Clio Linearity Calibration: Draft

Chris Bohlman

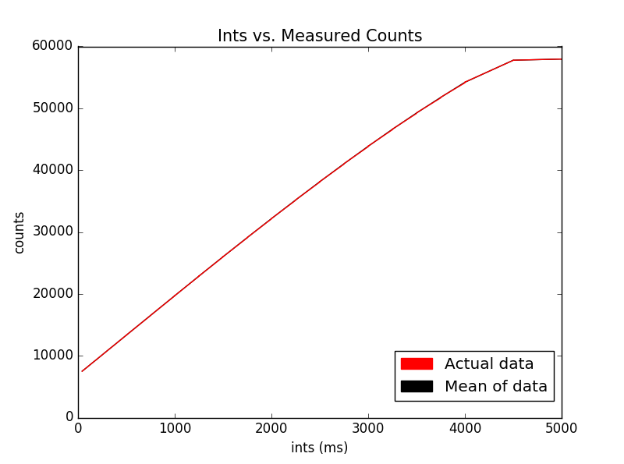
12/2/2016

Context:

For this project, I was given a stack of pictures and a little Python programming experience. And I was told to create a masterpiece.

Actually, not exactly. What I needed to do for this project was take the pictures from a certain cycle of pictures taken by the CLIO detector to provide some sort of linearity correction to future pictures taken. This was important, as it involved photometry and it served as an introduction towards my involvement with adaptive objects and how these detectors are used in real world situations.

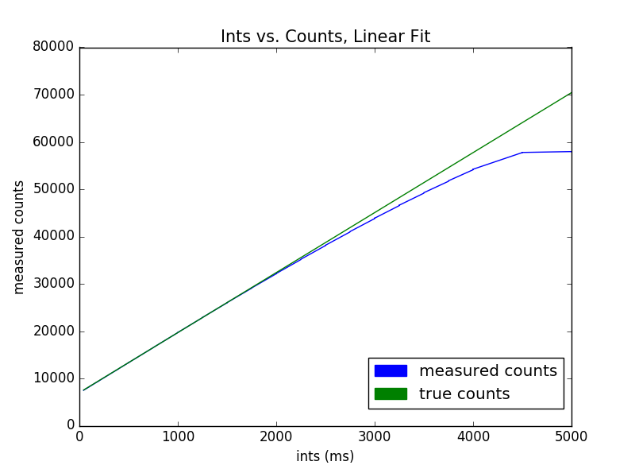
Ints vs. Counts:



First, I started by coding in Python and reading in each of the files through FITS. For this experiment, I chose to analyze the images in the entirety of their right side. I analyzed the number of counts in said section of the image, and found the integration time of the image in the FITS header. This was done through all of the given files, and the data was stored in 2 separated arrays, appropriately called ‘ints’ and ‘counts’.

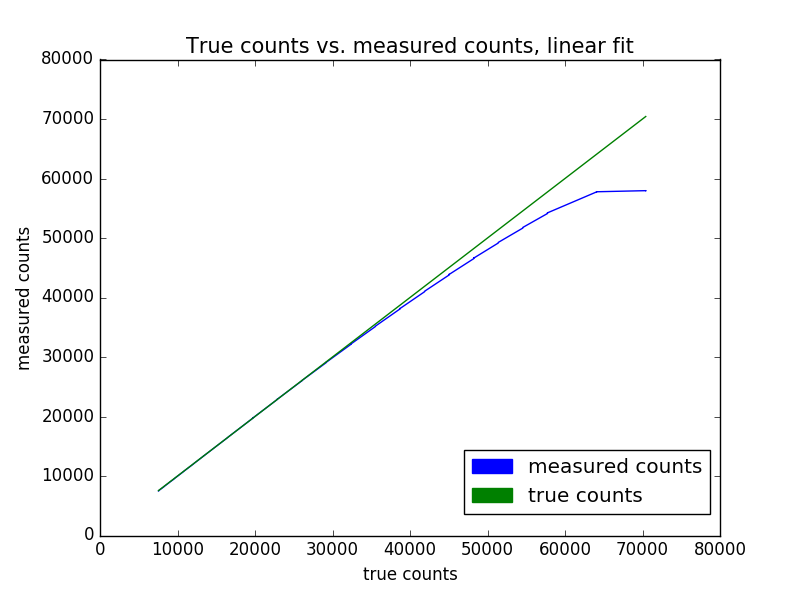
Now, for every picture, every 5 pictures had the same integration time, so I grouped all of them together and came up with a mean number calculation for each of the 5 pictures’ respective counts. I then graphed that alongside the raw ints versus counts, and the outcome was the graph above. As you can see, everything lined up, meaning that my calculations for the mean counts were correct.

Ints vs. Counts with linear relationship:



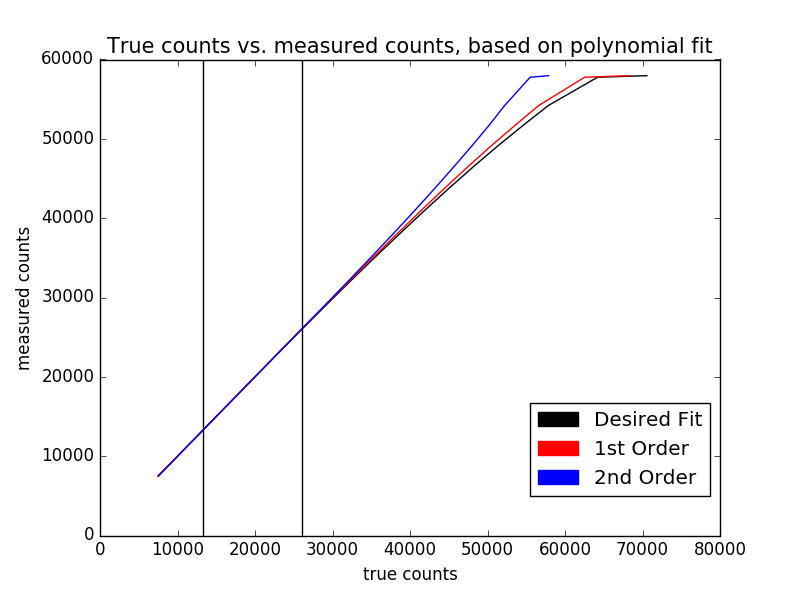
I needed to find a linear relationship with the data so I could find a method of determining the true counts, assuming that for true counts, I wanted it to be synonymous with integration time. I decided that the most linear part of the graph was between 500 ints and 1500 ints. Since there were 5 samples at each integration time, along with slightly different counts for each time in the group, I used polyfit from numpy to determine the coefficients of a line analogous with the area that I had picked out. I then plotted said line next to the data, showing where the data is linear and where it diverges from my calculated line.

True counts vs. Measured Counts with linear:



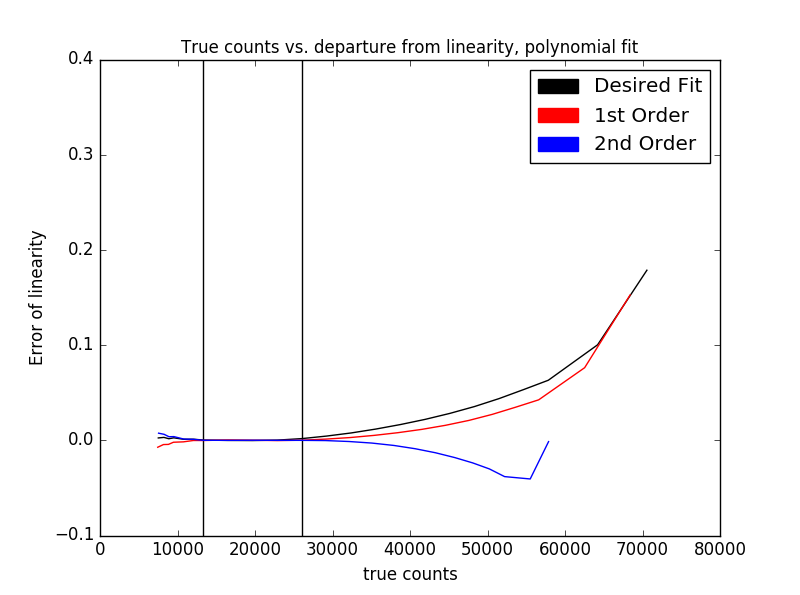
To determine the error of the counts as the measurement approached a higher integration time, a method of determining the true amount of counts (if the counts did not lessen) was needed. Fortunately, that was relatively simple to do with a linear approximation of the same part of the graph as before. I again used polyfit to formulate a linear approximation of the line between 500 and 1500 ints, but instead of directly graphing it onto the graph again, I evaluated the coefficients given by treating it as a polynomial. Because polyfit gives the ‘A’ and ‘B’ coefficients in ‘Ax + B’ when evaluating the section linearly, I made a true counts list by appending the polyfit polynomial evaluated with every int from 0 to the 114th int. The result was a list of true counts that was the same length as the ints list and the counts list. To graph it on the same graph as the original data, I first graphed true counts vs. true counts to give a line of comparison, and then I graphed true counts vs. the original counts to model the data.

True counts vs. Measured Counts with all:



In the end, what was wanted was to see which order equation could model most of the data the best. Therefore, once I had the linear version of the true counts, I calculated the coefficients of a second order equation per the data, and a third order equation to true counts and the measured counts, making it important to get the true counts. In fact, I used the mean counts array to generate the true counts array for the second and third order equations. I then used polyfit on these to model second and third order relationships, and redid the first order linear which came up with coefficients that were very close to the coefficients for the previous true counts line, so that did not cause a problem in my data, since that was going to be my desired relationship of data. I then graphed true counts vs counts for every single order the same way I did it in the method of above, and graphed them on the same graph as each other. The vertical lines are representations of where the coefficients for each of the lines were gathered from.

Error of Linearity vs. True Counts



In the end, we wanted to see which order (2nd or 3rd) was closest to the desired relationship of first order. Therefore, stemming from the last graph, I produced an error of linearity plot. Now the true counts axis stayed the same, but for the error of linearity, I used the equation

*Error of linearity* = (*True Counts – Measured Counts)/True Counts*

For every single count, I calculated the error of linearity and stuck that into a list. I then plotted the error of linearity compared to the true counts for first order, second order, and third order, and analyzed the results based off how close the second and third order errors were to the desired relationship of first order.

Analyzing the Data:

From the error of linearity plot, I discovered that the second order equation behaved more closely to the desired relationship. Now, the reason I couldn’t add fourth order is because there were only 23 items on each list (due to me averaging the ints and counts into separate lists), so the fourth order polyfit had some odd behavior. After consolidating my data and limits and methods across three python files, I finally came up with that error of linearity graph, which was a marked improvement over what I was producing previously. However, because I couldn’t get any order to exactly match up to the desired relationship, I was at a loss for things to do. That’s why I am writing this report and organizing all my files: to get some ideas or something, or to notice a method that I did wrong.

Here’s to hoping that happens.