README for Version 1.0.

The MACCsimulator creates a single signal ramp that can be tuned to emulate various features seen in the NISP detectors, and emulates onboard processing by the DPU:

- Generates a non-destructive SUTR at full sampling for any corresonding MACC(ng,nf,nd) mode.
- Can add configurable readout noise and a non-lineaity term.
- Can add two representations of RTN.
- Resampling into MACC groups and group averages is performed.
- Slope, Quality Factor (QF), and projected signal are calculated following DPU software specifications.
- Does some plotting.

The routine is intended to be used for statisitical studies of the effects of non-linearities and pixel/SCE defects leading to RTN before and after data compression. The QFs are of importance to help identify where non-linear behavior is difficult or impossible to distinguish from RTN in compressed data, which will affect how pixels should be flagged since RTN pixels will be regarded as uncalibratible, while non-linarities will be corrected in the pipeline.

Two routines run the simulator, MACCsimulator.py and make_random_trace.py. The latter is used to generate one of the RTN types ("RTN1").

The API looks like this:

```
def MACCsimulator( ng=15, nf=16, nd=11, darkcurrent = 0.04, lightcurrent = 10.,
                   nonlin = -5.0e-5, readnoise = 9.,
                   RTN1 = False, RTN2 = False,
                   plots = True ):
```

The configurables are pretty self-explanatory; there is also explanations with units in the code.

To complile and run the defaults, producing an RTN-less ramp (after opening a remote Xsession if running from the cluster or else from a Jupyter notbook for the plotting), just do:

```
>>> from MACCsimulator import MACCsimulator
>>> MACCsimulator()
```

Some examples are shown on the next pages.

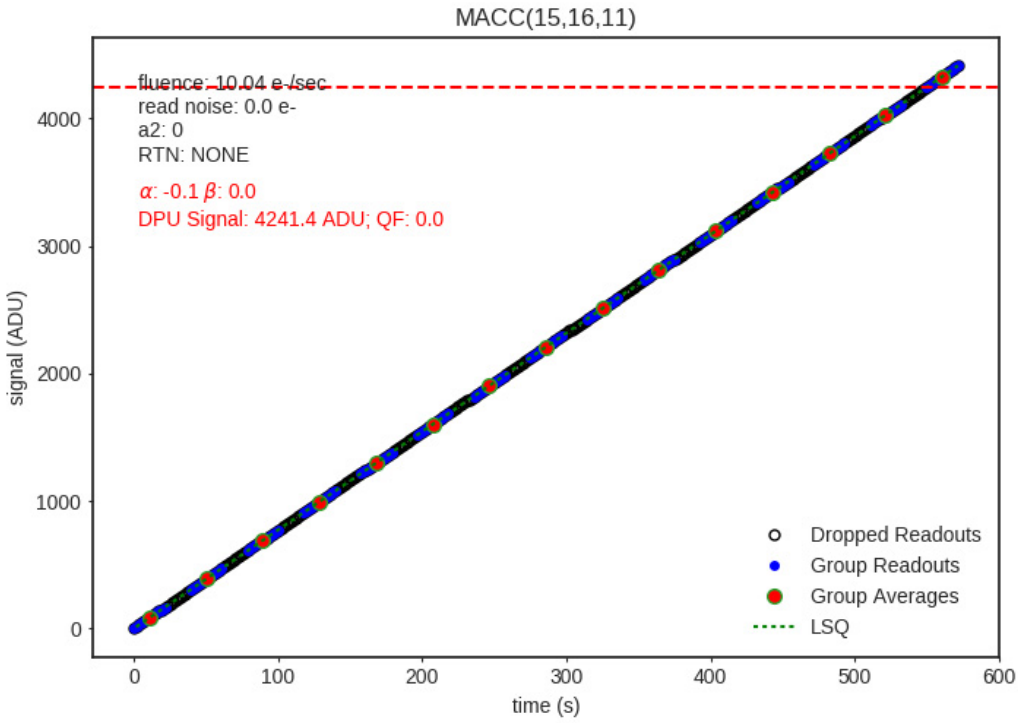
Advisements on Version 1.0, mainly to-do's:

1. The MACCsimulator code is a bit run-on, should be better modularlized into functions.
2. Only one type of RTN at a time; they are not meant to be both added since there is no need for this and might crash the code.
3. More RTN types will be added in the next version.
4. The characteristics of each RTN type, such as the maximum amplitudes and frequencies of the telegraphing readouts, are partly hard-coded. More user control will be added.

Examples

1. Produce a noiseless SPECTRO MACC(15,16,11) ramp for reference.

```
>>> MACCsimulator(readnoise=0,nonlin=0)
```



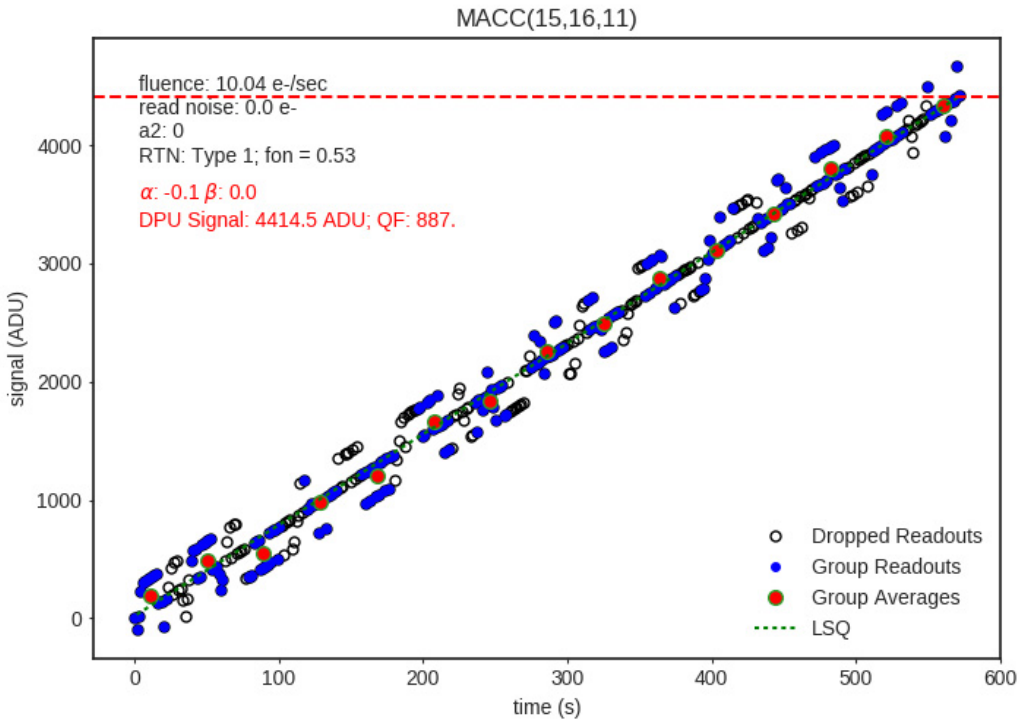
α encodes Poisson noise correlations (a constant that depends only on the exposure parameters common to all pixels), and β captures the dependence on readout noise weighted by α in the slope calculation.

The DPU Signal is the estimated flux which is radiated to the ground, measured from the Group Averages. Even for an unrealistically perfect ramp with no readout noise or non-linearity (thus QF = 0.0), the estimated flux is below the observed maximum because of the readout-to-readout correlation of 1/f noise modeled into the analytical flux estimator expression (see Kubik et al. 2016 PASP 128, 104504).

There is also some screen output giving inter-group slopes, explained in Example 3.

2. Add Type 1 RTN to Example 1.

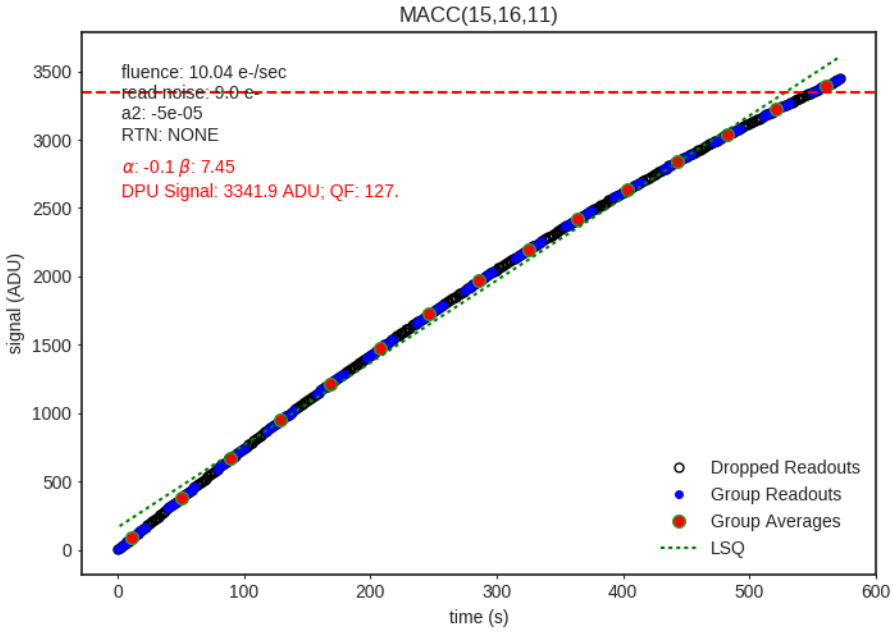
```
>>> MACCsimulator(readnoise=0,nonlin=0,RTN1=True)
```



The parameter fon is the fraction of readouts that are telegraphing. It is ranomly generated, but always > 0.5 for RTN1 and > 0.2 for RTN2.

3. Add default non-linearity and readnoise to Example 1.

```
>>> MACCsimulator()
```



Both readnoise and nonlin can be changed to whatever is sensible. QF is now higher.

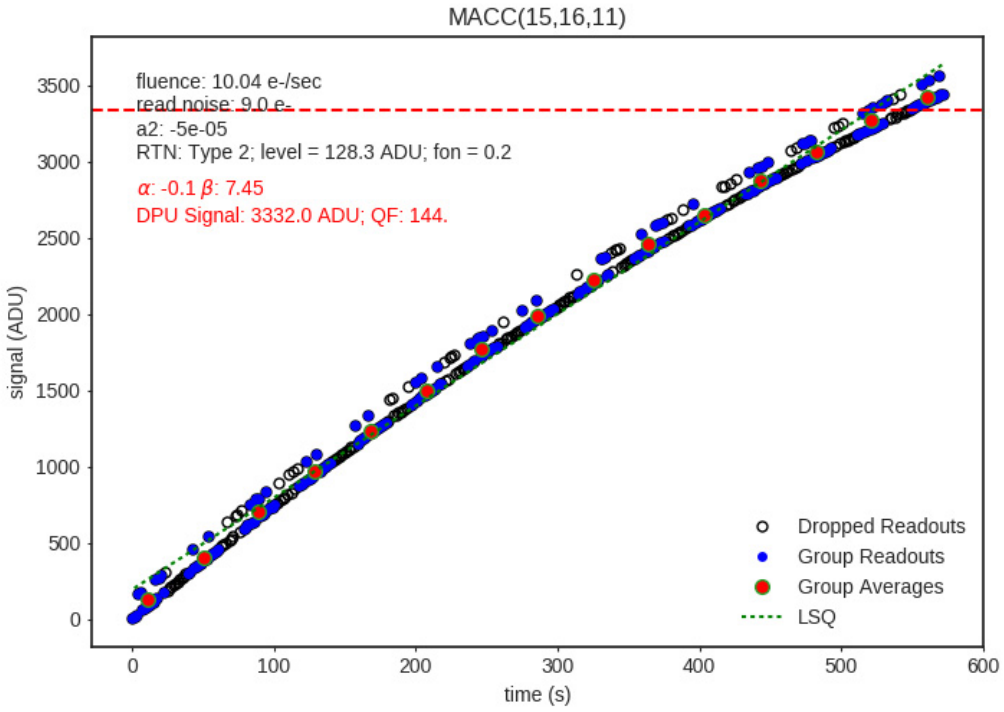
Here is the screen output:

```
>>> MACCsimulator()
sum =  851135.7903593874
Group slopes:
1   295.0290213185222
2   287.34858677231216
3   279.23686355723316
4   267.83084226499363
5   257.6466913133079
6   252.4406091149424
7   240.07299654142548
8   231.59832541601236
9   222.7832562777494
10   213.24207813906833
11   205.0636799233862
12   194.26333921842615
13   186.05300116085982
14   175.49971745917674
Slope DPU =  238.70727578008766
Signal DPU =  3341.901860921227
Pseudo flux ghat_x =  239.1085690615837
Quality Flag =  127.56884281774546
Slope Polyfit(1) =  167.113205995395  Chi-Squared =  2105.5021194874716
```

The group slopes are actually $I[i] - I[i-1]$ signal differences up the ramp rather than $\Delta I/\Delta t$ of each fragment, adhering to the definition of Slope DPU as computed by onboard software. The group slopes are calculated in the onboard signal estimator, but they are not stored and sent to the ground. The flattening values reflects the injected non-linearity, and may provide clues for disentangling non-linearities from RTN in collapsed data. ghat_x ($\hat{g}_x$) is the projected signal when 1/f noise correlations are neglected (i.e., $\alpha = 0$). Slope Polyfit(1) and Chi-Sq are from straight line fitting for reference. [Chi-Sq might not be computed correctly but it's not used and not important here.]

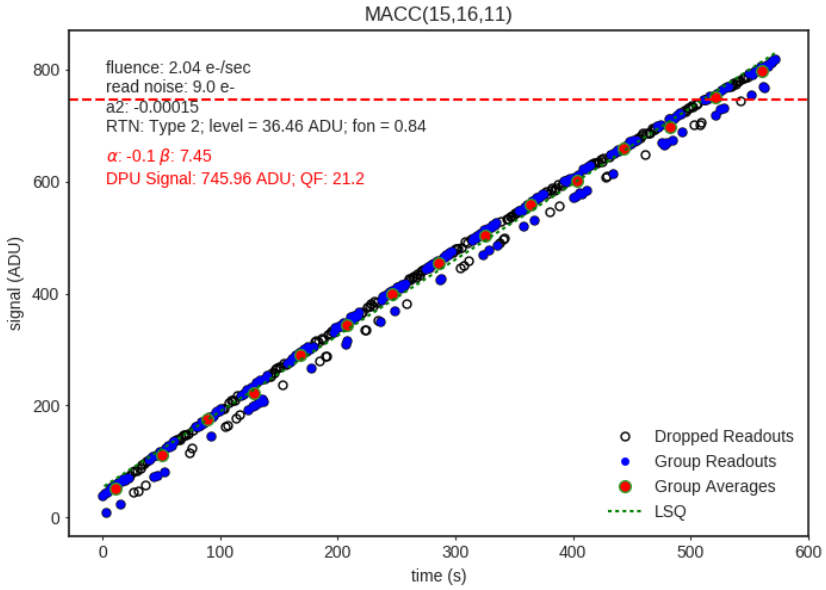
4. Add RTN Type 2 to Example 3.

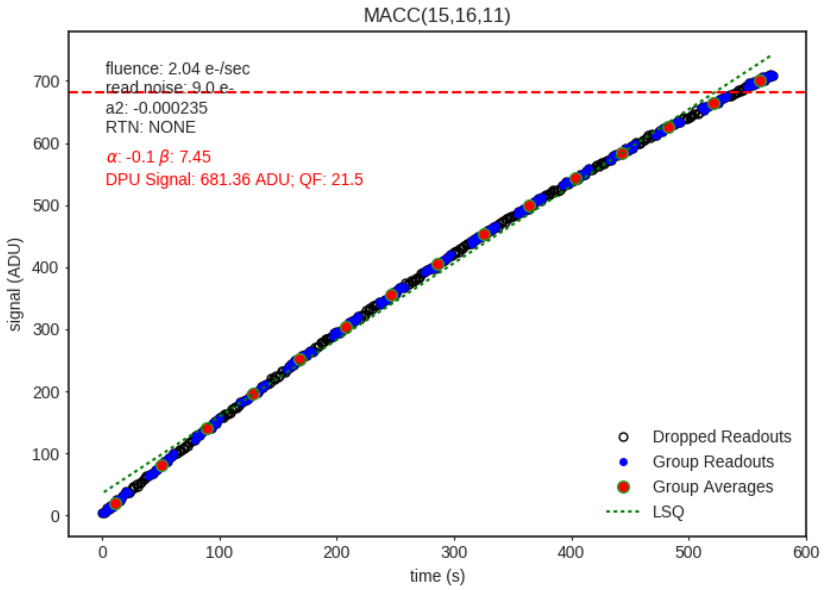
```
>>> MACCsimulator(RTN2=True)
```



5. Zodical light fluences

```
>>> MACCsimulator(lightcurrent=2., nonlin = -1.5e-4, RTN2=True)
>>> MACCsimulator(lightcurrent=2., nonlin - -2.35e-4)
```





Example of the same intrinsic ramp with different forms of noise leading to near-equal QFs but significantly different estimated signals--- due to weak non-linearity + high percentage of Type 2 telegraphing points (top) vs. stronger non-linearity + no telegraphing (bottom).