Original Data: Draft

Chris Bohlman

1/12/2017

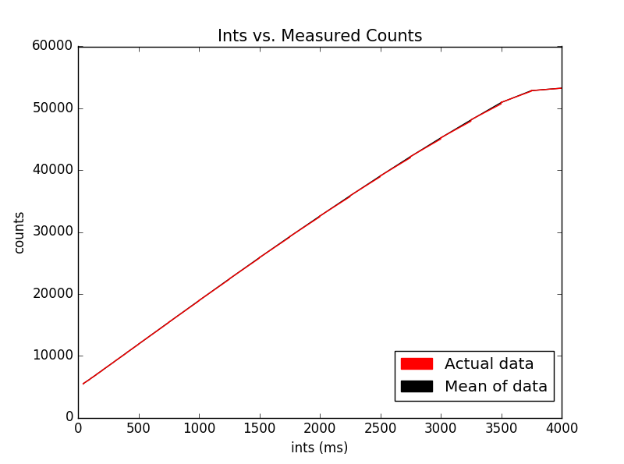
Context:

The CLIO infrared camera is an instrument used in the Magellan Adaptive Optics system, located at the Las Campanas Observatory located in the Atacama Desert in Chile. Now, infrared light is light that has a wavelength of 800 nm to 1 mm. Since the Earth’s atmosphere is mostly opaque to infrared light, infrared astronomy should be conducted at high elevations, past as much of the atmosphere as possible. Also, adaptive optics are used to gather data and correct for the opaqueness of the atmosphere at those long wavelengths.

The context to this report is to analyze a set of images to calibrate the infrared camera CLIO. To do this, a program must be written to judge how exactly the images should be corrected for linearity. Now, this data has previously been corrected for linearity by Katie Morzinski. However, I chose to also attempt to correct this data, as I want to use what I have done for this specific set of data as a template for correcting data that hasn’t been corrected yet.

This data was originally gathered on March 23, 2013, with the CLIO camera in the MagAO system at the Las Campanas observatory in Chile.

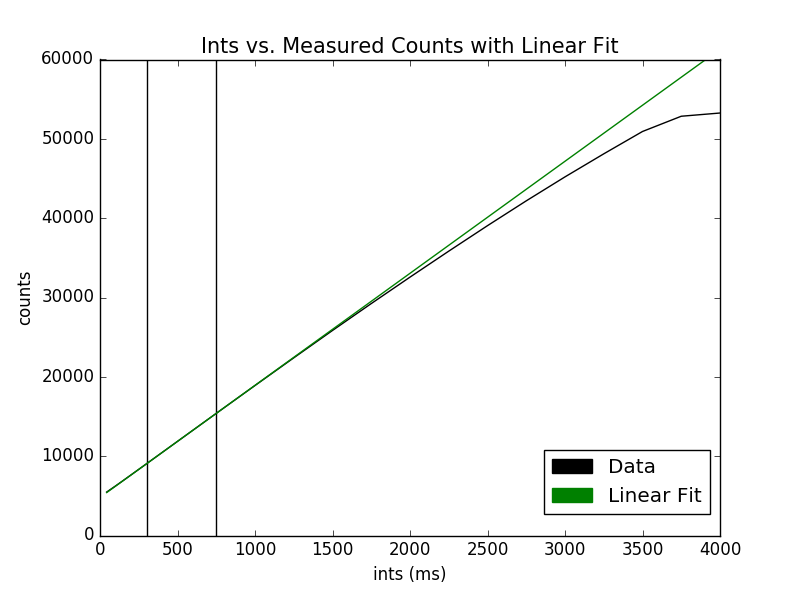
Ints vs. Counts:



To determine the integration time vs the counts of each picture, I read files in through the FITS package from astropy in Python, and adjusted the parameters of the read in’s area equal to original analysis’ parameters (200 to 350 in the x direction and 0 to 200 on the y axis) These parameters are due to the bright side of the images always being on the right half, so we wanted to really capture that part of the image. The program I wrote analyzed counts in that section of the image for each picture, and got the integration time’s measurement from the specific header for each image. In the context of this project, the integration time is best thought of the exposure time for the camera. Therefore, an increasing amount of ints means that the camera was open longer.

The data was stored in ‘int’ and ‘counts’ arrays, respectively. I graphed the integration time, which were in milliseconds, on the x axis and counts on the y axis.

Ints vs. Counts with linear relationship:

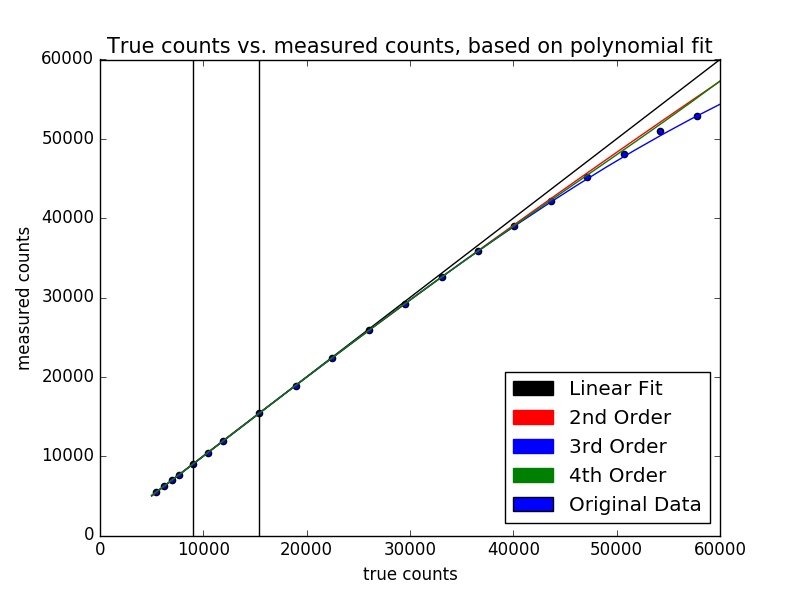


The first step in correcting for linearity was trying to find the straightest part of the data to add a linear fit to demonstrate the supposed linearity we were trying to achieve. I determined that the straightest part was between 300 ints and 750 ints, right at the beginning of the data set. The region of choice is in between the bars above. I chose this region among others because after graphing the linear fit in relation to other regions in the straight-line section of this graph, I zoomed into each of the fits, and found that this fit’s separation was minimal enough to provide a good fit for the rest and was a stellar fit.

The coefficients of the line were m = 14.12325 and b= 4802.21666667, if y = mx + b.

This line shows that the data is only strictly linear up to about 25,000 counts, so the linearity correction must be applied to any values above this.

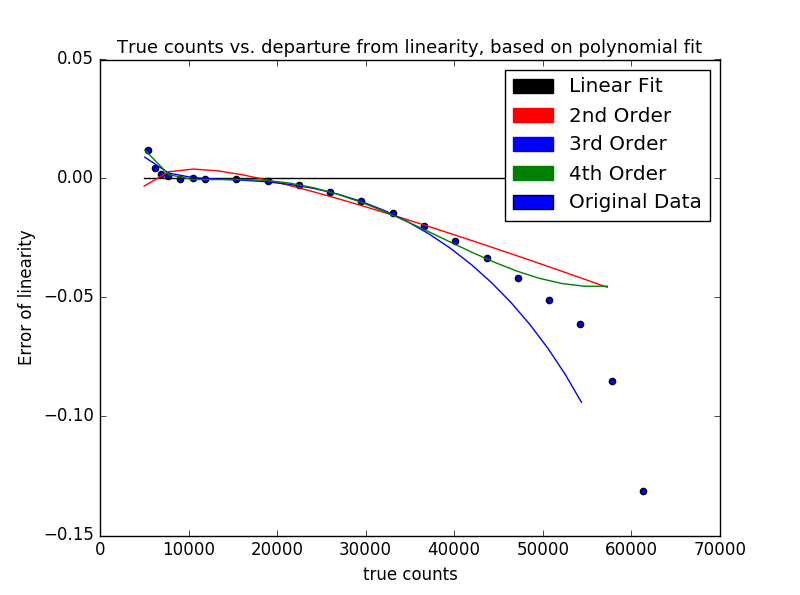
True counts vs. Measured Counts with all:



The next step was to calculate the true counts. The true counts represented what the data would look like if it was perfectly linear, and there was no calibration needed. On the above graph, this is the linear fit represents. However, we also wanted to fix the data for second, third, and fourth order polynomials, so we took the set of data up to about 40,000 counts, and created a function in each of those orders that would convert counts to true counts. By applying these function coefficients that were generated to counts for each order, 3 more true counts graphs were generated. By plotting the original data on the same graph, we could see the third order fit was the most accurate in this case.

The bars in the graph also represent where the linear fit was chosen from for all orders of the fits.

Error of Linearity vs. True Counts



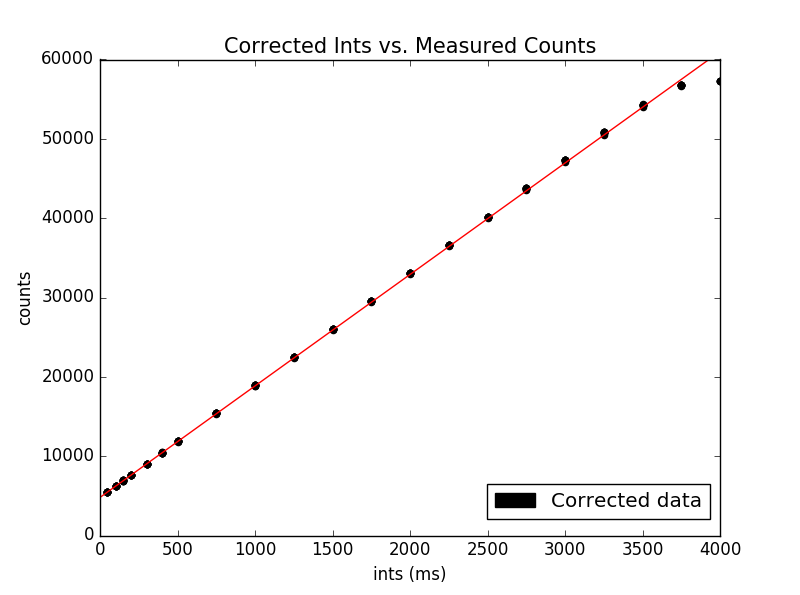
To confirm the results from the previous page that the third order fit was superior, we decided to calculate the error of linearity for each of the different polynomial fits. The error of linearity was calculated by:

Error of linearity =

With true counts referring to the specific true counts of each fit, calculated earlier.

As seen from the picture, 3rd and 4th order come very close to the fit, and diverge at about the same point from the original data. However, only the 3rd order fit still behaves in the same way as the original data. Therefore, from this graph, we can adequately support the claim that the 3rd order fit is the best fit for this data. Or maybe the fourth order. I don’t know right now ☹

Corrected Counts vs Ints:



This is the graph for 3rd order corrected data, which happens to be linear up to ~52,000 counts. Now, the process for this was as follows:

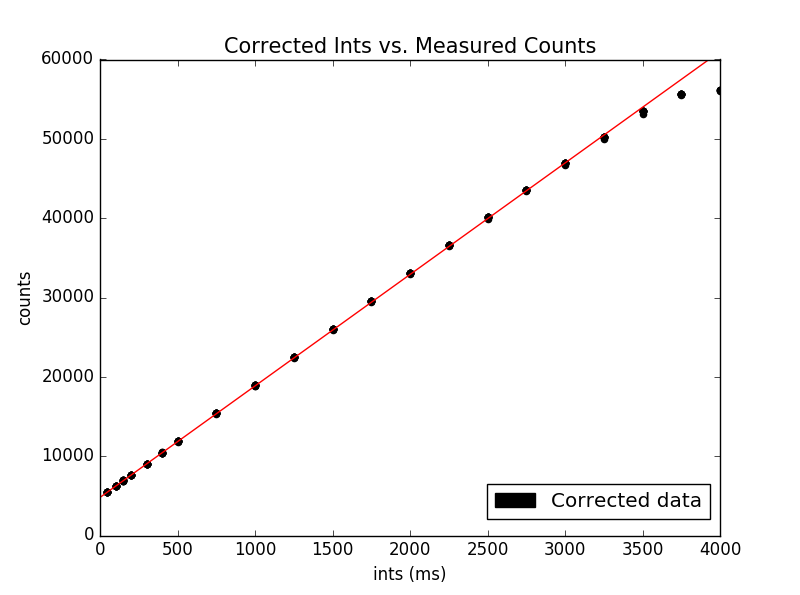
First, I made sure that the 3rd order coefficients were converting counts to true counts, so I personally had to recalculate my coefficients with counts in the x axis, and true counts in the y axis in the coefficients-calculating function in Python. I then reopened all the images back up, and calculated the ints as I did previously, reading them in through each image header. However, for the counts, if the counts read in were above 20,000, I took the counts from the image and applied my calculated third order coefficients to output a corrected and calibrated value for counts. I then graphed the resulting new counts, and found out that the corrected data was linear up to about 50,000 counts.

Equation that linearized the data:

Where x is the measurement of counts from the image, y is the corrected counts,

*a* = 6.28e-11, -2.55e-06, *c* = 1.036, and *d* = -1.736e+02

Bonus page: What about a fourth order correction?



Deviates faster than 3rd order, because this starts delinearizing at 45,000 counts.

Get this trash out of here.