MODS

Basic CCD Reduction with modsCCDRed

Document Number:	OSU-MODS-2012-002
Version:	2.0.1
Date:	2019 Feb 2
Prepared by:	R.W. Pogge The Ohio State University

Distribution List		
Recipient	Institution/Company	Number of Copies
Richard Pogge	OSU	[author]
Mark Wagner	LBTO	1 (PDF)
Rebecca Stoll	OSU	1 (PDF)
Dave Thompson	LBTO	1 (PDF)
Olga Kuhn	LBTO	1 (PDF)

Document Change Record			
Version	Date	Changes	Remarks
0.1	2012-03-28		First draft
0.2	2012-04-05	many and varied	internal beta release
0.3	2012-04-09	Added installation docs	first public beta release
0.4	2013-10-18	Started for MODS2	Not released
0.5	2018-07-22	Astropy.io fits handling	Release for handover
2.0	2019-02-02	GitHub Release	

Contents

1 Introduction

1.1 Scope

This document describes how to perform basic 2D data reduction steps unique to the MODS science CCD detectors. It describes the basic procedures, and how to use a suite of Python programs (modsCCDRed) to carry out these reductions. This document also describes the algorithms for those who may wish to implement the procedures in another package (e.g., IDL).

After processing with the modsCCDRed package, the user will have a set of 2D spectral images ready for further spectroscopic reduction and calibration with their standard astronomical packages (IRAF, IDL, etc.).

1.2 Reference Documents

1. *MODS Instrument Manual,* v1.2, R. Pogge, OSU-MODS-2011-003. (http://www.astronomy.ohio-state.edu/MODS/Manuals/MODSManual.pdf)
2. *NumPy 1.6 User Guide*, 2011 May 15, numpy.scipy.org
3. *NumPy 1.6 Reference Guide*, 2011 May 15, numpy.scipy.org
4. Astropy FITS File handling (astropy.io.fits), Release 3.0.3 (docs.astropy.org/en/stable/io/fits)

1.3 List of Abbreviations and Acronyms

Abbreviation	Description
2D	Two Dimensional
ADU	Analog-to-Digital Unit (sometimes called Data Numbers or DN)
ASCII	American Standard Code for Information Interchange
BPL	Bad Pixel List
CCD	Charge-Coupled Device
FITS	Flexible Image Transport System
Gb	Gigabyte
IDL	Interactive Data Language (ITT Visual Systems, Inc.)
IRAF	Image Reduction and Analysis Facility (NOAO)
LBT	Large Binocular Telescope
LBTO	LBT Observatory
MODS	Multi-Object Double Spectrographs
MOS	Multi-Object Spectroscopy
NOAO	National Optical Astronomy Observatory
OSU	The Ohio State University
OTF	Overscan, Trim, and Flat-Field
ROI	Region of Interest
TBD	To Be Determined

1.4 Organization of this Manual

The manual is organized top-down. The first section describes the MODS CCD format and readout architecture, and outlines the reduction steps required for bias, flat field, and bad pixel correction. The following section describes how to use a suite of Python programs (modsCCDRed) to perform these reduction steps. The manual then describes the details of the individual Python programs and the algorithms they use. All users should read the first section so they know why MODS requires special initial reduction procedures, then read the second section describing how to use the modsCCDRed tools. The final section describes the programs and their options in detail, as well as giving installation instructions.

This guide assumes you have read the MODS Instrument Manual and are familiar with the basic principles of CCD detector operation and standard imaging and spectroscopic calibration procedures.

1.5 System Requirements

The modsCCDRed Python programs have been tested on Linux (CentOS 5.0 and 6.0) and Mac OS X (v10.7 Lion) operating systems. Python 2.3 or later is required, and the programs have been used primarily with Python versions 2.3 through 2.7 to date. We strongly recommend adoption of the Anaconda Python distribution (www.anaconda.com) for all operating systems – we have found it uniformly stable and reliable.

Two Python modules are required:

numpy (numpy.scipy.org)
astropy.io.fits (docs.astropy.org/en/stable/io/fits)

This version of the program only runs on Python 2 system, and has been tested up to the last version, Python 2.7. It is also the last version to be distributed from the OSU Astronomy Department website.

The Python 3 port will be backwards compatible with Python 2, and distributed on GitHub. Release is expected in mid-2019.

Your computer should have at least 4Gb of memory in order to be able to perform the image median stacking steps, but otherwise the programs work with single images.

Runtime performance will depend on memory and processor speed, and is somewhat I/O limited given the large (24Mpixel) images involved. On an older 2-processor 2.8GHz Pentium 4 computer with 8Gb of RAM running CentOS 5.8, the time required to bias, flat-field, bad column fix and axis flip a single full-frame unbinned 8×3K MODS red-channel spectrum is about 13 seconds.

1.6 Installing modsCCDRed

The modsCCDRed programs are written in standard Python (v2.7), and requires numpy and astropy to run (we recommend using Anaconda). Both modules must be installed on your system in order for modsCCDRed to work.

Starting with version 2.0 we are distributing modsCCDRed using GitHub.

1.6.1 Downloading modsCCDRed

The modsCCDRed package is available on GitHub:

github.com/rwpogge/modsCCDRed

If you are familiar with using git on your host computer, you can install the package directly with by cloning this repository. First go to the folder where you want to keep all your programs, and then execute the "git clone" command to create a modsCCDRed/ folder and download the distribution:

```
cd /path/to/my/programs
git clone https://github.com/rwpogge/modsCCDRed.git
```

If you don't want to use git, go to the GitHub repository above, and then download a zip file (`modsCCDRed-master.zip`) into the folder where you want the code to reside, then go to that folder and unzip the file:

```
cd /path/to/my/programs
unzip modsCCDRed-master.zip
```

this will create the `modsCCDRed-master` directory and fill it with the programs and support files. Rename this folder to just `modsCCDRed`

```
mv modsCCDRed-master modsCCDRed
```

Either way, you will now have a `modsCCDRed` folder containing the individual python programs, readme and license files, and a single subfolder named `database/` with the default bad pixel list (bpl) files for the MODS CCD detectors.

1.6.2 Installation

You have two options: Personal or Public installation

For a personal installation, you have two ways to install the files to use:

1. Keep the modsCCDRed Python programs in place, and put the modsCCDRed directory in your default execution path.
2. Copy the executable Python programs into your ~/bin/ or ~/programs/ directory where you put executables in your default execution path.

For a public installation, e.g., on a central disk to share one copy of the package among many users, we suggest logging in as root and unpacking the tarball in your usual place for public add-on programs, e.g., `/usr/local/`, and then installing the executables in, e.g., `/usr/local/bin/`.

1.6.3 Common Bad Pixel Lists

To use the default bad pixel lists, users need to define this environment variable

```
setenv MODS_DBDIR /usr/local/LBT/modsCCDRed/database/
```

for a csh/tcsh shells with a public installation where the source code lives in `/usr/local/LBT/`. Other shells (e.g., bash) use a different syntax.

Users can always override the default bad pixel list used with one of these methods:

1. Redefining the `MODS_DBDIR` environment variable to point to the directory with their personal copies of bpl files with the same names (e.g., `mods1r.bpl`).
2. Undefining `MODS_DBDIR`, at which point the modsCCDRed programs default to the current working directory (`./`), and look for the standard names.
3. Using the –l flag with no path to use a custom file in `MODS_DBDIR` (or ./ if undefined), e.g.,
   ```
   modsFixPixl -l mym1r.bpl
   ```
4. Using the –l flag to give the full path and name of a particular bad pixel list file to use, e.g.,
   ```
   modsFixPix -l /my/path/to/mym1r.bpl
   ```

There is no requirement of using the MODS_DBDIR environment variable, but care should be taken to avoid having multiple, possibly conflicting versions of bad pixel lists spread through many directories.

1.6.4 Updates

If you have an old version of modsCCDRed you did not get from the GitHub repository, go through a completely new installation as described above.

github.com/rwpogge/modsCCDRed

You can either download a zip file.

If you have a version you got from GitHub, you can update in place using the git command

```
cd /path/to/wherever/you/put/modsCCDRed
git pull origin master
```

which will pull and commit any updates from the GitHub repository, or tell you that your version is up to date.

2 MODS Raw CCD Images

2.1 MODS CCD Detector Readout

The MODS science CCDs are e2v Technologies, Ltd. CCD231-68 monolithic backside-illuminated 8192×3088 15μm pixel CCDs operated with OSU MkIX detector controllers. Their detailed properties are described in the MODS Instrument Manual.

The MODS CCDs are divided into 4 quadrants that are read out simultaneously. Each readout channel is further split into even and odd readout channels, for a total of 8 readout channels. The even/odd readouts are staggered (the even channel reads while the odd channel's readout amplifier is resetting), allowing us to read this large (24Mpixel) array faster. The readout geometry is shown schematically in Figure 1. The thin strips at the left and right are the overscan columns.

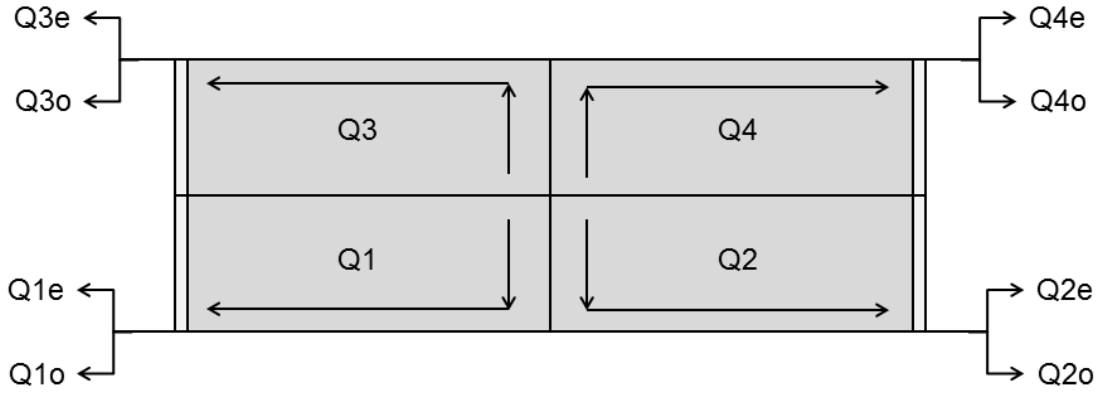

Figure 1: MODS CCD Readout Geometry

Each of the 8 output channels has its own DC bias level and conversion gain. Figure 2 shows examples of blue and red CCD full-frame bias images showing the different mean bias levels in each quadrant (left panel), and the even/odd pixel bias level differences (right panel).

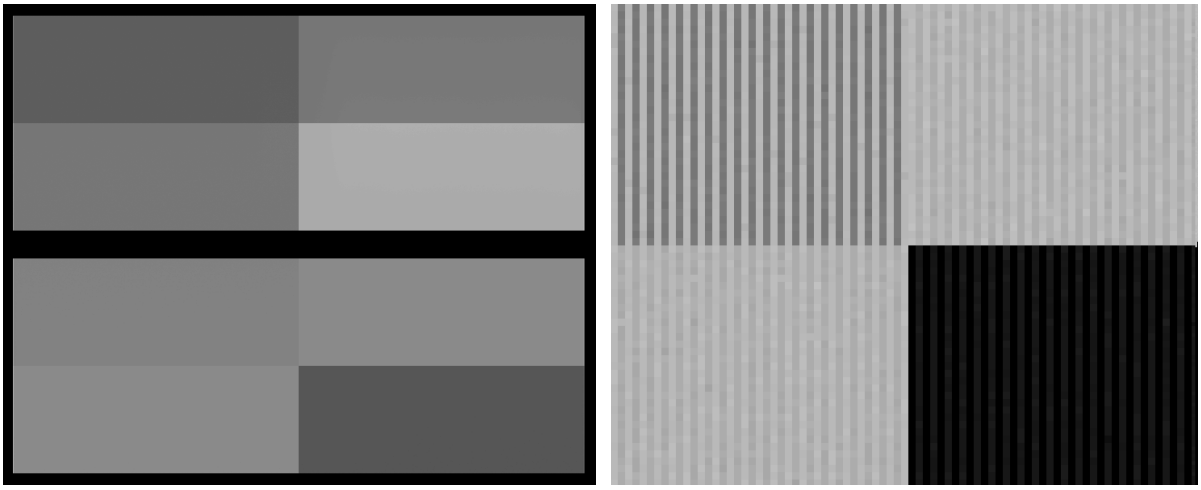

Figure 2: **Left:** MODS1 Bias Images (top: blue, bottom: red). Right: zoom in on the center of the Red bias image showing the even/odd effect in the bias.

The difference in the bias level between the even and odd pixel readout channels is larger than typical pixel-to-pixel variations due to readout noise. This is why the even/odd vertical striping pattern is so pronounced in raw images. In illuminated images like flat fields this "even/odd effect" is further enhanced by small (few percent) differences in the conversion gain (e^-/ADU) between the readout channels. The bias component can be removed using the overscan regions of the detector, while the conversion gain component can be corrected for using color-normalized spectra of continuum sources ("spectral pixel flats").

This readout scheme is unique to MODS, and in practical terms this means that raw MODS images cannot be bias corrected using the usual IRAF or IDL tasks, which assume that all pixels within a quadrant are read through a single output channel. Instead, we provide the modsCCDRed suite of Python programs to perform these basic 2D CCD reduction steps.

2.2 MODS Basic 2D CCD Reduction Procedure

The basic steps needed to bias and flat field MODS data to take account of the unique readout scheme are as follows:

1. Prepare normalized spectral flat field images:
   a. Bias correct and trim the flat fields
   b. Median combine the bias-corrected flats
   c. Fix bad columns using the bad pixel list for the detector
   d. Remove the color term to create a normalized "pixel flat"
2. Bias and flat field science target, standard star, and comparison lamp spectra, creating 2D images ready for subsequent reduction and analysis.

The modsCCDRed programs that perform the high-level tasks are:

`modsBias.py` – Bias correct images using overscan columns, and trim off overscan

`modsFixPix.py` – fix known bad columns listed in a bad pixel list (bpl) file

`modsPixFlat.py` – create a normalized, color-free "pixel flat" from a slitless spectral flat field image

`modsProc.py` – apply bias, bad pixel, and flat-field correction to science target, standard star, and comparison lamp spectra.

Details of how these programs are given in §4.

In addition, a set of convenience functions are provided for basic image arithmetic common to these steps:

`modsMedian.py` – median combine a list of images

`modsAdd.py` – add together one or more images

`modsSub.py` – subtract one image from another

These allow you to perform most of the basic 2D data reduction functions without having to use a second package like IRAF for intermediate steps.

3 Reducing Raw MODS Images with modsCCDRed

In this section we are showing the most common use: full-frame grating-mode dual-channel spectroscopy (long-slit or MOS, these are steps common to both). Steps for single-channel (blue-only or red-only) grating mode spectra are similar. The commands in this example are mostly used without command-line options; see the full command descriptions in §4.

3.1 Prepare for 2D Reduction

In the working directory, have copies of all of the raw data. For this example, we will have data with shortened versions of the usual raw filenames, e.g., `m1r.0001.fits` instead of `mods1r.20120329.0001.fits`.

You also need copies of the bad pixel lists for the two CCDs. For MODS1, these files are named

```
mods1r.bpl
mods1b.bpl
```

You have two options:

1. Put copies of these files in the same directory as the data and make sure that the `MODS_DBDIR` environment variable is not defined.
2. If you have defined a common MODS "database" directory, set the `MODS_DBDIR` environment variable to point to that directory, for example
   `setenv MODS_DBDIR /home/darkstar/pogge/MODS/Calib`
   and put the copies of the bad pixel lists there. When this environment variable is set, the modsCCDRed programs will look there by default if no custom bad pixel list is given on the command line (see §4.2).

The basic 2D image processing is described in the following sections.

3.2 Create the Normalized Color-Free Pixel Flat

For this procedure, we are assuming you have taken standard slitless spectral flats of the internal continuum lamps (e.g., using the grpixflats.cal script):

Red Channel: 5 images, `m1r.0001.fits` thru `.0005.fits`
Blue Clear Filter: `m1b.0001.fits` thru `.0005.fits`
Blue UG5 Filter: `m1b.0006.fits` thru `.0010.fits`

Step 1: Bias correct the raw spectral flats

```
modsBias.py m1r.000[1-5].fits
modsBias.py m1b.000[1-9].fits m1b.0010.fits
```

You will now have overscan bias and trim corrected images with names like

```
m1r.0001_ot.fits
```

The "`_ot`" suffix means "overscan [bias] and trim". Details of the bias correction process are given in §4.1.

Step 2: Median combine the flats

```
modsMedian.py m1r.000[1-5]_ot.fits rflat_med.fits
```

```
modsMedian.py m1b.000[1-5]_ot.fits bclrFlat_med.fits
modsMedian.py m1b.000[6-9]_ot.fits m1b.0010_ot.fits ug5Flat_med.fits
```

This yields three sets of median slitless flat field images

```
rflat_med.fits
bclrFlat_med.fits
ug5Flat_med.fits
```

Step 3a: Create the Normalized Red Pixel Flat

```
modsFixPix.py rflat_med.fits rflat_fix.fits
modsPixFlat.py rflat_fix.fits rpixFlat.fits
```

The first program (modsFixPix.py) uses the default bad pixel list (mods1r.bpl) that came with the modsCCDRed package (or obtained from the MODS website) to fix the known bad columns on the red CCD detector, producing the "fixed" median flat "rflat_fix.fits".

The second program (modsPixFlat.py) removes the color term from the median slitless flat field. An example of a red normalized color-free pixel flat is shown in Figure 3.

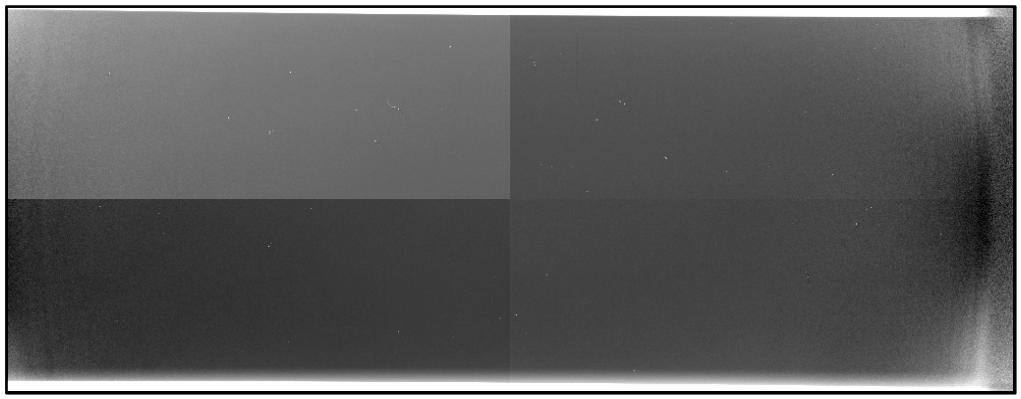

Figure 3: Normalized pixel flat from running modsPixFlat.py. The arc-shaped artifact at the right is due dramatic drop in signal well blue-ward of the dichroic cross-over wavelength.

A detailed breakdown of the pixel flat normalization process is given in §4.3.

Step 3b: Create the Normalized Blue Pixel Flat

For the blue pixel flat, we first combine the Clear and UG5 filter flats, the latter being used to boost the counts at the far blue end of the spectrum, and then create the normalized pixel flat using the same procedure as for the red pixel flat, as follows:

```
modsAdd.py bclrFlat_med.fits ug5Flat_med.fits bflat_med.fits
modsFixPix.py bflat_med.fits bflat_fix.fits
modsPixFlat.py bflat_fix.fits bpixFlat.fits
```

At end, we have two normalized, color-free pixel-to-pixel flat fields for the red and blue channel spectra:

```
rpixFlat.fits
bpixFlat.fits
```

Each has a mean value of 1.0, so when you divide a science or calibration image with these flats, it will preserve the raw signal levels to first order. The color response of the spectra will be calibrated explicitly with the standard star spectra.

3.3 Process the 2D Science and Calibration Spectra

Now that normalized color-free pixel flats have been prepared, the second step is to apply bias flat field corrections to the science and calibration frames. The processing will includes fixing known bad columns specified in the `mods1r.bpl` and `mods1b.bpl` bad pixel list files.

For this example, we have two images:

`m1r.0015.fits` – red science target spectrum, flat field = `rpixFlat.fits`
`m1b.0011.fits` – blue Ne comparison lamp spectrum, flat field = `bpixFlat.fits`

The reduction uses the modsProc.py script:

```
modsProc.py –b m1r.0015.fits rpixFlat.fits
modsProc.py –b m1b.0011.fits bpixFlat.fits
```

The output files will be

```
m1r.0015_otf.fits
m1b.0011_otf.fits
```

The suffix "`_otf`" means "overscan, trim, and flat". If we omitted the –b option, it would just bias and flat field the data without fixing bad columns (see §4.4 for a complete list of modsProc.py options)

The red OTF image is also flipped along the X axis so that the processed image has wavelengths increasing (blue to red) with increasing X pixel coordinate. The raw images have this order flipped. Blue images do not need to be flipped as the raw images are already aligned to have wavelengths increasing with X. An example of the results for a long-slit red spectrum is shown in Figure 4.

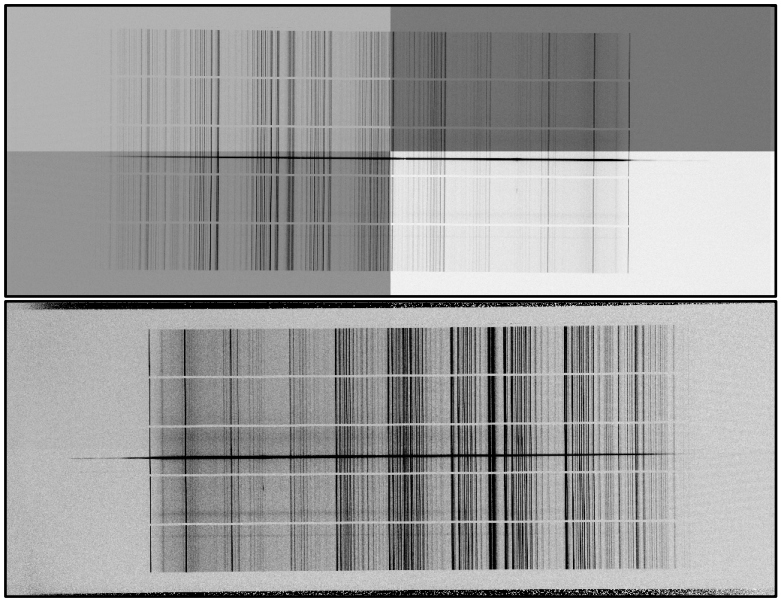

Figure 4: MODS1 red grating spectrum before (top) and after (bottom) modsProc.py processing.

A detailed breakdown of the modsProc.py processing steps is given in §4.4. A workflow diagram summarizing all of the steps described above is shown in Figure 5 in §3.4.

3.4 Basic 2D Image Reduction Work Flow

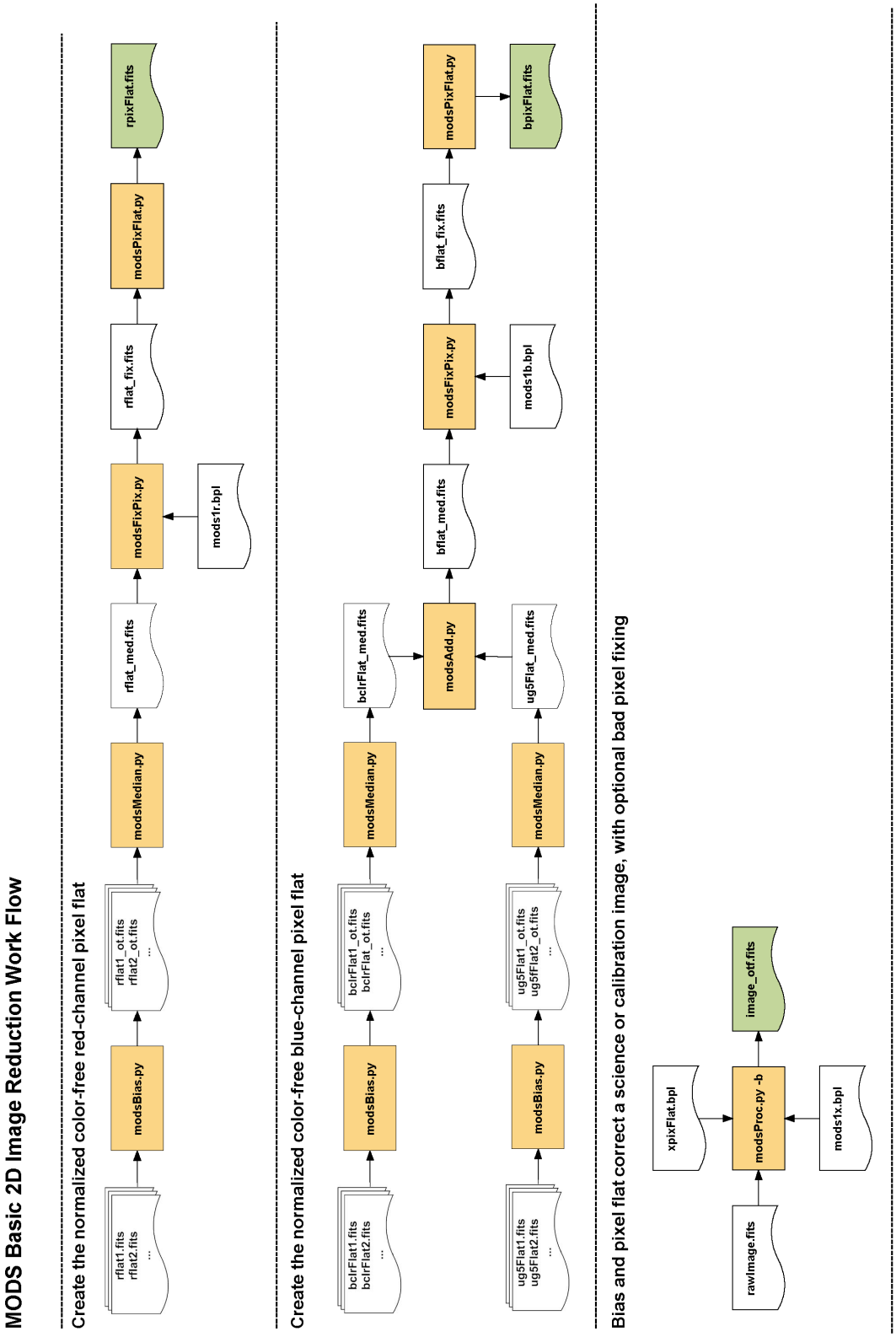

Figure 5: Summary of the MODS Basic 2D reduction work flow.

4 The modsCCDRed Programs

In this section we describe each of the programs in the modsCCDRed package and describe how they work.

All of the programs in modsCCDRed share a number of common properties:

1. Typing a command without arguments will print a brief "usage" message giving the command syntax and list of options.
2. When creating new files the default behavior is a "no clobber" rule that it will prevent you from overwrite an existing file. This can be overridden using the –f ("force") option, similar to the convention used by the Unix cp and mv commands.
3. All have a –V (--version) option that will report the program's version number and date of release and exit without further execution.
4. All do their work quietly unless there are errors that abort execution. More detailed verbose execution progress information can be viewed using the –v option.

4.1 **modsBias.py** – Bias correct MODS CCD images

Usage: **modsBias.py [options] rawFile1.fits [rawFile2.fits...]**

Where:

rawFile*.fits are the names of raw MODS files

Options:

-f force overwrite of existing output files ("clobber)
-v print verbose debugging information during execution
-V print version information and exit (no execution)

modsBias.py removes the quadrant-by-quadrant even/odd bias from one or more raw MODS images. The output files will have the _ot suffix appended (ot = Overscan and Trim).

The list of raw images may be given using standard Unix command-line wildcard expressions, thus

```
modsBias.py mods1r.20120328.00*.fits
modsBias.py mods1r.20120329.002?.fits
modsBias.py mods1b.20120328.000[1-8].fits
```

are all permitted. Because images are read in only as needed, there is no effective limit on the number of images that may be processed in one command session.

modsBias.py works as follows:

1. Divides the raw image into quadrants. Each quadrant consists of a data and overscan section (see Figure 1).
2. Deinterlaces each quadrant into even and odd overscan and data pixel sections.
3. For each line of pixels in a deinterlaced data section, subtracts the median bias level in that line of the deinterlaced overscan section.
4. Re-interlaces the even and odd bias-subtracted data pixels back into the quadrant

5. Trims off the overscan regions, leaving the bias-subtracted data section.

In the output FITS image, the following header cards are written summarizing the bias subtraction:

BIASPROC	Type of overscan procedure (currently only biasSingle)
BIASQnE	Bias level subtracted from even pixels in quadrant n (n=1..4)
BIASQnO	Bias level subtracted from odd pixels in quadrant n (n=1..4)

A HISTORY card is written recording the program version and date.

Note: this only removes the bias component of the 8-channel readout. An example is shown in Figure 6 for the center (four-corners) section of a bias image.

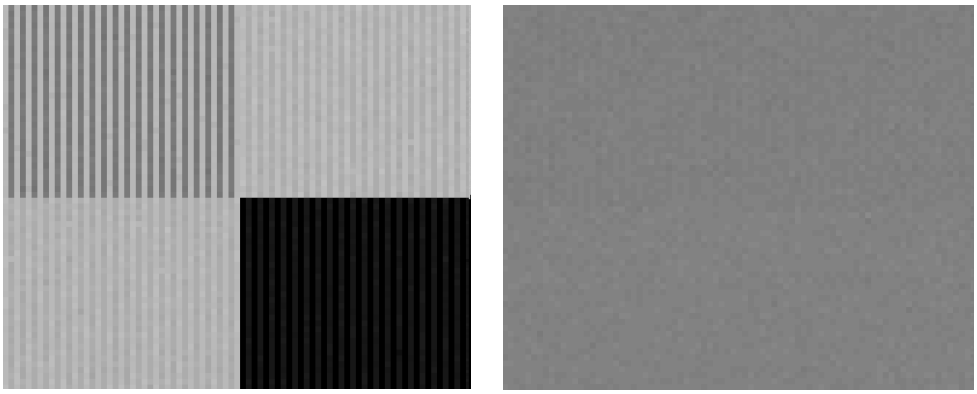

Figure 6: The four-corners region of a bias image before (left) and after (right) modsBias.py correction.

The mean bias level and even/odd effect are completely corrected by modsBias.py, leaving only the nominal readout noise in the output image.

The differences in conversion gain between channels, however, will remain visible in illuminated images until a flat field is applied. An example is shown in Figure 7, a zoom into the center of a science spectrum that has only been processed with modsBias.py:

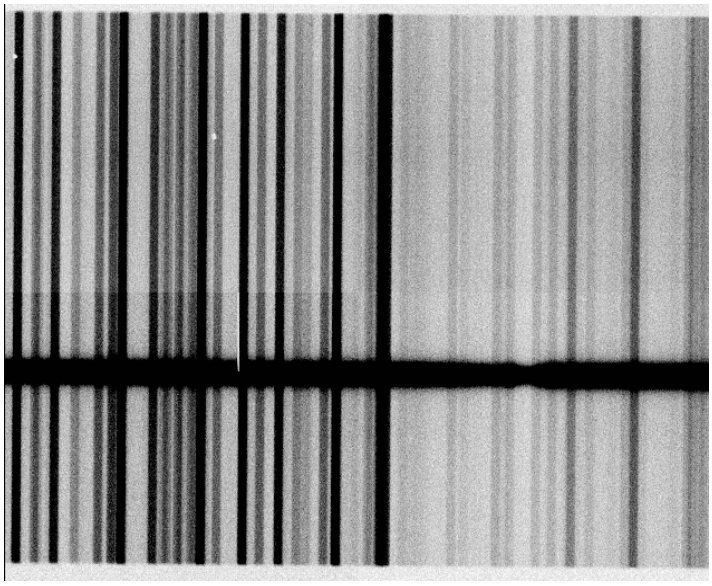

Figure 7: The four-corners region of a red long-slit spectrum processed with modsBias.py showing the residual quadrant pattern due to small quadrant-to-quadrant differences in conversion gain.

These remaining differences between quadrants and between even/odd pixels within quadrants are subtle, and will remain until the image is flat field corrected.

4.2 **modsFixPix.py** – Fix bad columns in MODS CCD images

Usage: **modsFixPix.py [options] inFile.fits outFile.fits**

Where:

inFile.fits	input FITS image file to fix
outFile.fits	output FITS file to create

Options:

-w avgWid	width of the averaging region to use (default 5 pixels)
-l bplFile	name of the bad pixel list (bpl) file to use
-f	force overwrite of existing output files (“clobber)
-v	print verbose debugging information during execution
-V	print version information and exit (no execution)

modsFixPix.py fixes bad columns on a MODS CCD image by replacing bad pixels within a region with the median of "good" pixels flanking that region by ±5 pixels. This default width may be overridden with the `-w` flag. The median in the flanking regions is computed and the pixels replaced by $0.5\times(\text{med}_{\text{left}}+\text{med}_{\text{right}})$.

The default bad pixel list is created from the INSTRUME keyword in the FITS header, for example: `mods1r.bpl` or `mods2b.bpl`. If the `MODS_DBDIR` environment variable is undefined, modsFixPix.py will look in the current working directory for the bad pixel list, otherwise it looks in `MODS_DBDIR` if no explicit path specification is given with the filename. A custom bad pixel list file may be applied by using the `-l` option.

Bad Pixel List Format:

The bad pixel list format is the same as used by IRAF: a 4-column ASCII text file containing the coordinates of the corners of the bad columns with spaces between the values:

```
#
# Bad pixel list for my CCD
#
xstart1 xend1 ystart1 yend1
xstart2 xend2 ystart2 yend2
...
```

The units of the XY coordinates are unbinned pixels starting with pixel (1,1) as the origin at the lower left-hand corner of quadrant 1 (see Figure 1). The relation between the bad column region and the averaging width is shown in Figure 8.

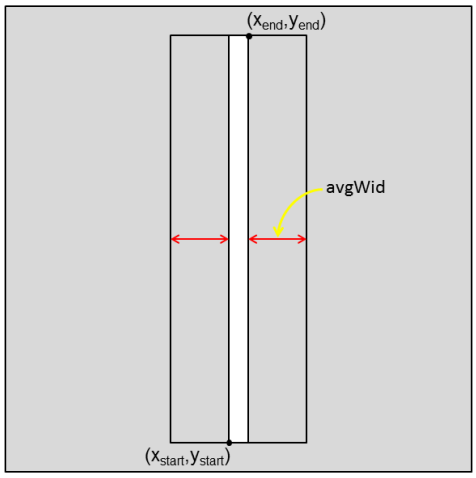

Figure 8: Bad Column and averaging width coordinates

In the output FITS image, the following header cards are written summarizing the pixel fixing:

BPLFILE	Name and path of the bad pixel list (bpl) file used
BPWIDTH	Width of the flanking averaging regions in pixels

A HISTORY card is written recording the program version and date.

4.3 **modsPixFlat.py** – Create a color-free normalized pixel flat field image

Usage: **modsPixFlat.py [options] inFlat.fits pixFlat.fits**

Where:

inFlat.fits	input bias-corrected flat field image
pixFlat.fits	output normalized pixel flat to create

Options:

-f	force overwrite of existing output files ("clobber)
-v	print verbose debugging information during execution
-V	print version information and exit (no execution)

modsPixFlat.py creates a normalized, color-free pixel flat from a bias-corrected MODS spectral flat field image. The processing steps are as follows:

1. Start with a median combined slitless spectral flat field image (Figure 9). The modsMedian.py program can create this (see §4.5.1).
2. Compute the mean color term by summing over all lines the slitless flat.
3. Smooth the mean color term with a 2-pixel boxcar to suppress any residual even/odd effect (even/odd gain difference, modsBias.py has already removed even/odd bias differences). An example is shown in Figure 10.
4. Divide each line of the slitless flat by the smoothed average color term. The result is a color-free pixel-to-pixel flat field image that has a mean data value of about 1.0 (Figure 11).

The following sequence of images shows the results at each of the processing above.

Start with a median slitless grating flat field, bias corrected and bad pixels fixed (Figure 9).

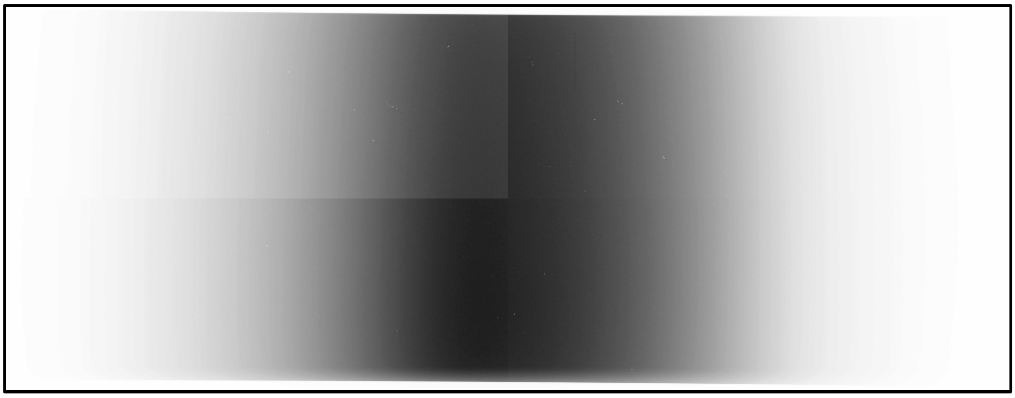

Figure 9: MODS1 red-channel median slitless grating flat field.

Extract the smoothed, mean color term derived from the average over lines (Figure 10).

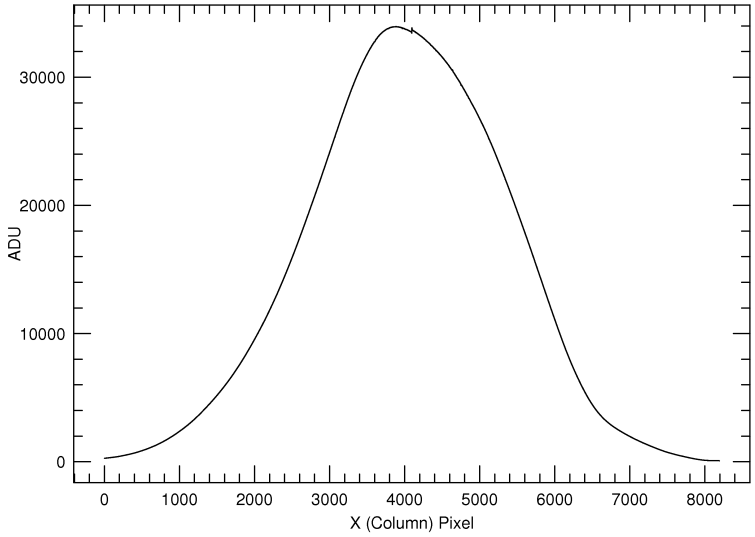

Figure 10: Smoothed mean color term for the slitless flat shown in Figure 9.

Divide each line of the median flat by the color vector to create a normalized color-free pixel flat field (Figure 11).

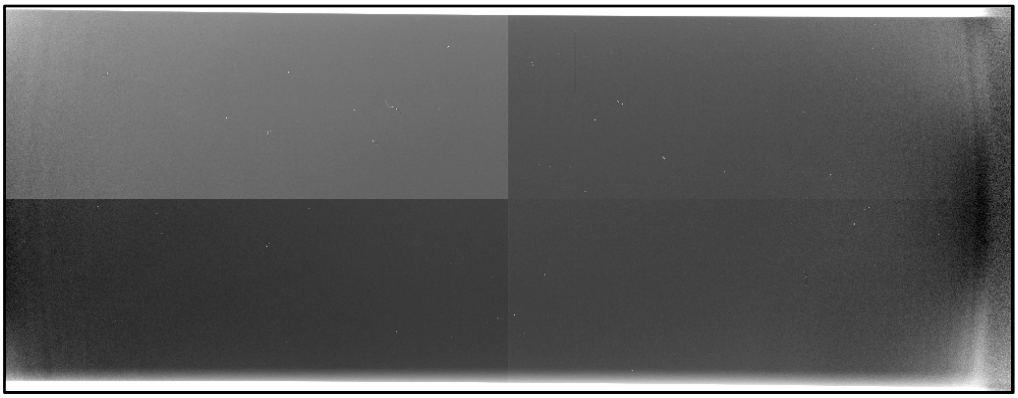

Figure 11: Normalized pixel flat resulting from running modsPixFlat.py on the slitless grating flat in Figure 9.

This normalized color-free pixel flat preserves the even/odd pixel and quadrant-to-quadrant differences in the detector gain. This is seen in a close-up in Figure 12.

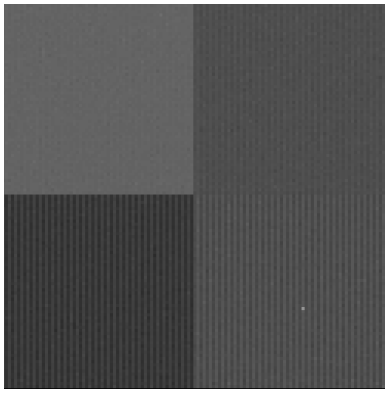

Figure 12: The four-corners region of the normalized pixel flat shown in Figure 11.

The output FITS image will preserve the FITS header of the input flat field image. In the output FITS image, a HISTORY card is written recording the program version and date along with the name of the original input file.

4.4 **modsProc.py** – Create a color-free normalized pixel flat field image

Usage: **modsProc.py [options] rawFile.fits pixFlat.fits**

Where:

rawFile.fits	raw MODS FITS image to be processed
pixFlat.fits	normalized pixel flat to apply

Options:

-b	fix bad pixels (see –l and –w to override defaults)
-l bplFile	bad pixel list (bpl) file to use (requires –b)
-w avgWid	width of the averaging region for bad pixel regions (default: 5 pix)
-f	force overwrite of existing output files (“clobber)
--noflip	do not flip a Red channel image X-axis X (ignored if blue channel)
-v	print verbose debugging information during execution
-V	print version information and exit (no execution)

modsProc.py processes a raw MODS CCD image as follows:

1. Subtract the quadrant-by-quadrant even/odd overscan bias (see modsBias.py, §4.1) and trim the overscan region.
2. Divide by the named normalized color-free pixel flat (see modsPixFlat.py, §4.3)
3. If –b option is given, fix bad pixels. The –l and –w options work the same as with modsFixPix.py (see §4.2).
4. If a red channel image, flip the X axis so that in the final output image the wavelength axis is oriented so that wavelength increases (blue to red) with increasing X. This step is not needed for blue channel images. Can be omitted with the `--noflip` option.
5. Write the image with the “`_otf`” suffix to mark it as “Overscan, Trimmed, and Flattened”.

modsProc.py encapsulates all of the basic reduction steps into one command for reducing science and calibration images.

As described in the MODS Instrument Manual, the red/blue channel optical geometry means that red-channel spectra map onto the red CCD with wavelength decreasing (red to blue) with increasing X pixel. Blue channel spectra, however, map into wavelength increasing with X. modsProc.py takes out this x-axis flip in red spectra automatically in processing. This is why we recommend processing science and calibration frames with modsProc.py. This feature can be overridden using the `--noflip` option. For blue-channel spectra this has no effect.

At present, bad pixel fixing is **optional**, and may be enabled using the –b option. The same –l and –w options are provided to let you use custom bad pixel lists or averaging widths. See §4.2 for details.

The output file's FITS header preserves the FITS header from the original file. Additional HISTORY and other processing keywords are added to record the processing. See the descriptions of modsBias.py and modsFixPix.py for header keywords added by their functions.

4.5 Convenience Functions

These generic image arithmetic functions are not MODS-specific, and are provided as convenience functions so that all of the basic tasks associated with MODS 2D image processing can be done without having to break out of the sequence and use similar functions in IRAF or IDL, etc.

4.5.1 **modsMedian.py** – Median combine images

Usage: **modsMedian.py [options] inFile1.fits [inFile2.fits ...] medFile.fits**

Where:

inFile*.fits	input FITS images to be median combined
medFile.fits	output median image to create

Options:

-f	force overwrite of existing median file ("clobber)
-v	print verbose debugging information during execution
-V	print version information and exit (no execution)

modsMedian.py will median combine a list of images. The output image contains the median value of the stack at each pixel. This is provided as a convenience function, for example to combine flat field images when creating normalized color-free pixel flats (see §3.2).

The FITS header for the median image will be taken from the first image in the input file list. modsMedian.py will add a HISTORY card to indicate how many files were combined, as well as recording the version/date of the modsMedian.py instance that was used.

4.5.2 **modsAdd.py** – Add or Average a set of images

Usage: **modsAdd.py [options] inFile1.fits [inFile2.fits ...] outFile.fits**

Where:

inFile*.fits	input FITS images to sum or average
outFile.fits	output FITS file with the sum or average

Options:

-a	compute the average instead of the sum (default: sum)
-f	force overwrite of existing median file ("clobber)
-v	print verbose debugging information during execution
-V	print version information and exit (no execution)

modsAdd.py will add together two or more images, creating the sum or, if the –a option is used, compute the average. No pixel rejection or clipping is done by modsAdd.py. Provided as a convenience function, for example to add together the blue channel Clear and UG5 filter flat fields (see §3.2, Step 3b).

By default, modsAdd.py will not overwrite an existing image, but you can override this with the –f option. The FITS header for the output image will be copied from the first input image in the list.

The list of images to sum (or average) can be given as a list or any valid Unix wildcard. The last image on the command line **must** be the name of the output file to create.

modsAdd.py will append an HISTORY card to the FITS header including the number of files added together (but not a list of their names), and if –a is used, note that this is the average. It will also record the version/date of the modsAdd.py instance that was used.

4.5.3 **modsSub.py** – Subtract two images

Usage: **modsSub.py [options] inFile1.fits inFile2.fits outFile.fits**

Where:

inFile1.fits	first input file
inFile2.fits	second input file
outFile.fits	output FITS file with difference: outFile=inFile1-inFile2.

Options:

-f	force overwrite of existing median file ("clobber)
-v	print verbose debugging information during execution
-V	print version information and exit (no execution)

modsSub.py subtracts image2 from image1, creating a third file with the difference. By default modsSum.py will not overwrite an existing output file, but this may be overridden with the –f option. The FITS header of the output difference image will be taken from the first image.

modsSub.py will add a HISTORY card to the FITS header that includes the names of the two images used to compute the difference. . It will also record the version/date of the modsSub.py instance that was used.