

# IRAF Tutorial for Exercise III

## Image Processing: Bias Frames and Flatfielding

### Learning Goals

- List the various sources that contribute to the amount of charge present in each pixel after an exposure.
- State which sources of unwanted charge are additive and which are multiplicative.
- Demonstrate proficiency in the IRAF tasks used in this tutorial and exercise.
  - imcopy, imarith, unlearn, imstat
  - disp, implot, colbias, sections, darksub
  - CCDRED, package, type
- Summarize the overall steps needed to process a raw CCD image.

### Files Used

/astro/classes/Astro\_480/exercises/exercise2/m92.tar.gz

### Introduction

Although CCDs have revolutionized astronomy, they are not perfect. Each individual pixel in a CCD array can be thought of as a device that linearly converts light to charge,  $q$  (electrons), so that for an integration time  $t$ ,

$$q_i = A_i t \int I_i(\lambda) QE_i(\lambda) d\lambda + B_i + D_i t$$

where  $A_i$  and  $B_i$  are the "gain" and "offset" parameters that characterize the pixel  $i$ ,  $I_i(\lambda)$  is the incident light flux and  $QE_i(\lambda)$  is the pixel's efficiency, which is wavelength dependent, integrated over some passband that is usually defined by an optical filter. There is also some thermally generated "dark current,"  $D_i t$ , (dark count per second times the number of seconds) that accumulates during this interval. You should think of a CCD as an array of pixels, each with their own individual gain and offset values.

The values of  $A_i$  and  $B_i$  vary from pixel to pixel across the CCD array, and we need to correct for this. As Craig Mackay, one of the grand practitioners of CCD astronomy, once said: "The only uniform CCD is a dead CCD." In addition, the electronics used to read out the detector also introduces a nonzero "bias" value that is added after the signal is converted to an analog voltage, so that the measured voltage associated with pixel  $i$  is

$$V_i = q_i + Bias(t).$$

This bias structure can be quite complex. Sometimes the bias level changes steadily during the course of the detector being read out. In other cases the bias structure changes along a row as the pixels are read out, and there is a constant bias "vector" that needs to be subtracted from each row. These effects arise because of non-idealities in the readout electronics, due to loading effects and temporal and thermal instabilities in the electronics. (You should have some sympathy for the challenge of maintaining microvolt stability over the environmental variations that occur in a telescope dome.)

If we illuminated the entire CCD with a uniform ("flat") intensity distribution, we could correct for the variations in pixel sensitivity, for the pixel-to-pixel differences in offset values, as well as for the dark current and the bias structure. In addition, the telescope and instrument that lie between the detector and the light source can also introduce both spatial ("vignetting") and wavelength-dependent variations in overall system sensitivity,

which must also be compensated for. The process of manipulating the raw CCD data file to correct for these effects is called "flatfielding." This involves compensating for both additive and multiplicative non-idealities.

Flatfielding is something of an art. You might think that the task of overcoming sensitivity variations would be simple: just take a frame of a uniformly illuminated field, use the image to determine the pixel-to-pixel variations, and divide all data frames by the appropriately normalized "sensitivity flat." This is certainly the basic approach we take, but the practical challenges include:

- How do you generate a genuinely uniform flux incident upon the system?
- Is the spectral energy distribution of the calibration source the same as the science targets?
- How do you disentangle the effects of the detector, the instrument, the optics, the filters, the telescope and the atmosphere?
- How can you best compensate for changes in sensitivity that depend on telescope orientation?

### **Flatfields: Dome Flats vs. Sky Flats**

There are two common ways that astronomers attempt to generate uniform illumination for characterizing CCD instruments: 1) dome flats and 2) sky flats (twilight or blank night sky). Regardless of the technique used, it is *essential* that you obtain flats **for every filter used**. One set of bias frames will work for all filters, though, as that part is wavelength independent.

Dome flats use a reflective screen painted on the inside of the telescope dome and illuminated appropriately. The observer points the telescope at it for calibration frames. Dome flats are often taken in the afternoon before an observing run. It is a good time to characterize and understand the detector and instrument for an unfamiliar setup. The dome flats can also be reduced before the sun sets, allowing for near real-time preliminary reductions of frames as they are being acquired. This has, in our experience, often proven very useful in maximizing the use of telescope time. There are some major disadvantages to dome flats, though. For one thing, there is often a lot of stray and scattered light that simply isn't present during night-time observing conditions. Also, the spectral energy distribution of the light from the flat-field screen is seldom much like the light from astronomical objects, so the integral across the passbands is not a good match to that of the data frames. It is still worth obtaining dome flats if possible, since it provides a consistency check and, as noted above, they are often good enough for mountaintop reductions.

Sky flats use the brightness of the sky itself for obtaining flatfield calibration data. There are even two kinds of sky flats: twilight flats and blank sky flats.

Twilight sky flats are taken while the sky is too bright for taking science data; typically the detector is halfway to saturation in just a few seconds or less. It is important to offset the telescope between successive sky flats to suppress the effects of bright stars in the field of view. **It is also important that the telescope be close to nominal focus, or the light path through the instrument will not mimic operating conditions.** Twilight is often the most hectic time during an observing run. The sky brightness is changing exponentially with time (more so in late autumn and early spring), and getting the right exposure level in a succession of frames and in multiple filters takes practice. An advantage to twilight sky flats is that you can make use of otherwise unproductive time on the telescope. A downside is that the twilight sky is somewhat polarized (since it is reflecting sunlight) and this can be a complication.

Blank sky flats, on the other hand, are often generated from the night's science images. If the science program involves multiple exposures of largely uncrowded fields, a median can be taken of the images to generate an overall flat field. There are significant advantages to this approach:

- Most astronomical objects of interest are fainter than the sky, so most of the flux hitting a pixel has the spectral character of the sky.
- This method makes maximum use of telescope time, as the calibration and science data are taken at the same time.

A downside is that it is often hard to get enough total counts to make a high signal-to-noise blank night sky flat. This is particularly true for narrow-band filters, or other systems where the optical throughput is low. The approach also does not work if, for example, you spend the whole observing run looking at large bright spiral galaxies. The method relies on the assumption that on average the flux distribution across the field of view is uniform, which would not be true in the last example.

A hybrid approach is to use the high flux in the dome flats to determine the small scale pixel-to-pixel sensitivity variations, and to use the sky flats to make an "illumination correction" to compensate for the non-uniform illumination that is common in dome flats. This, however, is a little beyond the scope of our course .

### Our steps for obtaining twilight flats at the A-Wing Observatory

- Know the linear regime for the particular CCD you are using.
- Start with the filters where the QE of the CCD is the lowest and the bandpass the smallest. This way, you can have a reasonable exposure and ADU counts while the sky is still fairly bright.
- Recall the idiosyncrasies of obtaining good twilight flats: angle from the Sun and polarization. [The Flat Sky: Calibration and Background Uniformity in Wide-Field Astronomical Images (Chromey & Hasselbacher, PASP, 108, 944, 1996)]
- Take trial exposures until you reach an ADU count that is certainly within the linear region; take a total of 5 exposures (at least) through that filter.
- Examine the set of flats for any irregularities (such as stars!) and make any needed adjustments. If stars are present, then shift the telescope slightly in between exposures and use the median when combining.
- Work fast, as once the Sun goes down, astronomical twilight comes quickly.
- Final set of flats are taken through the clear filter as the sky will have darkened significantly.

### Dark Frames versus Bias Frames

Similarly, the determination of the "bias" or "offset" structure turns out to be a little more complex than you might think. By taking "dark" frames at different exposure times, the contribution of the dark current can, in principle, be isolated from the other sources of structure on the CCD array. Even this is sometimes non-trivial, as light leakage and light emission from the detector itself can complicate matters.

For most instances where a liquid nitrogen cooled detector is used, the dark current is negligible. [Only for very high dispersion spectroscopy, where only a few photons strike each pixel, is this a concern. We will not encounter this case in Astronomy 480.] When using an instrument for the first time, you should nevertheless verify that dark current is small by taking a long (300 seconds or more) dark frame and comparing the imaging region with the overscan or bias level if there is no overscan region. You may find the imaging region to have a few ADUs due to spurious charge generation during the charge transfer process, but this will be independent of the exposure time and can be considered as part of the offset structure.



**NOTE:** As you work through this tutorial and exercise, you should always remember that the steps are set up to reduce **Kitt Peak data**. When we reduce images using other telescopes, some of the steps and certainly many of the parameters will necessarily change.

**Image Arithmetic in IRAF - The steps needed for the exercise answer sheet start here. You will be directed as to when to answer questions on the answer sheet.**

Obviously, for **our** flatfielding manipulations we will need to perform additive and multiplicative operations on images. Also, a variety of averaging procedures (average, median, mode) are also useful. IRAF provides nice utilities for these operations.



The exercises below assume that you have the images **m92000[1-6].fits** available. If you don't, then go back to the appropriate exercise where we grabbed those images, and follow the directions.

IRAF images named **m92\*.fits** should now be in the directory. There should be a bias frame as well as two flat fields and four images all taken through either V or B filters. The image headers will identify which is which.

Assume we have two frames, **im010** and **s011**, that we want to average. Copy two of the images in your exercise2 directory to new names so you can play a bit and not clobber files you will actually need.

```
ecl> imcopy m920006 im010 # NOTE: we could use cp or mv
ecl> imcopy m920007 s011 # to delete, we'd use imdelete or just del
```

We will do the averaging in two ways. We can do the same thing on the command line or by epar'ing the task.

```
ecl> unlearn imsum imarith # always important to start fresh!
ecl> lpar imsum
```

#Be careful about the spacing on the following command.

```
ecl> imsum im010,s011 aver1 pixt=r calct=r option=average v+ # or try "epar imsum", modify all of the
# parameters, and then type ":go"
```

[Note the concern in the above command about the pixel type, both for the calculation and for the output image.]

```
ecl> lpar imarith
ecl> imarith im010 + s011 aver2 pixt=r calct=r v+ # try "epar imarith", edit parameters, type
# ":go"
ecl> imarith aver2 / 2.0 aver3 # notice you can use scalars too!
ecl> unlearn imstat # start fresh
ecl> lpar imstat
ecl> imstat aver*
```



**NOTE:** When you change hidden parameters on the command line they are NOT "learn"ed! How do you "learn" parameters? The command line input for temporary parameter values differs from task to task. Understand pixel types. When programming, we sometimes need to declare variables. Since all a task is doing is calling a computer program or programs, number variables should be declared as well. Our options are (on the next page):

- double: double precision
- real: single precision
- long: 32 bit integers
- short: 16 bit integers
- int: machine default (usually short)

**After this practice, answer questions 1, 2, and 3 of the exercise on the answer sheet.**



**An aside on data representation:** You probably noted that in the above examples the data were to be specified as "real" rather than "ushort" numbers. This idea is probably familiar to people who have programmed in computing languages that allow you to specify the "type" of number held in a variable. There is a tradeoff between storage space and the dynamic range that can be accommodated in a number. Although CCD data are usually taken with 16 bit A/D converters, it is useful to change the data type of an image to a "real" early in the reduction process. This admittedly doubles the storage space needed for the images, but disks are getting cheaper with each passing day, so this is no longer much of a concern. The benefit of minimizing roundoff and truncation errors more than makes up for the extra storage space. The "imhead" task is a quick way to see the data type in an image.

## Subarrays

It is often useful to be able to specify subregions within an image. IRAF does this using the column and row values to designate any given pixel. The first pixel in the image "foo.fits" is designated by foo[1,1]. A region between rows R1 and R2, and columns C1 and C2 is designated as foo[C1:C2,R1:R2]. The syntax with commas and colons is important. For example, the first 100 rows and columns of the image "testframe" would be designated "testframe[1:100,1:100]." For example, you can compute the statistics on an image region by typing

```
ecl> imstat m920001[200:300,300:400]           # NO SPACES after m920001 nor within [ and ]
```

**Answer question 4. You will need to imstat again using a different section of the chip.**

## Flatfielding in IRAF

The rest of this exercise is designed to show you how IRAF deals with the preliminary reductions of CCD data, including the overscan subtraction, the bias or zero level subtraction, the dark subtraction, and the flat fielding. The images for this exercise are direct imaging data taken at the Kitt Peak National Observatory by Dr. George Jacoby. This exercise assumes that you have worked through IRAF Exercises 1 and 2 and feel comfortable with the basics of IRAF.

We will approach this exercise from two different paths. The first path is the "long" approach to the problem, but will allow you to step through this process one task at a time. The second path is the preferable way to do these preliminary reductions but for the first time user the actual steps may not be obvious. The second method takes full advantage of keywords in the FITS image header that distinguish between bias frames, flats and science images. This works very well, as long as you keep the header keywords current when you are taking the calibration data. Even if you forget, there are ways to edit the image header within IRAF to groom a stack of images for the streamlined processing pipeline.

You should be logged into IRAF in an xgterm window and have started IRAF. Get to the proper directory within IRAF where your images are. It will be helpful to display images using the DS9 window for this exercise.

```
ecl> ls
```

```
ecl> unlearn imhead
```

```
ecl> imhead m92*.fits
```

```
ecl> imhead m92*.fits lo+ | less
```

# This command lets you see all of the information from all of the image headers; the "less" command lets you scroll through them.

To reduce the noise (proportional to the square root of the number of frames averaged), the bias frame is an average of 25 frames. Each flat is an average of 5 frames to improve the signal to noise. Notice that the pixel type is "short" or 16-bit data. Look closely at all of the information contained in these image headers. When were the flat fields taken? Was the declination of any of the flats the same as the declination of the object M92? Does this matter? Does the imheader information give some indication of exactly which pixels are real and which range of pixels qualify as the overscan region?

Make sure that your IRAF display parameters have the scale set to linear and **not** log to highlight the overscan region. The average bias frame has an overscan region, as do the 4 science images. However, the flat fields (blue and visual filters) do not. The overscan region may display differently if you use the drop-menus from DS9 versus display command within IRAF. We found that the DS9 viewer did **not** show the overscan region if we opened a file from the drop menus.

## PATH 1. – Step by Step Flatfielding

We follow much the same steps in reducing photometric data no matter what equipment is used or types of objects observed. The first step would be to average the bias frames. This can be done with the task IMCOMBINE in the IMAGES.IMMATCH package. We would then do the same for the flats. Some type of pixel rejection could be used during this step to eliminate bad pixels or cosmic rays. Since these steps have already been done for us in this exercise, we can continue on to the overscan subtraction of the bias for images **m920004.fits** through **m920007.fits**.

We need to determine two things at this point: 1) the overscan region to subtract from the bias and science images and 2) the trimming parameters to determine the output image size. For this chip the overscan region is 32 columns wide but we often do not use all of the columns. The overscan region and any bad rows or columns along the edges of the frame are then trimmed from the image to produce our output image. [The columns and rows at the edge of the CCD often contain excess charge that diffuses in from the bulk silicon around the pixel array. It is often advisable to crop these out of the image.] We determine these parameters with IMPLOT using one of the flat field frames. IMPLOT uses the graphics window and keyboard commands to draw useful plots, especially cross-sections of images. The big advantage of IMPLOT over IMEXAMINE is its ability to average over rows and columns. Have the actual image itself displayed in the DS9 window while doing this.

```
ecl> disp m920006 1
```

# display in frame 1

```
ecl> implot m920006
```

Spend some time becoming familiar with "**implot**" as you will use it often. Try the global keys as well. Notice that when the **implot** screen first comes up, you can move the cursor around the graph and hit "I" (the letter 'ell') or "c" to produce a line or column plot at the location in the image that corresponds to the cursor location

in the graphics window. This takes a little getting used to. You can also specify a set of rows or columns to average over, as shown below. (Remember, use the letter 'l' not the number '1' for line.)

```
:c          #column plot, column read from cursor position
:l 100      #plot line 100
:c 150 200  #plot average of columns 150-200
```

How can you expand the plot other than with "Z"? look at the "e" key - put the cursor at the lower left corner of a box defining the region you wish to zoom, press "e", then move the cursor to the upper right corner of the box and press "e" again

```
:l 100      #get the full plot size back by typing the letter l
q          # exit
:x 100 500  #change the range of the x axis of the plot
:x        #recover the default
```

The same trick can be used for the y axis as well. Be sure you understand the distinction between setting the scaling for the plot and averaging over rows and columns in the image.

Notice that when you are in **line mode**, the columns are plotted, but the line numbers run up the right-hand axis. You can use this scale to plot other lines.

When in **column mode** and reading up across the lines, the column numbers are indicated on the right-hand axis. Rows 1 ~ 125 are bad in this bias image.

Determine the columns in image "**m920006**" to use for the overscan. We found using **e**, **:l n m**, **:c n m**, and **C** to be very helpful. Try to avoid using the sloping part of the overscan. You will need to expand the plot around the overscan region. Once you have determined the columns to use for the overscan, plot the average of these columns. What columns did you decide to use? We like columns 335-350; are yours close to those values? Write down your column numbers on the worksheet for this exercise.

***Use the worksheet for the following sections; move between the tutorial and worksheet as you work, starting with question 5.***

Now decide what columns and rows will be included in your output image. Plot several rows and columns and see what these values should be. Again you will need to expand the plot. Look at the rows first. Do you see any bad columns at the left edge? What about the right edge? We certainly want to trim off the overscan plus a bad column or two on the right edge of the plot. Then look at the columns and rows; there are some bad rows.

Expand each edge and determine the usable range of rows. The values that we determined for the trimming parameters are 1-318 for the columns, and 2-510 for the rows. Do you agree? Once we have this information we are ready to do the overscan subtraction and trimming. Load the packages. And then edit the task COLBIAS to reflect the values that we determined.



NOTE: Within IRAF if you see a different prompt, it is just indicating the package you are currently in. All packages that you loaded are still loaded, and their tasks can still be run no matter what the prompt is.

You may decide to put the packages that you use often into your **login.cl** file so that the packages are already loaded and you can run a task immediately.

```

ecl>noao
no>imred
im> bias
im> phelp colbias
im> unlearn colbias
im> epar colbias          #Remember that you can save the changes with :q or <ctrl> d

```

Edit as needed so that running LPAR on COLBIAS shows parameters similar to the following but where you insert YOUR values from questions 5 and 6 of the exercise where applicable. Notice that the overscan and trim values are entered as "image sections," the x-range and y-range in square brackets. The trim section is that part of the image we wish to keep.

```

input = "m92*.fits"          Input images
output = "%m%tr%92*.fits"    Output images
(bias = "[335:350,2:510]")    Bias section
(trim = "[1:318,2:510]")     Trim section
(median = no)                Use median instead of average in column bias?
(interactive = yes)          Interactive? (function = "chebyshev")
Fitting function (order = 1)  Order of fitting function
(low_reject = 3.)            Low sigma rejection factor
(high_reject = 3.)           High sigma rejection factor
(niterate = 1)               Number of rejection iterations
(logfiles = "")              Log files
(graphics = "stdgraph")      Graphics output device
(cursor = "")                 Graphics cursor input
(mode = "ql")

```



**Thought Question :** Do you understand the output image names? Try the following to see what the actual names on output will be. The task SECTIONS can be used to test image templates. In this case, the % sign brackets that part of the image name we wish to replace (**m**) and what we wish to replace it with (**tr**). `im> sections %m%tr%92*.fits`

We are ready to execute COLBIAS. This task will subtract the overscan from each image and then trim the image according to our specifications. Since the task is being run interactively we will first see a plot of the average of the overscan vector. We could modify the fitting parameters at this time but we like to use a straight line for these data. **Notice the fitting parameters at the top of the plot if we needed to use them.** A return is sufficient for the task queries - type q in plot mode to continue.

```

ecl> colbias
ecl> ls -l
ecl> imhead tr*.fits          # notice the new size of these images
ecl> display tr920007 1       # check your trimming

```

There is a philosophical decision you need to make at this point, namely whether to overwrite the original images or not. There are two schools of thought on whether intermediate steps in data reduction should be stored. Both schools agree that you should always save at least 2 copies of the original raw data, in different places/machines. Let's proceed with the idea that we will overwrite the trimmed images generated above. There

is an IRAF safety parameter that determines whether it will allow you to overwrite an existing file or not. It's set in your **login.cl** file in the "safe" mode, so the first step is to allow existing files to be overwritten ("clobbered").

The next step is to subtract the bias or zero frame from each of the images. This is best done with IMARITH. Before you start subtracting the bias frame (**m920001**), take a look at it. Is there structure in the bias image that is consistent from row to row?

Let us first create a file with a list of the images to process; we will use this as input and output to IMARITH, overwriting our input data. This will use the abilities of IRAF to batch process images, a real time-saver.



**NOTE:** For the next step you are going to edit a file from within IRAF. When you edit **zlist** you get to **vi**, unless you specified a different default editor in your login.cl file. (Even then, the program may default to vi. Beware!)

```
im> set clobber = yes                # temporarily allows files to be overwritten
im> files tr*.fits > zlist           # redirected output makes a nice list
im> edit zlist                       # delete bias frame (tr920001) from list [NOTE: You may be
                                     # dropped into the editor VI and not know what to do. Ask!]
im> imhead @zlist                    # this is IRAF's syntax for redirected input
im> unlearn imarith
im> epar imarith
```

Now edit your IMARITH parameter file so it looks like the following (the list continues on the next page). This will subtract the trimmed averaged bias frame from each of the images specified in **zlist**.

```
operand1 = "@zlist"    Operand image or numerical constant
op = "-"              Operator
operand2 = "tr920001" Operand image or numerical constant "
result = "@zlist"     Resultant image
(title = "")          Title for resultant image
(divzero = 0.)        Replacement value for division by zero
(hparams = "")        List of header parameters
(pixtype = "")        Pixel type for resultant image
(calctype = "")       Calculation data type
(verbose = yes)        Print operations?
(noact = no)          Print operations without performing them?
(mode = "ql")
```

```
im> imarith            # Execute IMARITH.
```

Notice that as IRAF is plowing through these tasks, the image's header information is updated, keeping some track of the operations that have been performed. You can see this by typing:

```
im> imhead tr*.fits lo+ | page
```

and looking at the entry "IRAF - TLM", for example.

At this stage, the images have been bias-corrected and cropped. This is the point where any dark subtraction would be done. That would be done using the task DARKSUB in the NOAO.IMRED.GENERIC package. The frames need to be scaled by exposure time before the subtraction is done, so this information would need to be in the header. If we were to subtract the average dark from the images and flats, the averaged bias frame would have to be subtracted from the dark first. For this exercise, we will skip the dark current subtraction step.

We finally arrive at the quantum efficiency correction stage. We have two flats and **they need to be normalized** before we divide them into our object frames. We will use IMSTATISTICS (imstat) to determine the normalization value for each flat, and then use IMARITH to create the normalized flats. You can use **imhead** to see that frames **tr920002** and **tr920003** are the averages of B and V flats, respectively.

We want to normalize the flats so that the typical pixel is unchanged. We do this by computing the **mode** of the values in the flatfield images.

```
ecl> phelp imstat
```

```
ecl> imstat tr920002,tr920003 fields="image,mode"
```

```
ecl> imarith tr920002 / 1313.0 Bflat           # normalizes the frame to unity; is 1313.0 the correct value?
```

```
ecl> imarith tr920003 / 1468.0 Vflat         # normalizes this frame, too; is 1468.0 the correct value?
```

```
ecl> implot Bflat                             # check Bflat
```

```
ecl> implot Vflat                             # always check!
```

```
ecl> disp Bflat 1                             # look at Bflat
```

```
ecl> disp Vflat 2                             # always check! Never assume all is well.
```



**Thought Question :** Why do we use the mode to normalize the frame? Why do we even want to normalize the flat frame? What does this end up doing to the image after we have flat-fielded it? The modes in these frames are fairly low and close in value. What would be the effect of your not normalizing the flat frames if your modes were, for example, 18,000 ADU's in one filter and 12,500 ADU's in another when it came to comparing the results?

Look at the two flat frames, using DS9. You can see quite a number of "features," including bad columns, dust spots (which appear as donuts), and the swirling patterns of "fringing."

We now divide each of the object frames by the appropriate flat. It is your responsibility to substitute in the correct image names for the operations. Note that each passband (filter) has a different flat.

```
ecl> imhead tr*                               #find out which ones are the taken through the V and B filters; don't
```

```
ecl> imarith tr920004 / Vflat n920004       just process like a bad robot
```

Also do the appropriate steps for images **tr920005 – tr920007**. **PAY ATTENTION TO THE PASSBANDS AND BE SURE YOU USE THE RIGHT FLAT!** Be sure you change the name of the **imarith** result images as appropriate, too. Once this is done you should have a set of 4 trimmed, bias-subtracted, and QE-corrected images.

```
ecl> imhead n92*
```

Look at these final images with DISPLAY and/or IMPLOT. Check to see if the sky is flat across the image. Sometimes the dome flats are not sufficient for flattening images - additional sky flats may need to be used. See the task MKSKYFLAT in the CCDRED package.



**You may want to delete the finished images since we are going to reprocess the raw data again, but using the other, more automatic path. However, unless you are quite sure you won't make a mistake in the next step, it is advisable to create a subdirectory where you move your finished, processed images. In any event, none of the images created so far should be kept in your working directory for the remainder of this exercise.**

```
ecl> imdelete tr*,n92*,Bflat,Vflat ver+
```

```
# no spaces except after imdelete and Vflat
```

```
ecl> del zlist
```

```
ecl> imhead m92*
```

## PATH 2. – CCDPROC

There is a very slick package in IRAF that can be used to batch process bias frames, flats, and science images. The CCDRED package contains CCDPROC, which is a very powerful utility. We have had you work through the elementary steps first because not all telescopes and detectors are set up to put in all of the correct keywords needed for CCDPROC. Besides, it's good to see the rudimentary guts of this animal called IRAF.

We **strongly** recommend that you now edit your **login.cl** file and set clobber = yes. If you do not clobber the files, then CCDPROC will create new files and the program may bomb.

Check to see what files are in our directory and refresh your memory as to what each image contains.

```
ecl>imhead m92*.fits
```

We want to use the tasks in the CCDRED package now to reduce these same data. Check to see what packages are loaded.

```
ecl> package
```

Now load the necessary packages - CCDRED is in NOAO.IMRED. The CCDRED package will process our data in the same way as we did previously. However, the steps are combined into one task, and we can use the information in the headers of the images to drive the task.

The CCDRED package looks for certain keywords and values in the header. If the keywords and values have different names than those expected by the package then a "translation" file can be used. The package expects the keywords IMAGETYP (with values "object," "flat," "zero," among others), EXPTIME (for dark subtraction), SUBSET (to define the filters), just to mention the ones we will be using. The task CCDLIST can be used as a check to be sure the package is picking up the header information correctly.

```

ecl> imhead m920005 lo+           # look for imagetyp, exptime, subset
ecl> unlearn ccdred
ecl> lpar ccdlist                 # get in habit of seeing what parameters need setting
ecl> ccdlist m92*.fits
ecl> ccdlist m92*.fits lo+       # what does this tell us?
Since this is KPNO data we already have a translation file set up so let's use it and see what happens.
ecl> setinstrument                # specify the translation file
?                                # This means you respond with a question mark
direct

```

[We are automatically put into EPAR mode for the package parameters for CCDRED - set the "**verbose**" parameter to "yes" – and exit by typing **:q**]

[We are automatically put into EPAR mode for the task parameters for CCDPROC - the task that does all of the work. Look at these parameters and see the similarity with the processing steps in **PATH 1.**]

[We do not want to do anything more here for now, so type **:q.**]

```

ecl> ccdlist m92*.fits           # do you see a difference?
ecl>type subsets                # subsets was created by ccdlist
ecl>dir ccddb$kpno             # kpno translation files
ecl>type ccddb$kpno/direct.dat
ecl>lpar ccdred

```

The above steps used a parameter file for the instrument used to collect the data, to distinguish between the different filters used and to use a designated section for the overscan subtraction. After finishing this step, it would be a good idea to do a **phelp subset** to better understand the process you just went through.

Now the CCDRED package knows about the headers. Notice that the package takes care of our pixel type for us as well. Remember that our pixel type is "short" but the "pixeltype" parameter will let us control both the calculation type and output type during processing. During the actual processing the input images are overwritten; the "backup" parameter would let us make copies of the original data first if we wanted.

Biases and flat frames can be combined using the tasks ZEROCOMBINE and FLATCOMBINE. But we will skip these steps since we have data that have already been combined. We are now ready to set up the parameters for CCDPROC. Notice the two parameters called "biassec" and "trimsec." These are currently set to "image." If these keywords have the correct value in the image header then we would need to do nothing. But closer inspection will show that the values that we computed earlier are different from the ones listed in the image headers. Run EPAR and modify the parameters.

```

ecl> imhead m920005 lo+
ecl> epar ccdproc

```

This is what we used:

```
images = "m92*.fits"           List of CCD images to correct
(ccdtype = "object")          CCD image type to correct
(max_cache = 0)                Maximum image caching memory (in Mbytes)
(noproc = no)                  List processing steps only?\n
(fixpix = no)                   Fix bad CCD lines and columns?
(overscan = yes)               Apply overscan strip correction?
(trim = yes)                    Trim the image?
(zero = yes)                    Apply zero level correction?
(dark = no)                     Apply dark count correction?
(flat = yes)                    Apply flat field correction?
(illum = no)                   Apply illumination correction?
(fringe = no)                  Apply fringe correction?
(readcor = no)                 Convert zero level image to readout correction?
(scancor = no)                 Convert flat field image to scan correction?\n
(readaxis = "line")           Read out axis (column|line)
(fixfile = "")                 File describing the bad lines and columns
(biassec = "[335:350,2:510]")  Overscan strip image section
(trimsec = "[1:318,2:510]")    Trim data section
(zero = "")                    Zero level calibration image
(dark = "")                    Dark count calibration image
(flat = "")                     Flat field images
(illum = "")                   Illumination correction images
(fringe = "")                  Fringe correction images
(minreplace = 1.)              Minimum flat field value
(scantype = "shortscan")       Scan type (shortscan|longscan)
(nscan = 1)                    Number of short scan lines\n
(interactive = yes)            Fit overscan interactively?
(function = "chebyshev")       Fitting function
(order = 1)                    Number of polynomial terms or spline pieces
(sample = "*")                 Sample points to fit
(naverage = 1)                 Number of sample points to combine
(niterate = 1)                 Number of rejection iterations
(low_reject = 3.)              Low sigma rejection factor
(high_reject = 3.)             High sigma rejection factor
(grow = 0.)                    Rejection growing radius
(mode = "ql")
```

Since the "zero" and "flat" images are in the input list it is not necessary to specify them. Try running the task and see what happens. Note that we will be clobbering the **m92\*.fits** images! If you mess up here, you will need to redo CCDPROC from the beginning, including getting new, unprocessed, raw images back in your directory.

```
ecl> ccdproc
```

```
ecl> page logfile
```

```
ecl> imhead m92*.fits
```

```
ecl> imhead lo+ | page # notice the processing flags in the headers
```

The CCDPROC task is a very useful way to reduce a large batch of data in an efficient way, but caution must be used.

**Do not delete these images since they will be used in later exercises!**

***This page is to certify that***

\_\_\_\_\_

***has successfully complete IRAF III and lived to tell the tale.***

***Dated this \_\_\_\_\_ day of \_\_\_\_\_, 2011***