Clio Linearity Calibration: November 2014 Data

Chris Bohlman

4/12/2017

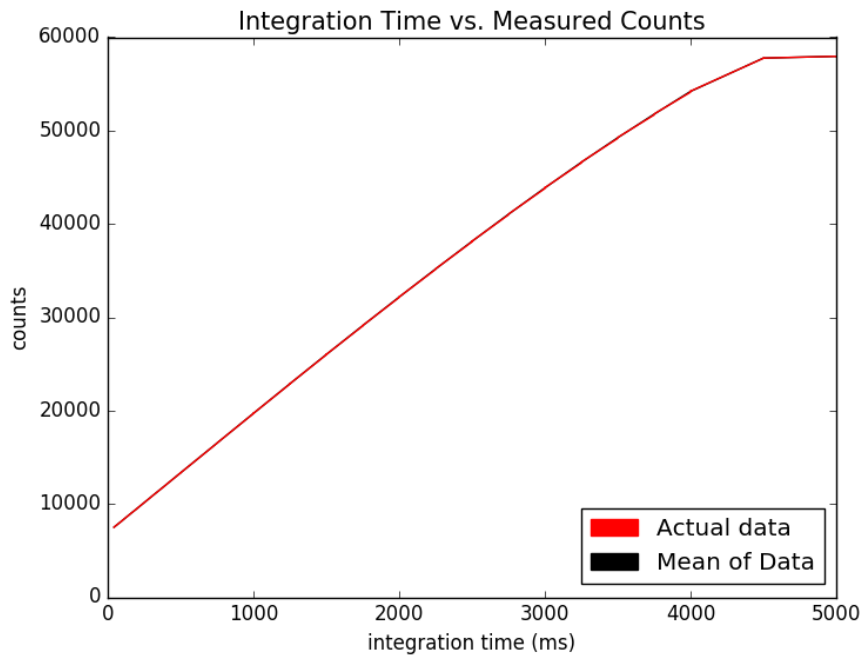
Context:

Las Campanas Observatory, at the Atacama Desert in Chile. Clio specializes in infrared photometry, and adaptive optics are used to ensure that the turbulence of the atmosphere are corrected for so clearer images can be taken. Specifically, Clio is sensitive from 1 μm to 5 μm.

Now, the goal of this report was to initially analyze a set of flat images to come up with a set of coefficients to calibrate the data from infrared camera Clio. The code to calibrate this flat data set was written in Python from August 2016 – January 2017 and adapted from code that I have previously wrote to analyze another data set from the Clio camera. One the flat data set was calibrated, we could take those same coefficients used to calibrate and apply them to another data. In the end, linearity is what we want to achieve with the data set. The supposed relationship in the data between the integration time and counts readings should be linear, however, due to saturation from increased brightness, non-linearity appears within the data trends. Thus, the nonlinear parts of the data are rendered useless. It is therefore the objective of this report to demonstrate that through curve fitting and other programming tools, we can correct this data and make sure that a larger portion of it turns out to be linear. This is known as linearity, and it is crucial to preserving the viability of the data.

This data was originally gathered in November of 2014, with the Clio camera in the MagAO system at the Las Campanas observatory in Chile.

Ints vs. Counts:



Initially, we found the ints and counts of each image. The ints, or integration time, is best thought of the exposure time for the detector. Therefore, an increasing amount of ints means that the camera was open longer. The counts can best be thought of as the brightness of a section of an image. To determine the integration time vs the counts of each image, I read files in through the FITS package from astropy in Python, and adjusted the parameters of the image equal to pixels 33 to 180 in the x direction and 20 to 180 on the y axis. These parameters are due to the bright side of the images always being on the right half, so we only wanted to process that part for data. The program I wrote analyzed counts in that section of the image for each image, and got the integration time's measurement from the specific header for each image. In the context of this project, the data was stored in 'int' and 'counts' arrays, respectively. I graphed the integration time of each image, which were in milliseconds, on the x axis and I graphed the counts on the y axis.

Where in my code this is calculated:

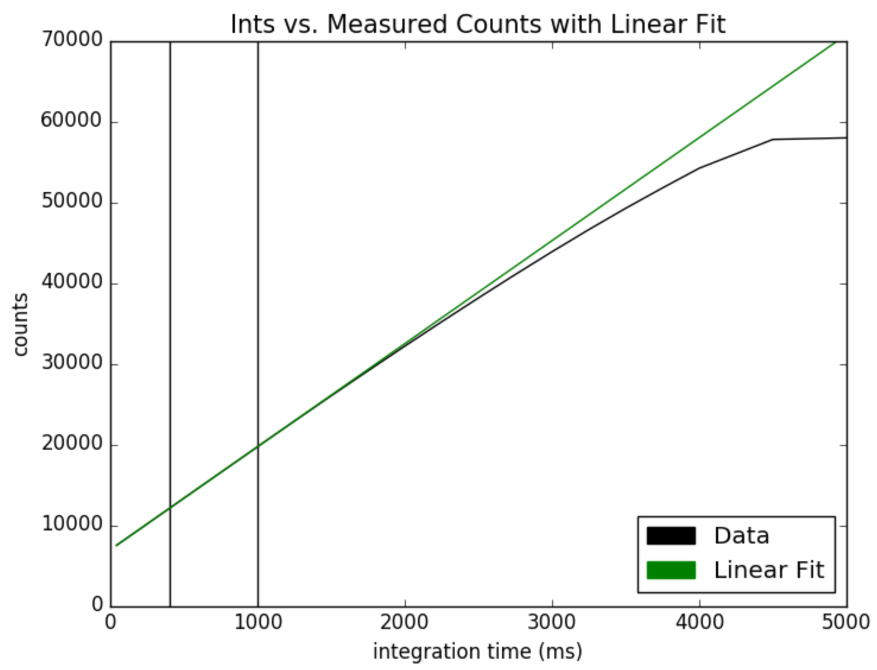
File: linearityReportNovember2014Data.py

Images are opened and data is read in: open_images(), lines 31-53

Mean of counts is taken: avg_ints_counts(), lines 55-63

Graph is printed: print_graph(), lines 168-180

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 400 ms and 1000 ms, making sure to not use the very low-int data (due dark current making noise). 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 = 12.76 and b= 6946.92 in terms of y = mx + b.

This line graphed with the ints and counts of each image show that the data is only strictly linear up to about 20,000 counts, so the linearity correction must be applied to any values above this.

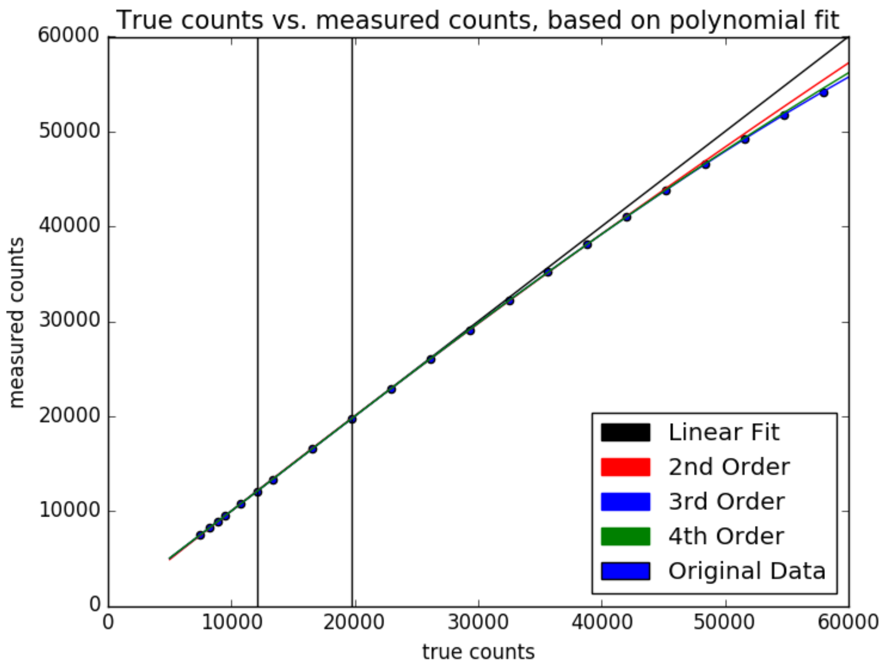
Where in my code this is calculated:

File: linearityReportNovember2014Data.py

Linear fit is found: make_true_counts(), lines 66-71

Graph is printed: print_graph(), lines 184-200

True counts vs. Measured Counts with linear:



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 what 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 44,000 counts, and created an equation 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. Therefore, measured counts were plotted for each order on the y axis, and true counts were graphed on the x axis. By plotting the original data on the same graph, we could see the third and fourth order fits were 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.

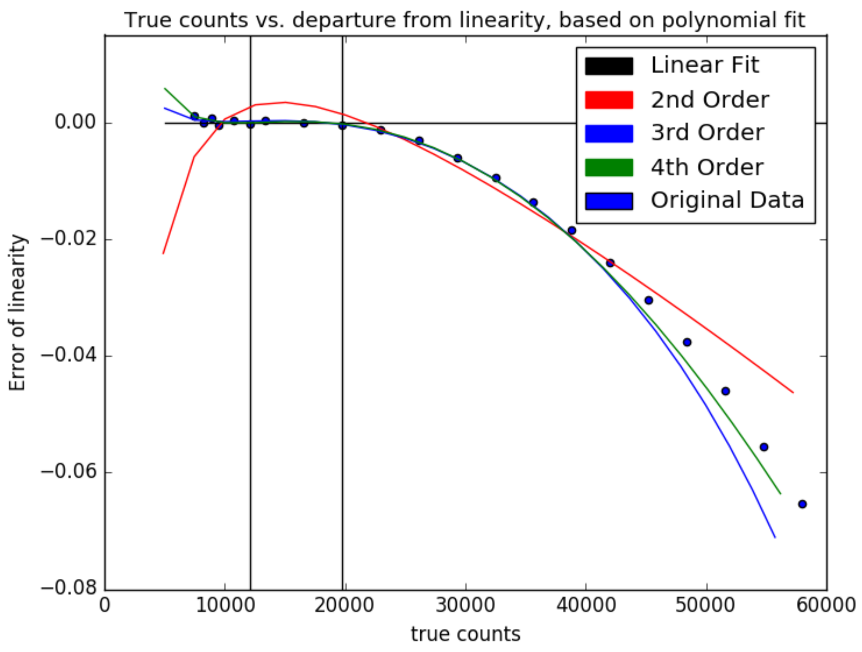
Where in my code this is calculated:

File: linearityReportNovember2014Data.py

True counts are found: make_true_counts(), lines 73-99

Graph is printed: print_graph(), lines 204-227

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:

$$\text{Error of linearity} = \frac{true\ counts - measured\ counts}{true\ counts}$$

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

As seen from the graph, 3rd and 4th order come very close to the fit, and diverge at about the same point from the original data. However, the fourth order fit is closer to the data, so I chose that order to correct the counts. After adjusting formulae and boundaries within my program, I have concluded this is the closest to the data error of linearity I could possibly get.

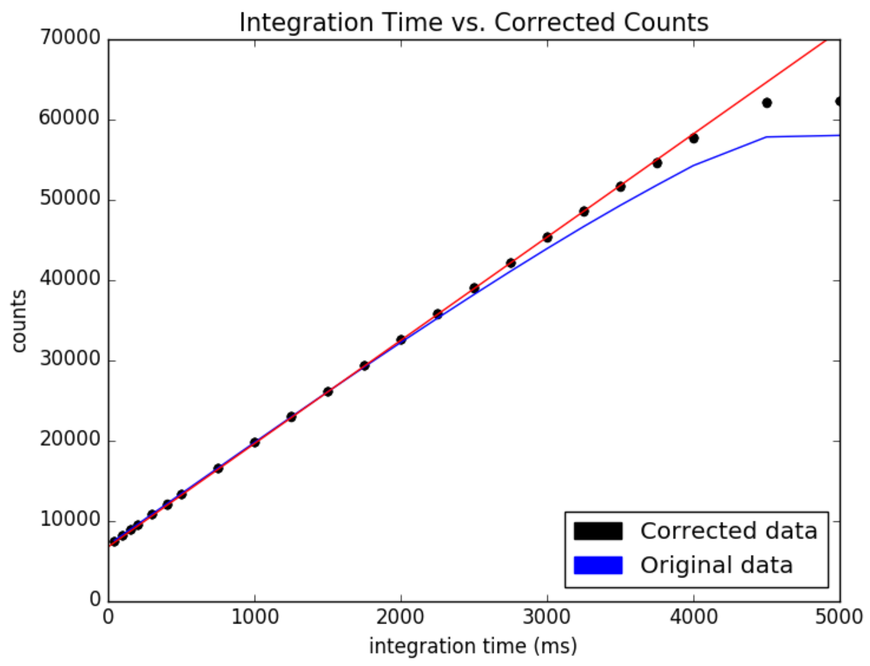
Where in my code this is calculated:

File: linearityReportNovember2014Data.py

Error is calculated: error(), lines 132-162

Graph is printed: print_graph(), lines 231-252

Corrected Counts vs Ints:



This is the graph for 4th order corrected data, which happens to be linear up to ~50,000 counts. Now, the process for this was as follows:

First, I made sure that the 4th 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 fourth 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: $y = ax^4 + bx^3 + cx^2 + dx + e$

Where x is the measurement of counts from the image, y is the corrected counts, and with:
a = 3.67e-16, b = -7.58e-11, c = 2.69e-06, d = 9.66e-01, e = 1.39e+02

Where in my code this is calculated:

File: linearityReportNovember2014Data.py

Correction is applied: make_true_counts(), lines 101-130

Graph is printed: print_graph(), lines 256-273