HxRGproc

Non-linearity modelling in HPF’s H2RG detector

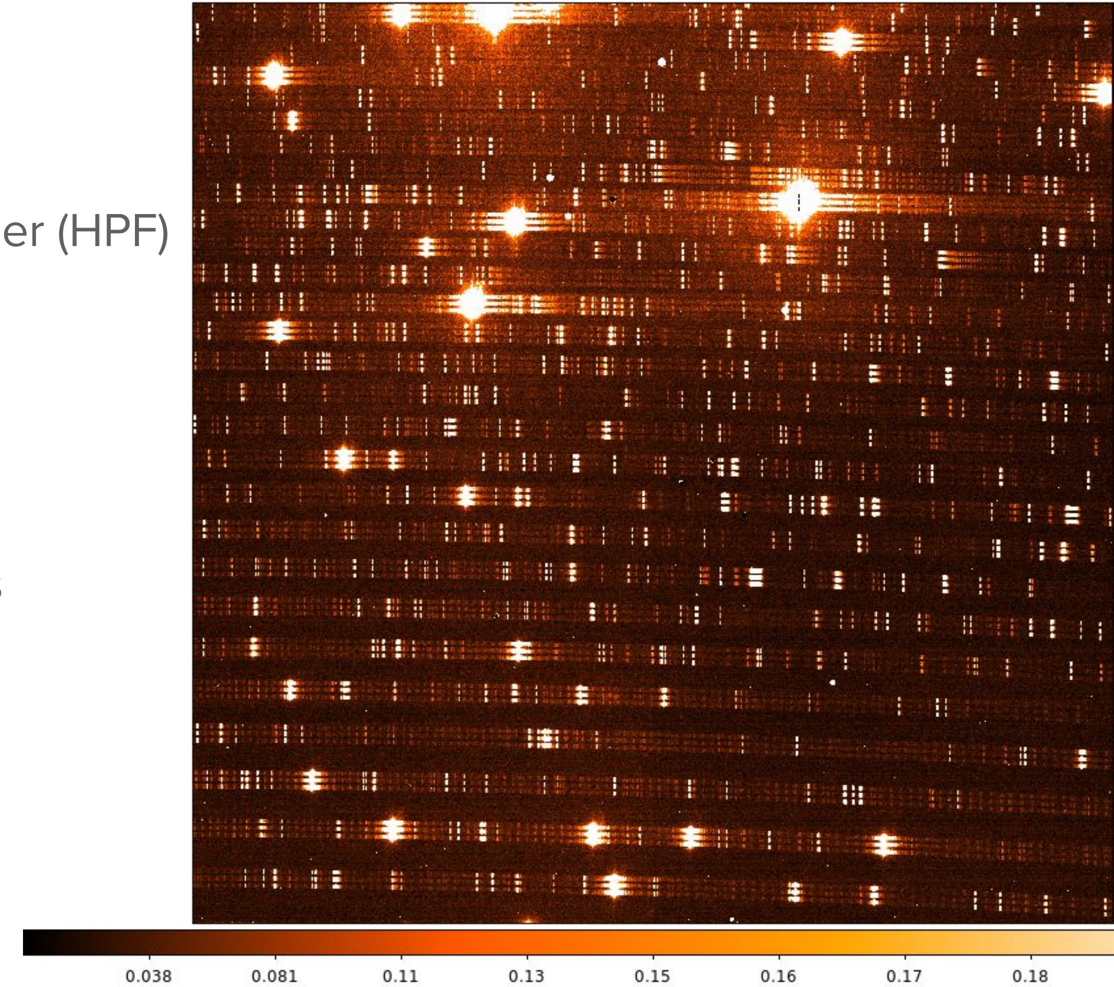
Joe Ninan & The HPF Team

(The Pennsylvania State University, USA)

Detector Modelling Workshop (DeMo) 2021

Overview

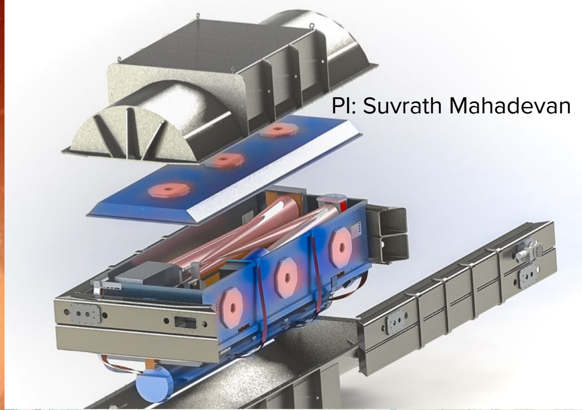
- The Habitable zone Planet Finder (HPF)
- HxRGproc
  - Capabilities
- Non-linearity in HxRG detectors
  - Formalism
  - Scalable and robust models

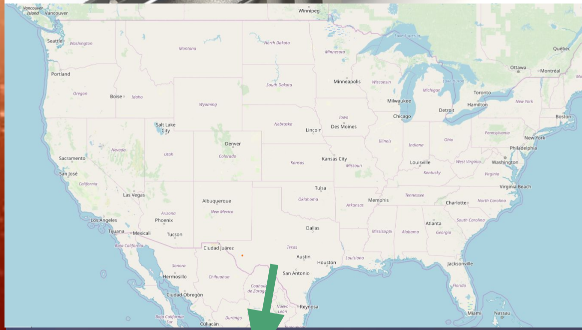


The Habitable-Zone Planet Finder (HPF)

Extreme precision radial velocity measurement spectrograph

- HPF wavelength coverage: **0.8 to 1.27 microns**
- Resolution = 55,000
- Located at 10m Hobby Eberly Telescope, McDonald Observatory, Texas, USA
- HPF uses a **1.7 µm cutoff H2RG** (Hawaii-2RG HgCdTe 2048x2048)





HxRGproc

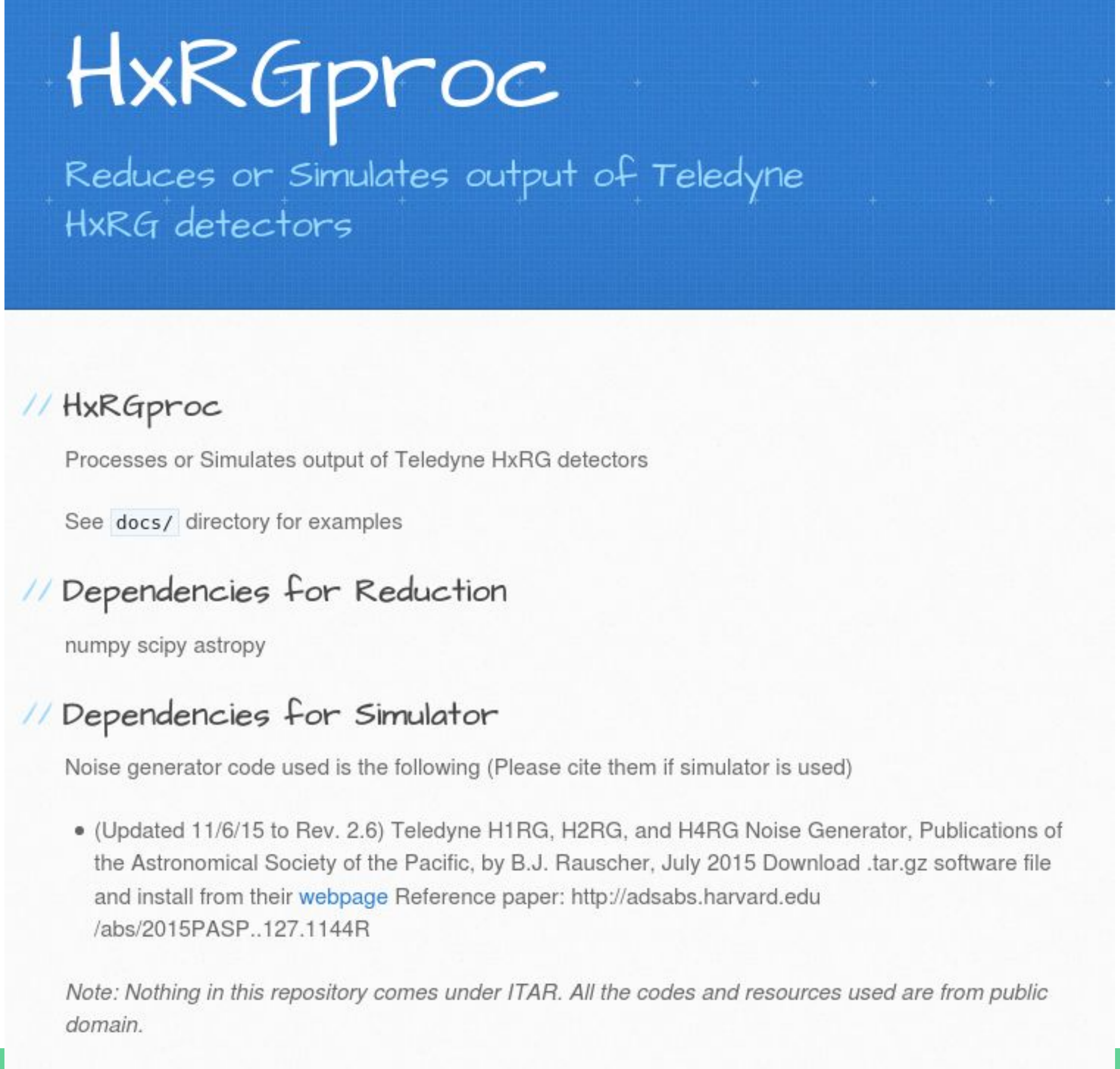
https://indiajoe.github.io/HxRGproc/

Python 2.7+ and 3.6+ support

Contains two sub-modules

Simulation

Reduction



HxRGproc

Reduces or Simulates output of Teledyne HxRG detectors

HxRGproc

Processes or Simulates output of Teledyne HxRG detectors

See `docs/` directory for examples

Dependencies for Reduction

numpy scipy astropy

Dependencies for Simulator

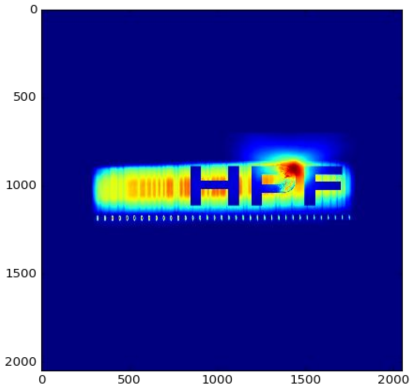
Noise generator code used is the following (Please cite them if simulator is used)

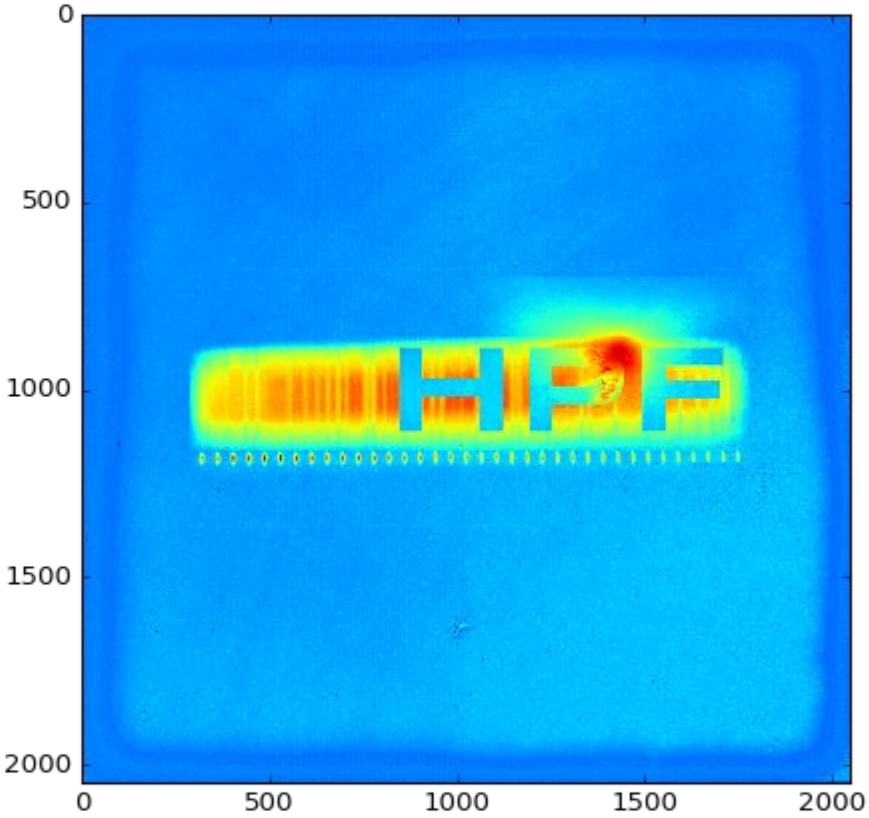
- (Updated 11/6/15 to Rev. 2.6) Teledyne H1RG, H2RG, and H4RG Noise Generator, Publications of the Astronomical Society of the Pacific, by B.J. Rauscher, July 2015 Download .tar.gz software file and install from their webpage Reference paper: http://adsabs.harvard.edu /abs/2015PASP..127.1144R

Note: Nothing in this repository comes under ITAR. All the codes and resources used are from public domain.

Simulation of raw data

HxRGproc uses the
Noise generator code developed by
B.J. Rauscher, et. al. 2015
for adding bias noise into simulated
up-the-ramp data.





Reduction **(3D up-the-ramp → 2D image)**

Modular enough to support multiple instruments which write up-the-ramp data in different file formats.

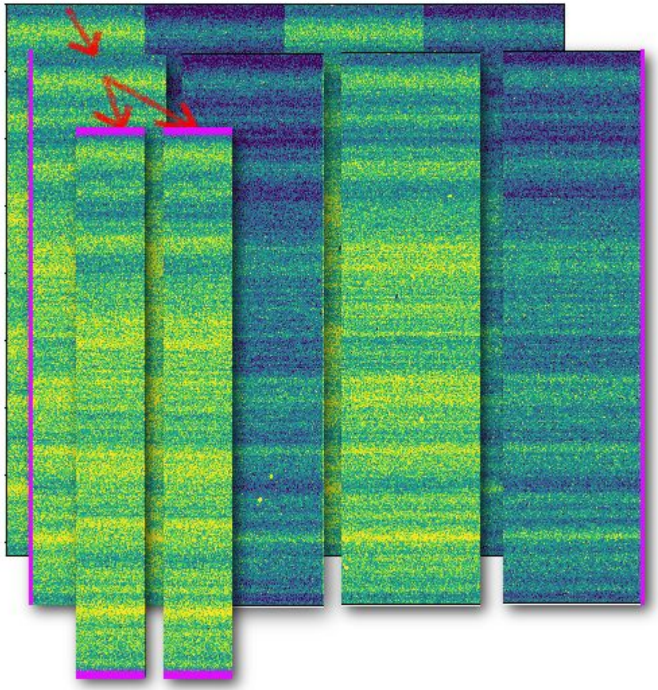
Hooks to update headers or other instrument specific pre-processing of data-cube.

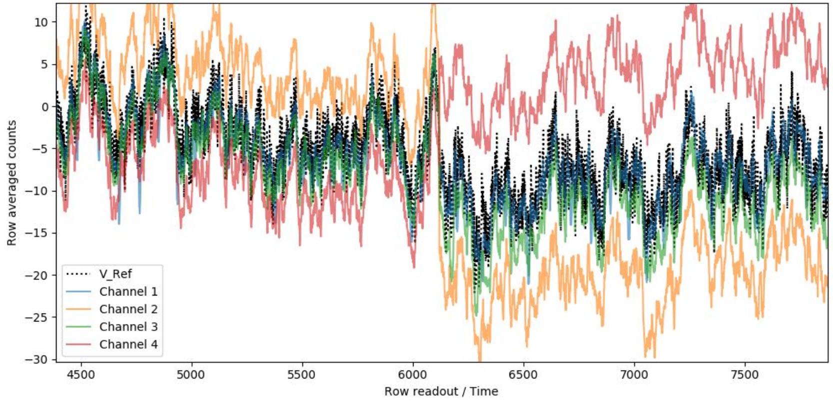
```
SupportedReadOutSoftware_for_slope = {
    'TeledyneWindows':{'RampFilenameString' : 'H2RG_R{0}_M', #Input filename structure with Ramp id substitution
                       'RampidRegexp' : 'H2RG_R(.+?)_M', # Regexp to extract unique Ramp id from filename
                       'HDR_NOUTPUTS' : 'NOUTPUTS', # Fits header for number of output channels
                       'HDR_INTTIME' : 'INTTIME', # Fits header for accumulated exposure time in each NDR
                       'filename_sort_func' : sort_filename_key_function_Teledyne,
                       'FixHeader_func': lambda hdr: hdr, # Optional function call to fix input raw header
                       'FixDataCube_func': lambda Dcube: Dcube, # Optional function call to fix input Data Cube
                       'estimate_NoNDR_Drop_G_func' : estimate_NoNDR_Drops_G_Teledyne,
                       'ExtraHeaderCalculations_func' : extra_header_calculations_Teledyne},

    'HPFLinux':{'RampFilenameString' : 'hpf_{0}_F', #Input filename structure with Ramp id substitution
                'RampidRegexp' : 'hpf_(.*_R\d*?)_F.*fits', # Regexp to extract unique Ramp id from filename
                'HDR_NOUTPUTS' : 'CHANNELS', # Fits header for number of output channels
                'HDR_INTTIME' : 'ITIME', # Fits header for accumulated exposure time in each NDR
                'filename_sort_func': sort_filename_key_function_HPFLinux,
                'FixHeader_func': fix_header_function_HPFLinux, # Optional function call to fix input raw header
                'FixDataCube_func': fix_datacube_function_HPFLinux, # Optional function call to fix input Data Cube
                'estimate_NoNDR_Drop_G_func':None,
                'ExtraHeaderCalculations_func':None},
    'HPFMACIE':{'RampFilenameString' : 'hpf_{0}_F', #Input filename structure with Ramp id substitution
                'RampidRegexp' : 'hpf_(.*_R\d*?)_F.*fits', # Regexp to extract unique Ramp id from filename
                'HDR_NOUTPUTS' : 'CHANNELS', # Fits header for number of output channels
                'HDR_INTTIME' : 'ITIME', # Fits header for accumulated exposure time in each NDR
                'filename_sort_func': sort_filename_key_function_HPFLinux,
                'FixHeader_func': lambda hdr: hdr, # Optional function call to fix input raw header
                'FixDataCube_func': lambda Dcube: Dcube, # Optional function call to fix input Data Cube
                'estimate_NoNDR_Drop_G_func':None,
                'ExtraHeaderCalculations_func':None},

}
```

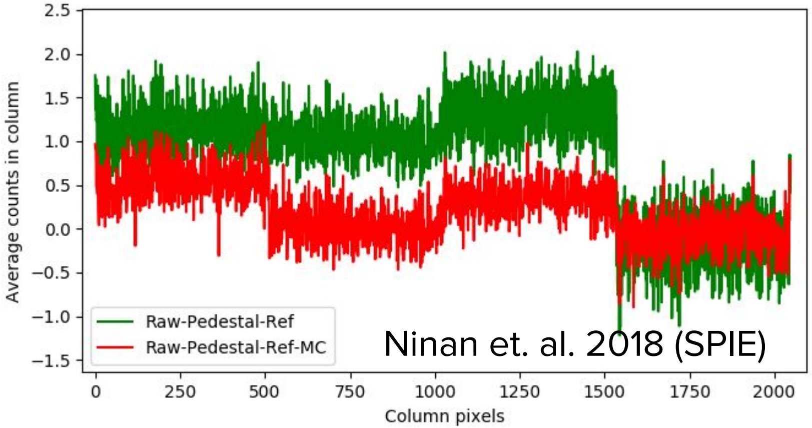
HxRGproc/reduction/instruments.py

Steps in HxRGproc before slope fitting

Advanced bias fluctuation correction



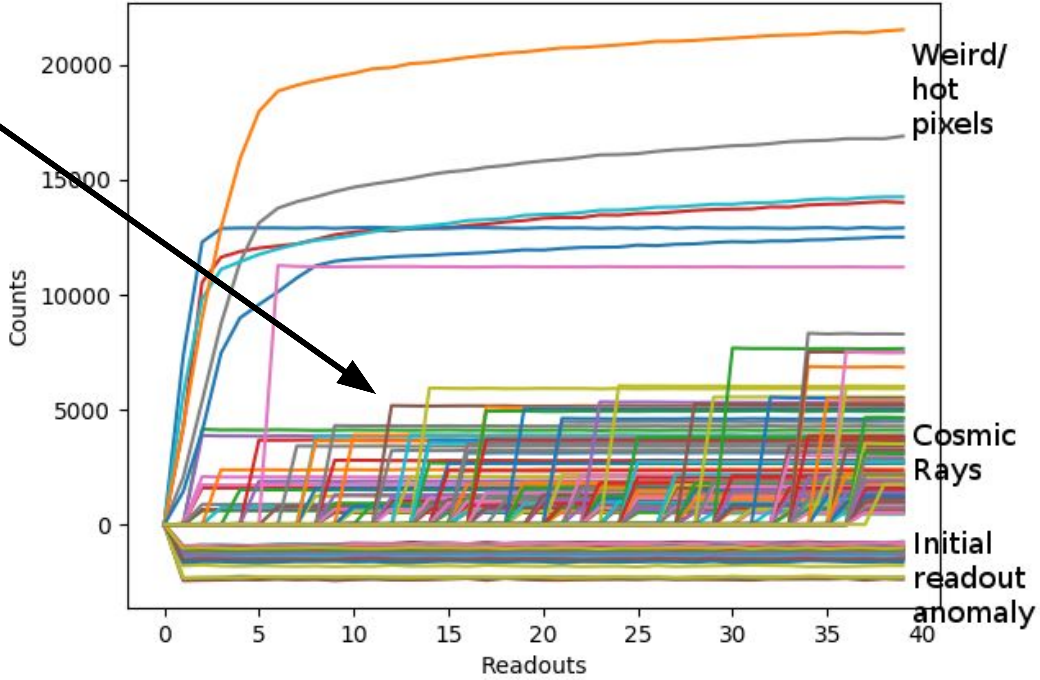




Ninan et. al. 2018 (SPIE)

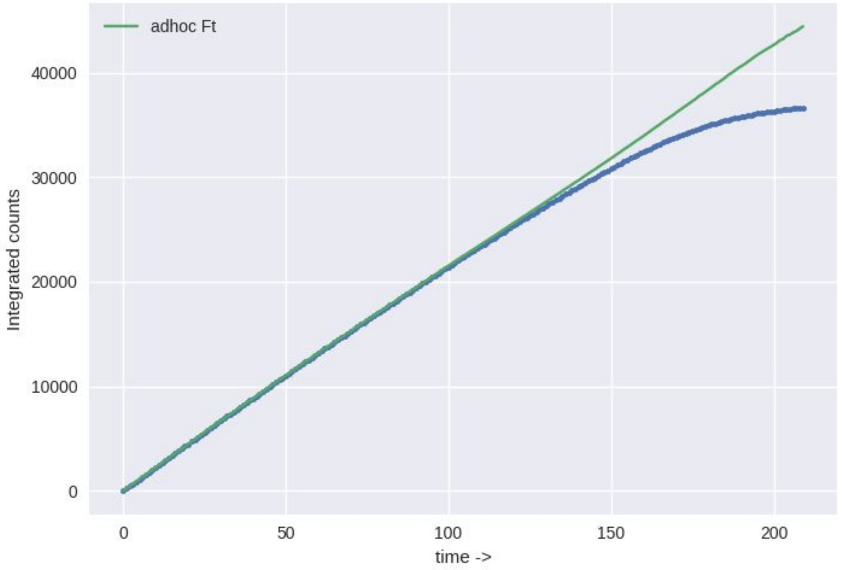
Steps in HxRGproc before slope fitting

- Cosmic Ray pixel recovery
- Saturated pixel recovery
- Non-linearity correction at pixel level



Non-linearity in H2RG detectors

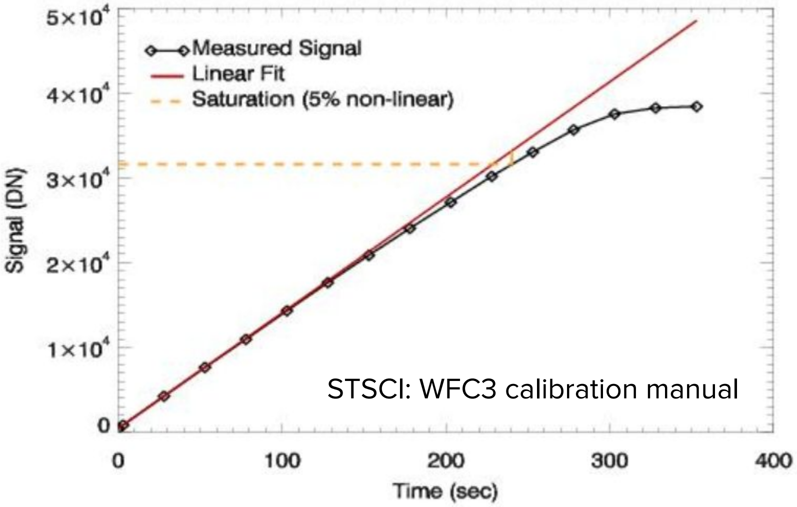
- Measured counts $\cancel{\propto}$ absorbed photons
- Unlike CCD non-linearity, it is different for each pixel!
    - Charge dependent capacitance of the p-n junctions in each pixel
    - Other amplifiers downstream
- Need 4.2 million independent functions to calibrate the measured counts to real counts!



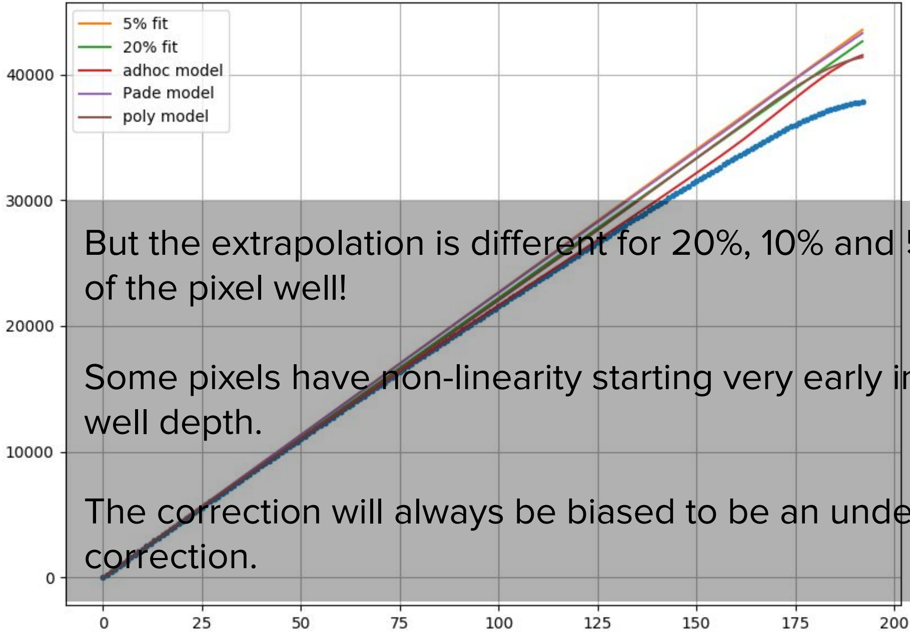
Issue with lower well signal extrapolation

Conventional technique: Extrapolate the signal in lower well to higher well

Figure 6.6: Measured non-linear response of an IR pixel.



STSCI: WFC3 calibration manual



But the extrapolation is different for 20%, 10% and 5% of the pixel well!

Some pixels have non-linearity starting very early in well depth.

The correction will always be biased to be an under correction.

Constrained model for Counts ➡ True Counts

- Motivated us to explore ways to model the non-linearity curve completely

- 4.2 million unique pixels imposes the requirement:
    - Should be flexible enough to all shapes of non-linearity across the pixels.
    - Should be very robust to fitting the data.
        - => Physically motivated constraints to the model are necessary.

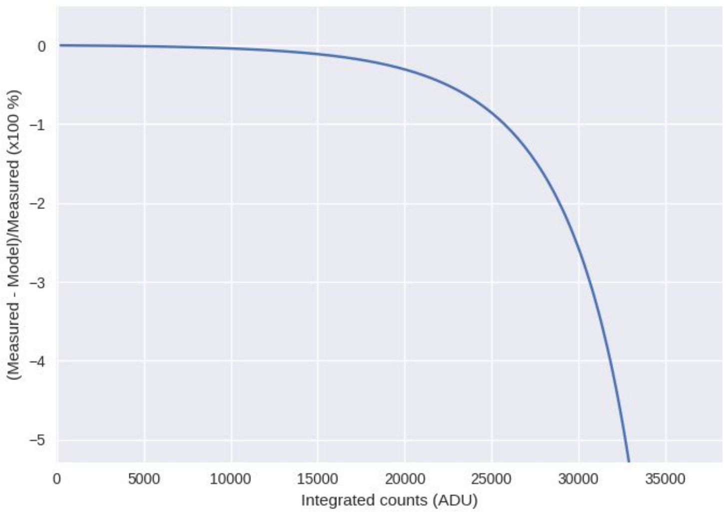
We want a function which maps the measured counts to true counts

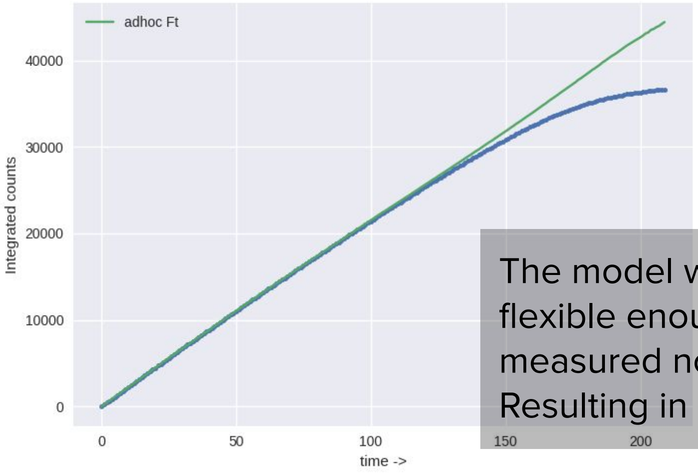
Biesiadzinski et. al. Model

Biesiadzinski + 2011

$$-a\left(\frac{b\log\left((a+1)^{\frac{S}{b}}-a-1\right)}{(a+1)\log(a+1)}-\frac{b\log\left((a+1)^{\frac{S}{b}}\right)}{(a+1)\log(a+1)}\right)$$

Three parameter model to describe change in junction capacitance in pixels as a function of the integrated signal.





The model was however not flexible enough to fit the measured non-linearity curve. Resulting in poor goodness of fit.

Inverse model in derivative space

Motivation: The signal response of the pixel weakens with the collected counts.

i.e., *the derivative of the correction function should be a monotonic function.*

Let F_o be a constant flux source, and c be the measured count at time t .

If $\epsilon(c)$ is the response at counts c . Then $\frac{dc}{dt} = F_o \epsilon(c)$

We can constrain $\epsilon(c)$ to be monotonic, and slope to be zero when c=0.

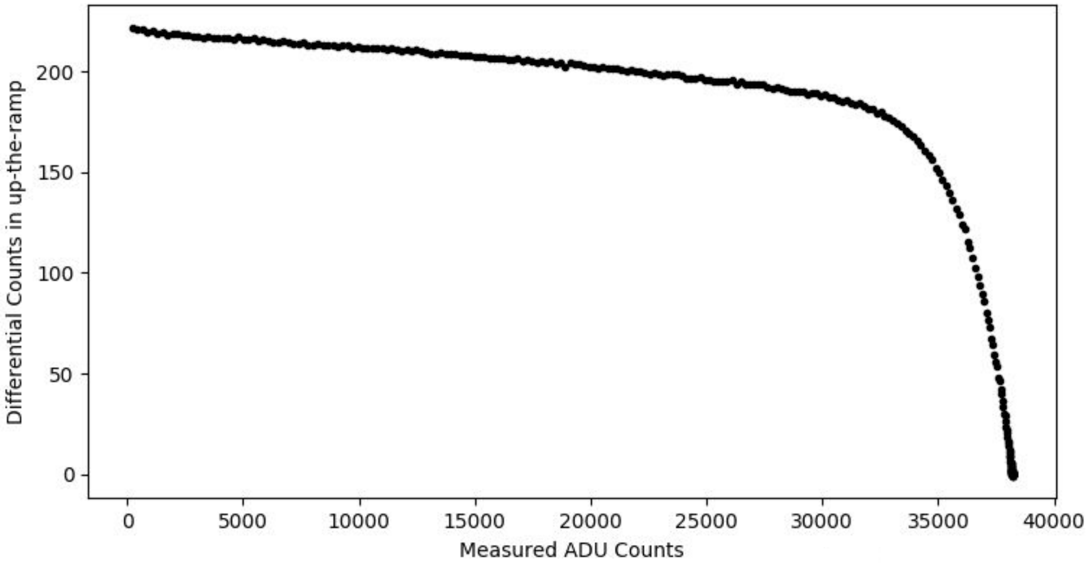
Integrating $$\int_0^T F_o dt = \int_0^C \frac{dc}{\epsilon(c)}$$

Gives us the non-linearity correction function which maps measured C to $F_o T$

Measuring $\epsilon(c)$ for HPF's H2RG detector pixels

Derivative of the up-the-ramp data of high signal-to-noise saturating flats

$\epsilon(c)$



c

Polynomial Models for $\epsilon(c)$

The pole at saturation makes it hard to fit $\epsilon(c)$ using simply polynomials.

Denominator in Padé approximant help to suppress the pole.

Padé model [3,1]

$$\frac{Sb_2}{a_2} + \frac{(a_2 b_1 - a_1 b_2)\log\left(S^2 a_2 + S a_1 + 1\right)}{2\,a_2^2} - \frac{\left(a_1 a_2 b_1 - 2\,a_2^2 - \left(a_1^2 - 2\,a_2\right)b_2\right)\arctan\left(\frac{2\,Sa_2+a_1}{\sqrt{-a_1^2+4\,a_2}}\right)}{\sqrt{-a_1^2+4\,a_2}a_2^2}$$

$$\frac{Sb_2}{a_2} + \frac{(a_2 b_1 - a_1 b_2)\log\left(S^2 a_2 + S a_1 + 1\right)}{2\,a_2^2} - \frac{\left(a_1 a_2 b_1 - 2\,a_2^2 - \left(a_1^2 - 2\,a_2\right)b_2\right)\log\left(\frac{2\,Sa_2+a_1-\sqrt{a_1^2-4\,a_2}}{2\,Sa_2+a_1+\sqrt{a_1^2-4\,a_2}}\right)}{2\sqrt{a_1^2-4\,a_2}a_2^2}$$

Analytical solution to the integral correction formula

Padé model [3,2]

$$\frac{S^2 a_2 b_3 + 2\left(a_2 b_2 - a_1 b_3\right)S}{2\,a_2^2} + \frac{\left(a_2^2 b_1 - a_1 a_2 b_2 + \left(a_1^2 - a_2\right)b_3\right)\log\left(S^2 a_2 + S a_1 + 1\right)}{2\,a_2^3} - \frac{\left(a_1 a_2^2 b_1 - 2\,a_2^3 - \left(a_1^2 a_2 - 2\,a_2^2\right)b_2 + \left(a_1^3 - 3\,a_1 a_2\right)b_3\right)\arctan\left(\frac{2\,Sa_2+a_1}{\sqrt{-a_1^2+4\,a_2}}\right)}{\sqrt{-a_1^2+4\,a_2}a_2^3}$$

$$\frac{S^2 a_2 b_3 + 2\left(a_2 b_2 - a_1 b_3\right)S}{2\,a_2^2} + \frac{\left(a_2^2 b_1 - a_1 a_2 b_2 + \left(a_1^2 - a_2\right)b_3\right)\log\left(S^2 a_2 + S a_1 + 1\right)}{2\,a_2^3} - \frac{\left(a_1 a_2^2 b_1 - 2\,a_2^3 - \left(a_1^2 a_2 - 2\,a_2^2\right)b_2 + \left(a_1^3 - 3\,a_1 a_2\right)b_3\right)\log\left(\frac{2\,Sa_2+a_1-\sqrt{a_1^2-4\,a_2}}{2\,Sa_2+a_1+\sqrt{a_1^2-4\,a_2}}\right)}{2\sqrt{a_1^2-4\,a_2}a_2^3}$$

Model: Inverse B-Spline model

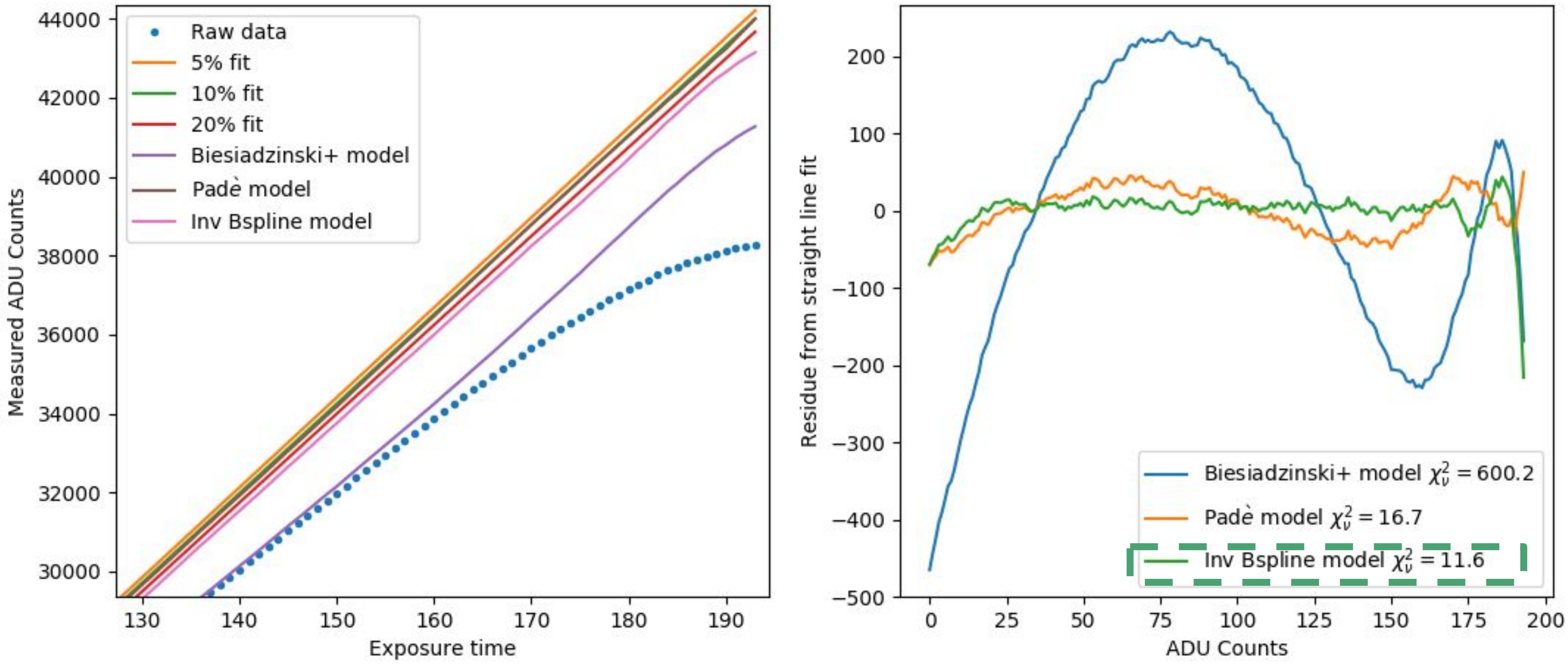
The motivation to use simple polynomials to model $\epsilon(c)$ is to have a fast analytical solution to the integral $\int_0^C \frac{dc}{\epsilon(c)}$

But numerical integration of B-spline models have an analytical formula!

B-spline offer more flexibility and at the same time robustness.

Hence, we fit the $\dfrac{1}{\epsilon(c)}$ curve with a B-spline for all 4.2 million pixels.

Final comparison of different model fits



Some practical notes

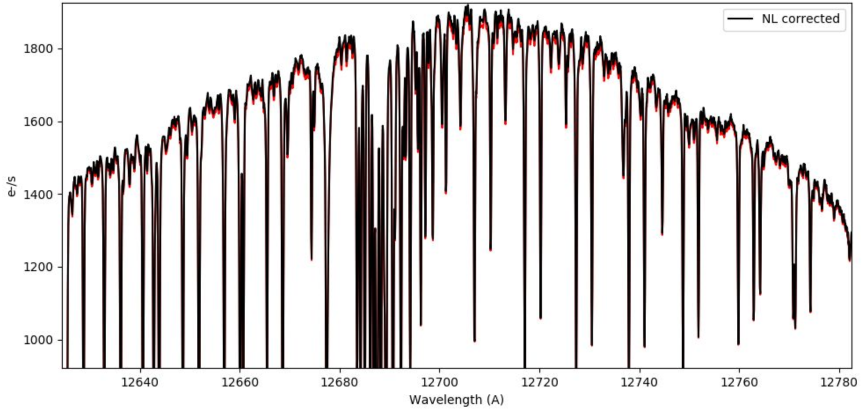
- Pedestal bias subtraction has to be done before non-linearity correction is applied

- Looping through 4 million pixels to apply a spline based function was too slow in Python. Hence, HxRGproc can outsource this calculation to other servers which does the spline calculation.
    - For instance, in HPF pipeline we have a Julia Non-linearity correction server which communicates to HxRGproc via unix sockets.

Summary

- **HxRGproc**: A free software tool for reducing H2RG up-the-ramp data

- Every pixel in H2RG has a **unique non-linearity** curve.
- Linear extrapolation under corrects non-linearity, and can cause extra noise.
- Modelling the correction function as an **integral of a monotonic**, yet flexible function performs well on **real life data**.
  - **Spline** based models under this category were found to be most robust while applied over 4.2 million pixels in the H2RG detector of HPF.



Thank you!