# Cleaning Images of Bad Pixels and Cosmic Rays Using IRAF

Lisa A. Wells David J. Bell

September 13, 1994

This document presents the possible uses and examples of the many tasks which may be used to clean images of bad pixels and cosmic rays with IRAF version 2.10. Basic knowledge of IRAF structure and syntax is assumed.

## Contents

# 1 Introduction

This document presents the many options within IRAF for fixing bad pixels and cleaning cosmic rays from CCD images. Images and spectra are treated somewhat differently, so both applications will be discussed. The text is based on the tasks available in IRAF Version 2.10.3. It is assumed that the reader has some preliminary knowledge of IRAF, including the package structure, getting help, listing and editing task parameters, and executing tasks in general. A beginner's guide and other useful documentation are listed in Appendix F.

Before reading any further, you should decide if you even *want* to clean your images. The techniques presented in this document range from cleaning through sophisticated statistical algorithms to straightforward data editing. In some cases, these methods are designed to extract as much useful information from an image as possible, while in others they involve data manipulation for purely cosmetic reasons. We will merely present your options and attempt to leave you well aware of what each task does. The decision as to which cleaning techniques, if any, should be performed will depend on the statistical and cosmetic requirements of your project.

There are many ways of fixing bad pixel regions in a CCD image. These features are due to minor flaws in the detector and thus will usually be seen in the same locations in every image. Sometimes, a pixel will be broken entirely, preventing the rest of its line or column from being accessed during readout. More commonly, a single pixel or a pixel region will simply be "hot" or "cold," with a sensitivity different from the rest of the chip. The flat field will take care of most of these features so it is a good idea to first process some test images and examine them to see if bad regions persist. If they do, you have the choice of fixing them before flat-fielding, or you may correct the already flattened data. There are several ways to do this, from setting constant values in these regions, to interpolation across the smallest dimension, to creating a mask image which may be applied using image arithmetic. If all images in a set have the same flaws, then a batch job can be used to process through all the images non-interactively.

Cosmic rays are treated differently, as they are random events and have obvious profiles. Since usually no two images of the same object will have cosmic rays in the same location, combining multiple registered images, using suitable pixel-rejection criteria, will generally remove these features. When dealing with individual images, cosmic ray features can often be detected and removed with statistical models. They can also be edited out by hand, which is more time consuming but may be necessary for cosmic rays with non-standard profiles, i.e., those that are trailed.

Spectra may be extracted from 2D images using optimal rejection techniques which correct for cosmic rays on the spectral profile. If background subtraction is being performed, statistical thresholds can be used to reject deviant data in the background as well. If these options are used, the need for cosmetic corrections at the end of the processing is often avoided. However, manual interpolation may also be performed on 1D spectra using **splot**.

# 2 About the Tasks

There are many tasks which may be used to clean bad pixels and cosmic rays from images, and often the same results can be obtained in different ways. Here we give a brief description of each task to help users decide which are best suited to their needs.

- **ccdproc** - The image processing task which allows specification of a bad pixel file. The parameter *fixpix* must be turned on, and a bad pixel list specified. Images are corrected in the same manner as in the task **fixpix**.

- **fixpix** - An input list of bad pixel positions are read and interpolation across the smallest dimension of each region specified is performed. This is used on each image individually.
- **imarith** - Uses image arithmetic to apply a mask image, flagging the bad regions by setting them to 0, a very high, or a very low value.
- **cosmicrays** - Locates and removes cosmic rays using a statistical model. This may have trouble distinguishing between multiple close events and stellar objects with small PSFs.
- **imsum** - Combines multiple images of the same object using *sum* or *average* options. Rejection of low or high valued pixels is used.
- **combine, imcombine** - Combine multiple images of the same object with many optional rejection algorithms.
- **lineclean** - Edits pixels using simple sigma clipping and replaces highly deviant pixels by the fit.
- **imreplace** - Replaces all pixels within a certain range of values with a given constant.
- **epix** - Edits an individual pixel by replacement with supplied value.
- **imedit** - Edits an image interactively, using the cursor to mark regions, either by interpolation or replacement with a specified value.

The first two tasks listed above, **ccdproc** and **fixpix**, are best used to fix bad pixel regions, which are usually well known for a particular CCD. These tasks require an input list of bad regions (assumed the same in every image), which are replaced by interpolation across the smallest dimension of each region. Replacement of bad regions by a constant may be done using **imreplace**, which can be given either the coordinates of a replacement region or a range in pixel values to be replaced. This should be used with caution since the value range may include pixels outside the truly bad regions. Image arithmetic (**imarith**) may be used to apply a mask to a list of images. This will simply replace the bad regions with a very large number, a very small number, or zero. All these tasks may be run in batch mode by specifying lists of input images. The tasks **ccdproc** and **fixpix** perform the operation in place, overwriting the input images, though **ccdproc** can be set to save the input images using the *backup* parameter in the *ccdred* package parameter set. Section 3.1 goes into more detail about these tasks. See Appendix D for information on making mask images.

Cosmic rays are random events with distinct profiles that are usually confined to one pixel. These can be removed by taking multiple exposures of a field and combining them, with appropriate rejection criteria, using **imsum**, **combine**, or **imcombine**. Hopefully no two images taken of the same object will have cosmic rays in the same place, although (by setting the appropriate parameter) more than one pixel value at any given coordinate may be rejected to get around this problem. If multiple exposures of an object are not available for use with the combine tasks, then **cosmicrays**[1] may be used to locate deviant pixels from image statistics.

[1]This task should be used with great caution on HST images, as stars in these images may be confused with cosmic rays and be deleted. Instead, the task **crrej** in the STSDAS.HST_CALIB.WFPC package should be used, though this task is not presented in this document. The STSDAS package is add-on software developed and distributed by STScI and is available by anonymous ftp to stsci.edu (130.167.1.2).

The task **lineclean** will get rid of highly deviant cosmic rays by specifying an appropriate level for the rejection limits. This task may remove good data so care should be taken when using it on 2D images. The above tasks are explained in more detail in Section 3.2.

The **imreplace** task performs replacement by a constant only. It is non-interactive and allows regions and/or pixel value ranges to be specified. The **epix** task works non-interactively, one pixel at a time, requiring a pixel position and new value for each pixel. It is most useful for cases in which there are only a few cosmic rays to be removed. The task **imedit** is an interactive task which uses the display device and image cursor. The replacement regions may be set to any radius interactively and the shape also may be changed. The task allows a variety of replacement algorithms from interpolation to replacement by a constant. The image can be automatically updated after each operation for inspection purposes. The above tasks may be used for cosmic ray or bad pixel removal (see Section 3.3).

Spectral images should also be flat-fielded before determining whether bad pixels are to be removed. The tasks best used for pixel fixing are **ccdproc** and **fixpix**, which are described in Section 3.1. This process of removing bad pixels is usually done before extraction, while cosmic rays may be removed before, during, or after the extraction process. The previously mentioned tasks can be used on any type of data, while the following are particular to spectral images:

- **apall** - Uses optimum extraction techniques to remove cosmic rays during the extraction process.
- **scombine** - Like the task **combine** for images, this averages 1D or multispec format spectra which have been dispersion corrected.
- **splot** - Used to interactively edit out bad regions in spectra.

The weighted extraction techniques in **apall** may take care of cosmic rays during the extraction to 1D spectra simply by setting the appropriate parameters. For multiple spectra taken of an object, **scombine** combines input spectra by interpolating them (if necessary) to a common dispersion sampling, rejecting pixels exceeding specified low and high thresholds. This works on 1D or multispec format spectra, where certain apertures may be specified to be combined into the final output spectrum. The spectra must have dispersion solutions since these are used in matching the input spectra, rather than their physical or logical pixel coordinates. For editing individual extracted spectra, an "etch-a-sketch" mode is available in **splot**. This may be used for interactive interpolation by eye to replace a cosmic ray or bad pixel region, including poorly subtracted sky lines, with a straight line defined by two positions of the cursor. The image must then be saved (since it is not done automatically) by replacing the new spectrum in the old name or renaming the output spectrum. More information and examples of these tasks may be found in Section 4.

Several additional tasks are mentioned in the Appendices:

- **rimcoord** - Used to find cursor positions in display device.
- **imexamine** - Used to examine an image and find pixel coordinates.
- **implot** - Used to find pixel coordinates without a display device.
- **badpiximage** - Creates a mask from a bad pixel coordinate file.
- **mkpattern** - Creates/modifies patterns in images or image sections.

- **imcopy** - Used to copy image sections or create pixel list masks.
- **imexpr** - Allows many enhanced arithmetic image expressions.
- **mknoise** - Adds noise or cosmic rays to an image or image section.

# 3 Editing Images

Each of these sections will describe the task and parameters, the input format for files, and give at least one good example of its use. Some tasks are better for fixing bad pixels while others are better at removing cosmic rays. The editing tasks described in the last section are useful for both applications. We advise you to at least briefly glance over each section before deciding on how to clean your images. Depending on the nature of your data and cleaning needs, the same operation that would be tedious with one task might be straightforward with another. Cleaning can be performed at several different stages in the reduction process, and a particular cleaning operation might be unnecessary or even undesirable depending on how the data will be used with other tasks. For instance, when combining several images of the same object, it is usually not necessary to clean cosmic rays from the individual images. Instead, statistical techniques can be used to reject deviant pixels during the combining process.

Bad regions can also be treated with a pixel "mask". A mask here refers to a map describing the good and bad regions of an image. For instance, instead of individually editing several images with the same bad pixel regions, we might instead create a binary mask. This could be an image with values of "0" in all the bad regions and "1" in all the good regions. Each data image could then be multiplied by the mask to set all the bad regions to 0. The tasks **combine** and **imcombine** can also use masks for pixel rejection during the combining process (instead of multiplication, pixels are simply used or not used based on the mask). In this case, the masks must be of the "pixel list" file type, designated in IRAF with a ".pl" extension. These ".pl" files store all the header and pixel information in one file, and are more compact and efficient than images for cases in which many of the pixels are of the same value (as in a mask). Nonetheless, these files can be created, edited, operated upon, graphed, and displayed with the same tasks as would be used for images. Possible uses of mask images and pixel lists are discussed in sections 3.1.3 and 3.2.3, and tasks useful for creating masks are described in Appendix D.

## 3.1 Fixing Bad Pixels

Bad pixel regions are unique for a given CCD and do not move around randomly as do cosmic ray events. These regions may be a pixel, line, column or 2D region of any shape where the statistics are higher or lower than the overall background statistics. Before using any of the tasks described below, the images should be flat-field corrected, since bad lines and columns will often be fixed by applying the flat image. In rare cases, bad regions may be made worse and so they must be fixed before processing the images. After processing, examining the images with **implot**, or **display** may reveal persistent bad regions, see Appendix C. In this case coordinates defining the edges of the bad regions must be acquired for input to some of the following tasks. See the help pages for these tasks for more information.

Be warned that the tasks **ccdproc** and **fixpix** perform their operations in place, overwriting the input image. If you wish to attempt a test processing procedure to learn whether pixel fixing should be performed before flat-fielding, you should do it on a test copy of one of the images. In either case, unless you are already familiar with the cosmetic characteristics of your CCD, you may wish to save local backup copies of your unprocessed data. For **ccdproc** this may be done by setting the **ccdred** package parameter *backup* to a directory (in which case it should end with a slash) or a prefix. Use "**epar ccdred**" for access to the package parameters.

### 3.1.1 ccdproc

This task for image processing is found in the IMRED.CCDRED package. It includes an option to fix bad pixel regions from a specified bad pixel file defining the regions to be fixed. This works like the task **fixpix** (see next section) in that both use interpolation across the smallest dimension of the region specified. The *fixpix* parameter must be set to **yes**, and the list of bad regions specified in a file, the name of which must be specified in the *fixfile* parameter. The format for this table is the first and last columns of the bad region followed by the first and last lines of the bad region, i.e., "xbegin xend ybegin yend" separated by spaces not commas. See Appendix C for information on tasks which may be used to obtain coordinates for the bad pixel regions.

```
cl> type images
obj0003
obj0004
obj0005
obj0006
obj0007
obj0008
cl> type badpix
189 189 258 258
480 562 378 378
493 521 390 397
cl> ccdproc @images fixpix=yes fixfile=badpix
```

In this example, a list of files is input to the task and processed. The **badpix** file in the example first defines one individual bad pixel, then a bad line, and then a larger rectangular region. If the bad region borders the edge of the image then the interpolation is by replication of the first good pixel in the direction of interpolation, otherwise linear interpolation between the bordering lines or columns is used. Note that if the files have already been processed without pixel fixing, rerunning the task as above will fix pixels without it repeating its other operations. If **ccdproc** has already been used to fix pixels in a image, the operation will not be done again, even if a new bad pixel file is given. If additional bad pixel regions are discovered after **ccdproc** fixing has been performed, one must delete the FIXPIX header keyword, or instead use the **fixpix** task.

### 3.1.2 fixpix

This task for fixing bad pixels is found in the PROTO package. It corrects bad pixel regions by interpolation using an input bad pixel file defining the regions to be fixed. Interpolation is performed across the smallest dimension of the region specified. Like **ccdproc**, **fixpix** overwrites the input images, but unlike **ccdproc** it does not include an option to save copies of the originals. Thus, it is safer to first try any new operation on a test image. The name of the image, or a list of images, to be fixed and the bad pixel table must be specified in the *images* and *badpixels* parameters respectively. The format for this bad pixel table is the first and last columns of the bad region followed by the first and last lines of the bad region, i.e., "xbegin xend ybegin yend" separated by spaces not commas. See Appendix C for information on tasks which may be used to obtain coordinates for the bad pixel regions.

```
cl> type fixlist
obj0003
obj0004
obj0005
obj0006
obj0007
obj0008
cl> type badpix
189 189 258 258
851 851 274 274
329 329 304 580
480 562 378 378
493 511 610 636
cl> fixpix @fixlist badpix
```

In this example, a list of files is input to the task and processed. The *badpix* file in the example first defines two individual bad pixels, then a bad column, a bad line, and then a larger rectangular region. If the bad region borders the edge of the image then the interpolation is by replication of the first good pixel in the direction of interpolation otherwise linear interpolation between the bordering lines or columns is used. NOTE: There is currently no way to convert a bad pixel image (see Appendix D) to a fixpix format input list.

### 3.1.3 imarith

Image arithmetic can be done with **imarith** in the IMAGES package. The best application of this task for fixing bad regions is to apply a "mask" to a data image by summation or multiplication (see beginning of Section 3 for definition of a mask). If one first creates a binary mask with values of "0" in all bad regions and "1" in all good regions, image multiplication can be used to set all "bad" pixels to 0 in a set of images. Optionally, a mask could be created in which the bad regions are set to a very large positive or negative value and the good regions set to "0". When this mask is added or subtracted from a data image, the good pixels will be unaffected while the bad ones will be given large negative or positive values. Such operations do not "fix" bad pixels in the sense that they make the images look nicer, but instead they can be used to flag the bad pixels (which may have had reasonable looking data values) for other tasks. For instance, when images are to be combined using a task such as **imcombine**, bad regions may first be set to clearly unreasonable values which will then be ignored by the combining task's threshold options (**combine** and **imcombine** can also use pixel list masks to reject pixels directly, making an **imarith** step unneccessary—see Section 3.2.3). In this way, images may be combined even when their bad regions are in different places. Tasks for creating masks are discussed in Appendix D.

The syntax for **imarith** is quite intuitive—it looks just like an arithmetic expression, but without the equals sign:

cl> **imarith obj0005 * mask mobj0005**

This example multiplies the object by the mask and renames the output image.

cl> **imarith obj0005 + mask obj0005**

The mask image is added to the object and the task operates in place by using the same name for the output image.

cl> **imarith @ilist - mask @olist**

The last example uses input and output lists of images and subtracts the mask from each image in the input list. These files may be identical to perform the arithmetic on the images in place or different to rename the output images. See the help page for this task for more information.

## 3.2 Fixing Cosmic Rays

Cosmic rays are random events which can occur at any place on an image. They are not corrected by flat fielding so other methods are used to clean these from an image. Normally a cosmic ray is seen as one very hot (high valued) pixel in an image though sometimes they do affect several adjacent pixels leaving a streak on the image. Due to their unique characteristics however, statistics may often be used to clean an image of these blemishes with **cosmicrays**. If multiple images were taken of an object, then combining these with some threshold limits will also remove cosmic rays. The last two sections here describe combining registered (i.e. properly aligned) images using **imsum**, **imcombine**, and **combine**.

### 3.2.1 cosmicrays

The **cosmicrays** task is found in the NOAO.IMRED.CCDRED package. This task searches for and corrects cosmic rays using selection criteria given by the parameters *threshold* and *fluxratio*. The *threshold* value determines the statistics used to identify deviant pixels; it should be set to 5 or more times the standard deviation in the background regions. The *fluxratio* parameter is used to choose which pixels should be corrected; they will be replaced with the mean of the 4 neighboring pixels. This parameter is the ratio (in percentage units) of the flux of the neighboring pixels, excluding the brightest neighbor, to that of the target pixel (after background subtraction). Thus, a value of 5 implies that the target pixel's value must exceed the mean of its neighbors by a factor of 20 to be deleted. Setting this parameter too high can delete good data so values between 2-6 are suggested. A bad pixel file may also be generated by specifying an output file name in the *badpix* parameter. This file may then be used to create a mask image using **badpiximage** (see Appendix D). The **tvmark** task may be useful for identifying the points which were corrected.

The following example first looks at the statistics of a region of sky in an image, using rough coordinates of a region free of objects, cosmetic defects, and cosmic rays. (See Appendix C for information on how the coordinates of such a region can be found with the display device and the cursor). The *threshold* is set to 5 times the sky background's standard deviation as given by the **imstatistics** task. Note that the *verbose* parameter in **ccdred** must be set to **yes** to get the output line shown below, though the information can also be found in the *logfile* and in the header of the output image.

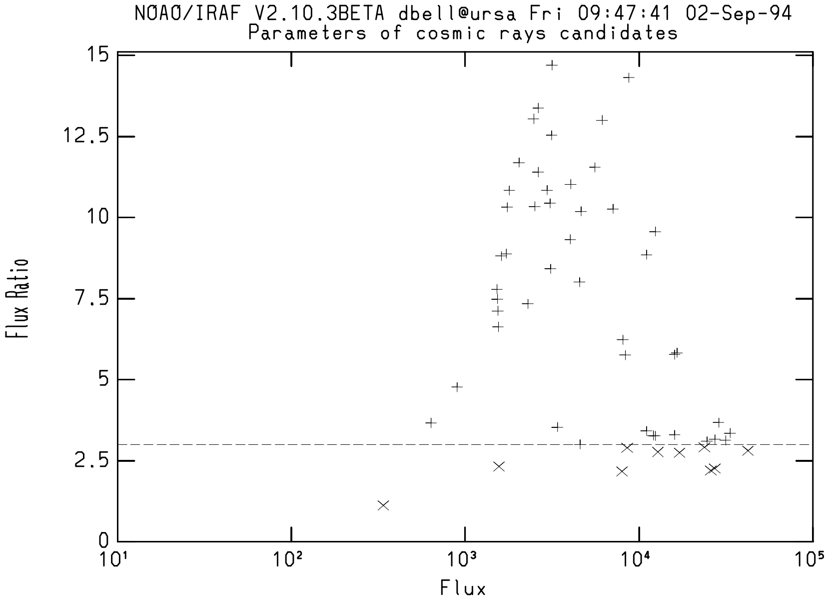


Figure 1: An example of an interactive plot in **cosmicrays**.

```
cl> imstat obj0008[107:164,152:227]
#                     IMAGE      NPIX      MEAN      STDDEV     MIN       MAX
obj0008[107:164,152:227]      4408      805.8      29.96      702.9      909.3
cl> cosmicrays obj0008 cobj0008 threshold=150 fluxratio=3
obj0008 - Examine parameters interactively?  (no|yes|NO|YES) (yes):  CR
cobj0008:  Sep 2 9:40 Threshold=150.0, fluxratio= 3.00, removed=33
```

Running this task interactively produces a plot of the pixels satisfying the condition set by the *threshold* parameter. The plot shows the flux versus the flux ratio in relation to the background sky. The value of the *fluxratio* parameter divides the plot between bad points to be replaced, represented by the diagonal crosses, and good points represented by the pluses. This value may be changed by setting the cursor at a new dividing point and typing a **t**. Crosses are changed to good points using **u** and pluses are deleted using **d**. The interactive portion of the task is exited by pressing **q**, at which time the corrections are made. In Figure 1, we see that there are several points in the lower right corner with large fluxes and small flux ratios. These represent very strong and extremely sharp features, and all of these are almost certainly cosmic rays. Thus, in this case, the *fluxratio* should be moved up so that all the points in this corner are included. Near $(10^4, 6.0)$ we see four more points which are likely to be cosmic rays, but which aren't quite as sharp. We could elect to delete these points by hand (by pressing **d** on each one) instead of moving the *fluxratio* up (crosses are always deleted and pluses are always ignored, even if they are on the wrong sides of the line). Most of the points at the top of the plot are probably stars and should not be corrected. The other points represent weak but sharp features, and these may be due to weaker cosmic rays or simply pixels which made it above the *threshold* due to statistical noise. Thus, if a lot of weak sharp points are appearing, it probably means the *threshold* value is set too low. Note that it is possible for a feature to have a negative flux and/or flux ratio, but that these points will not be visible unless the graph is re-windowed (e.g. by pressing **w a**).

Another interactive way of using this task is with a "training" option, invoked by setting *train=yes*. **Cosmicrays** will assume the user has already displayed the image in the display window (ximtool, saoimage, or imtool). An image cursor can then be used to mark features as cosmic rays with **c** or stars with **s**. The task uses this information to set **fluxratio** just high enough to include all the items labeled as cosmic rays. One may switch to the graphics plot by pressing **g** and back to the image display with **q**. A **q** from the image display is used to exit from the interactive portion of the task. To help decide if a point is a cosmic ray or a star, one may press **s** in graphics mode to show surface plots (from four different angles) of the point nearest the cursor. In addition, pressing **space** in graphics mode will give the pixel coordinates of a particular point (these options are also available when not using the training option). More information can be found in the help page for this task.

### 3.2.2 imsum

**Imsum** is found in the IMAGES package, and it not only sums images but can average or median them as well. The options for pixel rejection are not as extensive as those available in **combine** or **imcombine**, however, being limited only to rejection by rank order. This is done by setting the *high_* and *low_reject* parameters to reject those numbers of pixel values at each position in the image, not to exceed the number of input images. Setting these to values less than 1 rejects that percentage of high or low pixels. The task does not check the values against any statistical threshold, but merely throws out the requested number of pixels at each position. This means it will eliminate perfectly good points and pixel rejection should only be used with a large number of images to preserve statistics, though this task is much slower than the other available choices for such an operation. When only a few images are being combined, and pixel rejection is needed, it is probably better to apply a statistical rejection algorithm using **combine** or **imcombine**.

New values of certain header parameters may be computed for the combined output image by specifying them with the *hparams* parameter. For example, one would usually want to sum the exposure times in the image headers when the images are being summed. A new title for the output image must be specified, as this task will not overwrite existing images.

`cl>` **imsum obj008,obj009,obj010,obj011 sumout high_rej=1**

The first example adds 4 images (addition is the default operation for this task), throwing away the highest valued pixel at each point.

`cl>` **imsum obj008,obj009,obj010,obj011 sumout hparams="exptime"**

This example adds the images with no pixel rejection (by default when parameters are not given) and also computes a new exposure time for the output image by summing the input values.

`cl>` **imsum @comblist aveout option=average high_rej=2**

In the last example, an input list of images is used and the task outputs the average, rejecting the two highest pixel values per position.

By default, **imsum** will perform the calculations in the highest precision datatype of the input images. Thus, if several short integer images are being combined the output values can wrap around. This can be avoided by setting *calctype=real*.

This task is quite outdated and has been largely replaced by **combine** and **imcombine**. **Imsum** may still be used, however, for simple operations in which only a few images are being combined and statistical pixel rejection algorithms are not needed. See the help page for this task for more information about its parameters.

### 3.2.3 combine and imcombine

The **combine** task is found in the NOAO.IMRED.CCDRED package, and **imcombine** is found in the IMAGES package. These tasks perform the same operations though the parameter lists are not identical. This is mainly due to the context in which they are used—**combine** is set up to do automatic reductions employing a translation file to check header parameters. But the most important parameters for combining images are identical in the two tasks. There are many possible combinations of parameters so several examples will be detailed. We begin with an explanation of the available pixel rejection algorithms—*minmax*, *ccdclip*, *crreject*, *sigclip*, *avsigclip*, and *pclip*. This option may be turned off by setting *reject* to "none".

The *minmax* option works much like the high and low rejection in **imsum**. The number of low and high pixel values to reject is governed by the parameters *nlow* and *nhigh* which is turned into a fraction depending on the number of input images. This will reject the specified number of highest and lowest values at each pixel position so it is best to use this option only with a large number of images.

Better options for removing cosmic rays would be the *ccdclip* and *crreject* algorithms which use the readout noise and gain of the CCD to locate highly deviant values, based on the computed sigma value. The first of these deletes high and low values while the second rejects only high values. The *lsigma* and *hsigma* determine the deviation criteria for rejection. These options do not require a large number of input images, though they do require knowledge of the noise statistics of the detector.

The next rejection options are *sigclip* and *avsigclip*, which compute the median or average at each pixel and the standard deviation about this value. Pixels that deviate from the median or average by more than *lsigma* or *hsigma* times the standard deviation are rejected in an iterative process until no more deviant values are found. *Sigclip* works best with a large number of input images ($> 10$), while *avsigclip* can work with as few as three images. This is due to the different ways in which the two algorithms calculate the standard deviation. *Sigclip* simply computes the standard deviation based only on data from each pixel position. When using *avsigclip*, on the other hand, the standard deviation about the mean or median is assumed to be proportional to the square root of the mean or median at each point, allowing all data in a line to be used to determine the standard deviation as a function of the mean or median.

The last option is the *pclip* algorithm. It is similar to the sigma clipping options, however, the width of the distribution is characterized by the difference in the median value and a specified percentile pixel value. The percentile pixel is specified by the *pclip* parameter in the task. The pixel values are ranked from low to high at a given position in a set of images. The median is then the middle value for an odd number of inputs or the average of the two middle values for an even number. If *pclip* is a positive or negative integer value, then the percentile pixel is that number of input values above or below the median value respectively. If *pclip* is a positive or negative value between -1 and 1 then the percentage of the pixels above or below the median value is used, i.e., for *pclip* = -0.5, with 9 input values, there would be 4 pixel values below the median; 50% of 4 pixels is 2, which is the third pixel value in the rank. The difference between the median and the percentile value is multiplied by the *lsigma* and *hsigma* parameters which set the lower and upper rejection thresholds. This algorithm is good at removing very small excursions, such as low level wings of stars when several disregistered images are being combined to produce a sky flat. Each of the rejection options have associated parameters which are explained in more detail in Appendix A.

Threshold rejection is governed by the *lthreshold* and *hthreshold* parameters, which specify the lowest and highest good data values for pixels in each image. These thresholds are applied before all other rejection and combining operations. Threshold rejection is not performed when both parameters are set to "INDEF".

Pixel masks may be applied to each image, before the threshold or rejection processes take place. The mask files must be of the "pixel list" file type, which IRAF designates with a ".pl" extension (see beginning of Section 3 for the definitions of masks and pixel list files). Pixel masks are applied by setting the *masktype* and *maskvalue* parameters to specify the type of mask being used. For instance, if the mask is such that good pixels are marked with 0 and bad pixels marked with 1 (this logic is often used, as it makes for the most compact pixel list mask files), the appropriate values would be *masktype=goodvalue* and *maskvalue=0* (or, equivalently, *masktype=badvalue, maskvalue=1*). The mask file name must be specified in the header of its associated image under the BPM keyword, which may be added to the header using **hedit**. If no mask is associated with an image, it will be treated as if it had a mask will all values being zero. Thus, if combining a large number of images, only a few of which need masks, it is best to define 0 as the good value to avoid having entire good images rejected. Tasks useful for creating pixel mask files are described in Appendix D. Whether or not masks are being used for the input images, an output pixel list file, a map of the number of pixels rejected at each position, can be produced by setting the *plfile* parameter to a file name. This file name is added to the output image header under the keyword BPM.

For cases in which the input images do not have their objects at exactly the same pixel positions, there is an option for offsetting images which does not require registration and saves a lot of processing time. The offset table is given one object per line with the x and y shifts separated by a space. These shifts may be specified in relation to one of the images or to some arbitrary common point. The reference image shift would be "0.0 0.0" and negative values are allowed. This option should be used only when the images have large overlapping regions. The use of **imcombine** for mosaicing is explained in more detail in another document (see Appendix F).

*Average* and *median* are the choices for the *combine* parameter. This is the last operation performed on the input images, after offsetting, masking, thresholding, and rejection. In some cases, the number of pixel values going into the average or median may be lower than the *nkeep* parameter, resulting in the inclusion of those rejected pixel values with the lowest residuals calculated by the rejection algorithm. For example, if 2 of 9 pixels are deleted using the *hthreshold* parameter and 4 of the remaining 7 are thrown away by the rejection option used, then 3 pixels are checked against *nkeep*. If *nkeep* is set to 4, then the pixel value with the lowest residual from the rejection routine will be added back in to be combined. Pixels rejected by the thresholding are not affected by the *nkeep* parameter. Each of these examples uses the default *combine* parameter value of *average*.

> `cl>` **combine obj0002,obj0003,obj0004,obj0005 out reject=none **
> **hthreshold=28000**

This example uses the *hthreshold* to through out pixels above the given value for the four input images. All the other examples below use the input list of 9 images called **clist**.

```
cl> type clist
obj0003
obj0004
obj0005
obj0006
obj0007
obj0008
obj0009
obj0010
obj0011
cl> imcombine @clist comb reject=minmax nlow=0 nhigh=2
```

The **minmax** option is used to reject the highest 2 values at each pixel position.

```
cl> combine @clist comb reject=crreject rdnoise=3.6 gain=5.8
```

This example is using the cosmic ray rejection which requires the noise characteristics of the detector, the *gain* and *rdnoise*. The default rejection thresholds of 3.0 sigma are used.

```
cl> imcombine @clist comb reject=avsigclip nkeep=5 lsigma=2.5 hsigma=2.5
```

The last example uses the average sigma clipping algorithm and allows rejection of no more than 4 pixels at each position. If the noise parameters for the CCD are well-known then **crreject** may be used. If the noise parameters are not well-known and only a few images were taken, then **avsigclip** should be used. More information about this task is found in the help page.

### 3.2.4 lineclean

The task **lineclean** is found in the IMAGES package. The task fits a function to an image one line at a time and locates highly deviant pixels which are replaced by the fit. The shape of the function may be highly variable in the image however, and parameters resulting in a good fit to one line may delete good data from another line. This task should be used with great caution for this reason, and the task may be run interactively to examine the fit for many image lines. There is a choice of four function types for the fit:

- **Legendre** - polynomial of the specified *order*.
- **Chebyshev** - polynomial of the specified *order*.
- **Spline1** - linear spline with number of pieces set by *order*.
- **Spline3** - cubic spline with number of pieces set by *order*.

(Remember that a function's *order* in IRAF refers to the number of degrees of freedom, not the value of the highest exponent. *Order=2*, for the polynomial fits thus gives a straight line, not a parabola.) The *low_reject* and *high_reject* parameters set the rejection levels in units of the residual sigma. These values must be chosen carefully, and can be determined by estimating the level of the highest good value in the image compared with the average sky level.

```
cl> lineclean obj0003 cl0003 function=chebyshev order=4 low=3 high=3
```

This example uses a 4th order chebyshev function and sets the rejection level to 3 sigma above and below the function fit. It will prompt for a line to begin the fitting, and the user may type in the line number of interest and change the parameters accordingly. Some of the interactive commands are:

- **f** - Recalculates the fit.
- **r** - Redraws the graph.
- **?** - Shows the help menu, exited using **q**.
- **:function (new function)** - Changes the function type for the fit.
- **:order #** - Changes the order or number of spline pieces.
- **:niterate #** - Changes the number of rejection iterations.

The interactive fitting is exited using **q** which then prompts for the next line to be fit. When the parameters have been set properly, hitting a **CR** without a number will then fit the entire image line by line, using the last set of fitting parameters, and output the cleaned image. See the help page for this task for more information.

## 3.3 Fixing Images by Hand

This section describes several tasks used interactively or non-interactively to fix bad pixels and cosmic rays. In the non-interactive cases, the bad regions are input one at a time or in lists. The first two sections describe tasks which are non-interactive. The last section describes the task **imedit** which may be used either interactively or non-interactively. See Appendix C for more information on obtaining pixel positions.

### 3.3.1 imreplace

The **imreplace** task is found in the PROTO package. It simply replaces the pixel value of a given region specified by an image section and/or a range in current values. The replacement value must be specified and an imaginary part for a complex number may be incorporated. The range to be replaced is given with the *lower* and *upper* parameters, which may be set to "INDEF" to set no lower or upper bound. The replacement will be performed over the entire image if an image section is not specified.

```
cl> imreplace obj0003 1.0 lower=32000.0 upper=INDEF
cl> imreplace obj0003 1.0 lower=INDEF upper=-10.0
```

This example replaces all pixels in the image with values below -10.0 and above 32000.0 (the saturation level, for example) with the value 1.0.

```
cl> imstat obj0003[1:50,1:50]
#           IMAGE       NPIX     MEAN       STDDEV      MIN      MAX
obj0003[1:50,1:50]      2500     39.22      2.424       32.      92.
cl> imreplace obj0003[1:145,50:90] 39.22 lower=32000.0 upper=INDEF
cl> imreplace obj0003[1:145,50:90] 39.22 lower=INDEF upper=0.0
```

Here the mean in the background of the image is calculated and used as the replacement value. An image section is specified for replacement of values below 0.0 and above the saturation level. This task is also useful for creating mask images; see Appendix D for more information.

### 3.3.2 epix

The **epix** task is found in the PROTO package in IRAF. This task can be used to edit a single pixel at a time, for cases in which an image has only a few cosmic rays to be removed. Procedures to determine pixel coordinates are explained in Appendix C. The parameter *edit_image* must be set to **yes** in order to replace the specified pixel with the *new_value*. The following examples replace the specified pixel with the value 0.0:

```
cl> epix obj0004 345 267 0.0
cl> epix obj0004 678 465 0.0
cl> epix obj0004 723 682 0.0
```

Image statistics (**imstatistics**) may first be used on a section of the image to determine the replacement value. Or, if *new_value* is not specified on the command line, the task will compute the mean of the surrounding 8 pixels before prompting for the replacement value:

```
cl> epix obj0004 463 142
              462       463       464
       141  76.5472  74.9697  83.7819
       142   74.201  3741.14  55.4181
       143  34.7398  93.9899  85.8516
median 76.54721, mean 72.43739, sigma 18.93725, sample 8 pixels
new value for pixel (0.):  72.4374
```

See the help page for more information on this task.

### 3.3.3 imedit

The **imedit** task is found in the IMAGES.TV package. This task can be used to edit various types of regions of an image or to obtain statistics. Images may be edited interactively or non-interactively using a list of positions and commands. The images must be two dimensional. There are a number of replacement algorithms from interpolation, to replacement by a constant value, to replacing one region in a given aperture by another. An *input* list of images with a corresponding *output* list may be used, or one may simply edit one image at a time. If no output list is specified, then the modified images are saved under their old name. A **square** or **circular** *aperture* may be used and the *radius* set for the size of the region to be edited. The four parameters *buffer*, *width*, *xorder*, and *yorder* are used in the background replacement algorithm. If replacement by a constant is to be used, then the parameter *value* must be specified. The *cursor* parameter gives the name of a file to control non-interactive editing. It can contain either positions and editing keystrokes (a list appears at the end of this section), or bad regions in the **fixpix** format:

```
cl> type badpix
103 105 206 265
387 387 658 750
```

```
568 568 480 480
cl> imedit @inlist @outlist cursor=badpix fixpix=yes display=no
```

In the example above, the *display* option is set to **no** for non-interactive use of **imedit** and input and output lists are given for the images. The pixels are fixed by interpolation across the smallest dimension of the region specified as in the task **fixpix**.

The cursor file can instead give positions and the editing keys that would be used in interactive mode. The format for each line is first the x and y positions of the cursor, then a WCS coordinate (not used in this application, but some integer value in this field is required), followed by the editing keystroke. Those commands that require two cursor positions will also require two lines in the file.

```
cl> type pixlist
240 368 101 b
284 134 101 a
292 147 101 a
380 456 101 l
380 512 101 l
cl> imedit obj0003 edobj0003 cursor=pixlist display-
```

In this example, three regions are modified non-interactively using the editing commands which terminate each line. First is replacement of an aperture by the background **b**, then replacement of a rectangular region by the background **a**, and finally interpolation across a line **l**. If no editing command is given on a line, the task will perform the operation given by the *default* parameter.

To run the task interactively, use:

```
cl> imedit obj0003 edobj0003 display+ autodisplay+ radius=6
```

The *autodisplay* option shows the image after each modification. The following commands may be used in interactive mode:

- **+** - Increase *radius* by one.
- **-** - Decrease *radius* by one.
- **a** - Background values replace a rectangular region marked by opposite corners.
- **b** - Background values replace the aperture defined by *aperture* and *radius*.
- **c** - Interpolation across the columns marked.
- **d** - Constant value replaces a rectangular region marked by opposite corners
- **e** - Constant value replaces the aperture defined by *aperture* and *radius*.
- **f** - Interpolation across the smallest dimension replaces the region marked by opposite corners.
- **g** - Plots a surface graph.
- **i** - Starts over without saving changes for the current image.

- **l** - Interpolation across the lines marked.
- **m** - Replaces one aperture region with another aperture region.
- **n** - Adds the values of one aperture region to another region.
- **p** - Prints the pixel values and statistics for a box centered on cursor.
- **q** - Quits interactive mode and saves the modified image.
- **s** - Surface plot at the position of the cursor.
- **t** - Toggles between search for the maximum (positive) or minimum (negative) valued pixels in the *search* radius.
- **u** - Undo the last modification to the image. This is a toggle switch.
- **space bar** - Statistics for the region around the cursor.
- **?** - The help page is shown for the interactive commands, use a 'q' to get back to interactive mode.

The task parameters may be edited while in interactive mode using the **:** commands, for example **:radius** 4. The **+** and **-** can be used to change the *radius* interactively. Interactive mode is exited using a **q**. These commands are all explained in more detail in the help page for this task.

# 4 Fixing Spectral Data

Spectral data should be flat-fielded in a similar fashion as direct imaging, except for fiber data for which the flat-fielding is done after extraction. This processing may take care of the bad regions, though cosmic rays may still be a problem. Using optimal extraction algorithms with **apall** may remove most cosmic rays and perhaps bad pixels. Once spectra are extracted, however, combining spectra with **scombine** may take care of any residual cosmic rays and bad pixels, in cases for which multiple spectra were taken of an object. For individual spectra, there is also the option of removing bad pixels by hand using **splot**. These tasks are described in the next three sections.

## 4.1 apall

The **apall** task is found in NOAO.TWODSPEC.APEXTRACT and all the spectral packages in NOAO.IMRED, including KPNOSLIT, SPECRED, ECHELLE and others. Extraction of 1D spectra from 2D spectral images can be performed using optimal extraction techniques to distinguish between emission lines and cosmic rays. The cosmic rays are then rejected from the summation of the extraction window. Also, background or sky subtraction parameters may be set to reject cosmic rays in the background computation.

**Apall** has all the parameters required to define the extraction process for any kind of 2D spectral image. The boolean parameters at the beginning of the parameter list turn on and off the operations to be performed on the input images. A bright standard star may be traced and used as the reference for fainter program objects. The task will interactively extract spectra or run in batch on an input list of objects. The task could be used once interactively to define apertures, and again non-interactively for the actual extractions, or a reference spectrum may be specified for a fully non-interactive extraction.

Background sky lines may be removed by setting the *background* parameter to **fit** . This must be set to **none**, however, for comparison lamp extractions. The parameters beginning with *b_* are used to govern the background subtraction, with the most commonly changed parameters being *b_sample, b_naverage*, and *b_niterate*. *B_sample* defines the the background aperture relative to the center of the object aperture, and *b_naverage* gives the number of pixels to be averaged or medianed to produce data for a background fitting function (a constant by default). If *b_naverage* is positive, that number of pixels will be averaged; if negative, that number (in absolute value) will be medianed. Thus, if the background aperture is set to include 12 points on either side of the profile, a value of -3 will median the pixels in groups of three and produce 8 data values for the background fitting function. By default, these values would be averaged to determine the background value for that line or column perpendicular to the dispersion direction. A value of *b_naverage*=-3 will take care of single-pixel cosmic ray events; for CCDs with larger cosmic ray profiles, the magnitude can be increased to median more pixels. In addition, statistical rejection of the data points used in the background fitting function can be performed by setting *b_niterate* to the desired number of rejection iterations (0 by default). If this is done for the example above, the 8 data points (each a median of 3 pixels) would each be compared to their average, and any deviating by 3 sigma (default) would be rejected. This process would then repeat until no rejections occur or the number of requested rejection iterations is reached.

In addition to removing cosmic rays in the sky background, pixel cleaning can be performed on the object profile itself. The *weights* parameter should be set to **variance** and *clean* set to **yes**. This algorithm, described by Keith Horne (PASP, 1986, **98**, 609), uses the noise statistics of the CCD to detect deviant pixels in the profile. The parameters *saturation*, *readnoise*, *gain*, *lsigma*, and *usigma* are also used. *Saturation* may be set to the saturation limit of the CCD minus the DC-offset in ADUs, or it may be set to just above the largest real data value, allowing it to act as a threshold. If the readnoise and gain are not available, they may be calculated with the *findgain* task (in the NPROTO package) which uses pairs of raw biases and flats. The sigma parameters set the high and low rejection levels for deviant pixel values.

```
cl> apall spec0020 reference=spec0013 interactive- trace- recenter+ \
resize+ weights=variance clean+ readnoise=5. gain=1.8
```

This example non-interactively extracts a program object using a well defined star as the trace reference. The center of the object profile is automatically found and resized assuming its position on the slit was close to that of the reference star. The output spectrum will be called spec0020.0001 for onedspec format or spec0020.ms for multispec format. At each line or column perpendicular to the dispersion, the profile will be compared to an averaged profile and deviant pixels will be replaced by the fit. One must be careful to avoid rejecting good data, which can happen if the noise and rejection parameters are not properly set. One may rerun the task with *clean=no* and compare the output to make sure only bad data is being affected. Alternately, the task may be run with the *extras* parameter set to **yes**, which tells the task to also extract the raw spectrum (what would be output if simple profile summation was used with no cleaning or weighting), the subtracted sky spectrum, and the variance spectrum. These are extracted to the third dimension of the output image, and can be inspected with **splot** (will prompt for the band number). Help on the cleaning algorithms may be found in the help pages for this task, **apsum**, **approfiles**, and **apvariance**. Detailed instructions for extraction using this technique are given in the manual "A User's Guide to Reducing Slit Spectra with IRAF", by Phil Massey, Frank Valdes, and Jeannette Barnes (see Appendix F).

### 4.2 scombine

The task **scombine** is found in NOAO.ONEDSPEC and many of the spectral packages in IMRED such as KPNOSLIT, SPECRED, ECHELLE, and others. Combining spectra is done using pixels at common dispersion coordinates rather than physical or logical pixel coordinates. For spectra with different wavelength coverage and/or wavelength dispersions, interpolation is used to set the wavelength sampling. The first spectrum in an input list of spectra is used to define the dispersion sampling for the output spectrum if the *first* parameter is set to **yes**. The starting and ending wavelength, dispersion, and number of pixels (*w1*, *w2*, *dw*, and *nw* respectively) may be specified explicitly as well, in which case *first* should be set to **no**. The resulting image inherits the header parameters from the first input image.

Specific apertures may be specified for multispec format spectra. All the apertures are combined by setting this parameter to a blank value. The *group* parameter defines the type of operation to be performed.

- **all** - Combines all the input spectra into one output spectrum.
- **images** - Combines multispec format echelle spectra into one 1D output spectrum per input echelle spectrum.
- **apertures** - Combines the same aperture from all the input multispec format images into one output multispec format spectrum.

The spectra are combined using **average**, **median**, or **sum**, specified by the *combine* parameter. The rejection options and their governing parameters are the same as those for the **combine** and **imcombine** tasks, see Section 3.2.3 and Appendix A for more information about these options.

Onedspec format spectra may be combined using the following:

```
cl> imheader obj*.imh
obj0050.0001.imh[2001][real]:  m31 1
obj0051.0001.imh[2001][real]:  m31 2
obj0052.0001.imh[2001][real]:  m31 3
cl> scombine obj005*.0001.imh comb group=all combine=average \
reject=minmax
```

The 3 spectra are averaged throwing away high and low pixel values determined by the *nlow*, *nhigh*, and *nkeep* parameters, and the *output* is written to the image **comb**. In this example, *nlow* and *nhigh* are at their default values of 1, so that the operation is the same as a median (since there are only three input images). This is sometimes desired, but it does throw away good pixels as well as bad ones, and an **avsigclip** algorithm might alternately be used to reject bad pixels while keeping as many good ones as possible. Echelle spectra in multispec format may be combined into one long spectrum using the following:

```
cl> imheader @inlist
obj0001.ec.imh[256,9][real]:  m31 1
```

```
obj0002.ec.imh[256,9][real]:  m31 2
cl> type @outlist
obj001.1d.imh
obj002.1d.imh
cl> scombine @inlist @outlist apertures="" group=images
```

In this case, **inlist** is a list of the two images used in the example. The **outlist** contains the same number of entries but using different names to be used for the output spectrum. Any regions without overlapping pixel values will be set to the value specified by the *blank* parameter. For multispec format spectra where the aperture in one image corresponds to the same aperture in several others, use the **apertures** option for *group*:

```
cl> imheader @inlist
obj0100.ms.imh[2001,93][real]:  m31 1
obj0101.ms.imh[2001,93][real]:  m31 2
obj0102.ms.imh[2001,93][real]:  m31 3
cl> scombine @inlist comb.ms apertures=1-5,13,18,38,45,52 group=apertures
```

Here only apertures 1 through 5, 13, 18, 38, 45, and 52 are being combined from each input image. The *output* image **comb.ms.imh** is written in multispec format containing only those apertures which have been combined. The *input* here is in the form of a list, but the *output* is only one image.

When wishing to combine several apertures of one multispec format image into one spectrum, use the following:

```
cl> scombine obj0100.ms comb.ms apertures=1-5,13,18,38,45,52 group=all
```

Check the help page for this task for more information and examples.

### 4.3 splot

The task *splot* can be found in all the spectral packages in NOAO.IMRED, including ONEDSPEC, KPNOSLIT, SPECRED, and ECHELLE as well as others. Removing bad sky line subtractions or cosmic rays not detected by the cleaning extraction may be done using interactive mode with the **j** or **x** command. Several spectra of the same star may be graphed in order to find those bad sky lines or cosmic rays.

The input images may be done individually or an input list may be specified. Successive images are plotted following each **q** cursor command. Two dimensional images request the line or band to be graphed.

```
cl> splot obj0006.0001
```

A spectrum may be overdrawn by another spectrum by typing an **o** followed by a **g** or a **#**. Type in the name of the next spectrum to be graphed for the first option or the number of the order in the multispec format image. The last image plotted is the active one, so the last one overplotted should be the one with the bad pixels to be fixed. These are found by comparing the spectra plotted in this fashion. Regions may be enlarged using the **Z** command. The **P** command may be used to pan out from the zoomed region. Typing **0** will undo any zooms or pans, and **r** will redraw the last spectrum called via the **g** or **#** command. A 2D spectrum in echelle or multispec format may be input as follows:

```
cl> splot obj0006.ms options="auto,xydraw"
Image line/aperture to plot (0:) (1): 5
```

This will bring up the 5th order in an extracted 2D echelle spectrum, or the 5th object in a 2D multispec format spectrum. The *options* specified set the task to replot the graph after certain changes are made (*auto*—this default option does not force a replot when editing, but for several other options such as smoothing with a boxcar fit), and set the mode of use for the **x** interactive command (*xydraw*). The *xydraw* option sets the $x$ command such that a straight line is drawn between the two positions marked with the cursor position, using both $x$ and $y$ coordinates. Without this set, the $x$ command interpolates between the current pixel $y$ values of the two marked positions. Any of the following commands may be used in interactive mode:

- **g** - Plot another spectrum.
- **i** - Write current spectrum as new image.
- **j** - Set the value of the current pixel to the $y$ cursor position.
- **o** - Overplot next requested line/band (#) or spectrum (**g**).
- **q** - Quits interactive mode and saves the modified image.
- **x** - Interpolates or draws straight lines between positions marked.
- **?** - The help page is shown for the interactive commands, use a **q** to get back to interactive mode.
- **Z** - Zoom in on a region.
- **0** - Pan back from a zoom, preserving overplots.
- **#** - Plots a different line in multi-aperture spectra or 2D images.
- **:xydraw [yes/no]** - Defines the mode of use for the **x** command.

There are many more options which may be used interactively however, here we have described only those which pertain to the operation required. These options are all explained in more detail in the help page for this task.

# A Some Important Parameters

## A.1 Rejection Option Associated Parameters (combine, imcombine, & scombine)

The rejection options are discussed in detail in Section 3.2.3. Each of the rejection algorithms require that other parameters be defined. Table 1 lists the *reject* options with a complete list of associated task parameters.

Table 1

| Parameters Used by the Reject Options | | | | |
|---|---|---|---|---|
| minmax | ccdclip | crreject | sigclip<br>avsigclip | pclip |
| nlow<br>nhigh | mclip<br>nkeep<br>rdnoise<br>gain<br>snoise<br>lsigma<br>hsigma<br>sigscale | mclip<br>nkeep<br>rdnoise<br>gain<br>snoise<br>hsigma<br>sigscale | mclip<br>nkeep<br>lsigma<br>hsigma<br>sigscale | pclip<br>nkeep<br>lsigma<br>hsigma |

These parameters are common to all the following tasks: **combine**, **imcombine**, and **scombine**. A brief description of each follows:

- **nlow** Number of low pixel values to reject.
- **nhigh** Number of high pixel values to reject.
- **nkeep** Minimum number of pixels to keep (positive) or the maximum number of pixels to reject (negative).
- **mclip** Use the median value as the estimate of the true intensity when set to "yes" otherwise use the average.
- **lsigma** The lower threshold for sigma rejection of pixel values.
- **hsigma** The upper threshold for sigma rejection of pixel values.
- **rdnoise** The value of the readout noise in electrons for the detector used.
- **gain** The value of the gain in electrons per analog to digital unit (ADU) used for the observations.
- **snoise** The sensitivity noise expressed as a fraction.
- **sigscale** Determines if Poisson corrections are necessary for unscaled images.
- **pclip** Selects the percentile pixel or fraction of pixels to be used with **lsigma** and **hsigma** to determine the sigma clipping thresholds.

# B Technical Issues and Problems

There are many ways of manipulating images in IRAF and therefore problems may arise. Here we try to highlight some things to keep in mind while editing images.

- Pixel values in images should not be set to a value of "INDEF". IRAF does not work well with undefined pixel values.

- Fixing pixels in the background around regions where photometry is to be performed is not a good idea. Since the photometry routines have rejection options, it is best not to fix these unless statistical tests are also performed.

- Images should be registered before being masked. The interpolation routines will soften the edges of regions which have been set to very high or very low values with a mask.

- Images which have stars with small PSFs should not be corrected using the task **cosmicrays**. The stars will be confused with cosmic rays and be deleted from the image. The task **crrej** found in the STSDAS add-on package should be used instead.

- The **lineclean** task should be used with caution when fitting 2D images. A good fit to one line may result in the deletion of good data points in another line.

- One should never assume that IRAF will automatically know the difference between good and bad data. The rejection algorithms will simply apply whatever statistical model you ask; it is the user's responsibility to determine the parameters appropriate for their data.

# C Getting Bad Pixel and Cosmic Ray Positions

There are several ways of obtaining coordinate positions of bad pixel regions in an image. If the coordinates of the display window are trustworthy, then simply displaying an image and moving the cursor to the lower left and upper right corners of the bad regions will give the coordinates. Coordinates from the *imtool* device in SunView are brought up in the lower right corner by pressing the **F6** key within the window. The coordinate box is activated in the *ximtool* window by releasing on **Coords box** in the options menu or choosing the **Coords Box** in the control panel options in the lower right side of the panel. *SAOimage* displays the coordinates at the upper right. The coordinates may not always be exact but the edge of a bad region may be determined to within a pixel.

Using the display device, the task **rimcursor** could also be to determine image coordinates. Unlike the sophisticated centering algorithms used by other tasks, **rimcursor** simply reads the position of the image cursor, without calculating the center. The position is good to only .5 pixels in x and y. This is a good method for getting quick positions. The positions are output to the terminal but they can also be redirected to a file. An output file will look like the following:

```
cl> type rimcoord
379.5 66.5 101 \040
347.5 188.5 101 \040
224.5 130.5 101 \040
385.5 254.5 101 \040
405.  273.75 101 \040
217.75 439.  101 \040
```

This file may have to be edited for use as input to other tasks. The output formats for the coordinates may also be changed by setting *wxformat* and *wyformat*. See the help page for this task for more information about these parameters. To run **rimcursor** the image must first be displayed in the *display* window:

```
cl> display obj0003 1
cl> rimcursor obj0003
or
cl> rimcursor obj0003 > coords1
```

The cursor will move to the *display* window ready for interactive commands when **rimcursor** is executed. Move the cursor to the position of an object and hit any key. To exit this task once coordinates have been obtained for all the regions, type ˆ **d** or ˆ **z** where the ˆ character stands for the "control" key—press both at the same time. The default coordinate system is *logical* or pixel units, however this may be changed to obtain coordinates in another WCS (world coordinate system) by setting the parameter called *wcs* to some other coordinate system value. Note that there is currently a bug in IRAF's cursor readout that can cause the values to sometimes be off by one pixel. This only happens in image dimensions which are of an odd number of pixels, so this problem can be avoided by displaying image sections with even numbers of pixels in each dimension. Alternately, one may use coordinates determined from graphics plots, which are unaffected by this problem.

Another very useful task for finding coordinates is **imexamine**, found in the IMAGES.TV package. It can be used in interactive mode with the image display and graphics options. The operation of finding the object centers is governed by two parameter sets for this task, the **imexamine** parameters themselves and the *rimexam* pset[2] parameters. The standard **imexamine** parameters set up the environment for executing the task, while the *rimexam* parameters specify the options for the use of the *r* and *a* commands used to measure centers. Line plots **l** and column plots **c**, with specifications set by the *limexam* and *cimexam* psets, may also be used to find and identify the edges of bad regions. Output by default goes to the terminal screen so the *keeplog* parameter in *imexamine* should be set to **yes** and the *logfile* name specified if you want to save the output in a file. This logging facility can be turned on and off interactively. Subsequent executions of this task will reopen the *logfile* and append to the named file.

The centering parameter is found in the *rimexam* pset and must be set to **yes**. The cursor position is used as the initial point for computing the center moments of the marginal distributions in x and y. The marginal distributions are obtained from a square aperture with edge dimensions of twice the aperture radius parameter. Only pixel values above the mean are used in the computation and another iteration is done if the central moments are in a different pixel than that used for extracting the marginal distributions. This may be used for cosmic rays but may not work well on bad pixel regions.

The cursor commands which are most important are:

- **a** - centers and performs circular aperture photometry for a star.
- **c** - plot the column under the cursor.
- **g** - move to the graphics cursor from the image display.
- **i** - move to the image cursor from the graphics window.
- **l** - plot the line under the cursor.
- **m** - pixel statistics in cursor region.

[2]Psets are parameter sets within parameter sets. Psets provide a way to group a long list of parameters into smaller, more manageable parameter lists that can then be shared by several tasks in a package.

- **r** - the **a** key plus plotting the radial profile of the star.
- **s** - show a surface plot of region centered on the cursor.
- **w** - toggle writing to the output file.
- **x** - print coordinates
- **z** - print a table of pixel values for region near cursor.
- **C** - position of the cursor in the graphics window.
- **Z** - zoom in the graphics window.
- **0** - unzoom in the graphics window.
- **?** - print help menu.
- **q** - quit the task.

In interactive mode, any of these commands may be used. Running the task causes a circular cursor to appear in the display window. Use of the **c** command will plot the column at the position of the cursor. Typing **g** will move the interactive cursor to the graphics window. Now using the **C** command will give the position of the cursor in the graphics window. Interactive mode is exited using **q**.

In some instances, the user may need to find pixel coordinates without the use of an image display device. This can be done using **implot**, though finding bad regions may be difficult if you don't already have some idea of where to look. This task will allow the inspection of line and column plots similar to those obtainable with **imexamine**. The most useful options are:

- **c** - plot the column corresponding to the cursor position.
- **e** - expand plot by marking corners of viewport.
- **j** - move down in image (*step* number of lines of columns).
- **k** - move up in image (*step* number of lines of columns).
- **l** - plot the line corresponding to the cursor position.
- **C** - show coordinates of cursor.
- **space bar** - show coordinates of cursor, plus graph value at cursor $x$.
- **?** - print help menu for task.
- **:c #** - plot column number #.
- **:l #** - plot line number #.
- **:x x1 x2** - change range of x-axis in current plot.
- **:x** - restore displaying of full x-axis range.
- **:step #** - reset *step* parameter to new #.

- **q** - quit the task.

Running **cosmicrays** and specifying a *badpix* file will create a list of the cosmic-ray positions in the image being fixed. This output list may then be used with **badpiximage** to create a bad pixel mask. It would not be good at finding cosmic rays which produce streaks and it would not identify bad pixel regions, however.

# D Useful Tasks for Making Masks

The following tasks may be used to create mask files to be used with **imarith** or **imcombine**. Values for the mask may be set to 0.0 and 1.0 or to very large positive or negative values to set bad regions to highly deviant values. The task **mkpattern** may be used to create a mask or create an image to be edited further. The task **imreplace** is described in Section 3.3.1, however, it is also useful in creating masks. **Badpiximage** works with the table formatted for use with the **fixpix** task. Files of a special type called "pixel list files" may be created using a simple **imcopy** of an image and specifying an output image name with a ".pl" extension.

## D.1 mkpattern

The **mkpattern** task is found in the NOAO.ARTDATA package. This task creates an image of a size determined from the input dimensions specified by *ndims*, *ncols* and *nrows*. Higher dimension images may also be created. If the input image already exists, the requested pattern may be substituted for, added to, or multiplied by a given image section. The various *pattern* options are:

- **constant** - Set the image values to a constant value specified by *v1*.
- **grid** - A grid starting with the first pixel and going in steps of the pattern *size* with value *v2*. A minimum grid *size* of 2 is enforced.
- **checker** - A checkerboard with squares of the pattern *size* alternating between values *v1* and *v2* starting with *v1*.
- **coordinates** - Each pixel is numbered sequentially starting with 1 with the column dimension varying fastest.
- **slope** - A sloped plane starting with value *v1* for the first pixel and value *v2* for the last pixel in one or two dimensions.
- **square** - A checkerboard pattern in which the size of the squares begins with the pattern *size* and grow as the square of the coordinate.

The pattern size is specified by the parameter *size*. The values for the pattern are also specified in *v1* and *v2*. Sections of an image may also be edited by specifying a region to be edited. Certain header parameters may be set also such as *title* and *pixtype*.

`cl>` **mkpattern blank ncols=800 nlines=800**

This example creates a blank image of the given size. This image could then be edited with **epix**, **imreplace**, **imedit**, or **mkpattern** itself to create a bad pixel mask image.

cl> **mkpattern check pattern=checker option=add v1=0 v2=1.0 size=70**

This example adds a checkerboard pattern, with 70 pixel wide squares with 0.0 and 1.0 being the alternating values, to an existing image, or creates a 512x512 checkerboard image if the input image does not already exist. *Mkpattern* may also be used to modify a pattern in a section of an existing image:

cl> **mkpattern obj0005[245:360,300:400] pattern=constant option=replace**

Pixels in the region specified will be replaced by the constant value in the *v1* parameter. Check the help page for this task for more information.

## D.2 imreplace

The **imreplace** task is found in the PROTO package. This simply replaces the pixel values of a given region, specified by an image section, and/or a range in current values. The replacement value must be specified and an imaginary part for a complex number may be incorporated. All values between the values specified in *lower* and *upper* will be replaced if the whole image is defined. If for example an image has peak counts in the objects at 10000 ADU, but cosmic rays which peak at 24000 ADU, a replacement of cosmic rays can be done using:

cl> **imreplace obj0005 1.0 lower=11000.0 upper=30000**

This will set all pixels in the image with current values between 11000 and 30000 to a value of 1.0. However, this wouldn't handle any weaker cosmic ray features.

To create a pixel list file mask, one may begin with a blank file, all of whose pixels are 0.0, and edit bad pixel regions explicitly:

cl> **mkpattern mask.pl ncols=800 nlines=800**
cl> **imreplace mask.pl[356:450,689:693] 1.0 lower=INDEF upper=INDEF**

Notice that the tasks can create and edit pixel list files as well as images. Of course, if we had desired an image we would have left off the explicit ".pl" extensions. Making a detailed bad pixel mask will take time to create, but once it is ready it may be applied to all the images taken with that CCD, since bad regions do not usually jump around. Creating a mask using the badpix file format used by the task **fixpix** is described in the next section.

## D.3 badpiximage

The task **badpiximage** is found in the IMRED.CCDRED package. The bad pixel list used for the task **fixpix** may be input to create a mask image to be applied to the program images. The format for this table is the first and last columns of the bad region followed by the first and last lines of the bad region, i.e., "xbegin xend ybegin yend" separated by spaces not commas. There are several formats for the output mask. A standard IRAF image may be created for use with the various image arithmetic tasks. A mask image in a pixel list format file having a ".pl" extension is another option. If the **image** parameter is a name with the ".pl" extension, then the output is a pixel list, and an IRAF image is created if the ".pl" extension is not specified.

cl> **badpiximage badpix obj0003 obj3mask.pl goodvalue=0 badvalue=1**
cl> **hedit obj0003 BPM "obj3mask.pl" add+ verify+**

A pixel list format file is created setting the good values in the image mask to 0 and the bad regions specified in the badpix file to 1. The pixel mask image name is added to the header of the corresponding image to be applied when using tasks such as **combine** and **imcombine** (see Section 3.2.3).

cl> **badpiximage badpix obj0003 obj3mask goodvalue=0.0 badvalue=30000**

An image is created with good pixel values set to 0 and bad pixel values set to 30000. This image may then be added to an image to set the bad regions to very large numbers using the **imarith** task (see Section 3.1.3).

## D.4 imcopy

The **imcopy** task is found in the IMAGES package. It can be used to copy a section of one image into another image:

cl> **imcopy obj0005[194:207,452:489] obj0006[194:207,452:489]**

**Imcopy** may also be used for converting bad pixel images into pixel list files:

cl> **imcopy obj3mask obj3mask.pl**
cl> **hedit obj0003 BPM "obj3mask.pl" add+ verify+**

The name of the pixel list file is added to the header using the **hedit** task. The bad pixel list is used when running tasks such as **combine** and **imcombine** (see Section 3.2.3).

## D.5 imexpr

The task **imexpr** is found in the IMAGES package available only in the 2.10.3 or later releases of IRAF. This task allows for more complex image arithmetic calculations than those possible with **imarith**. Masks may be created with this task and applied to the images. The many arguments of the operation are specified in the operands *a* thru *z*. The *expression* is declared in terms of these operands. The operands may take three forms, an image name, an image header, and numeric constants. These are used in the *expression* along with any of the possible operators and functions.

The functions are too numerous to list here, however descriptions are found in the help page for this task. It is possible to replicate sections of an image, swap the right and left sides of an image, add several images and/or a constant, create new images, and make pixel list masks. The two parameters *intype* and *outtype* control the datatype for the input and output images. If the expression is to be evaluated in floating point precision, then the *intype* should be set to **real**.

For bad regions in an image below the values 200 adu and cosmic rays above the value of 20000 adu, a mask image is created using the following:

cl> **imexpr "a > 200 && a < 20000" mask a=obj0005**

The bad regions will be set to 0 while the good regions are set to 1 in the output image. The mask may instead be created as a pixel list file by adding the ".pl" extension.

## E Adding Noise to an Image: mknoise

The **mknoise** task is found in the NOAO.ARTDATA package. It will make or add noise and cosmic rays to a 1 or 2D image. The *gain* and *readnoise* may be specified for computing Poisson noise. The noise is added on top of the *background* defined. Positions of cosmic rays may be specified in a file or a number (*ncosrays*) are scattered randomly throughout the image. The list must include the positions and intensities of the cosmic rays to be produced.

```
cl> type crlist
20.5 40.6 1000
103.8 179.4 3000
230.0 15.8 300
647.2 480.2 400
690.4 275.9 1000
cl> mknoise obj0004 out=obj4cr cos=crlist
```

This example adds the cosmic rays listed in the file to the image and creates a new image.

```
cl> mknoise obj0006 back=350 rdnoise=8.5 gain=3.4 poisson+ ncos=30
```

This example adds Poisson noise to the specified background level and 30 cosmic rays to the image and then overwrites the image. Tasks such as **cosmicrays** may be used on the output image to test the limits of the search algorithms.

## F Other Useful Documentation

Help pages are available for all the tasks mentioned in this document. More information on topics covered in this manual may be found in the following documentation, accessed by anonymous ftp to *iraf.noao.edu*, or 140.252.1.1:

- **Cleaning Images of Bad Pixels and Cosmic Rays Using IRAF**, by Lisa Wells and David Bell, September 1994 (this manual). (iraf/docs/clean.ps.Z)
- **A Beginner's Guide to Using IRAF**, by Jeannette Barnes, August 1993. (pub/beguide.ps.Z)
- **A User's Guide to CCD Reductions with IRAF**, by Philip Massey, June 1992. (iraf/docs/ccduser2.ps.Z)
- **Rectifying and Registering Images Using IRAF**, by Lisa Wells, January 1994. (iraf/docs/reg.ps.Z)
- **Mosaicing Images Using IRAF**, (in progress). (iraf/docs/mosaic.ps.Z)
- **A User's Guide to Reducing Slit Spectra with IRAF**, by Phil Massey, Frank Valdes, and Jeannette Barnes, April 1992. (iraf/docs/spect.ps.Z)