STScI | SPACE TELESCOPE SCIENCE INSTITUTE

JWST TECHNICAL REPORT

Title: A New Non-Linearity Correction Method for NIRCam	Doc #: JWST-STScI-005167, SM-12 Date: 09 September 2016 Rev:-
Authors: A. Canipe, M. Robberto, B. Hilbert Phone: 410-338-6864	Release Date: 20 January 2017

1. Abstract

I evaluate a new method proposed to calculate a more accurate non-linearity correction for NIRCam data using a function of the ideal linear signal count rate. I demonstrate that using this algorithm allows the reconstruction of the true linear signal to within 0.2% over ~ 97% of the full dynamic range, a substantial improvement over the current correction strategy adopted (e.g. for WFC3/IR). Using this method, I analyze how the linearity correction coefficients change for different grouping methods, concluding that the original coefficients derived using the full ramp of ungrouped data are also adequate to reconstruct the true signal in the case of grouped frames. Additionally, the robustness of the method is tested using simulated ramps with high slopes. Finally, I verify that the method can successfully be applied to all 10 NIRCam detectors.

2. Introduction

JWST infrared detectors have an intrinsic non-linearity due to the change in PN junction capacitance as charge accumulates in the individual pixel capacitors. Correction of this non-linearity is a fundamental step in the JWST Science Calibration Pipeline, and is traditionally done through two rounds of curve fitting using the method described in Robberto (2010, 2011) and Hilbert (2008, 2014): the first passage uses the measured counts of a calibration ramp to determine the coefficients (slope and intercept) of the true linear signal, the second determines the correction coefficients to be applied to reconstruct any linear signal. For the work described here, the new method refers to the initial fitting of the ramp to determine the true linear signal, described in detail in the next section. Low and intermediate signal levels may require modest correction, while at the other extreme the full dynamic range of the detector may be correctable almost up to the saturation threshold. When the accumulated signal becomes large enough that the highly non-linear response is not correctable within some predetermined level of accuracy, the detector has essentially reached the saturation threshold. Saturated data are excluded from the curve-fitting routine.

2.1. Initial Curve-Fitting Method

An IR detector illuminated with constant flux returns a non-linear integrated signal. To derive the correction to be applied to the non-linear signal, one has to figure out the linear flux that an ideal detector would have detected. Traditionally one assumes that the first part of the

ramp is unaffected by non-linearity, and therefore estimates the true linear signal by fitting a line to a small range of low-signal groups in this regime and extrapolating with the resulting slope to create the "true" signal. This is the method currently used in the first step to generate coefficients for the pipeline. However, we propose instead to use the method of Robberto (2010, 2011), later applied by Hilbert (2014) for WFC3/IR. This method assumes non-linearity effects are present in the measured ramp at all signal levels; therefore, rather than presuming the existence of an initial linear regime, it is better to fit the full ramp using a polynomial expansion of the ideal linear signal rate. The coefficients of the second order and higher account for the deviation from linearity of the system response, as seen in Equation 1.

$$\boldsymbol{s_{meas,i} = B + At_i + C \cdot (At_i)^2 + D \cdot (At_i)^3 + E \cdot (At_i)^4 + F \cdot (At_i)^5 + \cdots} \quad (1)$$

Here, $s_{meas,i}$ (DN) is the measured signal at each frame i, A (DN/s) is the true signal rate in the absence of non-linearity effects, t_i (s) is the exposure time at each frame, and C, D, E, and F represent the magnitudes of deviation from the ideal linear signal. The order of the polynomial is a tunable parameter that can be determined after a comparison between fits to the ramp and the resulting signal corrections. Solving this equation deriving polynomial coefficients for each pixel provides an estimate of the signal rate A and the offset (bias) B, i.e. the ideal linear signal that one would have detected at each frame, given by Equation 2:

$$\boldsymbol{s_{ideal,i} = B + At_i} \quad (2)$$

Having determined the linear flux, the departure from linearity is then represented by the ratio between the estimated ideal signal, s_{ideal}, and the observed signal, s_{meas},

$$\boldsymbol{R_{meas,i} = \frac{s_{ideal,i}}{s_{meas,i}}} \quad (3)$$

for each frame i. This ratio is generally larger than 1.0 and increases with the accumulated signal. In practice, this ratio is a linearity correction curve relating the departure from linearity, $R_{meas,i}$ to the measured counts $s_{meas,i}$. A best-fit relation can be obtained through a second polynomial expansion (Equation 4) fit to the linearity correction curve, $R_{meas,i}$ vs. $s_{meas,i}$.

$$\boldsymbol{R_{fit,i} = \alpha + \beta s_{meas,i} + \gamma \cdot \left(s_{meas,i}\right)^2 + \delta \cdot \left(s_{meas,i}\right)^3 + \zeta \cdot \left(s_{meas,i}\right)^4 + \cdots} \quad (4)$$

The new correction coefficients α, β, γ, δ, and ζ can be generally used to correct the measured counts to recover the true linear counts s_{corr}, for any level of flux, using the equation:

$$\boldsymbol{s_{corr,i} = s_{meas,i} \cdot [\alpha + \beta s_{meas,i} + \gamma \cdot \left(s_{meas,i}\right)^2 + \delta \cdot \left(s_{meas,i}\right)^3 + \cdots]} \quad (5)$$

2.2. Saturation Levels

While the first polynomial fit returns by design the coefficients of a linear ramp, the second polynomial fit returns a set of data that only approximately will match a linear signal. The departure from linearity provides a measure of the quality of the correction and can be used to determine the saturation point for the measured signal. Using this type of criterion we can correct at high signal levels close to the top of the ramp, signals that depart by more than 20% from

linearity. This is much larger than the "soft" saturation limit at 5% non-linearity that was traditionally defined by instruments such as WFC3/IR (Hilbert 2008). In the case of NIRCam, a 5% threshold would in fact exclude most of the calibration ramp, as the signal non-linearity becomes larger than 5% after 300 seconds of the 2040 second integration. This is partially due to the fact that our method of estimating the true linear signal (first fit) returns a slope that is steeper than the one obtained by fitting the first few frames, being tangent to the slope at the beginning of the integration.

As we will see, by allowing the quality of our signal correction to determine the soft saturation limit, rather than using a predefined amount of non-linearity in the original data, we can push the linearity correction to very high signal levels. Because the linearity correction is only reliable up to this soft saturation limit, it is important to have a correction method that is highly accurate for as much of the ramp as possible. In our analysis, soft saturation is defined as the signal level at which the corrected signal values deviate by more than 0.2% from the ideal linear behavior.

3. Data Preparation

For this analysis, we use data set N815B-LIN-5365135655 from the Integrated Science Instrument Module (ISIM) Cryo-Vacuum Test 3 (CV3) at NASA Goddard Space Flight Center. In particular, this is one of the NIRCam linearity test exposures (test ID 8.15) in which long integrations of steady brightness (35-55 DN) are collected after the diffuser lamps are allowed to settle. It is a flat field integration composed of 190 readouts of the NIRCam B3 detector in sensor chip assembly (SCA) 488. The first frame of the data set with 10.7 seconds of illumination is shown in Figure 1. The full integration time is 2040 seconds. Data set details are summarized in Table 1.

Table 1: Summary of CV3 data set details.

Test Date	31 December 2015
Exposure ID	N815B-LIN-5365135655
SCA ID	488
Detector	NRCB3 (short wavelength)
Filter	F150W2
Pixel ID (x,y)	(750,600)
Time Between Frames	10.7 sec
Total Integration Time	2040 sec
Pipeline Calibration Steps	DQ initialization saturation check superbias subtraction reference pix correction
Readout Patterns Studied	RAPID (read 1, skip 0) BRIGHT1 (read 1, skip 1) SHALLOW4 (read 4, skip 1) MEDIUM2 (read 2, skip 8) DEEP8 (read 8, skip 12)

Several JWST calibration steps were applied to the data set before our analysis. The first step is data quality initialization, which flags pixels whose behavior is recognized as being significantly different from the other pixels (e.g. those with excessive dark current or low sensitivity). Next, the saturation check was performed to flag pixels considered saturated according to the latest JWST reference files, although this information was not used for the analysis. The saturation levels were determined using our fit to the data. Subsequently, superbias subtraction was performed to remove the spatially dependent bias signal present after detector reset. Finally, the reference pixel correction step was completed, removing spatially and temporally dependent bias signal using the signal measured in the outermost four rows and columns of the detector known as the reference pixels. Reference pixels are not light-sensitive, but are otherwise electronically identical to the regular pixels.

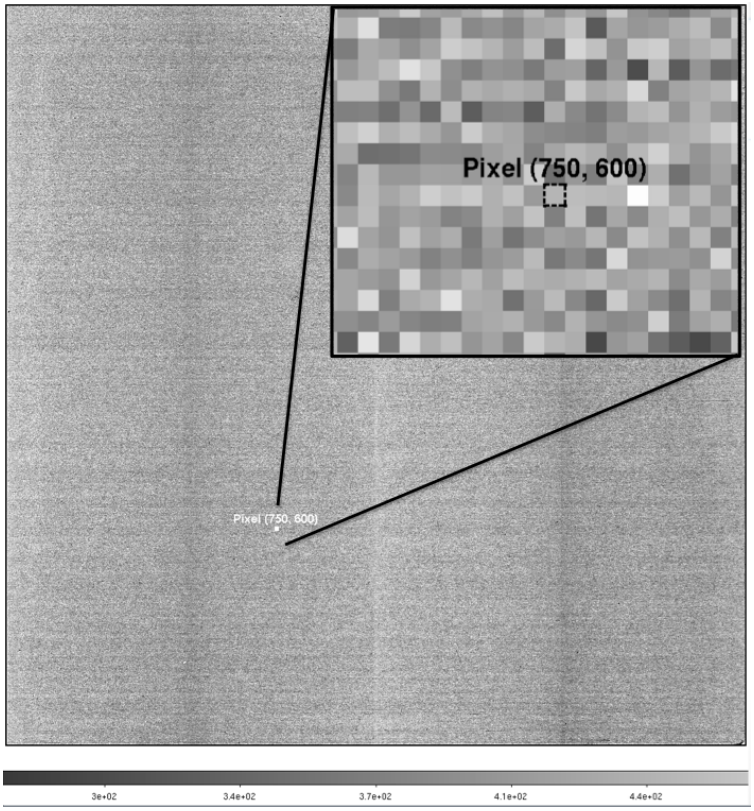


Figure 1: First frame of data set showing 2048 x 2048 pixel array for detector B3 and close up image of individual pixel used for analysis. Four readout channels for the detector are also visible (demarcated by faint vertical lines).

After applying calibration steps and isolating a randomly selected, well-behaved pixel (750,600) to guide our analysis, the signal ramp was constructed by plotting read time (190 x 10.73676 sec) versus signal level (DN). Once the accuracy of the correction method was verified for pixel (750,600), it was tested on the full 2048 x 2048 pixel array for detector B3 to output correction coefficients and coefficient errors for each pixel. Additionally, the correction method was tested on the other 9 NIRCam detectors to verify consistency across the full NIRCam Focal Plane Array (FPA). The final analysis section is dedicated to this topic, though a more in depth report detailing results for all CV3 exposures for each detector will follow.

In a second phase, the data were grouped according to the different NIRCam readout patterns listed in Table 1. For each read mode, a number of frames (*Ngroup*) are averaged together to reduce data volume, followed by a number of frames dropped to reduce the effects of averaging (*Nskip*). The choice of parameters is a trade-off between readout noise and loss of effective integration time due to data contaminated by cosmic rays (CR). The non-linearity correction algorithm was applied for each sampling method listed in the table.

As a final verification for the correction method, very steep ramps were simulated with Equation 1 by substituting large slope values for the signal rates and using interpolation to recreate signal saturation for the steep ramps. The curve-fitting procedure was then performed on the simulated ramps to test the accuracy of the non-linearity correction.

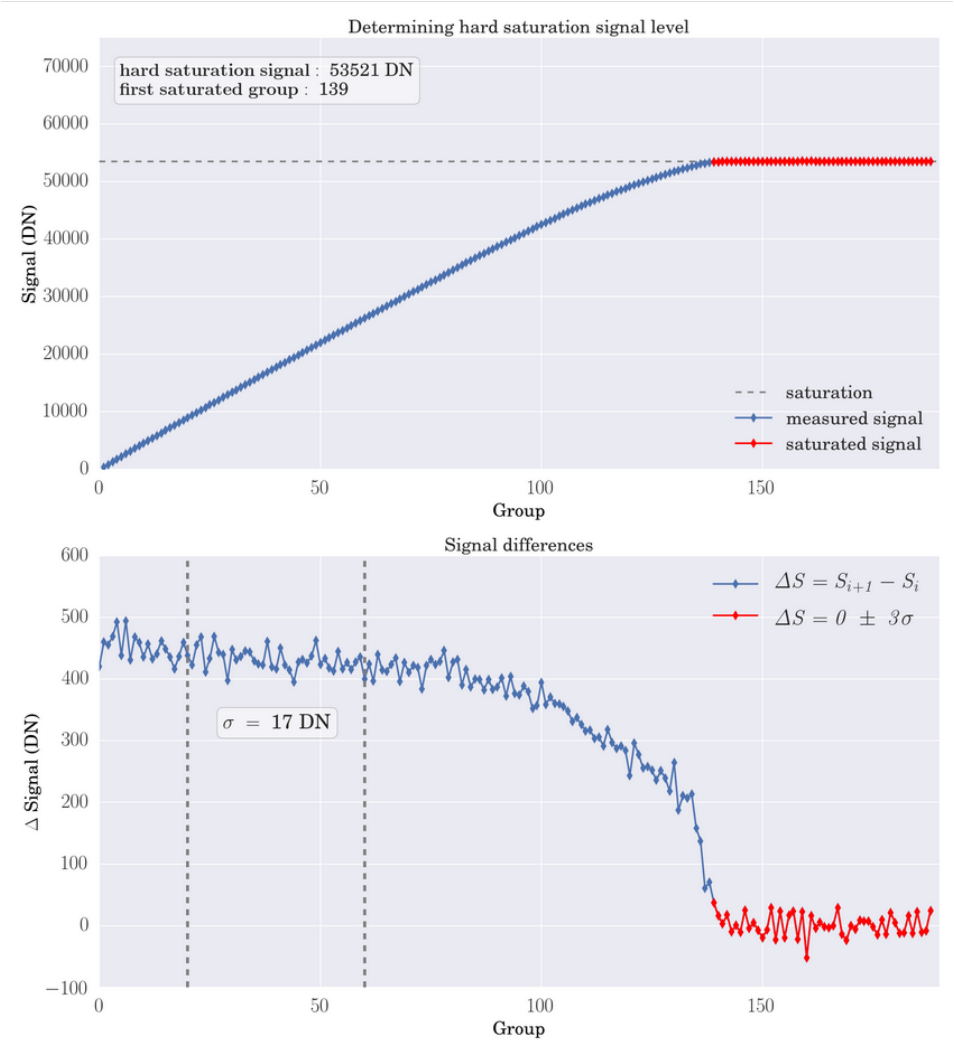


Figure 2: Plot showing full ramp of signal versus time with hard saturation at 53521 DN (top) and the signal value differences up the ramp (bottom)

4. Initial Analysis of a Representative Pixel

After calibrating the data, the two curve-fitting routines described in Section 2.1 were used to generate the correction coefficients for the randomly selected pixel, (750,600). Before the first round of curve fitting, it was necessary to determine the range of the detector data usable for analysis.

To determine the saturation level we looked at the differences between consecutive samples. Signal values early on in the ramp show large variations, whereas at the end their differences are compatible with zero. We flag samples in the range $0 \pm 3\sigma$ as saturated. By taking an average of signal values between the lowest saturated frame and the end of the integration, we determined a hard saturation level of 53521 DN. Figure 2 shows the full ramp of signal versus time and the sharp drop-off that occurs in the difference between signal values for groups above 140. The following analysis to determine the ideal signal and correction coefficients only uses data below this signal level.

4.1. Determining the Ideal Signal

After rejecting the saturated frames, we also ignored the 0th frame whose signal value showed residual deviation larger than 2% from the rest of the frames in the full ramp fitting. Then, using the 1st through 138th frames (this second value is the appropriate upper limit for the representative pixel under consideration), we performed the first round of curve-fitting to determine the ideal count rate by fitting Equation 1 to the ramp. To do this we used the ***curve_fit*** function in SciPy's ***optimize*** package, an open-source software package for mathematics, science, and engineering that provides commonly used optimization algorithms. ***curve_fit*** allows the user to define a function (in this case, Equation 1) with the associated input uncertainties in the measured signals. More details on the Python procedures for the following steps can be found in Appendix C and D.

My function returned the polynomial coefficients and coefficient uncertainties. For this study, an estimate of the signal uncertainties was provided as input to the $curve\_fit$ function. The signal uncertainty $\sigma_{s,i}$ was approximated as:

$$\sigma_{s,i} = \sqrt{F_R * g + s_{meas,i}}$$

where F_R is the readnoise in e$^-$/sec, g is the gain in e$^-$/ADU and $s_{meas,i}$ is the signal in DN. For this estimate, $F_R \sim 16$ e$^-$ and $g \sim 2$ e$^-$/ADU. A more in depth study of the uncertainties will be included in the follow up report containing results for all CV3 exposures for each detector.

The order of the polynomial for Equation 1 was left as a tunable parameter to determine the best order for the full ramp fit. A comparison of fit results for a 3rd, 4th, and 5th order polynomial can be found in Appendix A (Figure A1 and Table A1). A 5th order polynomial provided the best fit to the data (Figure 3) with a residual standard deviation of $\sigma = 0.0018$ and was therefore used to determine the ideal rate for a linear signal, although a 4th order polynomial fit the ramp as well with $\sigma = 0.0019$. Only the slope and intercept were taken from the set of coefficients and used to generate the ideal linear ramp (Equation 2) and thus the non-linearity values (Equation 3). The results of the full ramp fitting are found in Figure 3 and Table 2. The derived bias level is −122 DN. The B coefficient was left variable in the fit to avoid introducing biases at the lower signal values that would force the curve to approach zero at time $t = 0$. A non-zero bias level estimated for this bias-subtracted curve can be attributed partly to kTC (thermal) noise and partly to the known amplifier-reset variations of NIRCam data taken with ASICS.

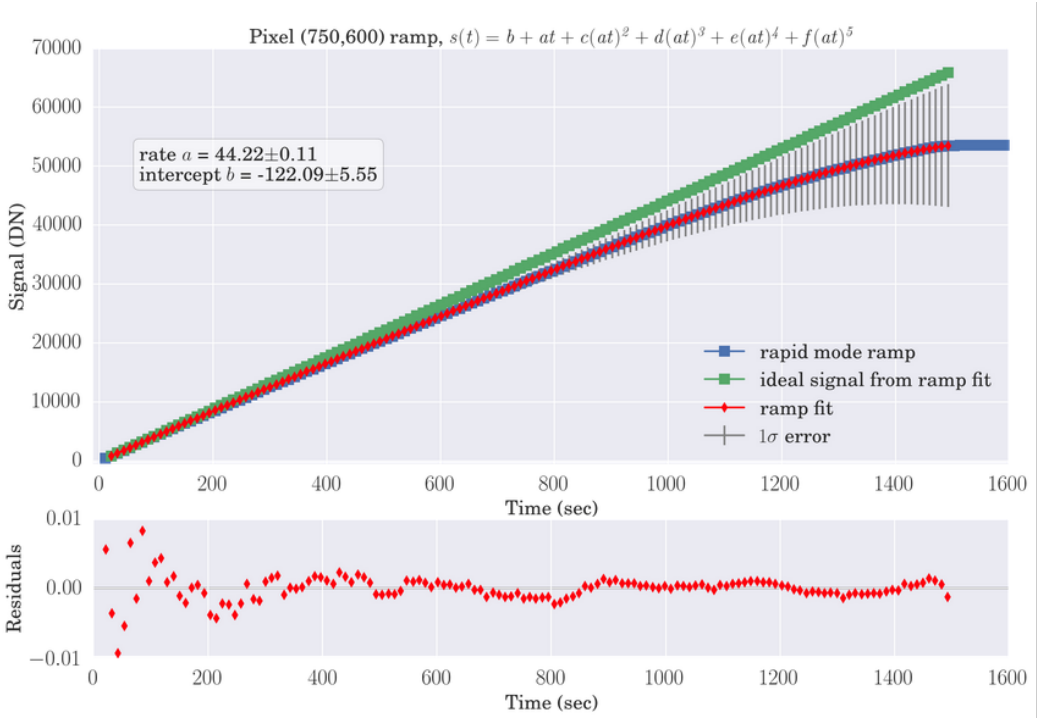


Figure 3: First round of curve fitting to determine the ideal signal rate. Ramp fit with 5th order polynomial (Equation 1) is plotted along with the resulting ideal linear signal and fit residuals.

4.2. Correcting the Non-Linear Signal

The next step to determine the correction coefficients was to fit Equation 4 to the non-linearity correction curve, R_{meas} vs. s_{meas}, once again using ***curve_fit*** to fit the polynomial. As in the previous round of curve-fitting, the polynomial order was left as a tunable parameter to determine the best order and data range for the correction. A table comparing the results of fitting the curve with various polynomial orders and the corresponding signal corrections can be found in Appendix A (Table A2). Once again, a 5th order polynomial provided the best fit to the non-linearity correction curve with residuals less than 0.2%, i.e. the resulting coefficients were able to correct the measured signal to within 0.2% of the ideal signal, up to the 133rd frame (19 frames higher than the result obtained with a lower order polynomial with the same error requirements). In comparison, correcting this pixel with the coefficients generated using the old method show residual non-linearity of 0.7% with soft saturation at 94% of the ramp, the 130th frame. The maximum signal correctable with the residual threshold of 0.2% was declared as the new soft saturation limit; for the pixel under examination it is 52613 DN, quite close (98.3%) to the hard saturation limit. Figures 4, 5 and Table 3 show the results of the 2nd curve-fitting routine to determine the correction coefficients.

Table 2: Table showing results from full ramp fitting with 5th order polynomial to determine ideal linear signal.

Parameter	Result (1 error)
Ideal count rate A (cts/s)	44.2 ± 0.1
B intercept	-122 ± 6
C coefficient	-5.2 x 10^{-6} ± 2.6 x 10^{-7}
D coefficient	1.3 x 10^{-10} ± 1.1 x 10^{-11}
E coefficient	-1.4 x 10^{-15} ± 2.1 x 10^{-16}
F coefficient	-1.4 x 10^{-21} ± 1.4 x 10^{-21}
Ramp fit residual σ	0.0018
Reduced χ^2	0.03 (132)

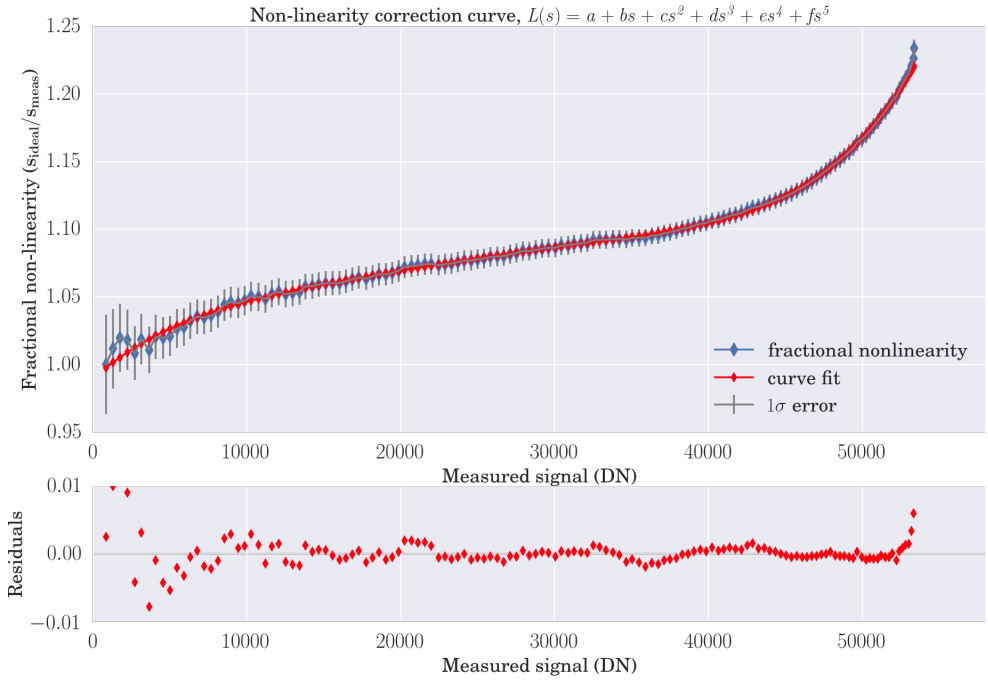


Figure 4: Second round of curve fitting to determine signal correction coefficients. Non-linearity correction curve, R_{meas} vs. s_{meas}, is fit with 5th order polynomial (Equation 4) to determine correction coefficients, with fit residuals shown underneath.

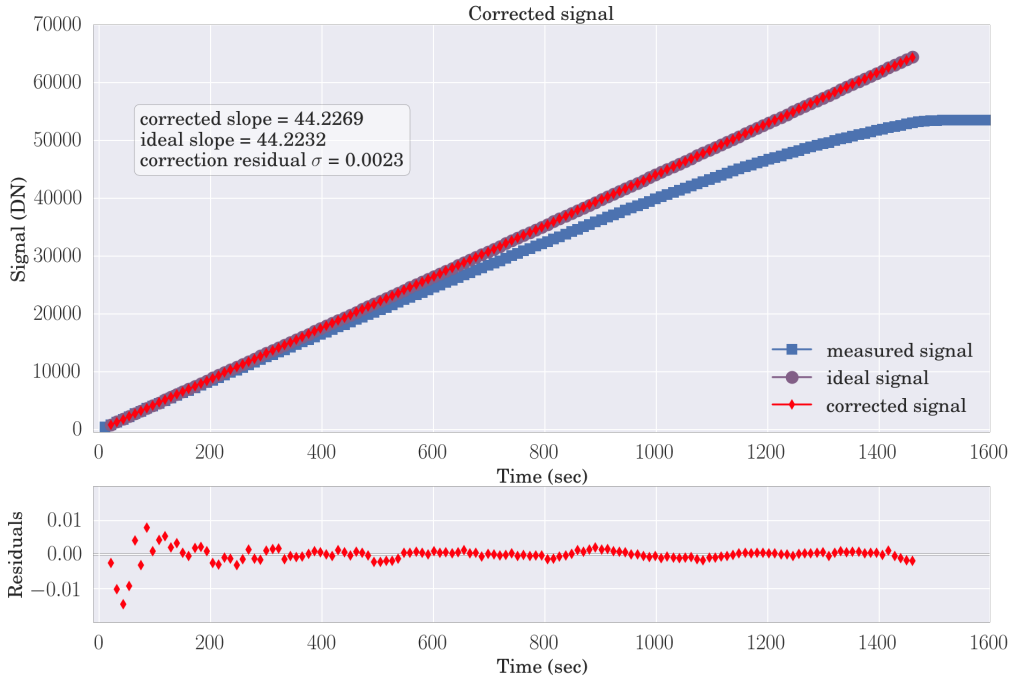


Figure 5: Ideal signal and ramp corrected according to Equation 5. Correction residuals show that the algorithm corrects to within ~ 0.2% of the ideal signal up to saturation.

Table 3: Table showing results from non-linearity correction curve fit with 5th order polynomial to determine correction coefficients, and corresponding result of signal correction.

Parameter	**Result (1 error)**
α coefficient	$9.9 \times 10^{-1} \pm 1.3 \times 10^{-2}$
β coefficient	$1.0 \times 10^{-5} \pm 3.5 \times 10^{-6}$
γ coefficient	$-6.6 \times 10^{-10} \pm 3.2 \times 10^{-10}$
δ coefficient	$2.8 \times 10^{-14} \pm 1.3 \times 10^{-14}$
ζ coefficient	$-5.9 \times 10^{-19} \pm 2.4 \times 10^{-19}$
η coefficient	$5.0 \times 10^{-24} \pm 1.6 \times 10^{-24}$
Non-linearity curve fit residual σ	0.0023
Curve fit reduced χ^2	0.03 (127)
Signal correction residual σ	0.0023

5. Applying the Correction to the Full Pixel Array

After determining the polynomial orders of the non-linearity correction algorithm for one pixel, the same method was applied to the full 2048 x 2048 pixel array for detector B3. The code was modified to output the coordinates for any pixels that had non-linearity corrections worse than 0.5% of ideal to facilitate comparison with the pixel data quality (DQ) array. Poor non-linearity corrections coincided with bad pixels and ramps that were generally not well-behaved, such as those showing jumps in the ramp or sharp drop-offs in the count-rate. A pixel data quality image flagging all pixels that do not match the 0.5% accuracy limit is shown in Figure 6.

Table 4 shows the results from running the algorithm for the full pixel array. The code successfully iterated over the full array and output correction coefficients for all pixels without fitting errors. 97% of the pixels had non-linearity corrections better than 0.5% of ideal, with an average of 0.3% and standard deviation $\sigma = 0.001$. Figure 7 shows data frame 15 and an image of slopes from the ramp fitting across the full pixel array. The superbias (i.e., the fixed pattern of the detector) has been removed, except for residuals that are described by the B coefficient array. Frame 15 is a representative image of the illumination pattern well before saturation, and the slope image is nicely similar. Figure 8 shows histograms of the ramp fit coefficients from Equation 1 that describe the intercept B, signal rate A, and 3rd order deviation from linearity, C. Outliers in the histogram tails correspond to pixels that had poor corrections. Figure 9 shows histograms of the first three correction coefficients from Equation 4.

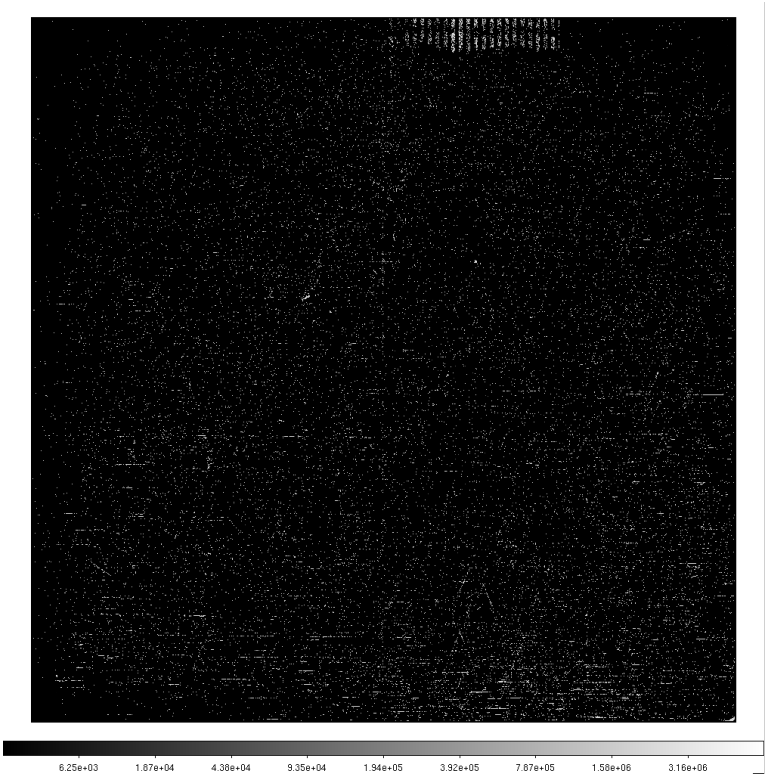


Figure 6: Pixel data quality image. Number scale corresponds to data quality flags that identify problem pixels, such as hot, noisy, or saturated pixels. Poor non-linearity corrections corresponded to flagged pixels that did not have well-behaved ramps.

Table 4: Results from using new non-linearity correction algorithm on full 2048 x 2048 pixel array.

# pixels w/ fit errors	32929 (1%)
# pixels w/ no fit errors	4161375
Average correction standard deviation	0.0028 (σ = 0.001)
# pixels w/ correction better than 0.5% from ideal	4051659 (97%)
Average ideal signal rate (DN/s)	42.03 (σ = 1.55)
Rate minimum, rate maximum (DN/s)	-2.86, 557.03
Average intercept value (DN)	-47 (σ = 97)
Intercept minimum, intercept maximum (DN)	-4122, 52938

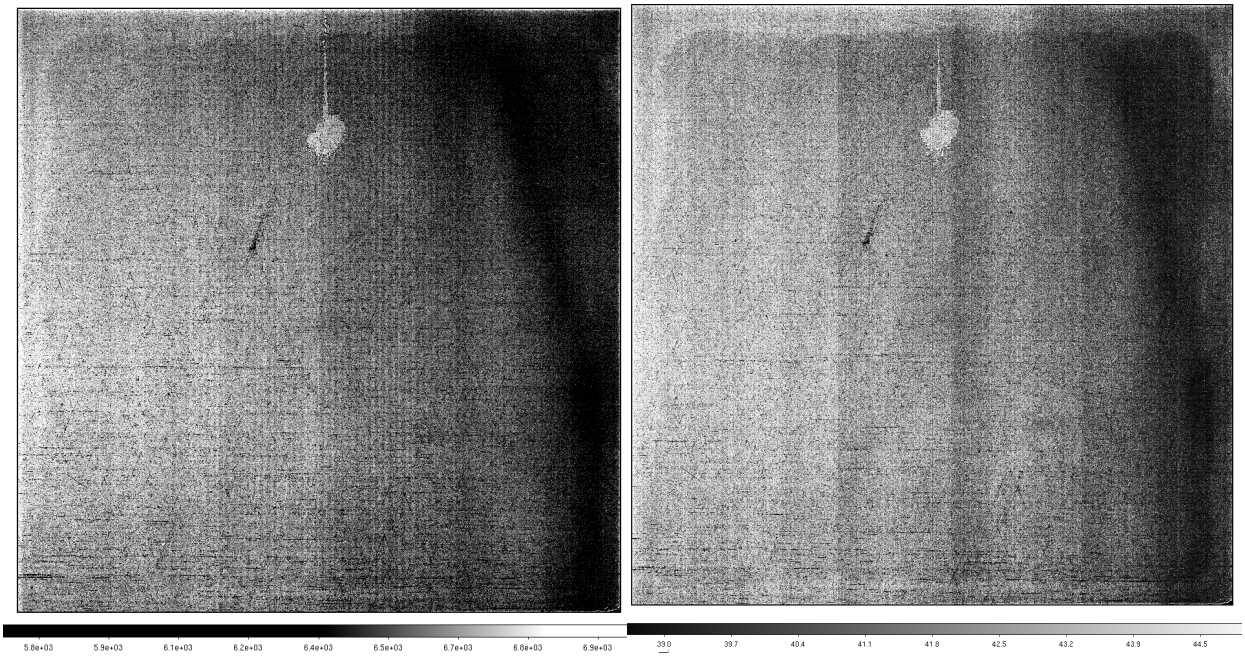


Figure 7: *Left*: Frame 15 of data cube, a representative image of detector illumination pattern before saturation. *Right*: Ramp fit slope image, nicely similar to frame 15.

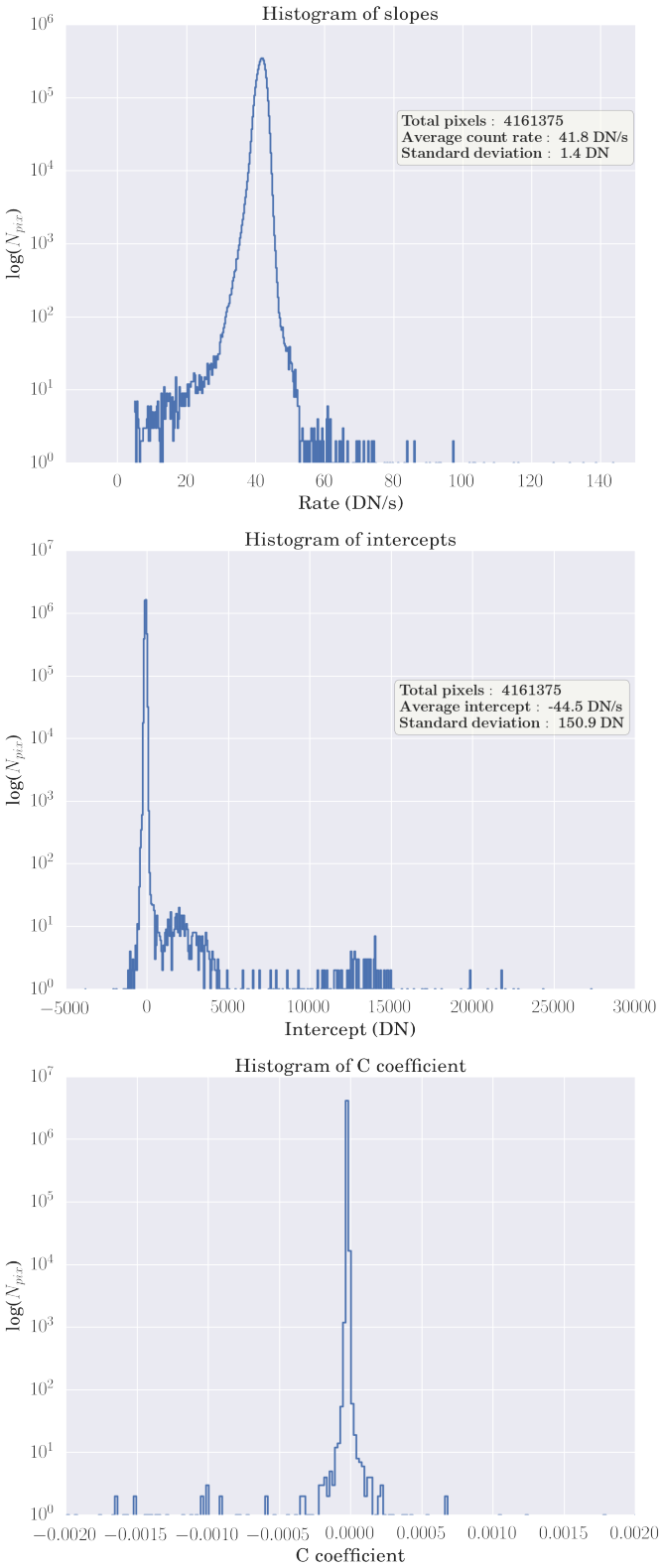


Figure 8: Histograms of first three coefficients in Equation 1, $B + At_i + C(At_i)^2 + D(At_i)^3 + E(At_i)^4 + F(At_i)^5$. Top: signal rates, coefficient A. Middle: signal intercepts, coefficient B. Bottom: First linearity deviation coefficient, C.

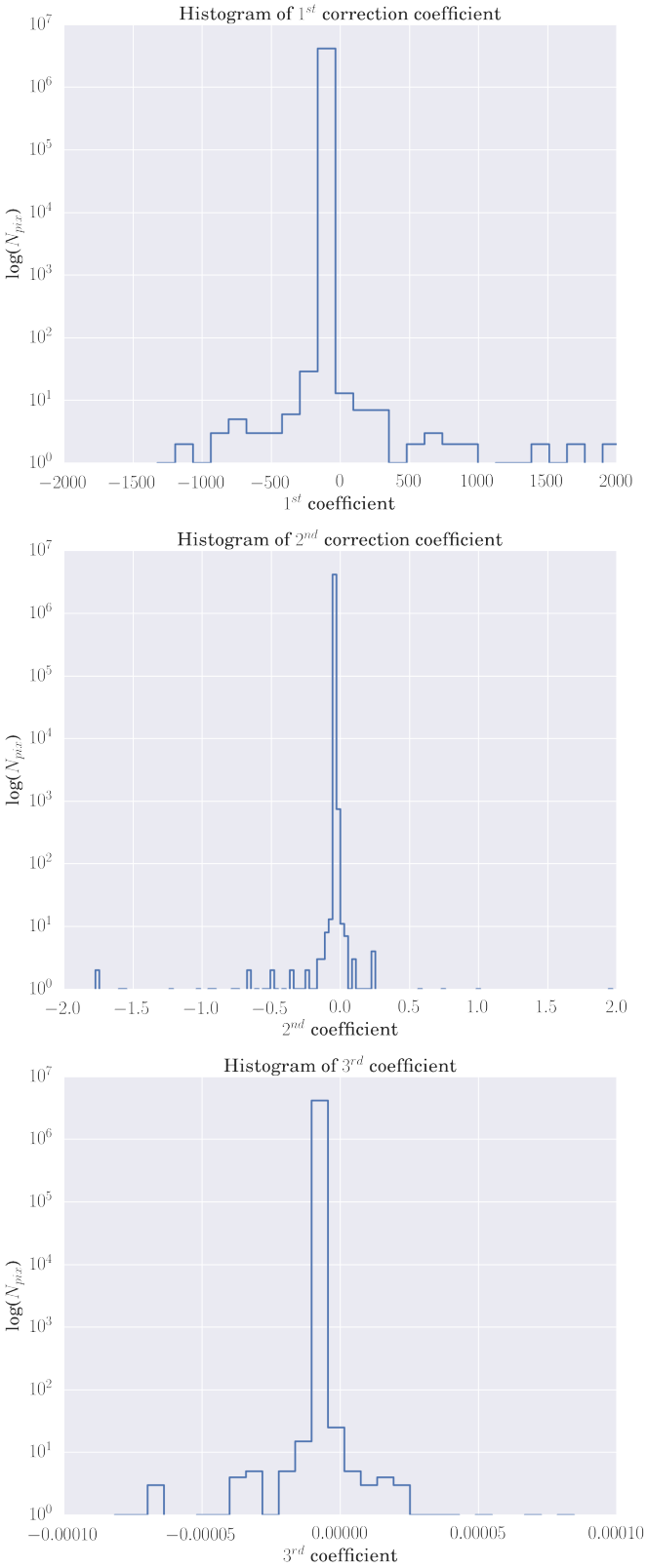


Figure 9: Histograms of first three *correction* coefficients in Equation 4, $\alpha + \beta s + \gamma \cdot (s)^2 + \delta \cdot (s)^3 + \zeta \cdot (s)^4 + \lambda \cdot (s)^5$. *Top*: α, coefficient 1. *Middle*: β, coefficient 2. Bottom: γ, coefficient 3.

6. Correcting Non-Linear Grouped Data

The data used in the previous sections for the analysis and determination of the ideal linear signal were obtained in RAPID mode, meaning all 190 frames of the ramp were read and none were skipped. Equipped with the ideal linear signal ramp, the correction coefficients can now be determined for the other readout patterns listed in Table 1. Grouping the data according to NIRCam's averaging method results in non-linear grouped ramps with slopes that are always slightly smaller (typically by ~ 1%) than the true linear slope.

To correct the non-linearity and get the proper slope value, we start by simply applying the same procedure used previously for RAPID mode data. The ideal linear signal is given again by $s_{true,i} = B + At_i$, where rate A and intercept B are those previously determined by the RAPID mode ramp fit. The fractional non-linearity, L_{group}, is given by s_{true}/s_{group}; one would expect that each sampling mode gives a particular non-linearity curve, but in fact all curves are nearly coincident (Figure 10). The global non-linearity correction curve, L_{group} vs. s_{group} can be fit with Equation 4 using curve fit to fit the polynomial and output the correction coefficients. Once again, the curve-fit residual standard deviation turns out to be less than ~ 0.2%, i.e. the coefficients can correct the measured grouped signals to within 0.2% of the ideal signal up to the last group before signal saturation. Correction results for grouped ramps are shown in Figure 11 and Table 5. A comparison of the correction coefficients for each readout mode can be found in Table B1 in Appendix B.

The fact that all readout modes provide nearly the same correction curves suggests that one could simply apply the best measured one, i.e. the RAPID mode correction coefficients, for all readout modes. RAPID mode coefficients, in fact, correct these non-linear ramps quite well, giving again grouped linear ramps within 0.2 % of the ideal linear signal. Results using this method are discussed further in Appendix B.

Table 5: Slope comparison for corrected linear grouped data. Coefficients are able to correct grouped data to within ~ 0.2% of ideal linear signal.

Mode	Corrected slope (DN/s)	$\frac{\text{corrected slope}}{\text{true slope}^a}$	Correction residual σ
BRIGHT1	44.227 ± 0.006	1.00009	0.0019
SHALLOW4	44.225 ± 0.007	1.00004	0.0010
MEDIUM2	44.217 ± 0.010	0.99986	0.0027
DEEP8	44.223 ± 0.005	0.99999	0.0002

[a] True slope = 44.2231 DN/s from full ramp fitting of rapid mode data.

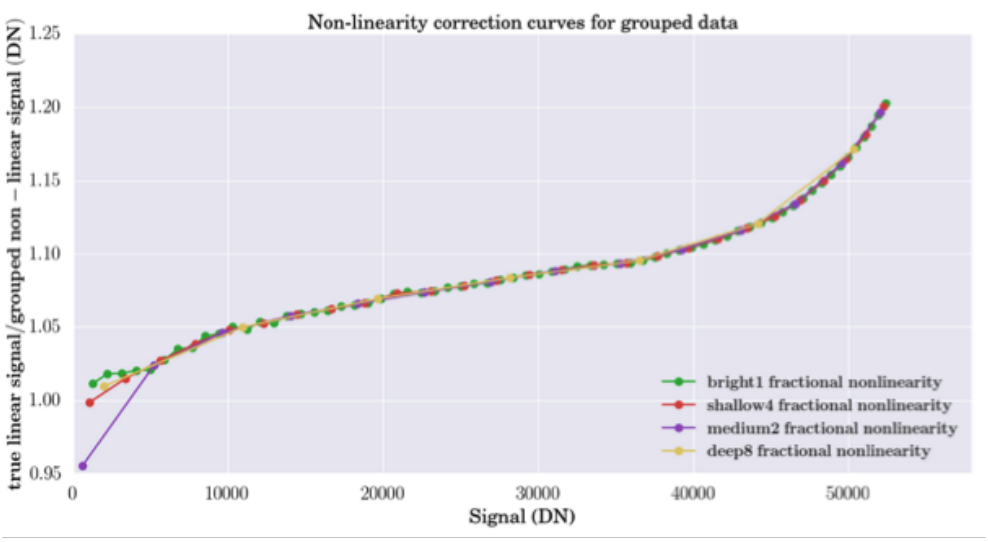


Figure 10: Non-linearity correction curves, L_{group} vs. s_{group}, for the readout patterns listed in Table 1. Curves are fit with 5th order polynomial (Equation 4) to determine correction coefficients.

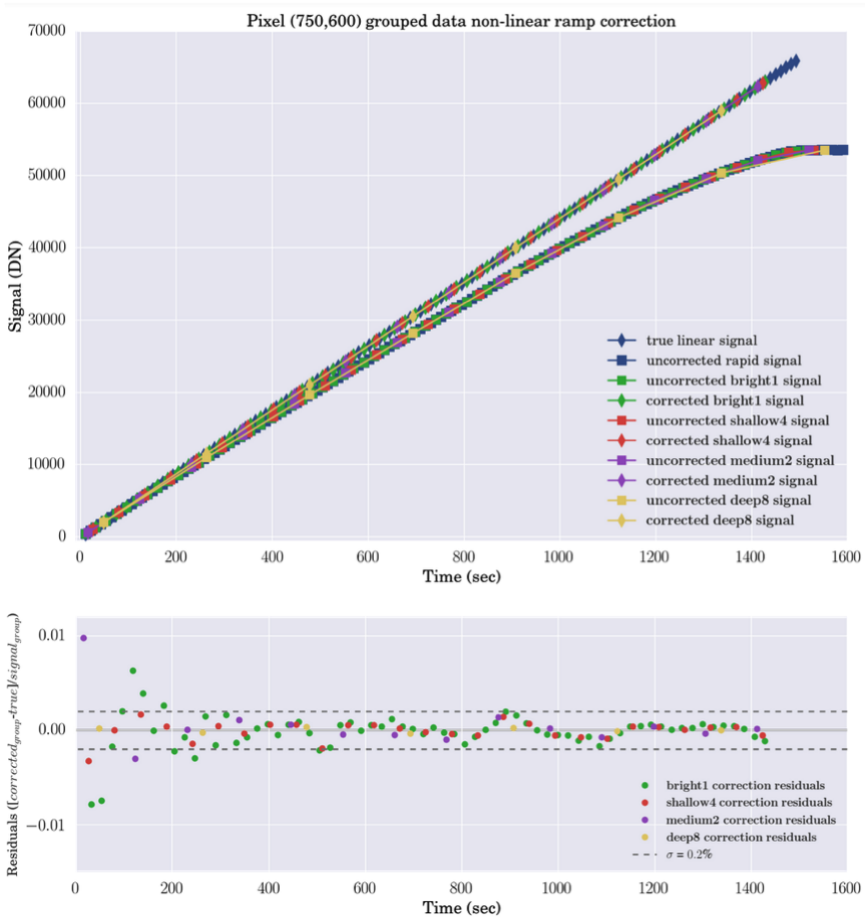


Figure 11: Ideal signal and grouped ramps corrected according to Equation 5. Correction residuals show that the algorithm also corrects grouped data to within ~ 0.2 % of the ideal signal up to saturation.

7. Correcting Steep Ramps

It turns out that the basic algorithm can correct the non-linearity in the ramp of pixel (750, 600), as well as the ramps of all other well-behaved pixels in the full array, and our "synthetic" grouped ramps for all different read modes. I have checked that the algorithm is also powerful enough to correct ramps with slopes 2 to 7 times steeper than the ramp for pixel (750, 600) analyzed in Sections 3.1 and 3.2. These non-linear ramps were simulated using the ramp equation (Equation 1) with the non-linearity coefficients from Table 2. However, instead of using A = 44 DN/s, we used steeper slopes of 120 DN/s and 300 DN/s; all other coefficients were left the same. We did not use steeper ramps, as the number of usable groups decreases to the point that the test cannot be effectively carried out. Saturated signals were simulated using SciPy's interpolation function $interp1d$, which takes arrays of values used to approximate some function f: $y = f(t)$. In this case, the saturated signal is $f(t)$. $interp1d$ returns a function whose call method uses interpolation to find the value of new points, taking the saturated signal from pixel (750,600) and recreating it for the simulated steep ramps. The ramps are shown in Figure 12, with the original pixel ramp included for comparison.

I then followed the same fitting procedure: signal saturation levels were determined, the full ramps were fit with 5th order polynomials to find the ideal signal rate and non-linearity correction curve, and the non-linear signal was corrected with the coefficients obtained by fitting the correction curve with another 5th order polynomial. Figure 13 shows the ramp fitting and Figures 14 and 15 show the non-linearity correction curves and corrected linear signals respectively. Fit results are listed in Table 6. The algorithm is still able to correct the steep non-linear signals to within ~ 0.2% of the ideal linear signal for 95% of the dynamic range, consistent with the results from the previous applications of the algorithm.

Table 6: Correction coefficients (Equation 5) and results for simulated steep ramps.

Parameter	**Simulated ramp 1 (slope = 120 DN/s)**	**Simulated ramp 2 (slope = 300 DN/s)**
α coefficient	$9.8 \times 10^{-1} \pm 2.0 \times 10^{-2}$	$9.8 \times 10^{-1} \pm 3.6 \times 10^{-2}$
β coefficient	$1.4 \times 10^{-5} \pm 5.2 \times 10^{-6}$	$1.2 \times 10^{-6} \pm 9.2 \times 10^{-6}$
γ coefficient	$1.1 \times 10^{-9} \pm 4.7 \times 10^{-10}$	$8.7 \times 10^{-10} \pm 8.2 \times 10^{-10}$
δ coefficient	$4.7 \times 10^{-14} \pm 1.8 \times 10^{-14}$	$3.6 \times 10^{-14} \pm 3.3 \times 10^{-14}$
ζ coefficient	$9.7 \times 10^{-19} \pm 3.3 \times 10^{-19}$	$7.5 \times 10^{-19} \pm 5.9 \times 10^{-19}$
η coefficient	$7.8 \times 10^{-24} \pm 2.2 \times 10^{-24}$	$6.1 \times 10^{-24} \pm 4.0 \times 10^{-24}$
Signal range (% full ramp)	94.5 (53 of 55 frames)	95.2 (20 of 21 frames)
Correction residual σ	0.0032	0.0017

A 5th order polynomial fit seems to leave no evidence of systematic effects, since the data are randomly scattered around the curves for all scenarios. In principle, one could increase the order of the polynomial, or consider modeling the ramp using different expressions, other than polynomial fits; a non-linearity correction model could be constructed purely based on the physics behind the IR detector non-linearity. However, the combination of accuracy and speed

reached with the algorithm presented in this report may be fully adequate for the needs of JWST instruments.

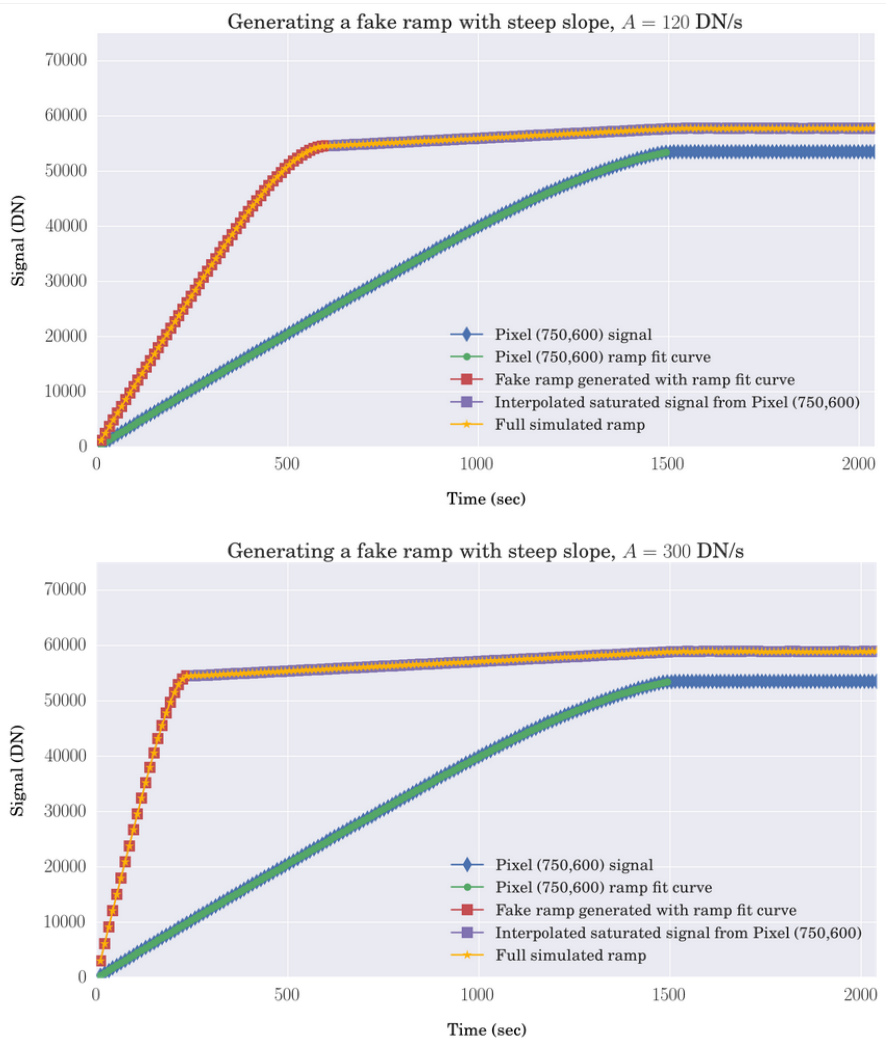


Figure 12: Simulated steep ramps. Top: Ramp with rate A coefficient of 120 DN/s. Bottom: Ramp with rate A coefficient of 300 DN/s.

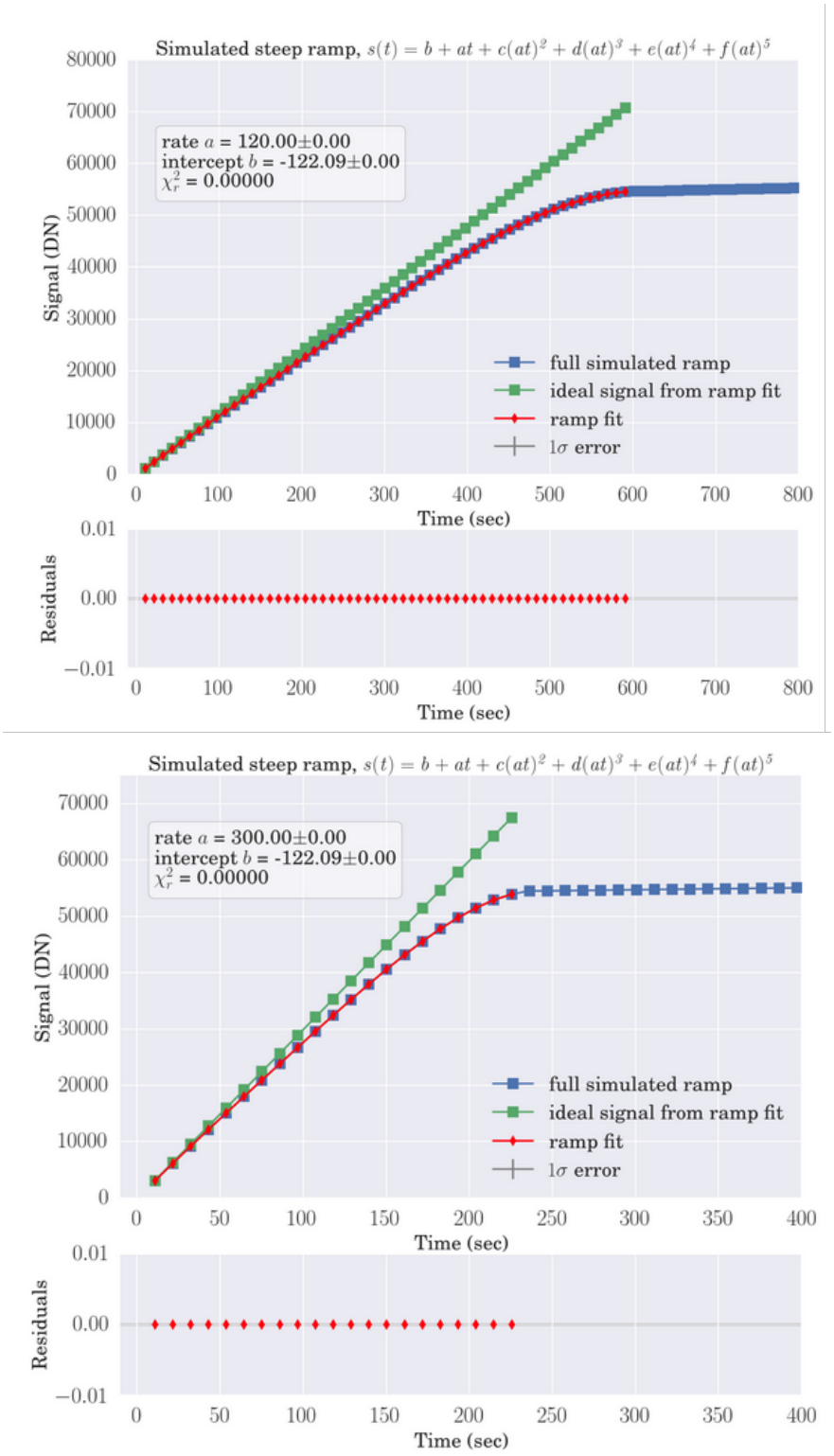


Figure 13: Ramp fit for simulated steep ramps. Top: Ramp fit for simulation with rate *A* coefficient of 120 DN/s. Bottom: Ramp fit for simulation with rate *A* coefficient of 300 DN/s. Ramps were generated with Equation 1, so residuals are ~ 0 and ideal signal slope is equal to input slope.

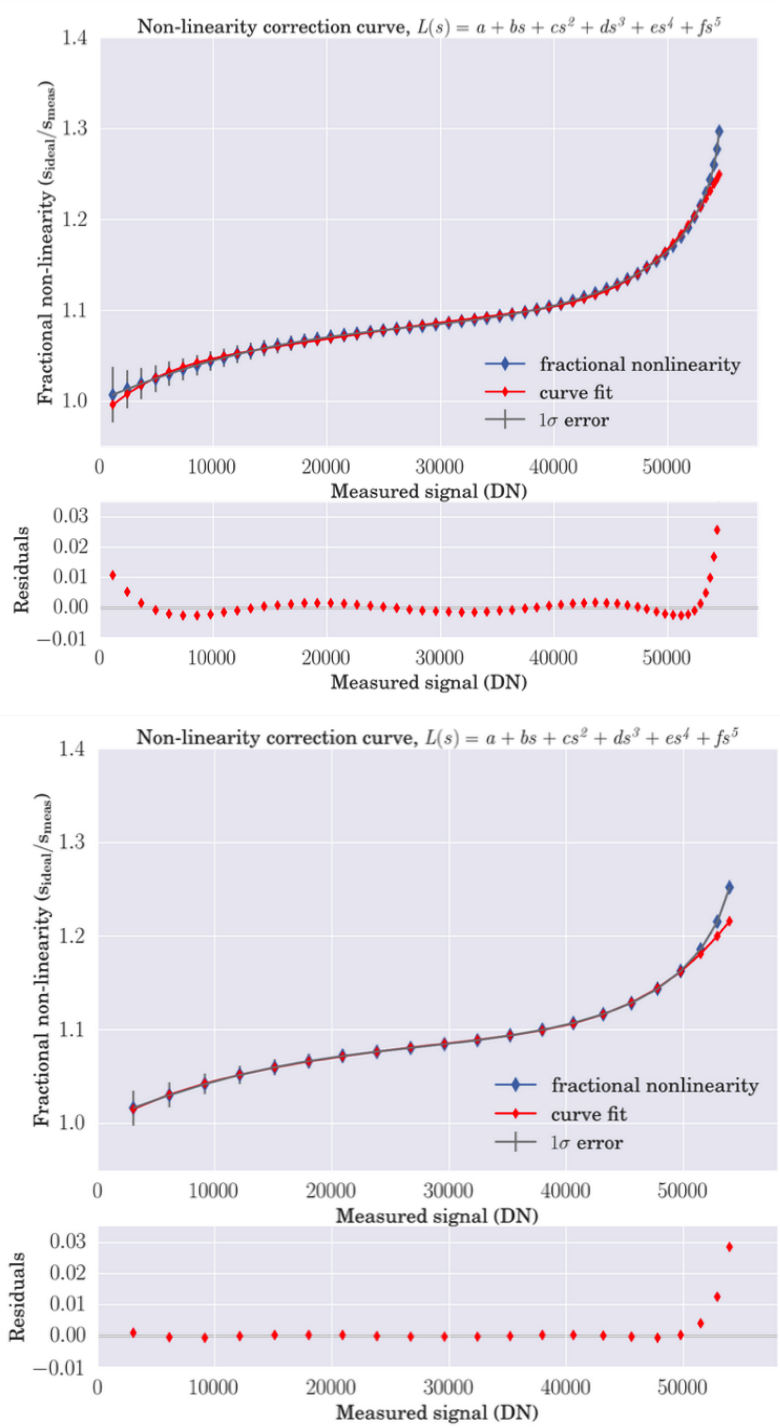


Figure 14: Non-linearity correction curves for simulated steep ramps. Top: Curve for simulation with rate *A* coefficient of 120 DN/s. Bottom: Curve for simulation with rate *A* coefficient of 300 DN/s.

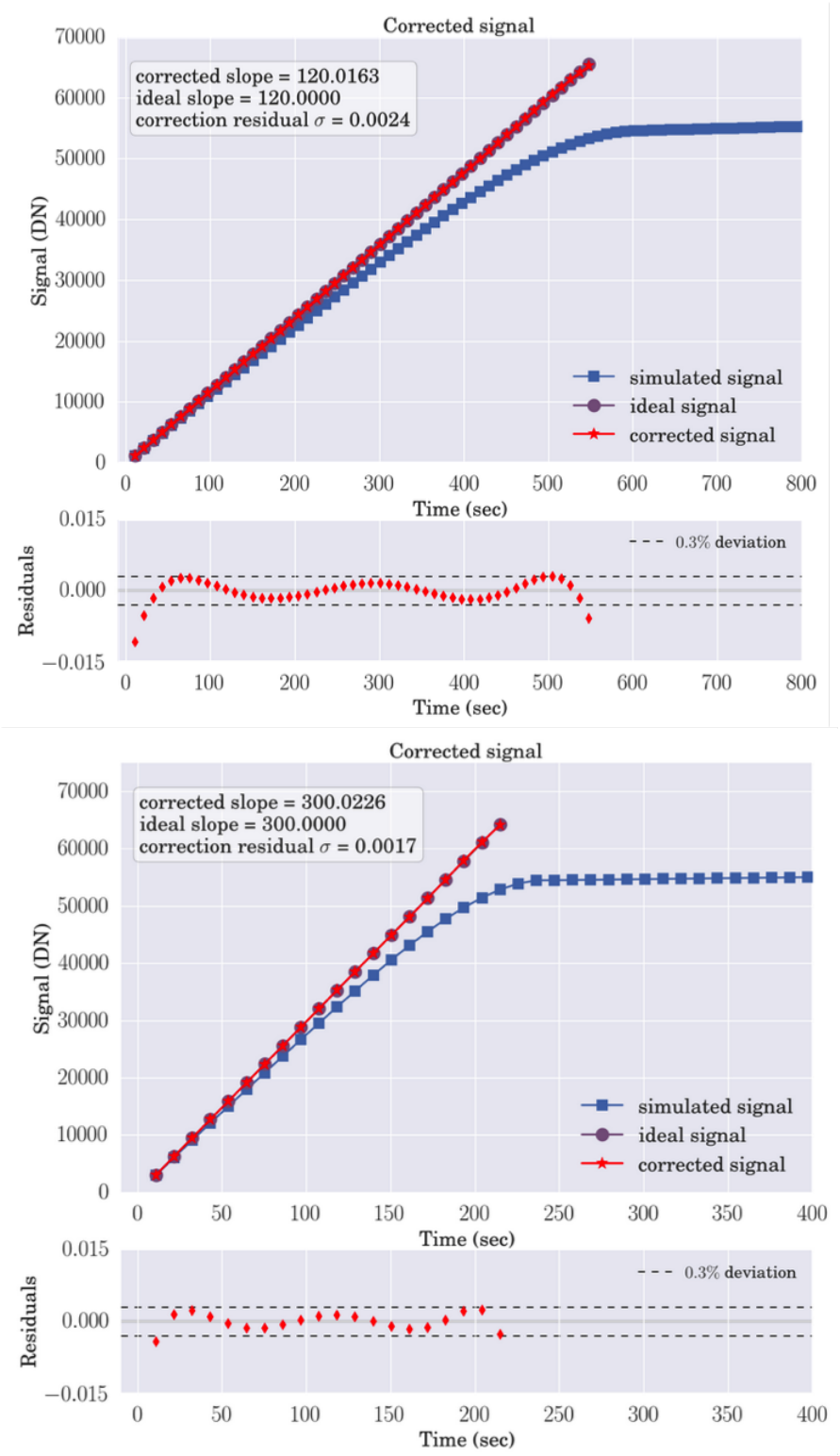


Figure 15: Corrected linear signals for simulated steep ramps. Top: Correction for ramp with slope A = 120 DN/s. Bottom: Correction for ramp with slope A = 300 DN/s.

8. Testing the Correction on the Other 9 Detectors

Detector B3 is just one of the detectors (Sensor Chip Assemblies, SCAs) of the 2 x 2 chip mosaic that composes the FPA of the short wavelength channel of module B. The algorithm was also tested on all other SCAs in module B and module A, both long and short wavelength channels, to determine if the results obtained for detector B3 are representative of all SCAs. We limited the analysis to a randomly chosen subarray of 350 x 350 pixels to reduce the time required to run the code on all 9 detectors. Results turn out to be fully consistent, with corrections within 0.2 – 0.3% of ideal for all other detectors. Detailed results for the entire set of detectors in the full array will be documented in a future report associated with the delivery of the linearity correction coefficients to the SSB pipeline. This report will use all available data sets from CV3 testing and include more analysis on the correction coefficient uncertainties.

9. Conclusions

Based on the results from the previous analysis, it appears that combining A) the method of correcting the measured signal by first fitting the full ramp with a 5^{th} order polynomial function of the ideal count rate to determine the linear signal, with B) a 5^{th} order polynomial to fit the resulting non-linearity correction curve, can provide a correction accurate to within 0.2% of the ideal linear signal over almost the entire ramp - specifically, up to ~ 97% of the full dynamic range of the detector. This residual non-linearity on corrected data is better by 0.5% compared to the current correction coefficients used by the pipeline. Additionally, the new method allows us to find the soft saturation limit, the range over which the correction is reliable, for each pixel. This limit is currently hard-coded into the correction files. The new algorithm has been tested on one pixel, the full array, grouped data, and simulated ramps with very steep slopes. It is a robust, highly accurate method that recovers the ideal linear signal better than existing methods by at least 0.5%.

References

Bryan Hilbert. WFC3 TV3 testing: IR channel nonlinearity correction. (WFC3 ISR 2008-39), 2008.

Bryan Hilbert. Updated non-linearity calibration method for WFC3/IR. (WFC3 ISR 2014-17), 2014.

Massimo Robberto. An improved algorithm for the correction of IR detector non linearity. (JWST-STScI-002163), 2010.

Massimo Robberto. Implementation of a new algorithm for the correction of non-linearity in JWST detectors. (JWST-STScI-002346), 2011.

Acknowledgements

I would like to extend my thanks to Massimo Robberto for initially outlining the purpose and background of this project for me, as well as taking the time to provide suggestions as needed. I would also like to extend my gratitude to Bryan Hilbert and Brian Brooks for their support and assistance as I worked through the project details and methods.

Appendices

A. Polynomial Fit Comparisons

This section contains a figure and tables comparing results for the two rounds of curve-fitting using various polynomial orders. In both cases, a 5th order polynomial provides the best fit over the widest range of the data.

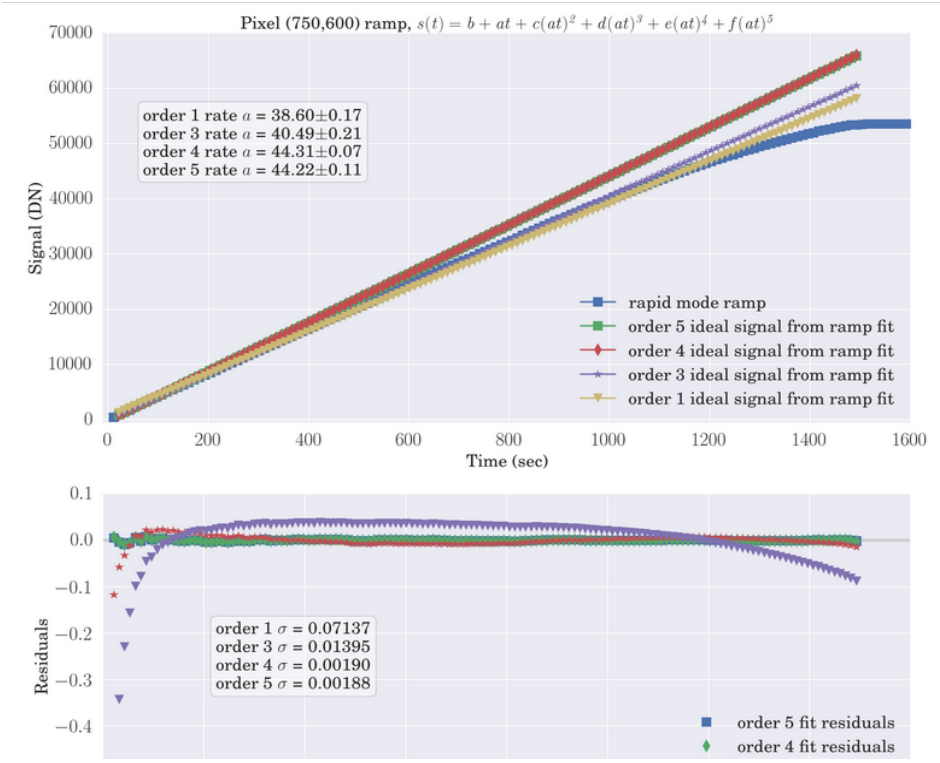


Figure A1: Upper plot shows ideal linear signals from the slopes of various polynomial order ramp fittings. Lower plot displays residuals from polynomial fits to the full ramp to get the ideal signal slope values in the upper plot.

Table A1: Table comparing non-linearity curve fit results for various polynomial orders and the corresponding signal correction residual standard deviations (σ).

Order	Fit range (frames)	Ideal linear rate	Intercept	Fit residual σ	Red. X^2
3	1 - 138	40.4 ± 0.2	54 ± 22	0.0139	1.23
4	1 - 138	44.31 ± 0.07	125 ± 5	0.0019	0.03
5	1 - 138	44.2 ± 0.1	122 ± 6	0.0018	0.03

Table A2: Comparison of results for full ramp fitting to determine ideal linear signal with various polynomial orders.

Order	Fit range	Soft saturation	Linearity curve fit reduced X^2	Correction residual σ
3	1 - 100	42510 DN	5.32	0.0022
4	1 – 114	47346 DN	1.23	0.0021
5	1 – 133	52613 DN	0.06	0.0021

B. Ramp Non-Linearity Correction Coefficient Comparison

This section contains a comparison between non-linearity correction coefficients for the different readout modes listed in Table 1. The coefficients correspond to Equation 5, with the correction $(\alpha + \beta s_{meas,i} + \gamma \cdot (s_{meas,i})^2 + \delta \cdot (s_{meas,i})^3 + \zeta \cdot (s_{meas,i})^4 + \eta \cdot (s_{meas,i})^5)$. Table B1 shows the coefficients for each readout pattern.

Section 5 briefly mentions using only the RAPID mode coefficients to correct non-linear data for grouped signals. Because the coefficients do not differ significantly between modes (within the error bars), simply applying RAPID mode correction coefficients to the non-linear grouped data produces linear ramps that are within 0.2 % of the true linear signal. Table B2, Figure B1, and Figure B2 show the corrected ramps and corresponding slope values after applying the same RAPID mode correction. Figure B3 shows the correction residuals up the ramp for two of the modes, SHALLOW4 and DEEP8, demonstrating that this method also corrects the non-linearity nearly up to saturation.

Table B1: Non-linearity correction coefficient comparison for all readout modes in analysis discussed in Section 5. Coefficients come from fitting the non-linearity correction curve, L_{group} vs. s_{group}, with 5th order polynomial (Equation 5).

Mode	α (1 error)	β (1 error)	γ (1 error)	δ (1 error)	ζ (1 error)	η (1 error)
RAPID	$9.9\text{x}10^{-1}$ ($\pm 1.3\text{x}10^{-2}$)	$1.0\text{x}10^{-5}$ ($\pm 3.5\text{x}10^{-6}$)	$6.6\text{x}10^{-10}$ ($\pm 3.2\text{x}10^{-10}$)	$2.8\text{x}10^{-14}$ ($\pm 1.3\text{x}10^{-14}$)	$5.9\text{x}10^{-19}$ ($\pm 2.4\text{x}10^{-19}$)	$5.0\text{x}10^{-24}$ ($\pm 1.6\text{x}10^{-24}$)
BRIGHT1	$9.9\text{x}10^{-1}$ ($\pm 3.1\text{x}10^{-3}$)	$9.4\text{x}10^{-6}$ ($\pm 8.1\text{x}10^{-7}$)	$6.0\text{x}10^{-10}$ ($\pm 7.4\text{x}10^{-11}$)	$2.5\text{x}10^{-14}$ ($\pm 3.0\text{x}10^{-15}$)	$5.5\text{x}10^{-19}$ ($\pm 5.6\text{x}10^{-20}$)	$4.7\text{x}10^{-24}$ ($\pm 3.8\text{x}10^{-25}$)
SHALLOW4	$9.8\text{x}10^{-1}$ ($\pm 3.3\text{x}10^{-3}$)	$1.0\text{x}10^{-5}$ ($\pm 8.3\text{x}10^{-7}$)	$7.5\text{x}10^{-10}$ ($\pm 8.1\text{x}10^{-11}$)	$3.1\text{x}10^{-14}$ ($\pm 3.3\text{x}10^{-15}$)	$6.3\text{x}10^{-19}$ ($\pm 6.2\text{x}10^{-20}$)	$5.2\text{x}10^{-24}$ ($\pm 4.2\text{x}10^{-25}$)
MEDIUM2	$9.5\text{x}10^{-1}$ ($\pm 6.5\text{x}10^{-3}$)	$1.8\text{x}10^{-5}$ ($\pm 1.7\text{x}10^{-6}$)	$1.3\text{x}10^{-9}$ ($\pm 1.6\text{x}10^{-10}$)	$5.2\text{x}10^{-14}$ ($\pm 6.4\text{x}10^{-15}$)	$9.9\text{x}10^{-19}$ ($\pm 1.2\text{x}10^{-19}$)	$7.5\text{x}10^{-24}$ ($\pm 8.1\text{x}10^{-25}$)
DEEP8	$9.9\text{x}10^{-1}$ ($\pm 3.4\text{x}10^{-3}$)	$8.5\text{x}10^{-6}$ ($\pm 1.0\text{x}10^{-6}$)	$4.7\text{x}10^{-10}$ ($\pm 1.1\text{x}10^{-10}$)	$1.8\text{x}10^{-14}$ ($\pm 4.8\text{x}10^{-15}$)	$3.9\text{x}10^{-19}$ ($\pm 9.4\text{x}10^{-20}$)	$3.5\text{x}10^{-24}$ ($\pm 6.8\text{x}10^{-25}$)

Table B2: Slope values for non-linear grouped ramps corrected with the same RAPID mode correction coefficients.

Mode	Corrected slope (DN/s)
RAPID	$44.227 \pm 5 \times 10^{-3}$
BRIGHT1	$44.232 \pm 7 \times 10^{-3}$
SHALLOW4	$44.220 \pm 8 \times 10^{-3}$
MEDIUM2	$44.197 \pm 1 \times 10^{-2}$
DEEP8	$44.229 \pm 2 \times 10^{-2}$

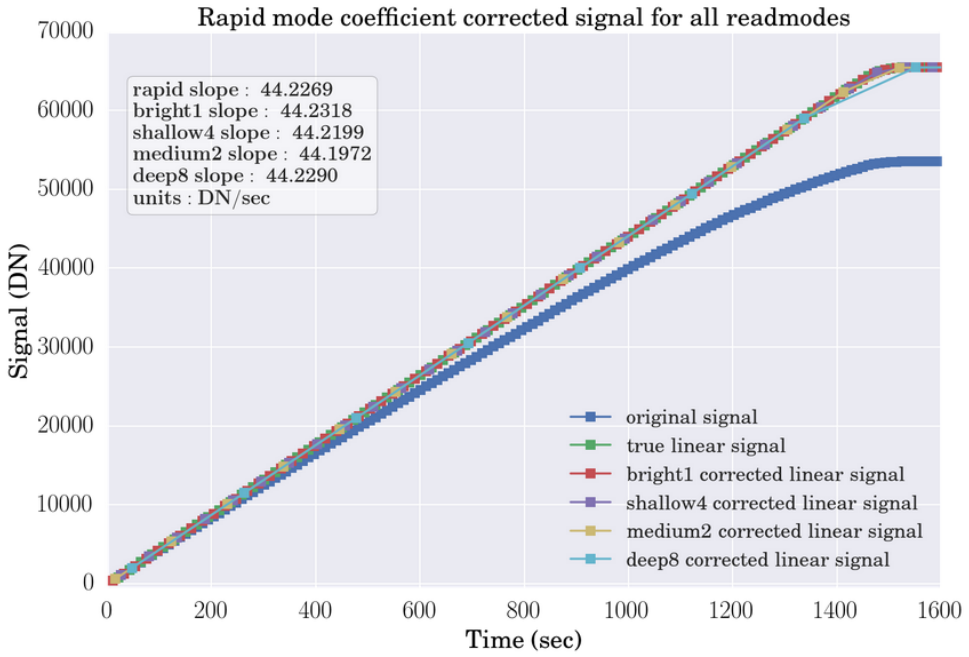
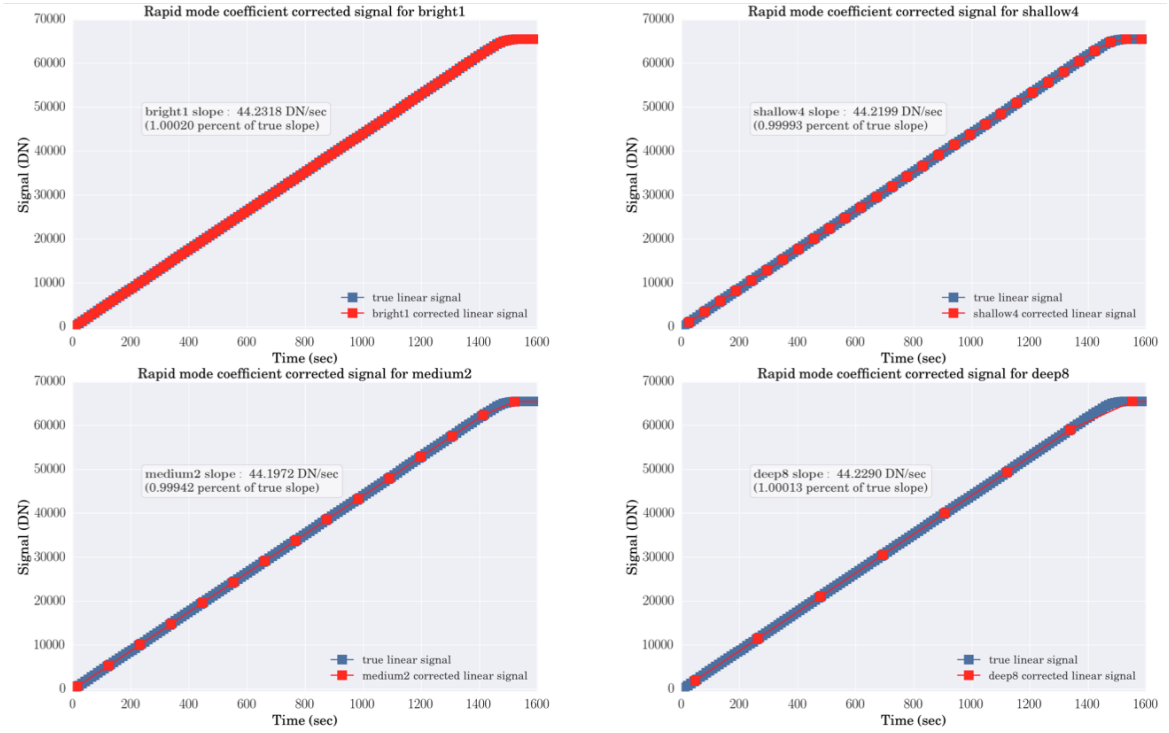


Figure B1: RAPID mode coefficient corrected ramps for different readout patterns. Slopes are within ~ 0.01% of the true slope value. Corrections are shown above hard saturation.



. **Figure B2:** Individual RAPID mode coefficient corrected ramps for each readmode.

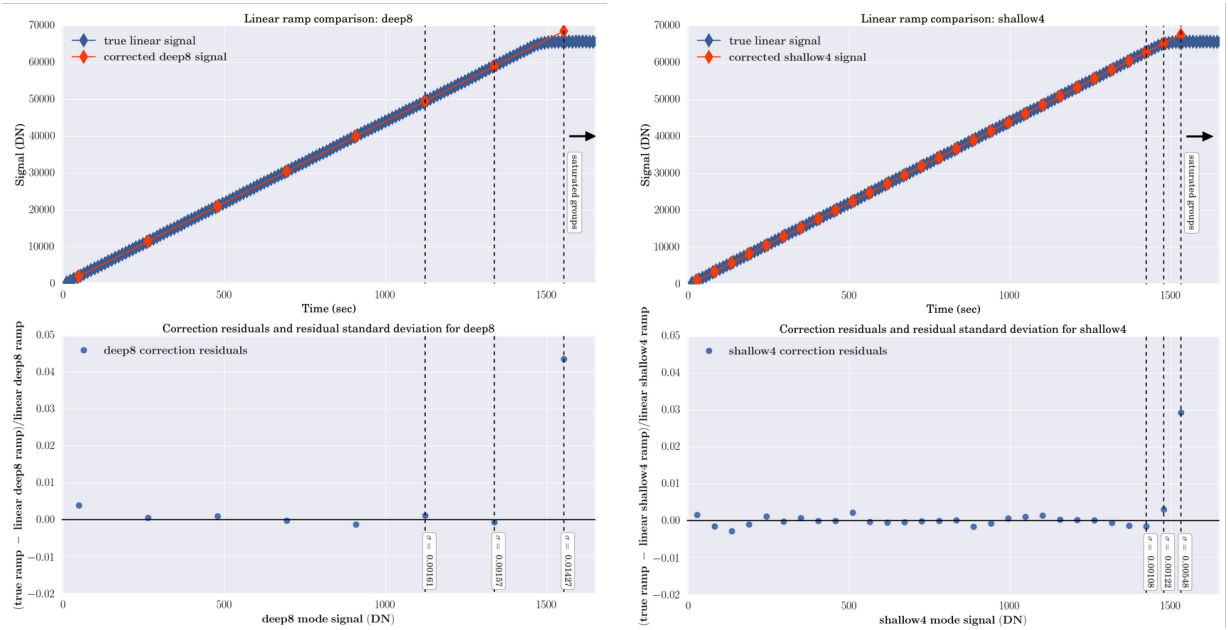


Figure B3: SHALLOW4 (left) and DEEP8 (right) ramps corrected with RAPID mode correction coefficients. Residual standard deviations are shown up the ramps above saturation.

C. Procedures to Generate Coefficients

This section contains the procedures written in Python for generating the correction coefficients to facilitate reproduction of the results presented. One module, *calib_nonlinCorrection.py*, contains the individual functions used for each step of the analysis. To simplify the code, scripts to make plots of the results from each step are excluded, as are the simple averaging functions for grouped data. Table C1 provides an explanation of the functions contained in *calib_nonlinCorrection.py* along with the sections corresponding to their use. It is first necessary to import the module, and then call the individual steps in order. An example is shown below.

```
>>> from calib_nonlinCorrection import *
>>> import calib nonlinCorrection as cal
>>> cal.get ramp("NRCN815B-LIN-5365135655_7_488_SE_2015-12-
31T18h07m58_cal.fits",750,600)

Ramp complete!
Time array shape:  (190,)
Signal array shape:  (190,)

>>> cal.def_hardSat(130)

Hard saturation signal level is: 53408.2 at group 139

>>> cal.rampFit(1,5))

Ideal rate for polynomial order 5 is:  44.0
B value for polynomial order 5 is:  351
Coefficients for order 5 are:  [ 3.51e+02 4.40e+01 -5.20e-06 1.38e-10
-1.46e-15 -1.45e-21]
Uncertainties in fit coeffs for order 5 are:  [21.2, 0.01, 6.2e-14,
1.3e-22, 4.7e-32, 2.3e-42]
Max ramp fit residual for order 5 is:  0.94 %
Standard deviation of ramp fit residuals for order 5 is:  0.00188
Chi squared for ramp fit:  4.48
The reduced chi squared for ramp fit:  0.03
Degrees of Freedom:  132

>>> cal.correctSignal(134,5)

Correction coefficients for order 5 are:  [ 9.85e-01 9.97e-06 -6.49e-10
2.70e-14 -5.74e-19 4.85e-24]
Uncertainties in coefficients are [1.84e-4, 1.26e-11, 1.04e-19, 1.70e-28,
5.74e-38, 2.68e-48]
Standard deviation for nonlinearity curve fit residuals = 0.00224
Chi squared for nonlinearity curve fit:  3.32
The reduced chi squared for nonlinearity curve fit:  0.03
Degrees of Freedom:  127
Standard deviation for corrected signal residuals:  0.00229
```

Table C1: Table listing the functions contained within the *calib_nonlinCorrection.py* module. The sections describing analysis procedures that correspond to the Python functions are included.

Function	Section	Purpose
get_ramp( )	3	Get signal ramp from input file (exposure time vs. signal value)
def_hardSat( )	4	Determine hard saturation levelCut data to only include signal values below hard saturation.
rampFit( )	4.1	Fit ramp with polynomial function to determine ideal signal rate
correctSignal( )	4.2	Fit non-linearity correction curve to determine correction coefficients Correct non-linear measured signal to get ideal signal

D. Python Code

This section contains a full copy of *calib_nonlinCorrection.py* and the functions contained within the module. The code is written in very basic Python and could certainly be improved for readability and efficiency. Once the preferred order of the polynomial has been determined, the loop over polynomial orders can be eliminated to shorten the code and run time.

```
"""

This module gets the signal ramp for the input dataset, determines the hard saturation
signal level, the ideal linear signal level, and the non-linearity correction coefficients.
It also corrects the non-linearity in the measured signal.


Example:
    To use the module, import it into python or a python notebook:

        £ ipython
        Python 3.5.1 |Continuum Analytics, Inc.| (default, Dec  7 2015, 11:24:55)
        Type "copyright", "credits" or "license" for more information.

        IPython 4.1.2 -- An enhanced Interactive Python.
```

```
        ?         -> Introduction and overview of IPython's features.
        %quickref -> Quick reference.
        help      -> Python's own help system.
        object?   -> Details about 'object', use 'object??' for extra details.

        In [1]: from calib_nonlinCorrection.py import *
        In [2]: import calib_nonlinCorrection as cal

        In [3]: # After importing the module, call the
        In [4]: # functions individually to complete each step, e.g. for get_ramp():

        In [5]: get_ramp("input_file.fits",750,600)


"""

#!/usr/bin/env python
from astropy.io import fits
import sys
import os
import string
import numpy as np
from scipy.optimize import curve_fit

def get_ramp(file,pixX,pixY):
    """Get signal ramp from input file and import necessary modules.

    Args:
        file: "file name" (string) **include path if not in current directory**
        pixX: 1st coordinate of pixel (integer)
        pixY: 2nd coordinate of pixel (integer)
    Returns:
        pixel: array with pixel coordinates
        tgroup: time between reads (float)
        new_cube: data cube (1, 190, 2048, 2048), 1 integration, 190 reads, 2048x2048 pix
        time_x: exposure time array (independent variable, sec)
        dn_y: signal value array (dependent variable, DN)
    Raises:
        FileNotFoundError: if no such file or directory
    Example:
        >>> get_ramp("NRCN815B-LIN-5365135655_7_488_SE_2015-12-31T18h07m58.fits",750,600)
        >>> print('Exposure time values:\n',time_x)
        Exposure time values:
         [  0.       10.73676  21.47352  32.21028  42.94704 ... ]
        >>> print('Signal values:\n',dn_y)
        Signal values:
```

```
        [ 407.17565918  827.328125  1287.11779785  1742.34204102  2211.53076172 ...]
    >>> print('Pixel is:\n',pixel)
    Pixel is:
     [750, 600]
    >>> print('Time between reads is:\n',tgroup)
    Time between reads is:
     10.73676

    """

    # Path and file name
    fileName = file


    # Get data and headers
    data = fits.getdata(fileName, 1)
    hdr = fits.getheader(fileName, 0)


    # Make output available for other function calls
    global pixel, tgroup, new_cube, x, y, time_x, dn_y


    # Get pixel and ramp data
    pixel = [pixX,pixY]                          # pixel coordinates
    tgroup = hdr['TGROUP']                       # time between groups (sec)
    int_data = data.shape[0]                     # 1 integration
    zdim=data.shape[1]                           # 190 groups
    ydim=data.shape[2]                           # 2048 pix (4 ref pix, top and bottom)
    xdim=data.shape[3]                           # 2048 pix (4 ref pix, left and right)

    new_cube = data[:,:,:,:]                     # data cube
    x = np.arange(1,191,1)                       # groups on x-axis
    y = new_cube[0,:,pixel[1],pixel[0]]          # counts on y-axis
    time_x = x * tgroup                          # time on x-axis (GROUPS * TGROUP)
    dn_y = new_cube[0,:,pixel[1],pixel[0]]       # DN on y-axis (be careful with units)
    print('\n\nRamp complete!\nTime array shape:',np.shape(time_x),\
          '\nSignal array shape:', np.shape(dn_y),'\n\n')


def def_hardSat(flat_grp):
    """Determine hard saturation level and cut data to only include signal values below
       hard saturation.

    Args:
        flat_grp: estimate for read where signal begins to flatten (integer)
```

```
    Returns:
        hard_sat: saturation signal level (float)
        first_sat_grp: 1st saturated group (integer)
        time_below_hardSat: time data array excluding saturation
        y_below_hardSat: signal data array excluding saturation
    Example:
        >>> def_hardSat(130)
        Hard saturation signal level is: 53408.2 at group 139

    """

    # Make output available for other function calls
    global hard_sat, first_sat_grp, time_below_hardSat, x_below_hardSat, y_below_hardSat

    # Get saturation signal level, hard_sat, by taking avg of saturated read signal values
    # saturated reads have standard deviation of signal differences = (0 +/- 3sigma)
    diff_array = []
    diff = [(dn_y[i+1]-dn_y[i]) for i in np.arange(0,(len(x)-1))] # signal differences
    sigma_diff = np.std(diff[20:60])                             # sigma(differences)
    diff_array = np.asarray(diff)
    saturation = np.where(((0-3*sigma_diff) < diff_array) & (diff_array < (0+3*sigma_TPD)))
    saturation_x = x[np.min(saturation)-1:-2]                    # saturated reads

    hard_sat = np.average(dn_y[np.min(saturation):])             # saturation level
    first_sat_grp = np.min(saturation)
    x_below_hardSat = x[:first_sat_grp]                          # x data < saturation
    time_below_hardSat = x_below_hardSat*tgroup
    y_below_hardSat = dn_y[:first_sat_grp]                       # y data < saturation


def rampFit(low_lim,order):
    """Fit the ramp with a polynomial function to determine the ideal signal rate.

    Args:
        low_lim: first read included in ramp fit
        order: polynomial order for ramp fitting
    Returns:
        y_ideal: ideal linear signal array
        y_ideal_unc: ideal linear signal uncertainties array
        nonlinearity: fractional non-linearity array
        nonlinearity_unc: fractional non-linearity uncertainty array
        residual_std: ramp fit residual standard deviation
        reduced_chi_squared: ramp fit reduced chi-squared
    Raises:
        UnboundLocalError: no function is defined for order outside of [1 < order < 5]
    Example:
```

```
    >>> rampFit(1,5)
    Ideal rate for polynomial order 5 is: 44.0038803781
    B value for polynomial order 5 is: 351.35960598
    Coefficients for order 5 are: [3.51359606e+02  4.40038804e+01 -5.20172387e-06
            1.37697717e-10  -1.46326014e-15  -1.44652078e-21]
    Uncertainties in fit coeffs for order 5 are:  [21.240676879450881,
            0.0099782012221226648, 6.2225955656646852e-14, 1.298832233435027e-22,
            4.731088017362644 9e-32, 2.2911600805689125e-42]
    Max ramp fit residual for order 5 is: 0.94 %
    Standard deviation of ramp fit residuals for order 5 is: 0.00188191356549
    Chi squared for ramp fit:  4.48
    The reduced chi squared for ramp fit:  0.03
    Degrees of Freedom:         132
    R-squared value for ramp fit is: 0.999996720855

"""


# Define our new polynomial with the order as a tunable parameter
# Here, 'b' will be constant signal rate for perfect detector
# 'c', 'd', 'e', etc. describe magnitudes of deviations from ideal signals

# Make output available for other function calls
global new_x, new_y

# Define values for ramp fitting
new_low_limit = low_lim                          # set lower limit for fit
x_cut = x_below_hardSat[new_low_limit:]          # truncate data at limit
time_x_cut = x_cut*tgroup                        # time data array above low limit
dn_y_cut = y_below_hardSat[x_cut]                # signal array above low limit
new_x = time_x_cut                               # renaming variables to simplify code
new_y = dn_y_cut
range_min = "%.0f" % np.min(x_cut)
range_max = "%.0f" % np.max(x_cut)


# Choose ramp fit hybrid polynomial order
orders = [order]                                 # can input multiple orders
                                                 # **if multiple orders**
                                                 # **change nonlin_correction.py**

for order in orders:

    # define polynomial function of the ideal linear rate 'b' for various polynomial
    # orders provide initial coefficient guess values in 'init_vals' array
    if order == 5:
```

```
    def func(t,a,b,c,d,e,f):
        return a + b * t + c * (b*t)**2 + d * (b*t)**3 + e * (b*t)**4 +
        f * (b*t)**5
    init_vals = [7.71249127e+02,3.82479646e+01,7.26905319e-03,-6.00091033e-06,\
    3.05167750e-08,-8.25686297e-11]

elif order == 4:

    def func(t,a,b,c,d,e): #this works with curve_fit
        return a + b * t + c * (b*t)**2 + d * (b*t)**3 + e * (b*t)**4
    init_vals = [7.71249127e+02,3.82479646e+01,7.26905319e-03,-6.00091033e-06,\
    3.05167750e-08]

elif order == 3:

    def func(t,a,b,c,d):
        return a + b * t + c * (b*t)**2 + d * (b*t)**3
    init_vals = [7.71249127e+02,3.82479646e+01,7.26905319e-03,-6.00091033e-06]

elif order == 2:

    def func(t,a,b,c):
        return a + b * t + c * (b*t)**2
    init_vals = [7.71249127e+02,3.82479646e+01,7.26905319e-03]

elif order == 1:

    def func(t,a,b,c):
        return a + b * t
    init_vals = [7.71249127e+02,3.82479646e+01]


# Provide uncertainties in the signal
unc = np.sqrt(8.**2 + new_y)

# Make output available for other function calls
global y_ideal, y_ideal_unc, nonlinearity, nonlinearity_unc, residual_std,\
        reduced_chi_squared
global new_ideal_rate, new_Bval


# Fit data and uncertainties with scipy's built in curve_fit function
# beta will be coefficients for hybrid polynomial, so beta[1] is our "ideal
# ramp rate" and b[0] is the constant value
# pcov contains uncertainty in the coefficients
```

```
beta, pcov = curve_fit(func, new_x, new_y,sigma=unc,maxfev=10000)
signal_fit_unc = [np.sqrt(pcov[j,j]) for j in range(beta.size)]
#print(pcov)

print('\n\nIdeal rate for polynomial order',order,'is:',beta[1])
print('B value for polynomial order',order,'is:',beta[0])
print('Coefficients for order',order,'are:', beta)
print('Uncertainties in fit coeffs for order',order,'are: ',signal_fit_unc)
new_ideal_rate = beta[1]
new_Bval = beta[0]
signal_fit_cov = pcov[1,0]


# Get ideal linear signal and signal uncertainties using beta[1], beta[0]
# coefficients from above
y_ideal = new_Bval + new_ideal_rate * new_x           # y = B + A*t
y_ideal_unc = np.sqrt(signal_fit_unc[0]**2 + (signal_fit_unc[1]*new_x)**2 +
        2.*signal_fit_cov**2)



# Get fractional nonlinearity to produce the correction curve for the
# second polynomial fit
nonlinearity = (y_ideal/new_y)                    # curve is nonlinearity vs. signal
nonlinearity_unc=np.sqrt(nonlinearity**2*(unc**2/new_y**2+ \
        y_ideal_unc**2/y_ideal**2))
residuals = (new_y - func(new_x,*beta))/new_y
residual_std = np.std((new_y-func(new_x,*beta))/new_y)
max_resid = np.max(residuals)
min_resid = np.min(residuals)
print('Max ramp fit residual for order',order,'is:', \
        np.round(np.max(np.absolute(residuals))*100,decimals=2),'%')
print('Standard deviation of ramp fit residuals for order',order, \
        'is:',np.std((new_y-func(new_x,*beta))/new_y))



# Get fit statistics
ndof = len(new_x) - len(beta)
chi_squared = np.sum(((func(new_x, *beta) - new_y)/unc) ** 2)
reduced_chi_squared = chi_squared / (len(new_x) - len(beta))
print('Chi squared for ramp fit: ', ("%.2f" % chi_squared))
print('The reduced chi squared for ramp fit: ', ("%.2f" % reduced_chi_squared))
print('Degrees of Freedom:\t', ndof,'\n\n')
```

```
def correctSignal(uppLim,nonlin_order):
    """Fit non-linearity correction curve to determine correction coefficients and
       correct the non-linear measured signal.

    Args:
        uppLim: maximum read for correction curve fit (find soft saturation limit)
        nonlin_order: polynomial order for curve fitting
    Returns:
        beta: array of correction coefficients
        curve_fit_resid_std: correction curve fit residual standard deviation
        nonlin_reduced_chi_squared: curve fit reduced chi-squared
        corrected_signal: array containing signal values corrected for non-linearity
        correction_resid_std: correction residual standard deviation
    Raises:
        UnboundLocalError: no function is defined for order outside of [1 < order < 5]
    Example:
        >>> correctSignal(134,5)
        Correction coefficients for order 5 are: [  9.84726510e-01   9.97236111e-06
                -6.48717477e-10   2.69665883e-14  -5.73891148e-19   4.85122446e-24]
        Uncertainties in coefficients are  [0.00018386390647398069, 1.2591386529098918e-11,
                1.0414993114807574e-19, 1.6965576533744201e-28, 5.739182210801211e-38,
            2.6846301157810076e-48]
        Standard deviation for nonlinearity curve fit residuals =  0.00224170020518
        Chi squared for nonlinearity curve fit:  3.32
        The reduced chi squared for nonlinearity curve fit:  0.03
        Degrees of Freedom:         127
        Standard deviation for corrected signal residuals:         0.00228906969392

    """

    # Set upper limit for linearity curve data range
    # ** Have option to set multiple upper limits to study change in correction **
    test_resid_lims = [uppLim]                                  # upper limit for fit

    for test_resid in test_resid_lims:


        # Cut data for new limit
        new_fitLim = np.where(new_x < test_resid*tgroup)
        test_x = new_x[new_fitLim]                              # time, new upper limit
        test_y = new_y[new_fitLim]                              # signal, new upper limit
        test_nonlinearity = nonlinearity[new_fitLim]            # non-linearity, new limit
        test_nonlinearity_unc = nonlinearity_unc[new_fitLim] # non-linearity uncertainty
        test_y_ideal = y_ideal[new_fitLim]                      # ideal signal new limit
```

```
test_range_max = np.max(test_x)/tgroup
test_range_min = np.min(test_x)/tgroup

# Make output available for function calls
global beta, curve_fit_resid, corrected_signal, correction_resid_std

# Choose order for non-linearity correction curve fit
nonlin_fit_orders = [nonlin_order]                    # order is tunable parameter

for nonlin_order in nonlin_fit_orders:

    # define polynomial function for various polynomial orders
    # provide initial coefficient guess values in 'nonlin_init' array
    if nonlin_order == 5:

        def poly_fit_func(t, aa,bb,cc,dd,ee,ff):
            return aa + bb*t + cc*t**2 + dd*t**3 + ee*t**4 + ff*t**5
        nonlin_init = np.array([1,3e-6,-1e-10,2e-15,1e-18,1e-21])


        # change data format for scipy's curve_fit function
        test_y = np.asfarray(test_y,dtype='float')
        test_nonlinearity = np.asfarray(test_nonlinearity,dtype='float')
        test_nonlinearity_unc = np.asfarray(test_nonlinearity_unc,dtype='float')


        # beta contains correction coefficients, pcov contains uncertainties in
        # coefficients
        beta, pcov = curve_fit(poly_fit_func, test_y,test_nonlinearity, \
                sigma=test_nonlinearity_unc,p0=nonlin_init,absolute_sigma=True, \
            maxfev=10000)
        poly_fit_unc = [np.sqrt(pcov[j,j]) for j in range(beta.size)]
        beta = np.asfarray(beta,dtype='float')

        print('\n\nCorrection coefficients for order',nonlin_order,'are:', beta)
        print('Uncertainties in coefficients are ',poly_fit_unc)


        # correct signal using coefficients
        corrected_signal = test_y * (beta[0] + beta[1] * test_y +
                beta[2] * test_y**2 + beta[3] * test_y**3 + beta[4] * test_y**4 +
            beta[5] * test_y**5)


        # output coefficients into a text file
        #np.savetxt('lincorr_ref_file_nonlinOrder4.txt',beta)
```

```
elif nonlin_order == 4:

    def poly_fit_func(t, aa,bb,cc,dd,ee):
      return aa + bb*t + cc*t**2 + dd*t**3 + ee*t**4
    nonlin_init = np.array([1.,3e-6,-1e-10,2e-15,1e-18])


    # change data format for scipy's curve_fit function
    test_y = np.asfarray(test_y,dtype='float')
    test_nonlinearity = np.asfarray(test_nonlinearity,dtype='float')
    test_nonlinearity_unc = np.asfarray(test_nonlinearity_unc,dtype='float')


    # beta contains correction coefficients, pcov contains uncertainties in
    # coefficients
    beta, pcov = curve_fit(poly_fit_func, test_y,test_nonlinearity, \
            sigma=test_nonlinearity_unc,p0=nonlin_init,absolute_sigma=True, \
        maxfev=10000)
    poly_fit_unc = [np.sqrt(pcov[j,j]) for j in range(beta.size)]
    beta = np.asfarray(beta,dtype='float')

    print('\n\nCorrection coefficients for order',nonlin_order,'are:', beta)
    print('Uncertainties in coefficients are ',poly_fit_unc)


    # correct signal using coefficients
    corrected_signal = test_y * (beta[0] + beta[1] * test_y + \
            beta[2] * test_y**2 + beta[3] * test_y**3 + beta[4] * test_y**4)


    # output coefficients into a text file
    #np.savetxt('lincorr_ref_file_nonlinOrder4.txt',beta)

elif nonlin_order == 3:

    def poly_fit_func(t, aa,bb,cc,dd):
      return aa + bb*t + cc*t**2 + dd*t**3
    nonlin_init = np.array([1.,3e-6,-1e-10,2e-15])


    # change data format for scipy's curve_fit function
    test_y = np.asfarray(test_y,dtype='float')
    test_nonlinearity = np.asfarray(test_nonlinearity,dtype='float')
    test_nonlinearity_unc = np.asfarray(test_nonlinearity_unc,dtype='float')
```

```
    # beta contains correction coefficients, pcov contains uncertainties in
    # coefficients
    beta, pcov = curve_fit(poly_fit_func, test_y,test_nonlinearity, \
            sigma=test_nonlinearity_unc,p0=nonlin_init,absolute_sigma=True, \
        maxfev=10000)
    poly_fit_unc = [np.sqrt(pcov[j,j]) for j in range(beta.size)]
    beta = np.asfarray(beta,dtype='float')

    print('\n\nCorrection coefficients for order',nonlin_order,'are:', beta)
    print('Uncertainties in coefficients are ',poly_fit_unc)


    # correct signal using coefficients
    corrected_signal = test_y * (beta[0] + beta[1] * test_y + \
            beta[2] * test_y**2 + beta[3] * test_y**3)


    # output coefficients into a text file
    #np.savetxt('lincorr_ref_file_nonlinOrder4.txt',beta)

elif nonlin_order == 2:

    def poly_fit_func(t, aa,bb,cc):
      return aa + bb*t + cc*t**2
    nonlin_init = np.array([1.,3e-6,-1e-10])


    # change data format for scipy's curve_fit function
    test_y = np.asfarray(test_y,dtype='float')
    test_nonlinearity = np.asfarray(test_nonlinearity,dtype='float')
    test_nonlinearity_unc = np.asfarray(test_nonlinearity_unc,dtype='float')


    # beta contains correction coefficients, pcov contains uncertainties in
    # coefficients
    beta, pcov = curve_fit(poly_fit_func, test_y,test_nonlinearity, \
            sigma=test_nonlinearity_unc,p0=nonlin_init,absolute_sigma=True, \
        maxfev=10000)
    poly_fit_unc = [np.sqrt(pcov[j,j]) for j in range(beta.size)]
    beta = np.asfarray(beta,dtype='float')

    print('\n\nCorrection coefficients for order',nonlin_order,'are:', beta)
    print('Uncertainties in coefficients are ',poly_fit_unc)
```

```
    # correct signal using coefficients
    corrected_signal = test_y * (beta[0] + beta[1] * test_y + \
            beta[2] * test_y**2)


    # output coefficients into a text file
    #np.savetxt('lincorr_ref_file_nonlinOrder4.txt',beta)


#print('Max nonlinearity curve fit residual for order',nonlin_order, \
# 'is =',np.round(np.max(np.absolute((test_nonlinearity- \
# poly_fit_func(test_y,*beta))/test_nonlinearity))*100,decimals=2),'%')
print('Standard deviation for nonlinearity curve fit residuals = ',
  np.std((test_nonlinearity-poly_fit_func(test_y,*beta))/test_nonlinearity))


# Get fit statistics
ndof = len(test_y) - len(beta)
chi_squared = np.sum(((poly_fit_func(test_y, *beta) - \
        test_nonlinearity)/test_nonlinearity_unc) ** 2)
nonlin_reduced_chi_squared = chi_squared / (len(test_y) - len(beta))
print('Chi squared for nonlinearity curve fit: ', ("%.2f" % chi_squared))
print('The reduced chi squared for nonlinearity curve fit: ', \
        ("%.2f" % nonlin_reduced_chi_squared))
print('Degrees of Freedom:\t', ndof)

# Correction accuracy
correction_resid_std = np.std((corrected_signal-(new_Bval + \
        new_ideal_rate * test_x))/test_y)
print('Standard deviation for corrected signal residuals:\t', \
        correction_resid_std,'\n\n')
```