*Astronomical Data Analysis Software and Systems XII*
*ASP Conference Series, Vol. 295, 2003*
*H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.*

# Calibration of COS data at STScI

Philip E. Hodge

*Space Telescope Science Institute, Baltimore, MD 21218*

**Abstract.** This paper describes the program for pipeline calibration of Cosmic Origins Spectrograph (COS) data at the Space Telescope Science Institute. CALCOS is written in Python. Image and table data are read from and written to FITS files using PyFITS, and the data arrays are manipulated using the numarray module, with C extension code for special cases not included in numarray.

## 1. Introduction

The Cosmic Origins Spectrograph is an ultraviolet spectrograph for installation in the Hubble Space Telescope in 2005. COS is being built by the Center for Astrophysics and Space Astronomy (CASA) at the University of Colorado, and Ball Aerospace. The far ultraviolet (FUV) detector system was built by the Experimental Astrophysics Group (EAG) at the University of California, Berkeley. For further information on the instrument, see `http://cos.colorado.edu/` and `http://ozma.ssl.berkeley.edu/~eagcos/`.

CALCOS performs the basic calibration of COS data, producing a flat fielded 2-D image and a 1-D flux calibrated spectrum. The high level portion of CALCOS uses three classes: Association, Observation, and Calibration. The Association class contains a list of Observation instances and information about the relationships between files in the association. It checks for consistency in the header keywords of all the input files, for example that they were taken with the same detector and grating, and that they use the same reference files. The Observation class contains information for an individual input file. The Calibration class contains the high level calibration methods. The lower level functions that do the actual calibration are procedural.

Further information can be found in Hodge (2002).

## 2. Processing

COS data may be taken in time-tag format (a table of photon events) or in image format (referred to as accum, or histogram mode). Some of the processing by CALCOS is identical for these two formats, partly for consistency but also to reduce duplication of code.

For time-tag data, tabular format is retained for the basic calibration steps. The thermal and geometric corrections are applied by changing the X and Y pixel coordinates of each photon event. Orbital and heliocentric Doppler corrections are done by changing the pixel coordinate in the dispersion direction (X for FUV, Y for NUV). Bad regions on the detector, bad time intervals, and events rejected because the pulse height is out of range are flagged by setting a bit in the data quality column. Flat field and deadtime corrections are applied by assigning a value in a weight column. After applying these corrections, the results are written to an output events table, which is similar in format to the input events table but with additional columns and different data types for some columns. The corrected table of photon events is also binned into an image. For each row in the table, the pixel nearest to the corrected X,Y position of the photon is incremented by the value in the weight column.

For accum image data, if the FUV detector was used, the image will be temporarily converted into a pseudo-time-tag list. For each count in the raw image, the X and Y coordinates of the pixel will be appended to in-memory lists; the time of arrival of the photon cannot be known, so there is no list of times. A pseudo-random number on the interval $[-0.5, +0.5]$ is added to each pixel coordinate to reduce aliasing effects. Thermal and geometric corrections are then applied using the same code as for time-tag data. The lists of X and Y coordinates are then binned back into an image, and subsequent calibration steps (flat field, deadtime) are performed on the image. The heliocentric correction for accum data is done by writing the radial velocity to a header keyword, which is later used during 1-D spectral extraction to shift the wavelengths.

After the basic calibration described above, a 1-D spectrum is extracted from the calibrated image. A wavelength is assigned to each pixel of the extracted spectrum, and the wavelength scale can be shifted to correct for an error in the grating adjustment (see below). One exposure yields two or three noncontiguous sections of spectrum. The FUV detector consists of two separate "segments" with a narrow gap between them. For NUV, three separate "stripes" of the spectrum are focussed onto the detector at one time. The gap between NUV stripes is large, approximately twice the length of a stripe. Each extracted 1-D spectrum is stored as one row in the output table, i.e., two rows for FUV and three rows for NUV, with the wavelength, flux, etc., stored as arrays in each row.

The mechanism to select or reposition a grating is not perfectly repeatable. For each exposure at a given position, the offset from the nominal position is determined using a "wavecal," an exposure using an internal emission-line lamp. A cross correlation of the 1-D extracted wavecal spectrum with a template spectrum gives the offset. The individual spectra within an exposure (for the two FUV detector segments or the three NUV spectral stripes) are processed independently, rather than averaging them to yield one offset per exposure. If multiple wavecal observations are available and two wavecals bracket a science observation, the offset to apply to the wavelength scale for the science observation is determined by linear interpolation with time. Two diagnostic tests have been implemented, to catch gross errors such as a failure of the grating mechanism: (1) the width of the cross correlation should be small, comparable to the spectral resolution; and (2) the maximum value of the cross correlation should be much larger than the median value.

The default observing mode is to take multiple exposures at four slightly different positions, offset in the dispersion direction, to smooth out flat-field irregularities and avoid detector defects. It is also possible to take multiple exposures at the same grating position, or to take single exposures. When multiple exposures were taken, an additional output file will be written that contains the averages of the individual 1-D extracted spectra. The flat-fielded images will also be averaged, for repeated exposures at the same grating position.

## 3. Python, PyFITS, Numarray, C

FITS files are used for raw data input, reference files (such as flat fields and dispersion coefficients), and calibrated output files. In all cases, the data are stored in FITS extensions, not in the primary header/data unit. Keywords in the primary header of COS data files specify the instrument configuration, which calibration steps to perform, and the names of the reference files to use for calibration. Binary table extensions are used for input time-tag data, calibrated time-tag data, output 1-D extracted spectra, and most reference files. Image extensions are used for input accum data, calibrated accum data, calibrated time-tag event data after binning into an image, and flat field and geometric distortion reference files.

CALCOS is written in Python, with some C code for row-by-row or pixel-by-pixel operations. The PyFITS module is used for FITS file I/O. The numarray module supports efficient array operations, and array arithmetic is as simple as scalar arithmetic. An image data array is represented in PyFITS as a numarray object, and a table is a 1-D array of records (rows). Each column of a table is a numarray object (or chararray for text strings). The numarray 'strides' attribute allows column access directly within the table data, without making a separate copy of the column. The total size and structure of a numarray object are fixed when the array is created (e.g., the columns and their data types, and the number of rows). Rather than actually deleting rows that are rejected during processing, CALCOS flags them as bad in the data quality column, because this doesn't change the number of rows; it also has the advantage of being reversible.

Most arithmetic operations use numarray. There are some operations that are not sufficiently generic to have been implemented in numarray, however, and for these a C extension module was written. A comparison of the flat field calibration step for time-tag and for accum data illustrates this issue. Time-tag data are corrected individually for each photon event (table row), and the entire procedure uses C code. For each row, the pixel coordinates of the event are gotten from the X and Y columns. The coordinates are rounded to the nearest integer, and the value is taken at that pixel in the flat field reference image. The weight for the event is then set equal to the reciprocal of the flat field value. A combination of C code and numarray operations are used for accum data. The first step is to convolve the flat field image in the dispersion direction, to account for the orbital Doppler shift during the exposure. (The on-board software shifts the pixel location before incrementing the image array in memory, so the flat field should be shifted by the same amount before being applied to the image.) This is done using C code, because it is a 1-D convolution of a 2-D image. After this, however, the actual flat-field calibration is just a division of two arrays, using numarray arithmetic. If the flat field is a subset of the entire detector (which would be the case for FUV data), just the matching subset of the input image will be corrected by using standard Python slice notation.

## 4. Memory Considerations

CALCOS processes the data in-memory, and some of the data files are in the 100-Mb range. The NUV detector is only 1K by 1K pixels, but the FUV detector is 16K by 1K. CALCOS writes error and data quality arrays in addition to the science data. Calibration takes approximately five minutes for a 100-Mb FUV time-tag file on a fast Sun or on an Intel/Linux machine. A typical FUV observation would include four exposures at slightly different positions, with perhaps four associated wavecal exposures. Since there are two FUV detector segments per exposure, this would involve a total of 16 files for the raw data. These 16 files are related, and CALCOS calibrates them as a set (one at a time, but within the same process), rather than independently. In-memory arrays (numarray objects) will be repeatedly created and destroyed as these files are processed. Memory allocation is handled efficiently on Intel/Linux machines, and 0.5 Gb of memory is sufficient for calibrating 100-Mb time-tag files. Using Sun/Solaris, however, memory is not necessarily available for reuse after being freed. Processing appears normal for the first few files (depending on the available real memory), but then it bogs down and may grind to a halt. Two Gb of memory appears to be sufficient, but 0.5 Gb is clearly not enough, due to the accumulation of allocated memory while processing multiple files. If sufficient memory is not available, a fallback option is to spawn a separate process for doing the basic calibration of each file. After that has completed for all of the files, the remainder of the calibration (wavecal processing, 1-D extraction, averaging of results) can be done. At the present time, memory mapping has been partially implemented in PyFITS, and further development is planned. Initial tests show that memory mapping does help. Raw accum image data use BZERO (in order to store unsigned 16-bit data in FITS files), and they need to be copied to scratch in order for the zero-point offset to be applied. This would defeat memory mapping for the raw data, but accum mode is expected to be used far less frequently than time-tag. It is likely that the CALCOS Python code will need to be customized in order to take full advantage of memory mapping.

**Acknowledgments.** The calibration algorithms were developed by the COS instrument development team together with COS instrument scientists at STScI.

## References

Hodge, P. 2002, in ASP Conf. Ser., Vol. 281, Astronomical Data Analysis Software and Systems XI, ed. D. A. Bohlender, D. Durand, & T. H. Handley (San Francisco: ASP), 273