HxRG Noise Generator

Bernard J. Rauscher
NASA Goddard Space Flight Center, Greenbelt, MD 20771

3 November 2015 (Rev. 2.6beta and 2.4stable)

eMail: Bernard.J.Rauscher@nasa.gov

1 INTRODUCTION

1.1 Overview

Noise generator (NG) is a tool for generating noise images and datacubes for Teledyne HxRG based detector systems. NG builds on principal components analysis of the James Webb Space Telescope (*JWST*) Near Infrared Spectrograph (NIRSpec) detector subsystem and our experience developing Improved Reference Sampling and Subtraction (IRS^2; pronounced "IRS-square").[1] Although NG was informed by *JWST*, it can be used to simulate many other HxRG based systems by changing the input parameters. This executable IPython Notebook contains several examples.

The NG distribution includes the python source code, a set of default noise parameters that produce noise similar to *JWST* NIRSpec, and a collection of worked examples for *JWST* and other HxRG detector systems.

1.2 Installation and Getting Started

To get started, you should unpack this distribution somewhere in your python search path and set the shell variable NGHXRG_HOME to point to it. The distribution contains the following files.

01_README.ipynb	README as an IPython Notebook
02_README.pdf	README in PDF format
nghxrg_v2.4stable.py	Previous release may be more stable
nghxrg_v2.6beta.py	The current NG source code
nghxrg.py	A soft link to the NG source code
ng_ms.pdf	Preprint of a journal article that describes the simulator
nirspec_pca0.fits	The measured NIRSpec PCA-zero component

By default, the more recent beta version is selected. If you would like to run the previous release, which may be more stable, update the hghxrg.py softlink to point to it.

NG 2.6 requires python-2.6 (or later) and the following python modules to run. The previous release, NG 2.4, required python-3.4 or later. We show the specific python and module versions that were used for initial development at NASA Goddard. For Mac computers, all of these modules are freely available from MacPorts.

- python34 @3.4.3_2
- py34-astropy @0.4.1_3
- datetime (included in python distribution)
- py34-numpy @1.9.2_0+gfortran
- os (included in python distribution)
- scipy, py34-scipy @0.14.0_0+gcc48
- warnings (included in python distribution)

1.3 About the Model

The noise model is informed by our experience doing principal components analysis of the *JWST* NIRSpec detector subsystem. It includes: (1) white read noise, (2) pedestal drifts, (3) correlated pink noise, (4) uncorrelated pink noise, (5) alternating column noise (ACN), and (6) PCA-zero (also known as "picture frame"). In items 3 and 4, correlated/uncorrelated refers only to whether the corresponding noise component is correlated across all outputs or not. In Fourier space, the pink components are $1/f$-like and the ACN components appear as $1/f$ modulating the Nyquist frequency. In science images, ACN appears as an alternating column pattern.

Although NG includes both stationary and non-stationary components, it assumes that the non-stationary components are completely uncorrelated with the stationary ones. Our measurements suggest that this is a pretty good, but imperfect, approximation to the real situation. The practical effect is that there will be some non-flight like mixing of $1/f$-like and "picture frame" noise.

ACKNOWLEDGMENTS

NG was initially developed by B.J. Rauscher of NASA Goddard Space Flight Center as part of the James Webb Space Telescope (*JWST*) Project. J.M. Leisenring of the *JWST* Near Infrared Camera (NIRCam) team and University of Arizona and Steward Observatory made a number of improvements including backward compatability to Python 2.x, more flexible subarrays, and speed enhancements.

References

2 EXAMPLES

These examples are provided to illustrate specific aspects of NG and to serve as templates for new simulations. In some cases, we refer to real systems and provide a set of (very approximate) parameters. For critical simulations, we recommend speaking to the instrument builders to ensure that your simulations use the most up to date parameters. In any of these simulations, you can turn on the verbose option to see runtime status information.

We begin by importing NG.

```
In [ ]: import nghxrg as ng
```

2.1 *JWST* NIRSpec H2RG and SIDECAR ASIC

The examples in this section use noise inputs that are roughly similar to *JWST* NIRSpec. In some cases, particular components are "turned up" to show them more clearly in the science images. We do this for illustrative purposes only. In a real system, one would clearly try to fix these artifacts.

2.1.1 Two Dimensional Science Image

In this example, we make a simulated 2048×2048 pixel H2RG noise image. The image contains the detector system noise components that would be seen after fitting up-the-ramp slopes and applying a basic reference pixel correction using only reference pixels in rows. On a MacBook Pro (Retina, 15-inch, Early 2013) with 2.7 GHz Intel Core i7 and 16 GB 1600 MHz DDR3 memory, it takes about 20 seconds to simulate a NIRSpec integration using 4 video outputs.

In [ ]:
```
# Instantiate a noise generator object for NIRSpec H2RGs. You
# can set verbose=True if you want to see diagnostic information. T
his
# will take about 20 seconds to execute on a modern laptop compute
r.
ng_h2rg = ng.HXRGNoise(verbose=False)

# Use parameters that generate noise similar to JWST NIRSpec
rd_noise=4.   # White read noise per integration
pedestal=4.   # DC pedestal drift rms
c_pink=3.     # Correlated pink noise
u_pink=1.     # Uncorrelated pink noise
acn=.5        # Correlated ACN
pca0_amp=.2   # Amplitude of PCA zero "picture frame" noise

# Do it
my_hdu = ng_h2rg.mknoise('ex_2.1.1.fits', rd_noise=rd_noise, pedest
al=pedestal,
                 c_pink=c_pink, u_pink=u_pink, acn=acn, pca0_amp=pca
0_amp)
```

2.1.1b Add Dark Current to Example 2.1.1

One could add astronomical sources, background light, and dark current to the image created in Ex. 2.1.1. The resulting simulation will be pretty good, but imperfect. Although it would not correctly account for temporal correlations in dark current in the up-the-ramp samples, the resulting simulation might still be useful for non-critical applications at very modest cost in computing time. We often work with two dimensional simulations like this because many different realizations can be made quickly. This example shows how to add dark current subject to these caveats. A later example will show how to add dark current including the correct up-the-ramp correlations.

The NIRSpec detectors have mean dark current, $i_{\rm dark} \sim 0.005\ e^-\ s^{-1}\ {\rm pix}^{-1}$, and the standard exposure time is $934\ s$.

In [ ]:
```
# Setup
i_dark = 0.005# e-/s/pix
t = 934.# s

# Open the result of Ex. 2.1.1
hdulist = fits.open('ex_2.1.1.fits')

# Add Poisson noise to the data
d = hdulist[0].data + np.random.poisson(i_dark*t, np.shape(hdulis
t[0].data))

# Write result
hduout = fits.PrimaryHDU(d)
hduout.writeto('ex_2.1.1b.fits', clobber=True)

# Clean up
hdulist.close()
```

2.1.2 Three Dimensional Datacube

In this example, we simulate an integration containing 88 non-destructive up-the-ramp reads. The resulting cube could be used as the basis for a more accurate observation simulation by adding sources, backgrounds, and dark current frame-by-frame. Compared to the result of Ex. 2.1.1, this would have the correct correlations in the up-the-ramp samples. The resulting datacube could also be used to validate low level calibration software.

This example takes much longer to run than others in this IPython Notebook. For this reason, we have commented it out. If you wish to run it, please remove the leading "#" from each line.

In [ ]:
```
## Instantiate a new object, this time a 2048x2048x88 pixel cube
#ng_h2rg_cube = ng.HXRGNoise(naxis3=88, verbose=False)
#
## Use parameters that generate noise similar to JWST NIRSpec
#rd_noise=4*4.  # White read noise per frame
#pedestal=4*4.   # DC pedestal drift rms
#c_pink=4*3.     # Correlated pink noise
#u_pink=4*1.     # Uncorrelated pink noise
#acn=4*.5        # Correlated ACN
#pca0_amp=4*.2   # Amplitude of PCA zero "picture frame" noise
#
## Do it
#my_hdu = ng_h2rg_cube.mknoise('ex_2.1.2.fits', rd_noise=rd_noise,
pedestal=pedestal,
#                c_pink=c_pink, u_pink=u_pink, acn=acn, pca0_amp=pc
a0_amp)
```

2.1.2b Add Dark Current to a Datacube

In contrast to Ex. 2.1.1b, this example includes more correct correlations in the up-the-ramp samples. Although this simulation is considerably more time consuming than Ex. 2.1.1b, it is better for critical applications. The datacube has 88 up-the-ramp frames. The frame readout time is $t_{\text{frame}} = 10.7368\ s\ \text{frame}^{-1}$.

Although this simulation is higher fidelity than Ex. 2.1.1b, it is still not perfect. Because dark current is integrated charge, there are also correlations in the spatial domain that must be handled for the highest fidelity work. Specifically, inter-pixel capacitance (IPC) should be considered, although it is not considered here. IPC is one of many effects that should be included when modeling how HxRG detectors respond to light (and dark current).

Because this code is dependent on Ex. 2.1.2, it is commented out. If you would like to run it, please uncomment the lines.

In [ ]:
```
## Setup
#i_dark = 0.005# e-/s/pix
#t_frame = 10.7368# s
#
## Open the result of Ex. 2.1.2 and get the datacube dimensions in
an
## easy to use format
#hdulist = fits.open('ex_2.1.2.fits')
#nz = hdulist[0].header['naxis3']# Number of up-the-ramp frames
#ny = hdulist[0].header['naxis2']# Number of rows
#nx = hdulist[0].header['naxis1']# Number of columns
#
## Make a cube that contains only the dark current since the 0th
## read. Work only on pixels in the range [4:2044] since reference
## pixels do not respond to light. The reference pixel border is 4
## pixels wide. The regular pixels are therefore a (ny-8)x(nx-8)
## area.
#dk_cube = np.zeros((nz,ny,nx), dtype=np.uint16)
#for z in np.arange(1,nz):
#    dk_cube[z,4:2044,4:2044] = dk_cube[z-1,4:2044,4:2044] + \
#        np.random.poisson(i_dark*t_frame, (ny-8,nx-8))
#
## Add dark current cube to NG noise cube
#result = dk_cube + hdulist[0].data
#
## Write result
#hduout = fits.PrimaryHDU(result)
#hduout.writeto('ex_2.1.2b.fits', clobber=True)
#
## Clean up
#hdulist.close()
```

2.1.3 H2RG Tuned (Badly) to Emphasize ACN

ACN is difficult to see in *JWST* NIRSpec darks because the system was tuned to minimize it. In this example, we adjust RNG's input parameters to clearly show ACN. Since this is for an H2RG, we re-use the object that we created in Ex. 2.1. The amount of ACN that is shown here is completely unrealistic for most systems.

In [ ]:
```
# Use parameters that generate noise similar to JWST NIRSpec
rd_noise=4.   # White read noise per integration
pedestal=4.   # DC pedestal drift rms
c_pink=3.     # Correlated pink noise
u_pink=1.     # Uncorrelated pink noise
acn=4*.5      # *** Add 4x as much ACN as before ***
pca0_amp=.2   # Amplitude of PCA zero "picture frame" noise

my_hdu = ng_h2rg.mknoise('ex_2.1.3.fits', rd_noise=rd_noise, pedest
al=pedestal,
                c_pink=c_pink, u_pink=u_pink, acn=acn, pca0_amp=pca
0_amp)
```

2.1.4 H2RG Tuned to Emphasize Picture Frame

In this example we adjust the tuning to clearly show picture frame (PCA0) noise. The amount of picture frame noise that is shown here is unrealistic for most systems.

In [ ]:
```
# Use parameters that generate noise similar to JWST NIRSpec
rd_noise=4.   # White read noise per integration
pedestal=4.   # DC pedestal drift rms
c_pink=3.     # Correlated pink noise
u_pink=1.     # Uncorrelated pink noise
acn=.5        # Add ACN
pca0_amp=4*.2 # *** Add 4x as much picture frame as before ***

my_hdu = ng_h2rg.mknoise('ex_2.1.4.fits', rd_noise=rd_noise, pedest
al=pedestal,
                c_pink=c_pink, u_pink=u_pink, acn=acn, pca0_amp=pca
0_amp)
```

2.1.5 H2RG with Fast Scan Directions Reversed

This example shows how to reverse the scan directions. Begin by generating an image that has the fast scanners reversed.

In [ ]:
```
# Create a new instance with reversed fast scanners
ng_h2rg_rev = ng.HXRGNoise(verbose=False, reverse_scan_direction=Tr
ue)

# Use the same parameters as in Ex. 2.1
rd_noise=4.   # White read noise per integration
pedestal=4.   # DC pedestal drift rms
c_pink=3.     # Correlated pink noise
u_pink=1.     # Uncorrelated pink noise
acn=.5        # Correlated ACN
pca0_amp=.2   # Amplitude of PCA zero "picture frame" noise

# Do it
my_hdu = ng_h2rg.mknoise('ex_2.1.5a.fits', rd_noise=rd_noise, pedes
tal=pedestal,
                 c_pink=c_pink, u_pink=u_pink, acn=acn, pca0_amp=pca
0_amp)
```

Use python's slice notation to reverse the slow scanners if desired.

In [ ]:
```
# Open the fits file
hdulist = fits.open('ex_2.1.5a.fits')

# Get the data
d = hdulist[0].data

# Flip vertically
d = d[::-1,:]

# Save the result (here we don't worry about maintaining the header
information)
hdu = fits.PrimaryHDU(d)
hdu.writeto('ex_2.1.5b.fits', clobber=True)

# Close the file since it is no longer needed
hdulist.close()
```

2.2 H4RG Examples

H4RG detectors are in development for both ground and space. In many cases, the requirements call for using at least 32 outputs.

2.2.1 *WFIRST* H4RG-10 and SIDECAR ASIC

NASA's Wide Field Infrared Survey Telescope (*WFIRST*) plans H4RG-10 detectors and SIDECAR ASICs. This example shows how to change the number of outputs from four to $\mathrm{n\_out} = 32$ and how to change the new row overhead (nroh) from 12 to 8 using $\mathrm{nroh} = 8$. Because *WFIRST*'s H4RG-10s are still in development, the parameters shown here are just place holders. They will need to be updated when real test data become available.

In [ ]:
```
# Instantiate a new object on account of the different array dimens
ions.
# Recall that the H4RG has 4096x4096 pixels. Run using
# 32 outputs. Also set the new row overhead to 8 pixels (a power of
2)
# which simplifies working with the data in Fourier space.
ng_h4rg = ng.HXRGNoise(naxis1=4096, naxis2=4096, n_out=32, nroh=8,
verbose=False)

# Make a noise file.
rd_noise=4   # White read noise per integration
pedestal=4   # DC pedestal drift rms
c_pink=3     # Correlated pink noise
u_pink=1     # Uncorrelated pink noise
c_ACN=1      # Alternating column noise
pca0_amp=.5  # Amplitude of PCA zero "picture frame" noise

my_hdu = ng_h4rg.mknoise('ex_2.2.1.fits', rd_noise=rd_noise, pedest
al=pedestal, c_pink=c_pink,
            u_pink=u_pink, acn=acn)
```

2.3 Other Situations

2.3.1 32×32 Pixel Subarray Datacube

This example shows how to generate a 32x32x128 pixel datacube. Unless they happen to fall along the edges of the detector, subarrays do not contain embedded reference pixels. This example shows how to use the parameter reference_pixel_border_width to address this.

In [ ]:
```
# Instantiate a new object having the correct dimensions. HxRG detectors read subarrays
# using only one output; hence n_ou=1. We furthermore assume that the camera builder was
# careful to minimize the new row overhead in subarray mode so that, nroh=8. Larger or
# smaller values of nroh are possible. You should ask the instrument builder for the
# correct value for your system.
ng_subarray = ng.HXRGNoise(naxis1=32, naxis2=32, naxis3=128, n_out=1, nroh=8,
                           reference_pixel_border_width=0, verbose=False)

# Use the same parameters as in Ex. 2.1
rd_noise=4.   # White read noise per integration
pedestal=0    # Assume no pedestal drifts for these short exposures
c_pink=3.     # Correlated pink noise
u_pink=1.     # Uncorrelated pink noise
acn=.5        # Correlated ACN
pca0_amp=0    # Turn off PCA0. It uses a file that is appropriate only for full frame data.

# Do it
my_hdu = ng_subarray.mknoise('ex_2.3.1.fits', rd_noise=rd_noise, pedestal=pedestal,
                c_pink=c_pink, u_pink=u_pink, acn=acn, pca0_amp=pca0_amp)
```