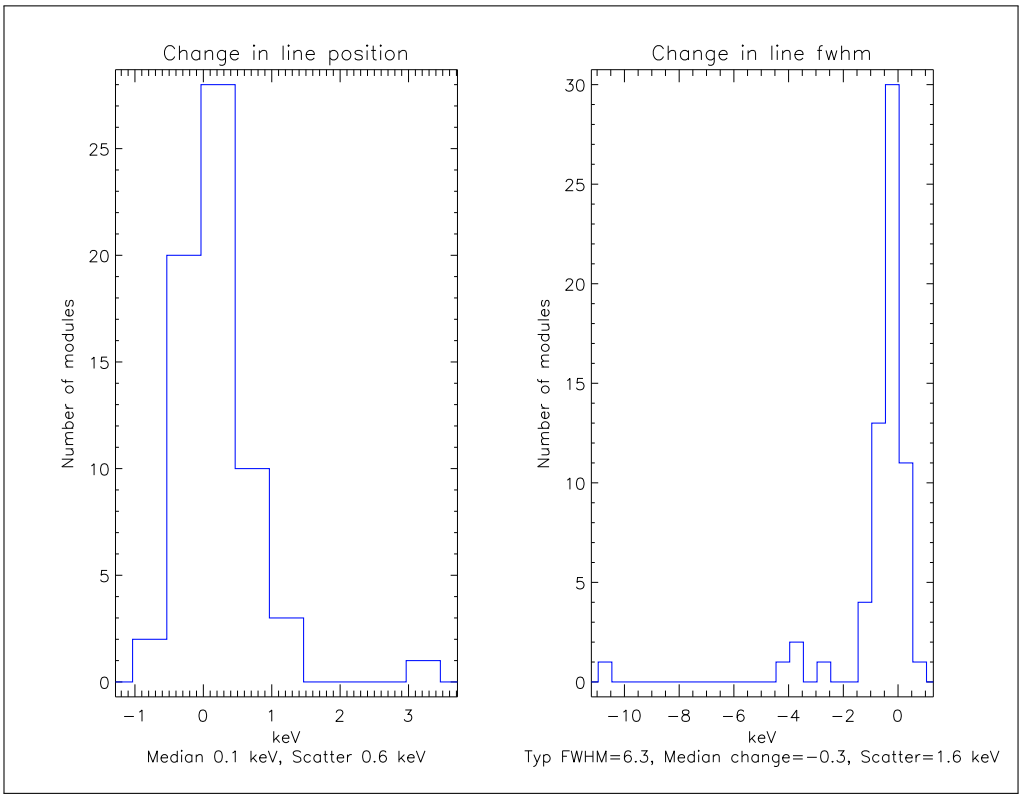

Figure 1: Comparison of pre– and post–thermovac spectral characteristics with `tvac_prepost_spec`.

The `IDL` *code for comparing noisy/dead pixels is* `plotrates.pro`*. The code and data products are both located at* `/data2/cztidata/vssc/thermovac/stability` *on the* `rohini` *server. The code for spectral comparisons is located at* `$SVN/trunk/users/varunb/tvac_prepost_spec.pro`

Part 5
Codes

Some of the IDL codes used in data analysis are listed above. A complete listing of IDL codes is given in Appendix C. Help for most of the codes can be obtained by simply calling them with a /help flag, like:

```
countmap, /help
```

Here, we describe some of the codes which have not already been touched upon in Part 3.

5.1 alphaveto

The alphaveto code reads event files, and converts them into pixel-wise spectra. The spectra are saved as 4096×4096 FITS images, with one row per pixel denoting the 4096 spectral channels. Output files are named as *.allspec.fits, and are usually compressed with gzip if alphaveto was called by basicproducts. These *.allspec.fits files have 4 extensions, corresponding to values of the alpha flag and the veto level in data. The four extensions are described in the help below:

```
alphaveto.pro, Varun Bhalerao, 2015-01-23
Version :        1.40000
SVN Revision : $Rev: 464 $
----- Documentation for ./alphaveto.pro -----

pro alphaveto
   INPUTS:
        infile          : (string) input .evt file
        fitsext         : (int)    FITS extension number to read. Default=1
        alphaveto       : (string) REQUIRED name of output fits file
                          containing pixel-wise spectra as follows:
                          Ext 0 : full spectra
                          Ext 1 : alpha = 1, any veto
                          Ext 2 : alpha = 0, veto > vth
                          Ext 3 : alpha = 0, veto <= vth
        vth             : (int)    Threshold for veto
                          Default: 50.
        /help           : display this message and exit

        pixid = module_id * 256 + pixel_number
```

5.2 apixmap

apixmap is the generic code that produces all the pixel maps seen in CZTI analysis products. It can be run in two ways: *list* mode and *counts* mode.

5.2A List mode

apixmap is run in list mode to show graphically different types of pixels, like dead and noisy pixels. It is called with a $3 \times N$ array that denotes types of the pixels, and a legend with names of each type. A sample output is shown in Figure 2.

The essential command for generating output like Figure 2 is as follows:

```
apixmap, [[0,0,0], [11,0,0], [13,9,0], [2,12,0], [1,15,0], [11,11,1]], $
legend=['Dead', 'Noisy'], colors=[64,252], filename='sample.pdf', $
title='Dead/noisy pixels in detector 0 in somefile', $
subtitle='5 Dead pixels (5.0-sigma), 1 Noisy pixels (5.0-sigma)'
```

Colours are taken from the rainbow colortable, loaded with loadct, 13. Other documentation for list mode is:

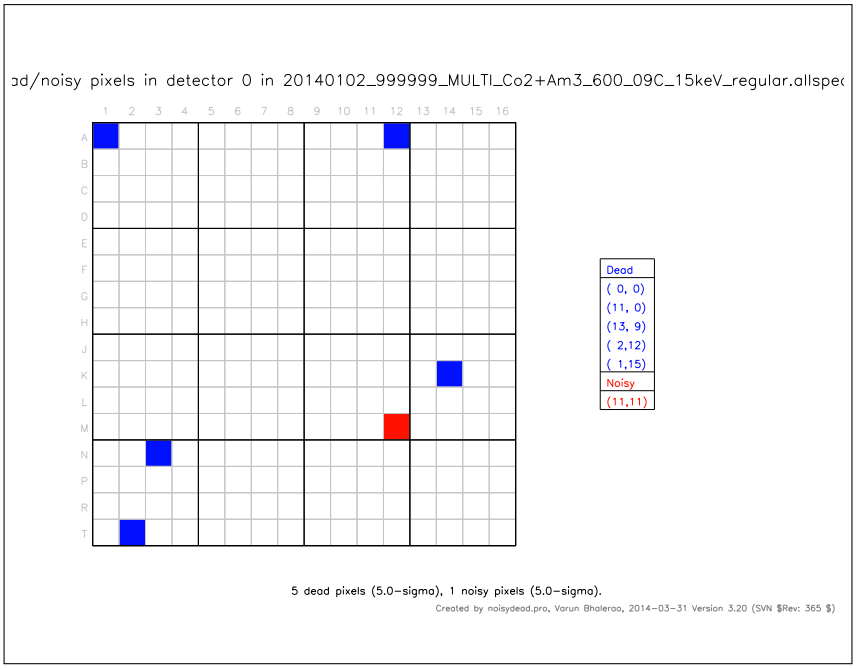

Figure 2: *List mode* output of `apixmap`.

```
apixmap.pro, Varun Bhalerao, 2013-10-03
Version :       1.13000
SVN Revision : $Rev: 405 $
----- Documentation for ./apixmap.pro -----

aim: take pixel list with colours, and make a plot.
make a table on the right with titles and the actual list

 plot type:
 0,0
   -----------------
   |  R            |     R
   |    R          |      3, 1
   |               |      5, 2
   |               |
   |               |     G
   |      G        |      6,13
   |               |
   -----------------
                  15,15

 Usage:
 pixelmap, pixlist, legend=legend, colors=colors, noaxes=noaxes, $
                filename=filename, title=title, subtitle=subtitle
        pixlist: 3*n array, with entries of type [[col,row,type], [col,row,type]]
                type must be from 0 to some number K
                pixlist need not be sorted in any order
                for int array, pixlist = transpose([[intarr mod 16], [intarr/16],
    [replicate(0, n_elements(intarr))]])
        legend: K element array, with string names for each type
                default = 'Type 0', 'Type 1', ...
        /shortlegend: Only give legend names, don't list pixels in the table
        colors: K element array, with color indices from the RAINBOW color table
                default colors are alloted if this is not specified
        /noaxes:do not overlay a light grid of pixels
        title   : string to be displayed at top of plot
        subtitle: string to be displayed at bottom of plot
        footer  : string to be put in lower right corner in smaller font
                 (usually for program name, version etc)
        filename: '/path/to/output.pdf'
        /help:  display this message
```

5.2B Counts mode

When called in *counts mode*, `apixmap` produces a pixel map on the left, and a histogram of counts on the right. This mode has relevant keywords to control the data limits of the histogram and so on. To demonstrate counts mode, we create an array of 256 random numbers (mean=0, sigma=1) and plot it to 'sample.pdf'.

```
apixmap, counts=randomn(seed, 256), dtype='Random number', $
countmin=-2, countmax=2, /saturate, filename='sample.pdf'
```

The result is shown in Figure 3. Note how the histogram is limited to the range [-2, 2]. Pixels like B12 which had < 2 counts are shown white. Pixels with > 2 ($>$ `countmax`) counts are shown with black hashing due to the `/saturate` flag.

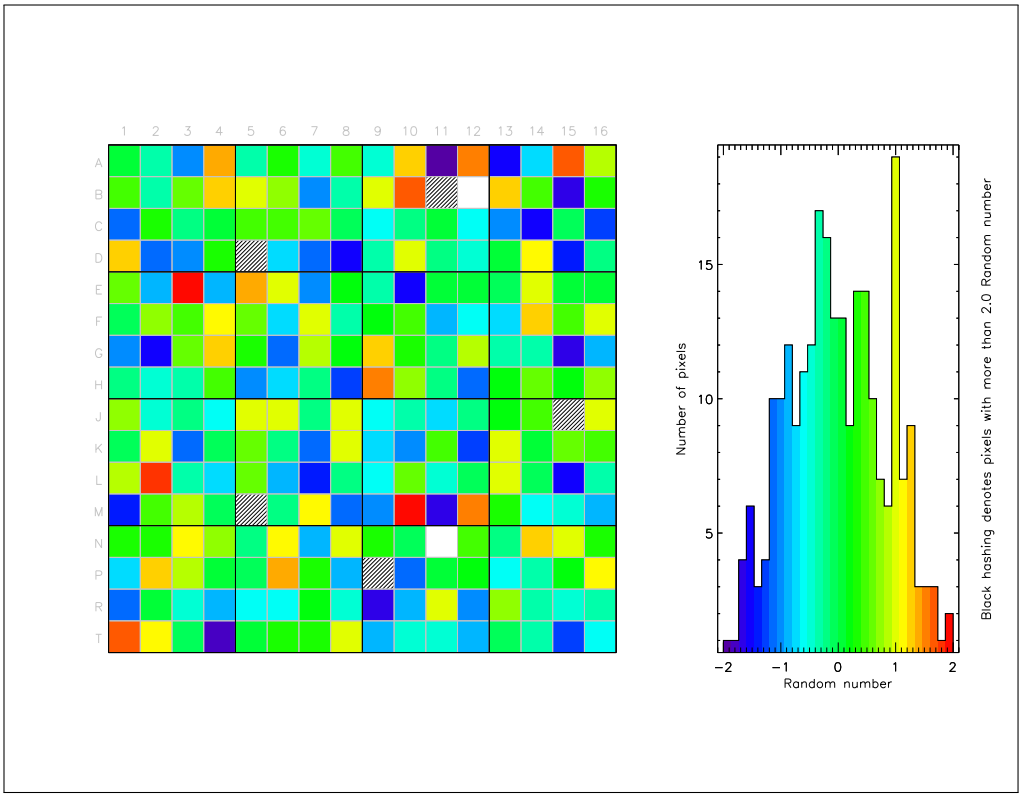

Figure 3: *Counts mode* output of `apixmap`.

The `title`, `subtitle` etc can be specified as in *list mode*. The granularity of the histogram can be controlled by `numbin` and `binsize` keywords, where `binsize` overrides the former.

5.3 fitline

This code fits a Gaussian to a line seen in spectra. The input spectrum is usually specified as a space-separated two-column text file with channel_number and counts, like the files produced by `modspec`. Alternately, the input file may be a `.evt` file—in which case `fitline` extracts the spectrum of a user–specified pixel in a user–specified detector. The fitting proceeds as follows:

1. User specifies a nominal line energy (`linepos`) and nominal one-sigma resolution (`linesig`).
2. `fitline` selects a spectrum in the range [`linepos`$-3\times$`linesig`, `linepos`$+3\times$`linesig`].
3. The centroid (calculated by `centroid.pro`) of this part of the spectrum is used as the starting guess for line center.
4. Only data within -1σ to $+2\sigma$ of this starting guess is used for fitting a Gaussian line profile. This final fitting range can be controlled by the `fitlow` and `fithigh` parameters. The default values -1 and $+2$ were selected on the basis of testing various ranges, to utilize

maximum counts for fitting the profile without being contaminated by the background or X-ray tailing.

`fitline` returns the line center, sigma and area; with corresponding uncertainties. It can also produce a PDF plot in a user-specified file. Detailed parameters for `fitline` are:

```
fitline.pro, Varun Bhalerao, Nilkanth V. 2015-02-06
Version :       1.40000
SVN Revision : $Rev: 471 $
----- Documentation for ./fitline.pro -----

pro fitline
   INPUTS: Specfiy either specfile, or the next four inputs
        specfile        : (string) Name of text spectrum (output by modspec etc)
                          Two column file listing energy, counts
        fitsfile        : (string) input .evt file
        fitsext         : (int)    FITS extension number to read. Default=1
        detid           : (int)    Id of detector for which spectrum should be
                          produced. Default=0
        pixid           : (int)    Id of pixel for which spectrum should be
                          produced. Default=0
   OUTPUTS:
        sigma           : (float)  Returned gaussian sigma of line
        err_sigma       : (float)  Error in returned gaussian sigma of line
        center          : (float)  Returned centroid of line
        err_center      : (float)  Error in returned centroid of line
        linecounts      : (float)  Returned value of counts in line
        err_linecounts  : (float)  Error in returned value of counts in line
        plotspec        : (string) optional name of the output pdf plot of
                          the spectrum for each pixel
   OPTIONAL INPUTS:
        gain            : (float) Gain in keV / channel. Default 0.0488
        offset          : (float) Offset in keV. Default 0
                          [Energy] = [Channels] * gain + offset
        linepos         : nominal position of line in keV
        linesig         : nominal sigma of line in keV
        fitlow          : lower side range for line fit (in units of linesig)
                          Default: 1.
        fithigh         : higher side range for line fit (in units of linesig)
                          Default: 2.
        rebin           : (float array*3) Optional rebinning parameters
                          rebin=[startenergy, stopenergy, binsize]
        plotspec        : (string) Name of output PDF plot
        plotmin         : Lower end of plot, in appropriate units
        plotmax         : Higher end of plot, in appropriate units
        /base           : (bool) whether to allow as additive base in the
                          gaussian fit for a line
        /help           : display this message and exit
```

5.4 peakfind

`peakfind` is a *function* that finds a local maxima. Given arrays `y` and `x` and a starting index value, it first selects a region of interest till the first inflexion points on either sides of the start point. Then, it returns the position of the highest point within this region of interest. If there are multiple points with the highest value (possible only with a flat top), then `peakfind` returns the lowest `x` coordinate from such values.

Note: the start point is specified as an index of the `x` array, but the returned value is a x coordinate rather than an array index.

```
IDL> print, peakfind(/help)
answer = peakfind(x, y, start, smoothval=smoothval)
Returns the x-coordinate of the local maximum around start
Optionally smooths the data by smoothval
NOTE: start is an index, not an x value
```

5.5 rcspec

rcspec is particularly useful when looking for module–induced systematic effects in spectra, with low SNR data. It processes a .evt file and produces plots spectra of each pixel row and each pixel column.

```
rcspec.pro, Varun Bhalerao, 2013-05-14
Version :       1.00000
SVN Revision : $Rev: 183 $
----- Documentation for ./rcspec.pro -----

 PRO RCSPEC
        Make row-wise and column-wise module level spectra
 Inputs:
        infile   : (string) input events file
        fitsext  : (int) FITS extension number to read. Default=1
        detid    : (int) Id of detector for which count map should be
                   produced. Default=0
        binsize  : (int) binsize of output spectrum (in channels)
                 : default = 16
        plotrange: (2-element float array)
                   Range of output plot in channels. Default = auto
        /help    : Show this help text
 Outputs:
        plotspec : (string) Name of pdf file to save combined spectrum to
```

5.6 groupspec

The modspec code mentioned in 3.1 creates spectra of an entire module. groupspec is a similar code that can combine spectra of a select list of pixels from a single module. If a gain–offset file is specified, then pixelwise corrections are applied before combining data. This facilitates pixel grouping, which is used for comparing mask data with uniform illumination data (Section 5.7), or for combining pixels with similar gain–offset values. A slight drawback is that groupspec v2.3 works only with .evt files, but not with .allspec.fits files. This makes the code execution slower.

For facilitating easy import of spectra into other software (eg Xspec), the default output of groupspec does not have any comments or metadata. Metadata can be added by specifying the /longtext flag.

Similar functionality can also be achieved from modspec, using the "ignorepix" input parameter. An example of this can be found in the source code for perfcomp (Section 5.7).

```
groupspec.pro, Varun Bhalerao, 2013-07-17
Version :       2.30000
SVN Revision : $Rev: 371 $
----- Documentation for ./groupspec.pro -----

 PRO GROUPSPEC
        Make module level spectra
 Inputs (required):
        infile   : (string) input events file
        pixlist  : (int array) list of pixels for making combined spectrum
        gainfile : (string) name of file with gains and offsets
                   if gainfile is not specified, make a combined
                   spectrum without any shifts.
                   [Energy] = [Channels] * gain + offset
 Inputs (optional):
        fitsext  : (int)    FITS extension number to read. Default=1
        detid    : (int)    Id of detector for which count map should be
                   produced. Default=0

        binsize  : (float)  binsize of output spectrum (in keV) Default=0.1
        erange   : (2-element float array)
                   Range of output spectrum in keV. Default = [0,200]

        /longtext: Produce a text spectrum with comments
                 : Default is no comments, the channels are in binsize units
```

```
        /help    : Show this help text
 Outputs:
        plotspec : (string) Name of pdf file to save combined spectrum to
        specfile : (string) Name of text file to save combined spectrum to
```

5.7 perfcomp

`perfcomp` is used for *comp*aring the *perf*ormance of the CZTI modules. For a given data file (eg: post-thermovac data), `perfcomp` locates the corresponding archival calibration ("old") data set. Pixels in the new data set are sorted by counts, and grouped into `numgroups` groups. Data for the same pixel groups are also extracted from the archival data set. Then gaussian profiles are fit to the new and archival data sets with `fitline` (Section 5.3). The code then writes text files with a table stating the following quantities for each pixel group: counts in pixel group, counts in line, old counts in group, old counts in line, new energy, new sigma, old energy, old sigma. It also generates PDFs for each pixel group (Figure 4), showing the new and old spectra and the line fits to them. Individual group plots are finally merged into a single `.perfcompare.pdf` file.

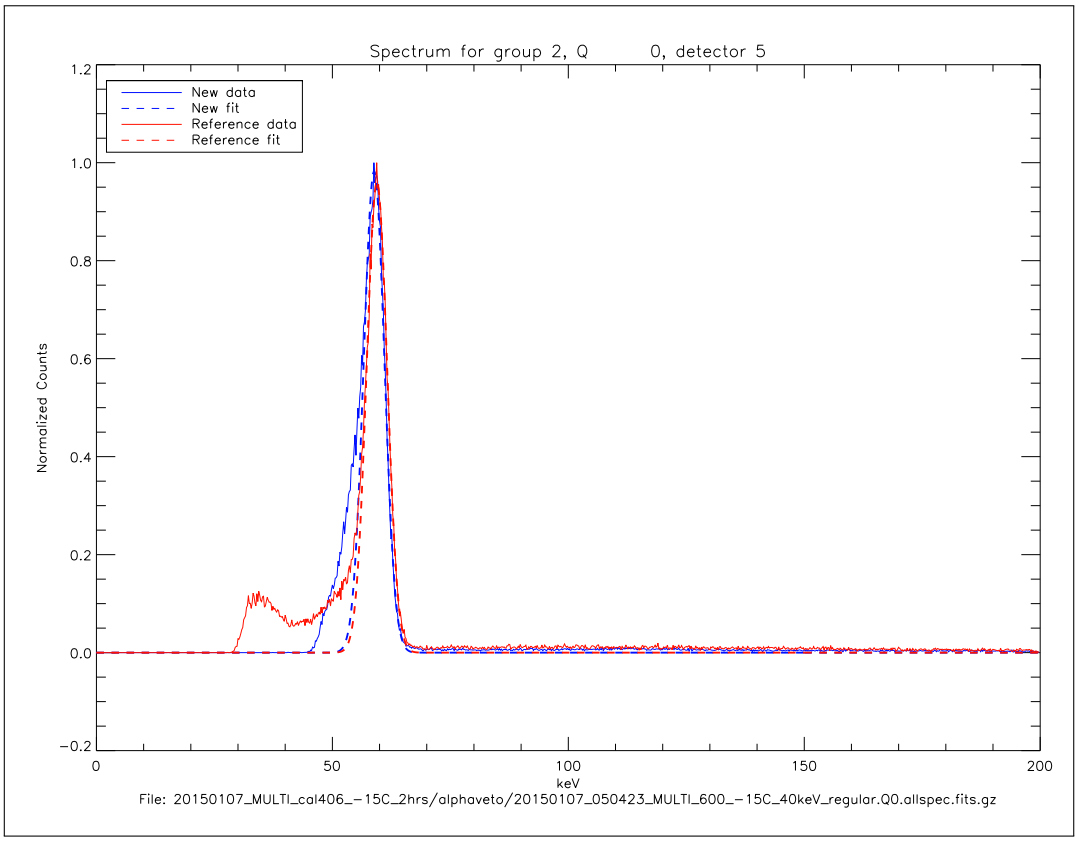

Figure 4: Comparing post-thermovac Am data for Quadrant 0, Module 5 with archival data.

These outputs can be further summarized by running the `perfcomp_summary` code. The `perfcomp` code supports following parameters:

```
perfcomp.pro, Varun Bhalerao, 2015-03-31
Version :       1.40000
SVN Revision : $Rev: 444 $
----- Documentation for ./perfcomp.pro -----

pro perfcomp
   INPUTS:
        infile     : (string) input .evt or .allspec.fits file
        allspec    : (bool) Specify if input file is .allspec.fits
                     Default file is assumed to be .evt file
```

```
        fitsext     : (int) FITS extension number to read. Default=1
                      Ignored for .allspec.fits files
        quad        : Quadrant number of the detector (0-3)
                      Required for locating correct reference data set
        detid       : (int) Id of detector for which count map should be produced
                    : Range 0 -- 15, default 0
        temp        : Temperature (e.g. '05C')

        source      : Source (e.g. 'Am2' / 'Am')
        linepos     : nominal position of line in keV
        linesig     : nominal sigma of line in keV

        numgroups   : Number of pixel groups to create (Default: 8)
        clip_perc   : Percentile point for clipping (0.5 = median, 1.0 = maximum)
                      Counts at this percentile are used as reference
        cliplim     : Multiplying factor for maximum allowed counts
                      For example, clip_perc = 0.5 and cliplim = 10 means pixels
                      with count rate above 10 times median are ignored
                      If clip_perc=1.0 and cliplim > 1 then no pixels are ignored
        /help       : print this message
   OUTPUTS PARAMETERS:
        outbase     : (string) base name and path to be prepended to output files
        binsize     : (float) binsize of output spectrum (in keV)
        plotrange   : (2-element float array)
                      Range of output plot in keV. Default = auto
```

5.8 threshold

This program calculates threshold values (Lower Level Discriminator; LLD) by fitting an error function (ERF[9]) to module spectra. The best–fit value and width (ERF sigma) are returned in the parameters `threshold` and `tsig`. The `threshold` code uses `modspec` as the backend, hence all `modspec` inputs are supported. Additionally, `threshold` can be given an input guess threshold value and a range for fitting to data. Input and output parameters specific to `threshold.pro` are:

```
IDL> threshold, /help
threshold.pro, Varun Bhalerao, 2014-05-07
Version :       1.20000
SVN Revision : $Rev: 401 $
----- Documentation for ./threshold.pro -----

 PRO THRESHOLD
        Calculate module thresholds
 Inputs (required):
       All inputs are of the same format as modspec
 Inputs (optional):
        threshold   : (float) Initial guess threshold value (keV)
                      Default: first non-zero element
        fitrange    : Range in which to fit for exact threshold
                      (2 element float array)
                      Default: threshold +- 40 keV
 Other flags:
        show        : Plot the fitrange and best-fit plots (in X server)
 Outputs:
        threshold   : (float) Final fit threshold value (keV)
                      Will overwrite any input variable that was set
        tsig        : (float) Gaussian sigma of threshold fitting
                      This is the sigma of the "Error Function" fit
                      in fitrange.
```

Internally, this code calls `modspec` for creating spectra from the input files. All `modspec` parameters are supported:

[9] ERF or Error Function is the integral of a Gaussian function.

```
----- Documentation for ./modspec.pro -----

 PRO MODSPEC
       Make module level spectra
 Inputs (required):
       infile     : (string) input events file or allspec.fits file
       /allspec   : Set /allspec to specify that input is a allspec.fits file
                    containing spectra of all pixels in that quadrant
       fitsext    : (int)    FITS extension number to read. Default=1
                    Ignored if input is allspec
       detid      : (int)    Id of detector for which count map should be
                    produced. Default=0
       gainfile   : (string) name of file with gains and offsets
                    if gainfile is not specified, make a combined
                    spectrum without any shifts.
                    [Energy] = [Channels] * gain + offset
       sigfile    : (string) name of file which has pixel energy
                    resolution information
       ignorepix  : (int array) list of pixels to be ignored
                    takes precedence over ignore_txt
       ignore_txt : (string) File containing list of dead pixels
                  : produced by noisydead.pro

       top        : (float)  fraction of pixels to coadd for spectrum
                    default = 1.0
       binsize    : (float) binsize of output spectrum (in keV)
       mincounts  : (int)   minimum counts in a pixel for calculating its
                    spectrum. Default = 10
       plotrange  : (2-element float array)
                    Range of output plot in keV. Default = auto
       /help      : Show this help text
 Outputs:
       plotspec   : (string) Name of pdf file to save combined spectrum to
       specfile   : (string) Name of text file to save combined spectrum to
       energy     : (double array) Optional output, x-axis (energies) of
                    the combined spectrum
       spectrum   : (double array) Optional output, y-axis (counts) of
                    the combined spectrum
```

Part 6
Appendix

A Sample testrecords.log file

```
# lines starting with # are ignored
# blank lines are also ignored

# initial trial files
# to be ignored
-1      raw/20110027_102600_UNKNO_UNKNO_UNKNO_UNKNO_UNKNO_200sec_N+D_DATA.raw
-1      raw/20110027_102900_UNKNO_UNKNO_UNKNO_UNKNO_UNKNO_400sec_N+D_DATA.raw
-1      raw/20110027_104900_UNKNO_UNKNO_UNKNO_UNKNO_UNKNO_200sec_N+D_DATA.raw
-1      raw/20110402_181400_UNKNO_Am0_600V_010C_60keV_200.raw
-1      raw/20110402_181900_UNKNO_Am0_600V_010C_60keV_200.raw
-1      raw/20110402_182300_UNKNO_Am0_600V_010C_60keV_200.raw
-1      raw/20110402_182800_UNKNO_Am0_600V_010C_60keV_200.raw
-1      raw/20110402_183300_UNKNO_Am0_600V_010C_60keV_200.raw
-1      raw/20110402_183800_UNKNO_Am0_600V_010C_60keV_200.raw
-1      raw/20110402_184300_UNKNO_Am0_600V_010C_60keV_200.raw
-1      raw/20110402_185200_UNKNO_Am0_600V_010C_60keV_200.raw
-1      raw/20110402_185700_UNKNO_Am_600V_010C_60keV_200.raw
# actual data begins
8       raw/20110402_190300_UNKNO_Am_600V_010C_60keV_200.raw
9       raw/20110402_190800_UNKNO_Am_600V_010C_60keV_200.raw
10      raw/20110402_191300_UNKNO_Am_600V_010C_60keV_200.raw
11      raw/20110402_191700_UNKNO_Am_600V_010C_60keV_200.raw
12      raw/20110402_192100_UNKNO_Am_600V_010C_60keV_200.raw
13      raw/20110402_192500_UNKNO_Am_600V_010C_60keV_200.raw
14      raw/20110402_192900_UNKNO_Am_600V_010C_60keV_200.raw
15      raw/20110402_193300_UNKNO_Am_600V_010C_60keV_200.raw
0       raw/20110402_193800_UNKNO_Am_600V_010C_60keV_200.raw
1       raw/20110402_194200_UNKNO_Am_600V_010C_60keV_200.raw
2       raw/20110402_194600_UNKNO_Am_600V_010C_60kev_200.raw
3       raw/20110402_195000_UNKNO_Am_600V_010C_60kev_200.raw
5       raw/20110702_110300_UNKNO_Am_600V_010C_60keV_200.raw
6       raw/20110702_110700_UNKNO_Am_600V_010C_60keV_200.raw
7       raw/20110702_111200_UNKNO_Am_600V_010C_60keV_200.raw
4       raw/20110702_111800_UNKNO_Am_600V_010C_60keV_200.raw
# no background files in this run
```

B Sample metadata in text files

Text files created by most data analysis codes have metadata at the top, commented out with the `#` symbol. A sample of metadata and the first few data rows for a gain-offset file created by the `linearity IDL` code for FQ1, Module 6 at 5°C is given below. For the purpose of this document, some long lines have been continued on a lower line. These are demarcated with ···, and appear as a single line in the actual file (`FQ1_mod06_ID17285_05C.gain.txt`).

```
# Gain / Offsets file
# Created by :  linearity.pro, Varun Bhalerao, 2014-06-09
# Version :  3.20000
# SVN Revision :  $Rev:  425 $
# Module ID : 17285
# Fits Extension :  1
# Data sets:  /data2/cztidata/vssc/analysis/gainoffset/filelist/···
·····FQ1_modid_17285_pos_06_05C_lin.files.lst
# filename energy(kev) 1sigma_linewidth(channels)
# /data2/cztidata/vssc/FQ4/20140131_FQ4_Co2_05C_4hrs/alphaveto/···
·····20140131_999999_MULTI_Cd+Co2_600_05C_10keV_regular.allspec.fits.gz 122.06 45.0
# /data2/cztidata/vssc/FQ4/20140204_FQ4_Cd_05C_4hrs/alphaveto/···
·····20140204_115525_MULTI_Am3+Cd_600_05C_10keV_regular.allspec.fits.gz 88.16 44.0
# /data2/cztidata/vssc/allquad/20140316_MULTI_Ba_cal401_05C_2.5hrs/alphaveto/···
```

```
......20140316_120041_MULTI_Ba_600_05C_10keV_regular.Q1.allspec.fits.gz 81.00 43.0
# /data2/cztidata/vssc/allquad/20140316_MULTI_Co2_cal401_05C_1hrs/alphaveto/...
......20140316_171357_MULTI_Co2_600_05C_10keV_regular.Q1.allspec.fits.gz 122.06 45.0
# /data2/cztidata/vssc/allquad/20140317_MULTI_Am3_cal401_05C_2.5hrs/alphaveto/...
......20140317_120001_MULTI_Am3_600_05C_10keV_regular.Q1.allspec.fits.gz 59.54 41.0
# /data2/cztidata/vssc/allquad/20140318_MULTI_Cd_cal401_05C_2.5hrs/alphaveto/...
......20140318_071400_MULTI_Cd_600_05C_10keV_regular.Q1.allspec.fits.gz 88.16 44.0
# /data2/cztidata/vssc/FQ4/20140131_FQ4_Co2_05C_4hrs/alphaveto/...
......20140131_999999_MULTI_Cd+Co2_600_05C_10keV_regular.allspec.fits.gz 136.47 48.0
# /data2/cztidata/vssc/FQ4/20140131_FQ4_Co2_05C_4hrs/alphaveto/...
......20140131_999999_MULTI_Cd+Co2_600_05C_10keV_regular.allspec.fits.gz 22.16 60.0
# /data2/cztidata/vssc/allquad/20140316_MULTI_Co2_cal401_05C_1hrs/alphaveto/...
......20140316_171357_MULTI_Co2_600_05C_10keV_regular.Q1.allspec.fits.gz 136.47 48.0
# /data2/cztidata/vssc/allquad/20140316_MULTI_Co2_cal401_05C_1hrs/alphaveto/...
......20140316_171357_MULTI_Co2_600_05C_10keV_regular.Q1.allspec.fits.gz 22.16 60.0
#
# Input gain-offset file:/data2/cztidata/vssc/analysis/go_AmCo/FQ1/05C/...
......FQ1_modid_17285_GO_AmCo_05C_allquad.txt
# Output gain-offset file (this file):/data2/cztidata/vssc/analysis/gainoffset/FQ1/05C/...
......FQ1_mod06_ID17285_05C.gain.txt
# Input gain/offset parameters:
# Mean gain (keV / channel) :  0.05589
# Variation in gain :  0.00123
# Mean offset :  4.67
# Variation in offset :  1.17
# Revised output gain/offset parameters:
# Mean gain (keV / channel) :  0.05644
# Variation in gain :  0.00124
# Mean offset :  4.18
# Variation in offset :  1.08
# [Energy] = [Channels] * gain + offset
#
# Good fits could not be obtained for 9 pixels
# Their gains and offsets were replaced by mean gain-offset values
# Their error bars were set to zero
# The 9 pixels are:  4 36 48 50 61 182 194 226 244
# Gain Gain Err Offset Offset Err
0.05754 0.00050 2.312 0.812
0.05648 0.00047 2.923 0.728
0.05704 0.00025 2.902 0.354
0.05626 0.00026 4.358 0.409
0.05646 0.00000 4.171 0.000
```

C List of all IDL codes

```
alphaspec_ctmap.pro
alphaspec.pro
alphastats.pro
alphaveto.pro
apixmap.pro
background_analysis.pro
backspace.pro
basicproducts.pro
centbox.pro
centroid.pro
checkmod_centfile.pro
countmap.pro
countrate.pro
data_search.pro
dead_list.pro
dead_props.pro
ene_res.pro
ene_res_txtfile.pro
fitline.pro
fitline_run.pro
gainoffset.pro
```

```
gainplots.pro
getmod.pro
goAmCo.pro
go_cluster
go_cluster.r
go_corr_mod.pro
groupspec.pro
inside.pro
linearity.pro
linearity_run.pro
lineprof.pro
lin_quadrant.pro
logup.pro
modid.pro
modres.pro
modspec.pro
noisydead.pro
noisy_list.pro
noisy_props.pro
parsefilename.pro
peakfind.pro
perfcomp.pro
perfcomp_summary.pro
perf_energy.pro
pixgroup.pro
pixsep.pro
pixspec.pro
plotvispix.pro
procpix.pro
ps2pdf_idl.sh
qe_calch.pro
qe_calc.pro
quadprod.pro
rcspec.pro
redist.pro
rmserror.pro
rms.pro
spec_plots.pro
splitproc.pro
stats.pro
temp_res.pro
temp_res_txtfile.pro
temp_res_txt.pro
threshold.pro
totvetospec.pro
```