Tutorial: Basic Image Reduction with IRAF

Kaustubh Vaghmare

In the tutorial introducing IRAF, we have seen how IRAF works in tandem with ds9. We have seen how to work with the basic interface of IRAF. We have also seen how to work with interactive tasks like imexamine. One important aspect of IRAF that we have not seen is that of handling parameters. It is not very useful to be able to learn how to edit and list parameters without proper examples. And thus the concept was deliberately pushed to this tutorial.

It is beyond the scope of this document to explain how images are taken using CCDs, a piece of knowledge important to understand all the steps we will perform in the following sections using IRAF. But nonetheless, we will summarize the working of CCDs first. For details, refer to books such *Handbook of CCD Astronomy* by *Steve T Howell.*

1 CCD Image Reduction in a Nutshell

How a computer image comes from a CCD?

A CCD (Charge Coupled Devices) is a rectangular arrangement of sensors called pixels (Picture Elements). Each pixel is made of a material capable of converting the light incident on it, into charge. In other words, each pixel converts photons, packets of light, into electrons, packets of charge. The amount of charge produced is generally linearly proportional to the amount of light incident. The process of allowing light to fall onto the CCD from the telescope is referred to as an *exposure* or *integration.*

Once a CCD has been exposed to a source of light focussed on it by a telescope, each pixel accumulates a charge proportional to the light in that region of the sky. Then the CCD is read out using a read-out circuit, pixel by pixel. Each pixel's charge (an analog signal) has to be converted into a corresponding number (a digital signal). This is done using an *Analog-Digital Converter* (ADC). A digital number (DN) is represented as a series of 0s and 1s which may correspond to two possible states in an electronic device. How large a number can be represented depends on the number of *registers* which can store 0s and 1s. An 8-bit ADC has 8 registers, each capable of storing either 0 or 1 and thus the largest number that can be stored will be $2^8 - 1$. The -1 accounts for the fact the first number is a 0.

Finally, the read out process gives us a 2-d array of numbers, each a digital number generated using the charge on the corresponding pixel. A image viewer like ds9 displays the image as follows. A suitable scheme is adopted for assigning, say, a shade of grey to a certain number or range of numbers. On the screen a 2-d array of colored patches is displayed, each patch corresponding to a pixel on the CCD, colored according to the number. For example, a 0 might be completely black and the highest number a perfect white. All numbers in between will be some shades of grey, darker ones indicating lower numbers, brighter ones indicating higher numbers. Of course, this is just an example.

And this leads us to our first mini tutorial.

1. Open a terminal. (We are assuming that demo data is downloaded on your machine.) If next steps do not work, it means data does not exist. Please execute following steps.

   ```
   $ wget www.iucaa.in/~kaustubh/IRAF_Demo_Data.zip
   $ unzip IRAF_Demo_Data.zip
   ```

2. Open an image using ds9. (We are not involving IRAF right now, ds9 is an independent program and can be used to display images without IRAF.)

```
$ cd IRAF_Demo_Data
$ ds9 m67v1.fits
```

3. Click on 'scale', this will present options such as 'linear', 'log' etc. Click on each of them to figure out what exactly happens to your image display when clicked.
4. Next, we are going to change the allocation of grey shades to the numbers using our mouse. Move the mouse pointer anywhere on the image. Right click and hold! Don't forget to hold the right click. And now move your mouse horizontally, you will the allocation of shades changing giving the image different appearance. You can also invert colors by holding right click and moving vertically. Try it . . . see if you can invert the greyscale.

   Which color scheme is best? Inverted or normal? Log or linear? There is no single answer to the question. Anything that displays the features of your object clearly is what is good.
5. Next, try clicking 'color' and try out the color schemes.
6. Close ds9 and the terminal. We are done with the exercise.

What is basic image reduction?

Point the telescope to the sky, switch on the CCD, take an exposure and you have a nice image of your object or star. Now, if you are someone who is simply happy with the image itself, go ahead, post it on Facebook and use your bragging rights. Oh, you want to do measurements, not just show off your image? Ok, in that case there are some tools you could use to do these measurements. So, if you know your tools you can now do measurements, right? *Wrong!*

But why? The image you obtain from a CCD has in it several artificial signatures added to the image by the instrument itself. So, the distribution of light that you see in the image is not a reflection of the reality but it is reality distorted by some defects or features of your instrument. If you measure directly, you won't measure, say, the true brightness of a star. Thus, it is necessary to remove or quantify various features of the instrument before you can use images for scientific measurements. This process is called *basic image reduction.* The images before this process are generally referred to as *raw images* while the final processed images are *science images.*

So, here is a list of effects you need to correct for.

- Bias. Remember the process of converting charge into a digital number using an ADC? Remember that the ADC has n registers to store 0s and 1s to represent a number? Remember that $2^n - 1$ numbers can be stored? Well, it is possible to store $2^n - 1$ numbers only if they are all positive numbers, generated using the same polarity of charge. But if a charge of opposite polarity is converted, then the ADC will generate a negative number. Then one of the registers is used for storing the sign, say 0 for positive and 1 for negative. So, if one register is lost in storing sign, what happens? The number of numbers that ADC can handle becomes $2^{n-1} - 1$. That's half of the original capacity, almost.

  Now, consider a pixel which does not receive any light. In the ideal world, this pixel will have zero charge. But because of noise in the circuit, there will always be some charge on that pixel. It can both be positive or negative. And this can cause that loss of one bit of the ADC in storing a sign. How does one prevent that? Add an artifical charge level to all the pixels so that even without exposure to light, there is always some charge present. This ensures that no opposite polarity charge develops and we can get the full range of numbers an ADC has.

  But when taking measurements, this has to be subtracted since it is an artifical level and not a real level of light. So, either a whole bunch of pixels (called overscan strip) are read out, which are never exposed to light and this is used for measuring bias levels. Or one can just take a 0 second exposure of a CCD (no light captured), called a bias frame, in order to subtract from the images.
- Dark Current. Close the lid! Start the CCD exposure process. No light falls on it. Continue for 100 seconds, say. Still no light falls obviously, since the lid is closed! Now, stop and read out the CCD. So, you have nothing but the bias, right? No! Due to some thermal effects, there

is some current flowing in the circuit which results in accumulation of charges. So, there is an artifical level of charge still present. This needs to be obviously subtracted.

Let t_e be exposure time and t_d be the time used for making a dark frame i.e. an image obtained by not allowing light to fall on the CCD but nonetheless integrating it for t_d seconds. The charge accumulated in the dark frame has happened in t_d seconds. But we need charge levels for t_e seconds. So, we multiply the image by t_e/t_d and subtract from images.

Many modern observatories cool down their CCDs to reduce thermal current and thus contribution from dark frames is miminal and a dark current subtraction may not even be needed.

- Flat Field. If 100 photons fall on a pixel and it generates 80 electrons, it means that the pixel is operating at an efficiency of 80%. Unfortunately, each pixel operates with its own slightly different efficiency as a result of which the final image is not a representation of the true light distribution but one distorted by unequal efficiencies.

  How to correct this? Take an image of a uniform source, a dome light or twilight sky. Assuming the source is uniform we must get for a 3 pixel CCD, for example, a count of [100, 100, 100]. But because pixels have different efficiencies, we get [80, 90, 100], say. Now what? Let us divide our image of the uniform source by highest number, we get [0.8, 0.9, 1]. Now, we divide our image by this. [80, 90, 100] / [0.8, 0.9, 1.0] and we get [100, 100, 100]. This is what we need!

- Cosmic Ray Corrections. Cosmic rays are charged particles which can penetrate through telescope domes etc. and produce a large charge in one of your pixels. These can be corrected using sophisticated cosmic ray detection algorithms.
- Fringe Correction. Fringes can appear in some images because of interference of light due to the thin film on the optics. These can be corrected by analyzing flat field images.
- Bad pixels. Some pixels die, some misbehave giving rise to dead and hot pixels. These can be corrected by marking them and asking a suitable interpolation algorithm to use the knowledge of neighbouring pixels to estimate the likely value of the pixel.
- Read Noise. Added by the read out circuit while converting charges into digital numbers. Cannot be corrected, only quantified.
- Other sources of systematics depending on the telescope and the instrument. Example: cross talk noise between detectors etc.

So, the basic final recipe for basic image reduction is given below. We are excluding cosmic ray corrections, bad pixel corrections etc. and highlighting the most important steps.

- Use multiple bias frames to get average bias frame. (Why should you take an average? Why not just take one exposure? Read about *Central Limit Theorem.*)
- Subtract mean bias from all other images including flats.
- Construct mean dark frame as needed and subtract from all other images except bias frames.
- Average flat field images.
- Normalize mean flat field image and divide all object images by this.

2 Using IRAF for Basic Image Reduction

Enough theory, let's do a practical example. The demo data should be on your machine now. If you are one of those who skipped previous sections, please ensure the data is present, else do the following in a terminal:

```
$ wget www.iucaa.ernet.in/~kaustubh/IRAF_Demo_Data.zip
$ unzip IRAF_Demo_Data.zip
```

If you have the data, proceed as follows:

1. Start IRAF. (Refer to previous tutorials if you don't know or don't remember how to do this.)

2. One important feature of IRAF. It supports basic shell commands like 'ls', 'cd' and 'pwd' within it. So, let's go to our example directory and list what files we have.

```
vocl> cd IRAF_Dema_Data/
vocl> ls
bias1.fits  bias5.fits    flat_i1.fits  flat_i5.fits  flat_r4.fits  m67b2.fits  m67b6.fits  m
bias2.fits  flat_b1.fits  flat_i2.fits  flat_r1.fits  flat_r5.fits  m67b3.fits  m67i1.fits  m
bias3.fits  flat_b2.fits  flat_i3.fits  flat_r2.fits  logfile       m67b4.fits  m67i2.fits  m
bias4.fits  flat_b3.fits  flat_i4.fits  flat_r3.fits  m67b1.fits    m67b5.fits  m67i3.fits  m
vocl>
```

The files bias1, bias2 etc are the bias frames. There are image files of M67 open star cluster in 3 bands - b, i and r. For each band, there is a set of flat field images. (Why do we need separate flat fields for three bands? Why don't we have separate bias frames for different filters? Discuss or think!)

3. Let us see what a bias frame looks like.

```
vocl> display bias1.fits
```

You can see the bias image having a nearly constant level of 100 with some noise level. You can examine this clearly by moving the cursor over the frame in ds9 and seeing the recorded value in the top panel.

4. We need to construct an average bias frame. Thus we need to add the five bias frames and divide by 5. The task that allows us to deal with image arithmetic is 'imarith'. (Adding 2 images means making one image where each pixel has a value which is sum of the two corresponding pixels in the two images being added)

Obviously imarith will need some input to be able to do anything. These inputs are called parameters. Let us list these parameters.

```
vocl> lpar imarith
  operand1 = ""             Operand image or numerical constant
        op = "+"            Operator
  operand2 = ""             Operand image or numerical constant
    result = ""             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 = no)            Print operations?
    (noact = no)            Print operations without performing them?
     (mode = "ql")
```

As you can see, it needs at least 4 inputs - operand1, op, operand2 and result. Refer to help to understand other parameters - later!

5. Next, we need to edit the parameters.

```
vocl> epar imarith
```

Now, you can edit these parameters, the black cursor is on operand1. Type 'bias1.fits'. NOTE: The backspace key behaves funny in xgterm, so if you make a mistake in typing, simply press Enter, use arrow keys to go the top line again and try. Enter 'bias2.fits' in operand2 and enter 'temp1.fits' in result. The 'op' is already +.

After entering all this information, press the following key combination ':wq' to save (w) and quit (q). You will get back to the prompt.

6. To execute imarith with these values, you can type imarith and press Enter. imarith will confirm the entered values, keep pressing Enter and finally 'ls' to check that temp1.fits is created. Go ahead, display temp1.fits and check the values of the pixels. They should be around 200.

7. Next, we need to add temp1.fits and bias3.fits to produce temp3.fits, the sum of three bias frames. But it is painful to keep saying epar all the time, so we will learn another powerful method of executing IRAF tasks, by directly specifying input while calling the task itself. We show below how to keep adding these files and finally divide by 5 to construct average bias, also called master_bias.

```
vocl> imarith temp1.fits + bias3.fits temp2.fits
vocl> imarith temp2.fits + bias4.fits temp3.fits
vocl> imarith temp3.fits + bias5.fits temp4.fits
vocl> imarith temp4.fits / 5 master_bias.fits
vocl> ls
bias1.fits  flat_b1.fits  flat_i3.fits  flat_r3.fits  m67b2.fits  m67i1.fits  m67i6.fits  m67
bias2.fits  flat_b2.fits  flat_i4.fits  flat_r4.fits  m67b3.fits  m67i2.fits  m67r1.fits  m67
bias3.fits  flat_b3.fits  flat_i5.fits  flat_r5.fits  m67b4.fits  m67i3.fits  m67r2.fits  mas
bias4.fits  flat_i1.fits  flat_r1.fits  logfile       m67b5.fits  m67i4.fits  m67r3.fits  tem
bias5.fits  flat_i2.fits  flat_r2.fits  m67b1.fits    m67b6.fits  m67i5.fits  m67r4.fits  tem
vocl>
```

Go ahead, display the file and examine it. If you are observant, you will notice one or two bright spots. Where did they come from? Open bias1.fits, you will find that same spot there. But not in bias2.fits or other bias files. Why?

These are hot pixels or likely hit by some cosmic ray particles. One problem with average is the following. Let's say I take 5 measurements, I have a huge error in one. Say my measurements are [1, 2, 3, 4, 100]. The fifth measurement has gone bad. So, I take average of these five numbers, I get 22. So, my average value is affected by the presence of one large wrong measurement. This is what is happening when we take mean of five bias frames - one bad frame causes problems in the final frame.

So, what do we do? Take the median! The median of [1, 2, 3, 4, 100] is 3, a better estimate of the central value in this case. But how to take median of images? This cannot be done using imarith. For this purpose we have another task called 'zerocombine'. Let's see how to use that. But before, let's delete this master_bias.fits we created. The command in Linux for this is 'rm'. For safety or some other reason I don't know, this command is not supported by IRAF. To ask IRAF to pass a command to the actual shell I must start the command with an '!'. (Try without !, IRAF will complain because IRAF does not understand 'rm'.)

```
vocl> !rm master_bias.fits
```

8. Let's open the packages where zerocombine exists.

```
vocl> noao
      artdata.      astrometry.   digiphot.     imred.        nobsolete.    observatory   one
      astcat.       astutil.      focas.        mtlocal.      nproto.       obsutil.      rv.

noao> imred
      argus.      ccdred.     ctioslit.   echelle.    hydra.      irred.      kpnocoude.  qua
      bias.       crutil.     dtoi.       generic.    iids.       irs.        kpnoslit.   spe

imred> ccdred
      badpiximage     ccdinstrument   ccdproc         darkcombine     mkillumcor      mkskyfl
      ccdgroups       ccdlist         ccdtest         flatcombine     mkillumflat     setinst
      ccdhedit        ccdmask         combine         mkfringecor     mkskycor        zerocom
```

Now, say, `epar zerocombine` and enter the following information.

```
        input = "bias*.fits"    List of zero level images to combine
```

```
     (output = "master_bias.fits") Output zero level name
    (combine = "median")      Type of combine operation
     (reject = "none")        Type of rejection
    (ccdtype = " ")           CCD image type to combine
    (process = no)            Process images before combining?
     (delete = no)            Delete input images after combining?
    (clobber = no)            Clobber existing output image?
      (scale = "none")        Image scaling
    (statsec = "")            Image section for computing statistics
       (nlow = 0)             minmax: Number of low pixels to reject
      (nhigh = 1)             minmax: Number of high pixels to reject
      (nkeep = 1)             Minimum to keep (pos) or maximum to reject (neg)
      (mclip = yes)           Use median in sigma clipping algorithms?
     (lsigma = 3.)            Lower sigma clipping factor
     (hsigma = 3.)            Upper sigma clipping factor
    (rdnoise = "0.")          ccdclip: CCD readout noise (electrons)
       (gain = "1.")          ccdclip: CCD gain (electrons/DN)
     (snoise = "0.")          ccdclip: Sensitivity noise (fraction)
      (pclip = -0.5)          pclip: Percentile clipping parameter
      (blank = 0.)            Value if there are no pixels
       (mode = "ql")
```

Make sure you type a blank space for 'ccdtype'. Else program will not work.

Go ahead, execute zerocombine, you should see something as below. IF YOU DO NOT SEE THE FOLLOWING OUTPUT AND GET AN ERROR, see Appendix.

```
ccdred> zerocombine
List of zero level images to combine (bias*.fits):

Dec  4 14:23: IMCOMBINE
  combine = median, scale = none, zero = none, weight = none
  blank = 0.
                Images
             bias1.fits
             bias2.fits
             bias3.fits
             bias4.fits
             bias5.fits

  Output image = master_bias.fits, ncombine = 5
```

Open master_bias.fits and check. The bright spots would have gone.

9. So, you now have a bias file, subtract it from all other images. Let's do it. But how? Will you keep typing as many 'imarith' commands as there are images to be subtracted? You don't have to. Here is what we will do - we will use a nice trick to create a list of all files other than the bias and use the '@' character to supply this list to imarith.

```
 vocl> ls -1 flat*.fits m67*.fits > other_than_bias.list
 vocL> imarith @other_than_bias.list - master_bias.fits @other_than_bias.list
```

Take a moment to understand what's happening. @ tells IRAF to take every file listed in the file following @. And since this is both given as input as well as output, the files will be changed in place i.e. no new files will be created but existing files will be modified.

10. Next, we will combine the flat frames. This should be similar to zerocombine. Remember this should be done for each filter separately. DO NOT COMBINE flats from all filters. We show below the process for images taken using b- filter.

```
ls -1 flat_b*.fits > flat_b.list
```

Then combine using flatcombine, parameters are listed below.

```
    input = "@flat_b.list"    List of flat field images to combine
      (output = "master_flat_b")  Output flat field root name
     (combine = "average")      Type of combine operation
      (reject = "sigclip")      Type of rejection
     (ccdtype = " ")            CCD image type to combine
     (process = no)             Process images before combining?
     (subsets = no)             Combine images by subset parameter?
      (delete = no)             Delete input images after combining?
     (clobber = no)             Clobber existing output image?
       (scale = "mode")         Image scaling
     (statsec = "")             Image section for computing statistics
        (nlow = 1)              minmax: Number of low pixels to reject
       (nhigh = 1)              minmax: Number of high pixels to reject
       (nkeep = 1)              Minimum to keep (pos) or maximum to reject (neg)
       (mclip = yes)            Use median in sigma clipping algorithms?
      (lsigma = 3.)             Lower sigma clipping factor
      (hsigma = 3.)             Upper sigma clipping factor
     (rdnoise = "rdnoise")      ccdclip: CCD readout noise (electrons)
        (gain = "gain")         ccdclip: CCD gain (electrons/DN)
      (snoise = "0.")           ccdclip: Sensitivity noise (fraction)
       (pclip = -0.5)           pclip: Percentile clipping parameter
       (blank = 1.)             Value if there are no pixels
        (mode = "ql")
```

Please enter each value very carefully. Not doing so will mostly give errors. Next, we use imstat to find out the mean pixel value in combined flat. You can also normalize to maximum value.

```
ccdred> imstat master_flat_b.fits
#               IMAGE      NPIX      MEAN    STDDEV       MIN       MAX
   master_flat_b.fits   1742000    35706.     1101.     2.895    37839.
ccdred>
```

Next, we normalize by saying.

```
ccdred> imarith master_flat_b.fits / 35706 master_flat_b.fits
```

You may want to check your final flat field using display. Most values will be centered around 1. Next, we apply this flat on b-band images.

```
ccdred> ls -1 m67b*.fits > m67_b.list
ccdred> imarith @m67_b.list / master_flat_b.fits @m67_b.list
```

That's it! B-band images have been flat-fielded (and also bias corrected from previous step). You should try repeating this process for i and r bands too.

11. The final step is to combine all flat-fielded, bias subtracted images of M67 for each band into one final image. This is done using 'imcombine'. Do epar imcombine and enter following parameters. (Most parameters are default except first two.)

```
ccdred> lpar imcombine
        input = "@m67_b.list"   List of images to combine
       output = "m67_b_final.fits" List of output images
     (headers = "")             List of header files (optional)
     (bpmasks = "")             List of bad pixel masks (optional)
    (rejmasks = "")             List of rejection masks (optional)
   (nrejmasks = "")             List of number rejected masks (optional)
    (expmasks = "")             List of exposure masks (optional)
      (sigmas = "")             List of sigma images (optional)
       (imcmb = "$I")           Keyword for IMCMB keywords
     (logfile = "STDOUT")       Log file\n
     (combine = "average")      Type of combine operation
      (reject = "none")         Type of rejection
     (project = no)             Project highest dimension of input images?
     (outtype = "real")         Output image pixel datatype
   (outlimits = "")             Output limits (x1 x2 y1 y2 ...)
     (offsets = "none")         Input image offsets
    (masktype = "none")         Mask type
   (maskvalue = "0")            Mask value
       (blank = 0.)             Value if there are no pixels\n
       (scale = "none")         Image scaling
        (zero = "none")         Image zero point offset
      (weight = "none")         Image weights
     (statsec = "")             Image section for computing statistics
     (expname = "")             Image header exposure time keyword\n
  (lthreshold = INDEF)          Lower threshold
  (hthreshold = INDEF)          Upper threshold
        (nlow = 1)              minmax: Number of low pixels to reject
       (nhigh = 1)              minmax: Number of high pixels to reject
       (nkeep = 1)              Minimum to keep (pos) or maximum to reject (neg)
       (mclip = yes)            Use median in sigma clipping algorithms?
      (lsigma = 3.)             Lower sigma clipping factor
        (hsigma = 3.)             Upper sigma clipping factor
     (rdnoise = "0.")           ccdclip: CCD readout noise (electrons)
        (gain = "1.")           ccdclip: CCD gain (electrons/DN)
      (snoise = "0.")           ccdclip: Sensitivity noise (fraction)
    (sigscale = 0.1)            Tolerance for sigma clipping scaling corrections
       (pclip = -0.5)           pclip: Percentile clipping parameter
        (grow = 0.)             Radius (pixels) for neighbor rejection
        (mode = "ql")
```

The final execution of imcombine looks something like below.

```
ccdred> imcombine
List of images to combine (@m67_b.list):
List of output images (m67_b_final.fits):

Dec  5  0:08: IMCOMBINE
  combine = average, scale = none, zero = none, weight = none
  blank = 0.
                Images
             m67b1.fits
             m67b2.fits
             m67b3.fits
             m67b4.fits
             m67b5.fits
             m67b6.fits

  Output image = m67_b_final.fits, ncombine = 6
```

You can repeat this procedure for i and r-bands.

This concludes our tutorial on basic image reduction. All the above steps can be fully automated using a task called 'ccdproc'. This is a very sophisticated task. You can refer to the help pages to understand how this task works.

Appendix

If 'zerocombine' or 'flatcombine' give error about instrument not being found or something, do the following.

```
ccdred> setinst
Instrument ID (type ? for a list):
```

Type 'direct' and press Enter, you will see a new set of parameters. Don't change anything, simply ':wq' and quit. You will (surprisingly) be taken to another big set of parameters. Again, don't edit anything, say ':wq'.

Some versions of IRAF do not give such an error. I am not sure why! All I do know is that doing the above seems to help in resolving the error in many cases.