pyNRC Documentation

Release 0.9.0dev

Jarron Leisenring

Jun 06, 2020

GETTING STARTED

CHAPTER

ONE

PYNRC - PYTHON ETC AND SIMULATOR FOR JWST NIRCAM

pyNRC is a set of Python-based tools for planning observations with JWST NIRCam. It includes an ETC, a simple image slope simulator, and an enhanced data simulator. This package works for a variety of NIRCam observing modes including direct imaging, coronagraphic imaging, slitless grism spectroscopy, DHS observations, and weak lens imaging. All PSFs are generated via WebbPSF (https://webbpsf.readthedocs.io) to reproduce realistic JWST images and spectra.

Developed by Jarron Leisenring and contributors at University of Arizona (2015 - 2019).

1.1 pyNRC

1.1.1 A JWST NIRCam ETC and Simulator

Authors: Jarron Leisenring (UA)

Contributors: Everett Schlawin (UA), Jonathan Fraine (STScI)

!!Under Development!!

pyNRC is a set of Python-based tools for planning observations with JWST NIRCam, such as an ETC, a simple slope image simulator, and an enhanced data simulator.

While special attention has been placed on NIRCam coronagraphic modes, this package also works for a variety of NIRCam observing modes including:

- direct imaging
- coronagraphic imaging
- weak lens imaging
- slitless grism spectroscopy
- DHS observations (TBI)

All PSFs are generated via WebbPSF (https://webbpsf.readthedocs.io) to reproduce realistic JWST images and spectra.

Documentation can be found at https://pynrc.readthedocs.io.

Note: pyNRC enables more modes than are officially allowed by the Observatory, (ie., filter + coronagraphic combinations, subarray sizes, etc.). Just because you can do something with pyNRC does not mean it will be supported. Check out https://jwst-docs.stsci.edu/display/JTI/NIRCam+Observing+Modes for more information.

Similar to some of its dependencies, pyNRC requires a host of input data files in order to generate simulations. Due to the size of these files, they are not included with this source distribution. Please see the documentation for instructions on how to to download the required data files.

1.2 Installation

1.2.1 Requirements

pyNRC requires Python 3.5+ along with the following packages:

- Recent version of NumPy, SciPy, and matplotlib
- Astropy 2+
- pysynphot 0.9.7+
- WebbPSF 0.8.0+ and its dependencies.

Recommended Python packages:

- jwst_backgrounds 1.1.1+
- psutil

1.2.2 Installing with conda

Todo: pyNRC has been placed on conda-forge so you can manage the package through your Conda installation. Simply add conda-forge to your `.condarc` file, which appends the appropriate URL to Conda's channel search path:

```
$ conda config --add channels conda-forge
```

With the conda-forge channel added, it's a simple matter to run:

```
$ conda install pynrc
```

1.2.3 Installing with pip

You can install the `pynrc` package through pip:

```
$ pip install pynrc
```

Note that the pip command only installs the program code. You still must download and install the data files, as *described below*.

1.2.4 Installing from source

To get the most up to date version of `pynrc`, install directly from source, though stability is not guarenteed. The development version can be found on GitHub.

In this case, you will need to clone the git repository:

```
$ git clone https://github.com/JarronL/pynrc
```

Then install the package with:

```
$ cd pynrc
$ pip install .
```

For development purposes:

```
$ cd pynrc
$ pip install -e .
```

in order to create editable installations. This is great for helping to develop the code, create bug reports, pull requests to GitHub, etc.

1.2.5 Installing the data files

Files containing such information as the instrument throughputs, SCA biases and darks, stellar models, and exoplanet models are distributed separately. To run `pynrc`, you must download these files and define the `PYNRC_PATH` environment variable. This is also the location that PSF coefficients will be saved to during normal operations of `pynrc`.

1. Download the following file: pynrc_data_v0.6.1.tar.gz [approx. 2.3 GB]
2. Untar into a directory of your choosing.
3. Set the environment variable `PYNRC_PATH` to point to that directory. For bash, for example:

   ```
   $ export PYNRC_PATH=$HOME/data/pynrc_data
   ```

   You will probably want to add this to your `.bashrc`.

You should now be able to successfully `import pynrc` in a Python session.

1.2.6 Testing

Todo: If you want to check that all the tests are running correctly with your Python configuration, you can also run:

```
$ python setup.py test
```

in the source directory. If there are no errors, you are good to go!

1.3 Install from New Conda Environment

This installation tutorial assumes a clean installation with Anaconda and has been verified on both Python 2.7 and 3.6 using the following modules:

- Numpy 1.14
- Matplotlib 2.1
- Scipy 1.0
- Astropy 2.0

1.3.1 Configure Conda to use the AstroConda Channel

We will be install a few packages that live in AstroConda. If you're already working in an AstroConda environment, then you should be all set and can probably skip this step.

If you have some other Conda, installation, then you can simply add the AstroConda channel to your `.condarc` file, which appends the appropriate URL to Conda's channel search path:

```
$ conda config --add channels http://ssb.stsci.edu/astroconda
# Writes changes to ~/.condarc
```

1.3.2 Installing Pysynphot

With the AstroConda channel added, it's a simple matter to run:

```
$ conda install pysynphot
```

Otherwise, install the standalone release:

```
$ pip install git+https://github.com/spacetelescope/pysynphot.git@0.9.8.8
```

Pysynphot Data Files

Data files for Pysynphot are distributed through the Calibration Reference Data System. They are expected to follow a certain directory structure under the root directory, identified by the `PYSYN_CDBS` environment variable that *must* be set prior to using this package.

1. Download the following file: cdbs.tar.gz [approx. 760 MB]
2. Untar into a directory of your choosing.
3. Set the environment variable `PYSYN_CDBS` to point to that directory. For example, in .bashrc shell file, add:

   ```
   export PYSYN_CDBS='$HOME/data/cdbs/'
   ```

You should now be able to successfully `import pysynphot` in a Python session.

1.3.3 Installing WebbPSF

The AstroConda copy of WebbPSF has a `webbpsf-data` installation dependency, which we do not want in our slightly customized installation, because the WebbPSF data files get downloaded separately. Instead, we will do this in two parts to first install the most of the dependencies first, then WebbPSF with the `--no-deps` flag:

```
$ conda install jwxml poppy
$ conda install webbpsf --no-deps
```

For other installation methods see the WebbPSF documentation.

Caution: A note about backends.

In many cases `matplotlib` crashes when using the default backend (at least on Mac OS X and certain Linux distributions). Given the propensity for these crashes, it may be preferable to use a different graphics backend such as `TkAgg`. This can either be accomplished by setting `matplotlib.use("TkAgg")` after importing `matplotlib` or setting the default backend via your *matplotlibrc file <https://matplotlib.org/users/customizing.html#the-matplotlibrc-file*. The latter option is probably preferred for most cases.

WebbPSF Data Files

For the user's convenience, WebbPSF data files can be found here: webbpsf-data-0.6.0.tar.gz [approx. 240 MB] Follow the same procedure as with the Pysynphot data files, setting the `WEBBPSF_PATH` environment variable to point towards your `webbpsf-data` directory.

1.3.4 Installing JWST Backgrounds

`jwst_bakcgrounds` is a a simple program to predict the levels of background emission in JWST observations. It accesses a precompiled background cache prepared by Space Telescope Science Institute, requiring an internet connection to access. However, `pynrc` comes a simpler background estimator in the event no there is no internet functionality. In this sense, `jwst_backgrounds` is not a strict requirement for running `pynrc`.

This module requires `healpy` to run:

```
$ conda config --add channels http://ssb.stsci.edu/astroconda
$ conda install healpy
```

If `healpy` asks you to downgrade some of its dependencies, it is suggested that you only install the missing dependencies manually, then run conda with the `--no-deps` flag. For instance:

```
$ conda install pytest-runner --no-deps
$ conda install healpy --no_deps
```

Then install JWST Backgrounds with pip:

```
$ pip install jwst_backgrounds
```

1.3.5 Installing pyNRC

Installing with pip

You can install the `pynrc` package through pip:

```
$ pip install pynrc
```

Note that the pip command only installs the program code. You still must download and install the data files, as described below.

Installing from source

To get the most up to date version of `pynrc`, install directly from source, though stability is not guarenteed. The development version can be found on GitHub.

In this case, you will need to clone the git repository:

```
$ git clone https://github.com/JarronL/pynrc
```

Then install the package with:

```
$ cd pynrc
$ pip install .
```

For development purposes:

```
$ cd pynrc
$ pip install -e .
```

in order to create editable installations. This is great for helping to develop the code, create bug reports, pull requests to GitHub, etc.

pyNRC Data Files

Similarly, `pynrc` comes with its own set of data files, such as instrument throughputs, SCA biases and darks, stellar models, and exoplanet models. To run `pynrc`, you must download these files and define the `PYNRC_PATH` environment variable. This is also the location that PSF coefficients will be saved to during normal operations of `pynrc`.

1. Download the following file: pynrc_data_v0.6.1.tar.gz [approx. 2.3 GB]
2. Untar into a directory of your choosing.
3. Set the environment variable `PYNRC_PATH` to point to that directory. For example, in .bashrc shell file, add:

   ```
   export PYNRC_PATH='$HOME/data/pynrc_data'
   ```

   You will probably want to add this to your `.bashrc`.

You should now be able to successfully `import pynrc` in a Python session.

1.4 Basic Usage

This tutorial walks through the basic usage of the `pynrc` package to calculate sensitivities and saturation limits for NIRCam in a variety of modes.

[1]:
```
# Makes print and division act like Python 3
from __future__ import print_function, division

# Import the usual libraries
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

# Enable inline plotting at lower left
%matplotlib inline

from IPython.display import display, Latex, clear_output
```

1.4.1 Getting Started

We assume you have already installed `pynrc` as outlined in the documentation.

[2]:
```
# import main module
import pynrc
from pynrc import nrc_utils

# import pysynphot instance
#from pynrc.nrc_utils import S
```

Log messages for `pynrc` follow the same the logging functionality included in `webbpsf`. Logging levels include `DEBUG`, `INFO`, `WARN`, and `ERROR`.

[3]:
```
pynrc.setup_logging()
```

```
pyNRC log messages of level INFO and above will be shown.
pyNRC log outputs will be directed to the screen.
```

If you get tired of the `INFO` level messages, simply type:

```
pynrc.setup_logging('WARN', verbose=False)
```

1.4.2 First NIRCam Observation

The basic NIRCam object consists of all the instrument settings one would specify for a JWST observation, including filter, pupil, and coronagraphic mask selections along with detector subarray settings and ramp sampling cadence (i.e., `MULTIACCUM`).

The NIRCam class makes use of high order polynomial coefficient maps to quickly generate large numbers of monochromatic PSFs that can be convolved with arbitrary spectra and collapsed into a final broadband PSF (or dispersed with NIRCam's slitless grisms). The PSF coefficients are calculated from a series of WebbPSF monochromatic PSFs and saved to disk. These polynomial coefficients are further modifed based on focal plane position and drift in the wavefront error relative to nominal OPD mao.

There are a multitude of posssible keywords one can pass upon initialization, including certain detector settings and PSF generation parameters. If not passed initially, then defaults are assumed. The user can update these parameters at any time by either setting attributes directly (e.g., `filter`, `mask`, `pupil`, etc.) along with using the `update_detectors()` and `update_psf_coeff()` methods.

For instance,

```
nrc = pynrc.NIRCam('F210M')
nrc.module = 'B'
nrc.update_detectors(read_mode='DEEP8', nint=10, ngroup=5)
```

is the same as:

```
nrc = pynrc.NIRCam('F210M', module='B', read_mode='DEEP8', nint=10, ngroup=5)
```

To start, we'll set up a simple observation using the `F430M` filter. Defaults will be populated for unspecified attributes such as `module`, `pupil`, `mask`, etc.

Check the function docstrings for more detailed information

[4]:
```
nrc = pynrc.NIRCam(filter='F430M')
print('Filter: {};  Pupil: {};  Mask: {};  Module: {}'\
      .format(nrc.filter, nrc.pupil, nrc.mask, nrc.module))
```

```
[     pynrc:INFO] Initializing SCA 485/A5
[     pynrc:INFO] RAPID readout mode selected.
[     pynrc:INFO] Setting nf=1, nd1=0, nd2=0, nd3=0.
[     pynrc:INFO] Updating PSF coeff with fov_pix=33 and oversample=4
[     pynrc:INFO] Generating and saving new PSF coefficient
Filter: F430M;  Pupil: CLEAR;  Mask: None;  Module: A
```

Keyword information for detector and PSF settings are stored in the `det_info` and `psf_info` dictionaries. These cannot be modified directly, but instead are updated via the `update_detectors()` and `update_psf_coeff()` methods.

[5]:
```
print('Detector Info Keywords:')
print(nrc.det_info)
print('')
print('PSF Info Keywords:')
print(nrc.psf_info)
```

```
Detector Info Keywords:
{'wind_mode': 'FULL', 'xpix': 2048, 'ypix': 2048, 'x0': 0, 'y0': 0, 'read_mode':
↪'RAPID', 'nint': 1, 'ngroup': 1, 'nf': 1, 'nd1': 0, 'nd2': 0, 'nd3': 0}

PSF Info Keywords:
{'fov_pix': 33, 'oversample': 4, 'offset_r': 0, 'offset_theta': 0, 'tel_pupil': None,
↪'save': True, 'force': False, 'opd': ('OPD_RevW_ote_for_NIRCam_requirements.fits.gz
↪', 0), 'jitter': 'gaussian', 'jitter_sigma': 0.007}
```

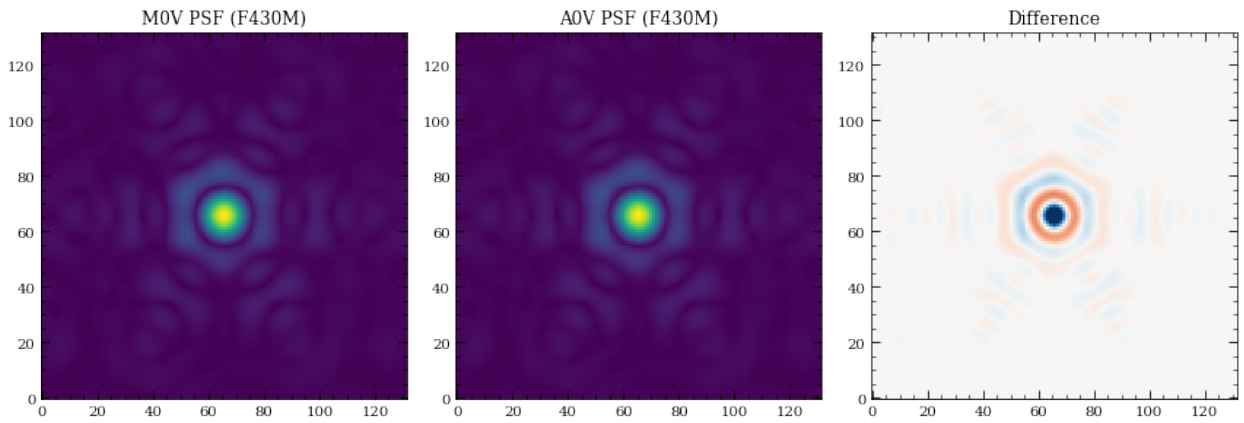
PSF coefficient information is stored in the `psf_coeff` attribute. This data is accessed by many of the NIRCam class functions to generate PSFs with arbitrary wavelength weights, such as the `gen_psf()` function.

[8]:
```
# Demonstrate the color difference of the PSF for different spectral types, same␣
↪magnitude
sp_M0V = pynrc.stellar_spectrum('M0V', 10, 'vegamag', nrc.bandpass)
sp_A0V = pynrc.stellar_spectrum('A0V', 10, 'vegamag', nrc.bandpass)

# Generate oversampled PSFs (counts/sec)
```

(continues on next page)

(continued from previous page)

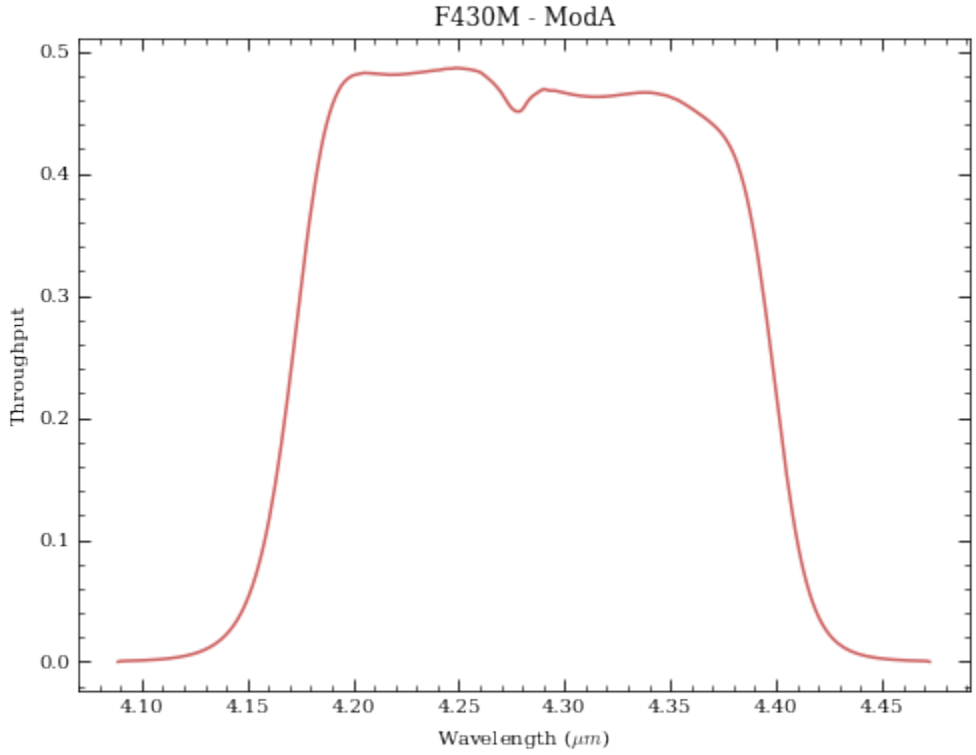
```
_, psf_M0V = nrc.gen_psf(sp_M0V, return_oversample=True)
_, psf_A0V = nrc.gen_psf(sp_A0V, return_oversample=True)

fig, axes = plt.subplots(1,3, figsize=(12,4))

axes[0].imshow(psf_M0V**0.5)
axes[0].set_title('M0V PSF ({})'.format(nrc.filter))
axes[1].imshow(psf_A0V**0.5)
axes[1].set_title('A0V PSF ({})'.format(nrc.filter))

diff = psf_M0V - psf_A0V

minmax = np.abs(diff).max() / 2
axes[2].imshow(diff, cmap='RdBu', vmin=-minmax, vmax=minmax)
axes[2].set_title('Difference')

fig.tight_layout()
```



Bandpass information is stored in the `bandpass` attribute and can be plotted with the convenience function `plot_bandpass()`.

[9]:
```
nrc.plot_bandpass()
```

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c2d8a6978>
```



1.4.3 1. Saturation Limits

One of the most basic functions is to determine the saturation limit of a CDS observation, so let's try this for the current filter selection. Generally, saturation is considered to be 80% of the full well.

[10]:
```
# Turn off those pesky informational texts
pynrc.setup_logging('WARN', verbose=False)

# Configure the observation for CDS frames (ngroup=2)
# Print out frame and ramp information using verbose=True
nrc.update_detectors(ngroup=2, verbose=True)
```

```
New Ramp Settings:
  read_mode :    RAPID
  nf        :        1
  nd2       :        0
  ngroup    :        2
  nint      :        1
New Detector Settings
  wind_mode :     FULL
  xpix      :     2048
  ypix      :     2048
  x0        :        0
  y0        :        0
New Ramp Times
  t_group   :   10.737
  t_frame   :   10.737
  t_int     :   21.474
  t_int_tot :   32.210
```

(continues on next page)

(continued from previous page)

```
  t_exp     :   21.474
  t_acq     :   32.216
```

The `sat_limits()` function returns a dictionary of results. There's the option in include a Pysynphot spectrum, but if none is specificed the it defaults to a G2V star.

[11]:
```
# Set verbose=True to print results in a user-friendly manner
sat_lims = nrc.sat_limits(verbose=True)

# Dictionary information
print("\nDictionary Info:", sat_lims)
```

```
F430M Saturation Limit assuming G2V source: 12.18 vegamag

Dictionary Info: {'satmag': 12.180647544008966, 'units': 'vegamag', 'Spectrum': 'G2V',
↪ 'bp_lim': 'F430M'}
```

By default, the function `sat_limits()` uses a G2V stellar spectrum, but any arbitrary spectrum can be passed via the `sp` keyword. In addition, using the `bp_lim` keyword, you can use spectral information to determine the brightness in some other bandpass that saturates the source within the NIRCam filter.

[12]:
```
# Spectrum of an M0V star (not normalized)
sp_M0V = pynrc.stellar_spectrum('M0V')
# 2MASS Ks Bandpass
bp_k = pynrc.bp_2mass('K')

sat_lims = nrc.sat_limits(sp=sp_M0V, bp_lim=bp_k, verbose=True)
```

```
Ks-Band Saturation Limit for F430M assuming M0V source: 12.37 vegamag
```

Now, let's get the same saturation limit assuming a 128x128 subarray.

[13]:
```
nrc.update_detectors(wind_mode='WINDOW', xpix=128, ypix=128)
sat_lims = nrc.sat_limits(sp=sp_M0V, bp_lim=bp_k, verbose=True)
```

```
Ks-Band Saturation Limit for F430M assuming M0V source: 7.94 vegamag
```

You can also use the `saturation_levels()` function to generate an image of a point source indicating the fractional well fill level.

[14]:
```
# Spectum of A0V star with Ks = 8 mag
sp = pynrc.stellar_spectrum('M0V', 8, 'vegamag', bp_k)
sat_levels = nrc.saturation_levels(sp, full_size=False, ngroup=nrc.det_info['ngroup'])

print('Max Well Fraction: {:.2f}'.format(sat_levels.max()))
```

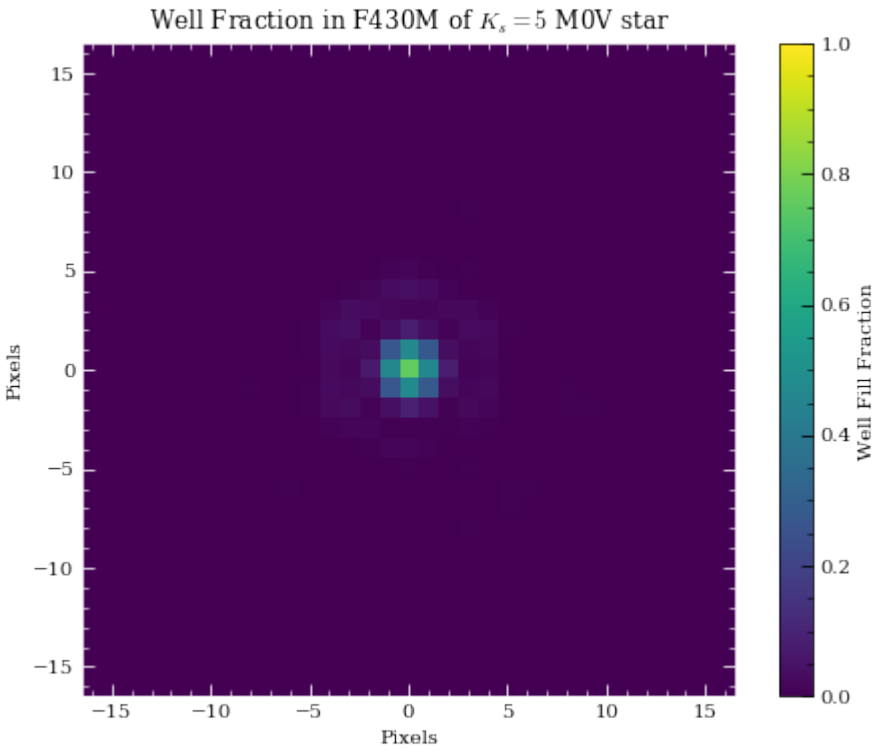
```
Max Well Fraction: 0.76
```

[15]:
```
# Plot the well fill levels for each pixel
fig, ax = plt.subplots(1,1)

extent = 0.5*nrc.psf_info['fov_pix'] * np.array([-1,1,-1,1])
cax = ax.imshow(sat_levels, extent=extent, vmin=0, vmax=1)
ax.set_xlabel('Pixels')
ax.set_ylabel('Pixels')
ax.set_title('Well Fraction in {} of $K_s = 5$ M0V star'.format(nrc.filter))
```

(continues on next page)

(continued from previous page)

```
cbar = fig.colorbar(cax)
cbar.set_label('Well Fill Fraction')

ax.tick_params(axis='both', color='white', which='both')
for k in ax.spines.keys():
    ax.spines[k].set_color('white')
```



Information for slitless grism observations show wavelength-dependent results.

[16]:
```
nrc = pynrc.NIRCam('F444W', pupil='GRISM0', ngroup=2, wind_mode='STRIPE', ypix=128)
sat_lims = nrc.sat_limits(sp=sp_M0V, bp_lim=bp_k, verbose=True)
```

```
Ks-Band Saturation Limit for F444W assuming M0V source:
   Wave    Sat Limit (vegamag)
--------- -------------------
     3.90                4.50
     4.00                4.49
     4.10                4.38
     4.20                4.26
     4.30                4.11
     4.40                3.87
     4.50                3.69
     4.60                3.53
     4.70                3.35
     4.80                3.18
     4.90                2.97
     5.00                2.38
```

1.4.4 2. Sensitivity Limits

Similarly, we can determine sensitivity limits of point sources (and extended sources) for the defined instrument configuration. By default, the `sensitivity()` function uses a flat spectrum. In this case, let's find the sensitivities NIRCam can reach in a single ~1000sec integration with the F430M filter. Noise values will depend on the exact `MULTIACCUM` settings.

[17]:
```
nrc = pynrc.NIRCam('F430M')
nrc.update_detectors(read_mode='MEDIUM8', ngroup=10)

# The multiaccum_times attribute describes the various timing information
print(nrc.multiaccum_times)
```

```
{'t_frame': 10.73677, 't_group': 107.3677, 't_int': 1052.20346, 't_exp': 1052.20346,
↪'t_acq': 1062.94547, 't_int_tot': 1062.94023}
```

[18]:
```
sens = nrc.sensitivity(nsig=5, units='vegamag', verbose=True)
```

```
Point Source Sensitivity (5-sigma): 23.32 vegamag
Surface Brightness Sensitivity (5-sigma): 21.22 vegamag/arcsec^2
```

The sensitivity function also includes a keyword `forwardSNR`, which allows the user to pass a normalized spectrum and estimate the SNR For some extraction aperture.

[19]:
```
sp = pynrc.stellar_spectrum('M0V', 20, 'vegamag', nrc.bandpass)
snr = nrc.sensitivity(sp=sp, forwardSNR=True, units='vegamag', verbose=True)
```

```
Point Source SNR (20.00 vegamag): 75.53 sigma
Surface Brightness SNR (20.00 vegamag/arcsec^2): 14.47 sigma
```

1.4.5 3. Ramp Optimization

Armed with these two basic functions, we can attempt to determine the best instrument settings to optimize for SNR and efficiency. In these types of optimizations, we must consider observational constraints such as saturation levels, SNR requirements, and limits on acquisition time.

Note: The reported acquisition times do not include obsevatory and instrument-level overheads, such as slew times, filter changes, script compilations, etc. It only includes detector readout times (including reset frames and Fast Row Resets).

For instance, we want to observe an M-Dwarf (K=18 mag) in the F430M filter. What is the most efficient configuration to obtain an SNR of 100?

[23]:
```
# Setup observation
nrc = pynrc.NIRCam('F430M', wind_mode='WINDOW', xpix=160, ypix=160)

# Spectrum of an M2V star
bp_k = pynrc.bp_2mass('K')
sp_M0V = pynrc.stellar_spectrum('M2V', 18, 'vegamag', bp_k)
```

[24]:
```
# Run optimizer. Result is a ranked list sorted by efficiency.
tbl = nrc.ramp_optimize(sp_M0V, snr_goal=100, ng_min=5, nint_min=10, verbose=True)
```

```
BRIGHT1
BRIGHT2
DEEP2
DEEP8
```

(continues on next page)

(continued from previous page)

```
MEDIUM2
MEDIUM8
RAPID
SHALLOW2
SHALLOW4
 Pattern   NGRP NINT   t_int     t_exp     t_acq     SNR      Well     eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
DEEP8         5   10     24.52    245.20    248.04    124.5    0.006    7.904
MEDIUM8       6   12     16.16    193.93    197.34    100.2    0.004    7.133
MEDIUM8       6   13     16.16    210.09    213.78    104.3    0.004    7.133
MEDIUM2       8   10     20.06    200.62    203.46     98.6    0.005    6.913
MEDIUM2       8   11     20.06    220.68    223.80    103.4    0.005    6.913
SHALLOW4     10   15     13.65    204.80    209.06     99.4    0.003    6.875
SHALLOW4     10   16     13.65    218.45    222.99    102.7    0.003    6.875
DEEP2         5   10     22.85    228.48    231.32    102.9    0.005    6.766
MEDIUM8       5   18     13.37    240.74    245.85    104.3    0.003    6.654
SHALLOW4      9   20     12.26    245.20    250.88    104.3    0.003    6.583
...         ...  ...       ...       ...       ...      ...      ...      ...
RAPID        10  511      2.79   1423.85   1568.87    101.1    0.001    2.551
RAPID        10  512      2.79   1426.64   1571.94    101.2    0.001    2.551
RAPID        10  513      2.79   1429.42   1575.01    101.3    0.001    2.551
RAPID        10  514      2.79   1432.21   1578.08    101.4    0.001    2.551
RAPID        10  515      2.79   1435.00   1581.15    101.5    0.001    2.551
RAPID        10  516      2.79   1437.78   1584.22    101.6    0.001    2.551
RAPID        10  517      2.79   1440.57   1587.29    101.7    0.001    2.551
RAPID        10  518      2.79   1443.36   1590.36    101.7    0.001    2.551
RAPID        10  519      2.79   1446.14   1593.43    101.8    0.001    2.551
RAPID        10  520      2.79   1448.93   1596.50    101.9    0.001    2.551
RAPID        10  521      2.79   1451.71   1599.57    102.0    0.001    2.551
Length = 50 rows
```

For a slightly more complicated scenario, consider an additional foreground source. In this scenario, the F0V star will saturate much more quickly compared to the fainter M2V, so it limits which ramp settings we may want to use (assuming we want unsaturated frames, which isn't always necessarily true).

[25]:
```
sp_F0V = pynrc.stellar_spectrum('F0V', 10, 'vegamag', bp_k)
tbl = nrc.ramp_optimize(sp_M0V, sp_bright=sp_F0V,
                        snr_goal=100, ng_min=5, nint_min=10, verbose=True)
```

```
BRIGHT1
BRIGHT2
DEEP2
DEEP8
MEDIUM2
MEDIUM8
RAPID
SHALLOW2
SHALLOW4
 Pattern   NGRP NINT   t_int     t_exp     t_acq     SNR      Well     eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
RAPID        10  500      2.79   1393.20   1535.10    100.0    0.778    2.551
RAPID        10  501      2.79   1395.99   1538.17    100.1    0.778    2.551
RAPID        10  502      2.79   1398.77   1541.24    100.2    0.778    2.551
RAPID        10  503      2.79   1401.56   1544.31    100.3    0.778    2.551
RAPID        10  504      2.79   1404.35   1547.38    100.4    0.778    2.551
RAPID        10  505      2.79   1407.13   1550.45    100.5    0.778    2.551
RAPID        10  506      2.79   1409.92   1553.52    100.6    0.778    2.551
```

(continues on next page)

(continued from previous page)

```
RAPID        10  507      2.79   1412.70   1556.59    100.7    0.778    2.551
RAPID        10  508      2.79   1415.49   1559.66    100.8    0.778    2.551
RAPID        10  509      2.79   1418.28   1562.73    100.9    0.778    2.551
...         ...  ...       ...       ...       ...      ...      ...      ...
BRIGHT1       5  960      2.51   2407.45   2679.90    101.5    0.700    1.960
BRIGHT1       5  961      2.51   2409.96   2682.69    101.6    0.700    1.960
BRIGHT1       5  962      2.51   2412.47   2685.48    101.6    0.700    1.960
BRIGHT1       5  963      2.51   2414.97   2688.27    101.7    0.700    1.960
BRIGHT1       5  964      2.51   2417.48   2691.06    101.7    0.700    1.960
BRIGHT1       5  965      2.51   2419.99   2693.86    101.8    0.700    1.960
BRIGHT1       5  966      2.51   2422.50   2696.65    101.8    0.700    1.960
BRIGHT1       5  967      2.51   2425.00   2699.44    101.9    0.700    1.960
BRIGHT1       5  968      2.51   2427.51   2702.23    101.9    0.700    1.960
BRIGHT1       5  969      2.51   2430.02   2705.02    102.0    0.700    1.960
BRIGHT1       5  970      2.51   2432.53   2707.81    102.0    0.700    1.960
Length = 85 rows
```

If there are no objections to saturating the bright source, then we can set the `well_frac_max` parameter to something like 5 times the hard saturation limit. This allows for more efficient exposure settings.

[26]:
```
tbl = nrc.ramp_optimize(sp_M0V, sp_bright=sp_F0V, well_frac_max=5,
                        snr_goal=100, ng_min=5, nint_min=10, verbose=True)
```

```
BRIGHT1
BRIGHT2
DEEP2
DEEP8
MEDIUM2
MEDIUM8
RAPID
SHALLOW2
SHALLOW4
 Pattern   NGRP NINT   t_int     t_exp     t_acq     SNR      Well     eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
MEDIUM8       6   12     16.16    193.93    197.34    100.2    4.510    7.133
MEDIUM8       6   13     16.16    210.09    213.78    104.3    4.510    7.133
SHALLOW4     10   15     13.65    204.80    209.06     99.4    3.810    6.875
SHALLOW4     10   16     13.65    218.45    222.99    102.7    3.810    6.875
MEDIUM8       5   18     13.37    240.74    245.85    104.3    3.732    6.654
SHALLOW4      9   20     12.26    245.20    250.88    104.3    3.421    6.583
MEDIUM2       7   13     17.28    224.58    228.27     98.3    4.821    6.506
MEDIUM2       7   14     17.28    241.86    245.83    102.0    4.821    6.506
SHALLOW2     10   19     13.10    248.83    254.22    100.2    3.655    6.283
SHALLOW2     10   20     13.10    261.92    267.60    102.8    3.655    6.283
...         ...  ...       ...       ...       ...      ...      ...      ...
RAPID        10  511      2.79   1423.85   1568.87    101.1    0.778    2.551
RAPID        10  512      2.79   1426.64   1571.94    101.2    0.778    2.551
RAPID        10  513      2.79   1429.42   1575.01    101.3    0.778    2.551
RAPID        10  514      2.79   1432.21   1578.08    101.4    0.778    2.551
RAPID        10  515      2.79   1435.00   1581.15    101.5    0.778    2.551
RAPID        10  516      2.79   1437.78   1584.22    101.6    0.778    2.551
RAPID        10  517      2.79   1440.57   1587.29    101.7    0.778    2.551
RAPID        10  518      2.79   1443.36   1590.36    101.7    0.778    2.551
RAPID        10  519      2.79   1446.14   1593.43    101.8    0.778    2.551
RAPID        10  520      2.79   1448.93   1596.50    101.9    0.778    2.551
RAPID        10  521      2.79   1451.71   1599.57    102.0    0.778    2.551
Length = 49 rows
```

[ ]:

1.5 Ramp Optimization Examples

This notebook outlines an example to optimize the ramp settings for a few different types of observations.

In these types of optimizations, we must consider observations constraints such as saturation levels, SNR requirements, and limits on acquisition time.

Note: The reported acquisition time does not include obsevatory and instrument-level overheads, such as slew times, filter changes, script compilations, etc. It only includes detector readout times (including reset frames).

[1]:
```
# Makes print and division act like Python 3
from __future__ import print_function, division

# Import the usual libraries
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

# Enable inline plotting at lower left
%matplotlib inline

from IPython.display import display, Latex, clear_output
```

[2]:
```
import pynrc
from pynrc import nrc_utils
from pynrc.nrc_utils import S
from pynrc.pynrc_core import table_filter

pynrc.setup_logging('WARNING', verbose=False)

from astropy.table import Table
```

1.5.1 Example 1: M-Dwarf companion (imaging vs coronagraphy)

We want to observe an M-Dwarf companion (K=18 mag) in the vicinity of a brighter F0V (K=13 mag) in the F430M filter. Assume the M-Dwarf flux is not significantly impacted by the brighter PSF (ie., in the background limited regime). In this scenario, the F0V star will saturate much more quickly compared to the fainter companion, so it limits which ramp settings we can use.

We will test a couple different types of observations (direct imaging vs coronagraphy).

[3]:
```
# Get stellar spectra and normalize at K-Band
# The stellar_spectrum convenience function creates a Pysynphot spectrum
bp_k = S.ObsBandpass('k')
sp_M2V = pynrc.stellar_spectrum('M2V', 18, 'vegamag', bp_k)#, catname='ck04models')
sp_F0V = pynrc.stellar_spectrum('F0V', 13, 'vegamag', bp_k)#, catname='ck04models')
```

[4]:
```
# Initiate a NIRCam observation
nrc = pynrc.NIRCam('F430M', wind_mode='WINDOW', xpix=160, ypix=160)
```

[5]:
```
# Set some observing constraints
# Let's assume we want photometry on the primary to calibrate the M-Dwarf for direct imaging
#  - Set well_frac_max=0.75
# Want a SNR~100 in the F430M filter
#  - Set snr_goal=100
res = nrc.ramp_optimize(sp_M2V, sp_bright=sp_F0V, snr_goal=100, well_frac_max=0.75, verbose=True)
```

```
BRIGHT1
BRIGHT2
DEEP2
DEEP8
MEDIUM2
MEDIUM8
RAPID
SHALLOW2
SHALLOW4
 Pattern   NGRP NINT   t_int     t_exp     t_acq     SNR      Well     eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
DEEP8         7    4     35.67    142.66    143.80    101.7    0.628    8.478
DEEP8         7    5     35.67    178.33    179.75    113.7    0.628    8.478
DEEP2         8    4     39.57    158.27    159.40    101.1    0.696    8.009
DEEP2         8    5     39.57    197.83    199.25    113.1    0.696    8.009
MEDIUM2      10    7     25.63    179.44    181.43    101.3    0.451    7.524
MEDIUM2      10    8     25.63    205.08    207.35    108.3    0.451    7.524
MEDIUM8       7    9     18.95    170.53    173.08     98.5    0.333    7.488
MEDIUM8       7   10     18.95    189.48    192.31    103.8    0.333    7.488
MEDIUM8       6   13     16.16    210.09    213.78    104.2    0.284    7.125
SHALLOW4     10   15     13.65    204.80    209.06     99.3    0.240    6.867
...         ...  ...       ...       ...       ...      ...      ...      ...
RAPID        10  513      2.79   1429.42   1575.01    101.1    0.049    2.547
RAPID        10  514      2.79   1432.21   1578.08    101.2    0.049    2.547
RAPID        10  515      2.79   1435.00   1581.15    101.3    0.049    2.547
RAPID        10  516      2.79   1437.78   1584.22    101.4    0.049    2.547
RAPID        10  517      2.79   1440.57   1587.29    101.5    0.049    2.547
RAPID        10  518      2.79   1443.36   1590.36    101.6    0.049    2.547
RAPID        10  519      2.79   1446.14   1593.43    101.7    0.049    2.547
RAPID        10  520      2.79   1448.93   1596.50    101.8    0.049    2.547
RAPID        10  521      2.79   1451.71   1599.57    101.9    0.049    2.547
RAPID        10  522      2.79   1454.50   1602.64    102.0    0.049    2.547
RAPID        10  523      2.79   1457.29   1605.71    102.1    0.049    2.547
Length = 54 rows
```

[6]:
```
# Print the Top 2 settings for each readout pattern
res2 = table_filter(res, 2)
print(res2)
```

```
 Pattern   NGRP NINT   t_int     t_exp     t_acq     SNR      Well     eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
RAPID        10  501      2.79   1395.99   1538.17     99.9    0.049    2.547
RAPID        10  502      2.79   1398.77   1541.24    100.0    0.049    2.547
BRIGHT1      10  135      5.29    714.71    753.02     99.7    0.093    3.632
BRIGHT1      10  136      5.29    720.01    758.60    100.1    0.093    3.632
BRIGHT2      10   87      5.57    484.83    509.52     99.6    0.098    4.414
BRIGHT2      10   88      5.57    490.41    515.38    100.2    0.098    4.414
SHALLOW2     10   19     13.10    248.83    254.22    100.0    0.230    6.274
SHALLOW2     10   20     13.10    261.92    267.60    102.6    0.230    6.274
```

(continues on next page)

(continued from previous page)

```
SHALLOW4     10   15     13.65    204.80    209.06     99.3    0.240    6.867
SHALLOW4     10   16     13.65    218.45    222.99    102.5    0.240    6.867
MEDIUM2      10    7     25.63    179.44    181.43    101.3    0.451    7.524
MEDIUM2      10    8     25.63    205.08    207.35    108.3    0.451    7.524
MEDIUM8       7    9     18.95    170.53    173.08     98.5    0.333    7.488
MEDIUM8       7   10     18.95    189.48    192.31    103.8    0.333    7.488
DEEP2         8    4     39.57    158.27    159.40    101.1    0.696    8.009
DEEP2         8    5     39.57    197.83    199.25    113.1    0.696    8.009
DEEP8         7    4     35.67    142.66    143.80    101.7    0.628    8.478
DEEP8         7    5     35.67    178.33    179.75    113.7    0.628    8.478
```

[7]:
```
# Do the same thing, but for coronagraphic mask instead
#pynrc.setup_logging('DEBUG', verbose=False)
nrc = pynrc.NIRCam('F430M', mask='MASK430R', pupil='CIRCLYOT',
                   wind_mode='WINDOW', xpix=320, ypix=320)

# We assume that longer ramps will give us the best SNR for time
patterns = ['MEDIUM8', 'DEEP8']
res = nrc.ramp_optimize(sp_M2V, sp_bright=sp_F0V, snr_goal=100,
                        patterns=patterns, even_nints=True)

# Take the Top 2 settings for each readout pattern
res2 = table_filter(res, 2)
print(res2)
```

```
 Pattern   NGRP NINT   t_int      t_exp      t_acq      SNR       Well      eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
MEDIUM8      10   84    104.77   8800.34   8891.81    100.4    0.001    1.065
MEDIUM8      10   86    104.77   9009.87   9103.52    101.6    0.001    1.065
DEEP8        20   12    414.79   4977.45   4990.52    102.5    0.003    1.451
DEEP8        19   12    393.41   4720.88   4733.95     99.3    0.003    1.442
```

RESULTS

Based on these two comparisons, it looks like direct imaging is much more efficient in getting to the requisite SNR. In addition, direct imaging gives us a photometric comparison source that is inaccessible with the coronagraph masks.

1.5.2 Example 2: Exoplanet Coronagraphy

We want to observe GJ 504 for an hour in the F444W filter. - What is the optimal ramp settings to maximize the SNR of GJ 504b? - What is the final background sensitivity limit?

[8]:
```
# Get stellar spectra and normalize at K-Band
# The stellar_spectrum convenience function creates a Pysynphot spectrum
bp_k = pynrc.bp_2mass('ks')
sp_G0V = pynrc.stellar_spectrum('G0V', 4, 'vegamag', bp_k)

# Choose a representative planet spectrum
planet = pynrc.planets_sb12(atmo='hy3s', mass=8, age=200, entropy=8, distance=17.5)
sp_pl = planet.export_pysynphot()

# Renormalize to F360M = 18.8
bp_l = pynrc.read_filter('F360M') #
sp_pl = sp_pl.renorm(18.8, 'vegamag', bp_l)
```

[9]:
```
# Initiate a NIRCam observation
nrc = pynrc.NIRCam('F444W', pupil='CIRCLYOT', mask='MASK430R', wind_mode='WINDOW',␣
↪xpix=320, ypix=320)
```

[10]:
```
# Set even_nints=True assume 2 roll angles
res = nrc.ramp_optimize(sp_pl, sp_bright=sp_G0V, tacq_max=3600, tacq_frac=0.05,
                        even_nints=True, verbose=True)
```

```
BRIGHT1
BRIGHT2
DEEP2
DEEP8
MEDIUM2
MEDIUM8
RAPID
SHALLOW2
SHALLOW4
 Pattern   NGRP NINT   t_int     t_exp     t_acq     SNR      Well     eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
DEEP8         6   30    115.46   3463.69   3496.36    665.3    0.717   11.251
DEEP8         6   32    115.46   3694.60   3729.45    687.1    0.717   11.251
MEDIUM2      10   34     98.35   3343.96   3380.98    652.6    0.611   11.222
MEDIUM2      10   36     98.35   3540.66   3579.86    671.5    0.611   11.222
MEDIUM2      10   38     98.35   3737.36   3778.74    689.9    0.611   11.222
MEDIUM8      10   32    104.77   3352.51   3387.36    652.0    0.651   11.202
MEDIUM8      10   34    104.77   3562.04   3599.07    672.1    0.651   11.202
MEDIUM8      10   36    104.77   3771.57   3810.78    691.6    0.651   11.202
DEEP2         6   32    109.04   3489.35   3524.19    660.4    0.677   11.125
DEEP2         6   34    109.04   3707.43   3744.46    680.8    0.677   11.125
...         ...  ...       ...       ...       ...      ...      ...      ...
SHALLOW2     10   72     50.24   3617.63   3696.04    639.0    0.312   10.510
BRIGHT2      10  158     21.38   3378.17   3550.22    526.0    0.133    8.828
BRIGHT2      10  160     21.38   3420.93   3595.16    529.3    0.133    8.828
BRIGHT2      10  162     21.38   3463.69   3640.10    532.6    0.133    8.828
BRIGHT1      10  166     20.31   3371.75   3552.52    477.3    0.126    8.007
BRIGHT1      10  168     20.31   3412.38   3595.32    480.1    0.126    8.007
BRIGHT1      10  170     20.31   3453.00   3638.12    483.0    0.126    8.007
RAPID        10  304     10.69   3249.88   3580.93    371.1    0.066    6.200
RAPID        10  306     10.69   3271.26   3604.48    372.3    0.066    6.200
RAPID        10  308     10.69   3292.64   3628.04    373.5    0.066    6.200
RAPID         9  334      9.62   3213.53   3577.25    343.4    0.060    5.741
Length = 26 rows
```

[11]:
```
# Take the Top 2 settings for each readout pattern
res2 = table_filter(res, 2)
print(res2)
```

```
 Pattern   NGRP NINT   t_int     t_exp     t_acq     SNR      Well     eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
RAPID        10  304     10.69   3249.88   3580.93    371.1    0.066    6.200
RAPID        10  306     10.69   3271.26   3604.48    372.3    0.066    6.200
BRIGHT1      10  166     20.31   3371.75   3552.52    477.3    0.126    8.007
BRIGHT1      10  168     20.31   3412.38   3595.32    480.1    0.126    8.007
BRIGHT2      10  158     21.38   3378.17   3550.22    526.0    0.133    8.828
BRIGHT2      10  160     21.38   3420.93   3595.16    529.3    0.133    8.828
SHALLOW2     10   68     50.24   3416.65   3490.70    621.0    0.312   10.510
SHALLOW2     10   70     50.24   3517.14   3593.37    630.0    0.312   10.510
SHALLOW4     10   66     52.38   3457.28   3529.15    634.4    0.325   10.679
```

(continues on next page)

(continued from previous page)

```
SHALLOW4     10   68     52.38   3562.04   3636.09    644.0    0.325   10.679
MEDIUM2      10   34     98.35   3343.96   3380.98    652.6    0.611   11.222
MEDIUM2      10   36     98.35   3540.66   3579.86    671.5    0.611   11.222
MEDIUM8      10   32    104.77   3352.51   3387.36    652.0    0.651   11.202
MEDIUM8      10   34    104.77   3562.04   3599.07    672.1    0.651   11.202
DEEP2         6   32    109.04   3489.35   3524.19    660.4    0.677   11.125
DEEP2         6   34    109.04   3707.43   3744.46    680.8    0.677   11.125
DEEP8         6   30    115.46   3463.69   3496.36    665.3    0.717   11.251
DEEP8         6   32    115.46   3694.60   3729.45    687.1    0.717   11.251
```

[12]:
```
# The DEEP and MEDIUMs are very similar.
# Let's go with MEDIUM2 for more GROUPS & INTS
# and slightly better efficiency over MEDIUM8
nrc.update_detectors(read_mode='MEDIUM2', ngroup=10, nint=36)

keys = list(nrc.multiaccum_times.keys())
keys.sort()
for k in keys:
    print("{:<10}: {: 12.5f}".format(k, nrc.multiaccum_times[k]))
```

```
t_acq     :   3579.86304
t_exp     :   3540.66048
t_frame   :      1.06904
t_group   :     10.69040
t_int     :     98.35168
t_int_tot :     99.44064
```

[13]:
```
# Background sensitivity (5 sigma)
sens_dict = nrc.sensitivity(nsig=5, units='vegamag', verbose=True)
```

```
Point Source Sensitivity (5-sigma): 21.75 vegamag
Surface Brightness Sensitivity (5-sigma): 23.01 vegamag/arcsec^2
```

1.5.3 Example 3: Single-Object Grism Spectroscopy

Similar to the above, but instead we want to obtain a slitless grism spectrum of a K=12 mag M9V dwarf. Each grism resolution element should have SNR~100.

[14]:
```
# M9V star at K=12 mags
bp_k = S.ObsBandpass('k')
sp_M9V = pynrc.stellar_spectrum('M9V', 12, 'vegamag', bp_k)
```

[15]:
```
nrc = pynrc.NIRCam('F444W', pupil='GRISM0', wind_mode='STRIPE', ypix=64)
```

[16]:
```
# Set a minimum of 10 integrations to be robust against cosmic rays
# Also set a minimum of 10 groups for good ramp sampling
res = nrc.ramp_optimize(sp_M9V, snr_goal=100, nint_min=10, ng_min=10, verbose=True)
```

```
BRIGHT1
BRIGHT2
DEEP2
DEEP8
MEDIUM2
MEDIUM8
RAPID
```

(continues on next page)

(continued from previous page)

```
SHALLOW2
SHALLOW4
 Pattern   NGRP NINT   t_int     t_exp     t_acq     SNR      Well     eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
DEEP8        10   10     64.03    640.35    643.81    249.0    0.041    9.814
DEEP2        10   10     61.99    619.91    623.37    244.2    0.040    9.779
MEDIUM8      10   10     33.38    333.80    337.26    173.2    0.021    9.433
MEDIUM2      10   10     31.34    313.36    316.82    166.9    0.020    9.375
SHALLOW4     10   10     16.69    166.90    170.36    115.7    0.011    8.867
SHALLOW2     10   10     16.01    160.09    163.55    110.7    0.010    8.654
BRIGHT2      10   27      6.81    183.93    193.27     98.5    0.004    7.082
BRIGHT2      10   28      6.81    190.74    200.43    100.3    0.004    7.082
BRIGHT2      10   29      6.81    197.55    207.59    102.0    0.004    7.082
BRIGHT1      10   36      6.47    232.98    245.43     98.9    0.004    6.313
BRIGHT1      10   37      6.47    239.45    252.25    100.3    0.004    6.313
BRIGHT1      10   38      6.47    245.92    259.07    101.6    0.004    6.313
BRIGHT1      10   39      6.47    252.39    265.89    102.9    0.004    6.313
RAPID        10  116      3.41    395.11    435.23     99.9    0.002    4.786
RAPID        10  117      3.41    398.51    438.98    100.3    0.002    4.786
RAPID        10  118      3.41    401.92    442.74    100.7    0.002    4.786
RAPID        10  119      3.41    405.33    446.49    101.1    0.002    4.786
RAPID        10  120      3.41    408.73    450.24    101.6    0.002    4.786
RAPID        10  121      3.41    412.14    453.99    102.0    0.002    4.786
RAPID        10  122      3.41    415.54    457.74    102.4    0.002    4.786
```

[17]:
```
# Print the Top 2 settings for each readout pattern
res2 = table_filter(res, 2)
print(res2)
```

```
 Pattern   NGRP NINT   t_int     t_exp     t_acq     SNR      Well     eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
RAPID        10  116      3.41    395.11    435.23     99.9    0.002    4.786
RAPID        10  117      3.41    398.51    438.98    100.3    0.002    4.786
BRIGHT1      10   36      6.47    232.98    245.43     98.9    0.004    6.313
BRIGHT1      10   37      6.47    239.45    252.25    100.3    0.004    6.313
BRIGHT2      10   27      6.81    183.93    193.27     98.5    0.004    7.082
BRIGHT2      10   28      6.81    190.74    200.43    100.3    0.004    7.082
SHALLOW2     10   10     16.01    160.09    163.55    110.7    0.010    8.654
SHALLOW4     10   10     16.69    166.90    170.36    115.7    0.011    8.867
MEDIUM2      10   10     31.34    313.36    316.82    166.9    0.020    9.375
MEDIUM8      10   10     33.38    333.80    337.26    173.2    0.021    9.433
DEEP2        10   10     61.99    619.91    623.37    244.2    0.040    9.779
DEEP8        10   10     64.03    640.35    643.81    249.0    0.041    9.814
```

[18]:
```
# Let's say we choose SHALLOW4, NGRP=10, NINT=10
# Update detector readout
nrc.update_detectors(read_mode='SHALLOW4', ngroup=10, nint=10)

keys = list(nrc.multiaccum_times.keys())
keys.sort()
for k in keys:
    print("{:<10}: {: 12.5f}".format(k, nrc.multiaccum_times[k]))
```

```
t_acq     :    170.36264
t_exp     :    166.89890
t_frame   :      0.34061
t_group   :      1.70305
```

(continues on next page)

(continued from previous page)

```
t_int     :     16.68989
t_int_tot :     17.03574
```

[19]:
```
# Print final wavelength-dependent SNR
# For spectroscopy, the snr_goal is the median over the bandpass
snr_dict = nrc.sensitivity(sp=sp_M9V, forwardSNR=True, units='mJy', verbose=True)
```

```
F444W SNR for M9V source
   Wave      SNR    Flux (mJy)
--------- --------- ----------
     3.80      6.03       5.01
     3.90    166.73       4.85
     4.00    177.67       4.62
     4.10    169.44       4.40
     4.20    159.49       4.17
     4.30    143.53       3.70
     4.40    132.39       3.57
     4.50    120.79       3.20
     4.60    110.68       3.16
     4.70     99.12       2.98
     4.80     92.55       3.03
     4.90     80.37       2.92
     5.00     38.29       2.67
     5.10      0.75       2.76
```

Mock observed spectrum

Create a series of ramp integrations based on the current NIRCam settings. The gen_exposures() function creates a series of mock observations in raw DMS format by default. By default, it's point source objects centered in the observing window.

[20]:
```
# Ideal spectrum and wavelength solution
wspec, imspec = nrc.gen_psf(sp=sp_M9V)

# Resize to detector window
nx = nrc.det_info['xpix']
ny = nrc.det_info['ypix']
```

[21]:
```
# Create a series of ramp integrations based on the current NIRCam settings
# Output is 10 HDULists
im_slope = imspec + nrc.bg_zodi()
res = nrc.gen_exposures(im_slope=im_slope, return_results=True, targ_name='sp_M9V')
```

[22]:
```
header = res[0]['PRIMARY'].header
tvals = (np.arange(header['NGROUPS'])+1) * header['TGROUP']
slope_list = []

for hdul in res:
    header = hdul['PRIMARY'].header
    data = hdul['SCI'].data
    ref = pynrc.ref_pixels.NRC_refs(data, header, DMS=True, do_all=False)
    ref.calc_avg_amps()
    ref.correct_amp_refs()

    # Linear fit to determine slope image
    cf = nrc_utils.jl_poly_fit(tvals, ref.data, deg=1)
```

(continues on next page)

(continued from previous page)

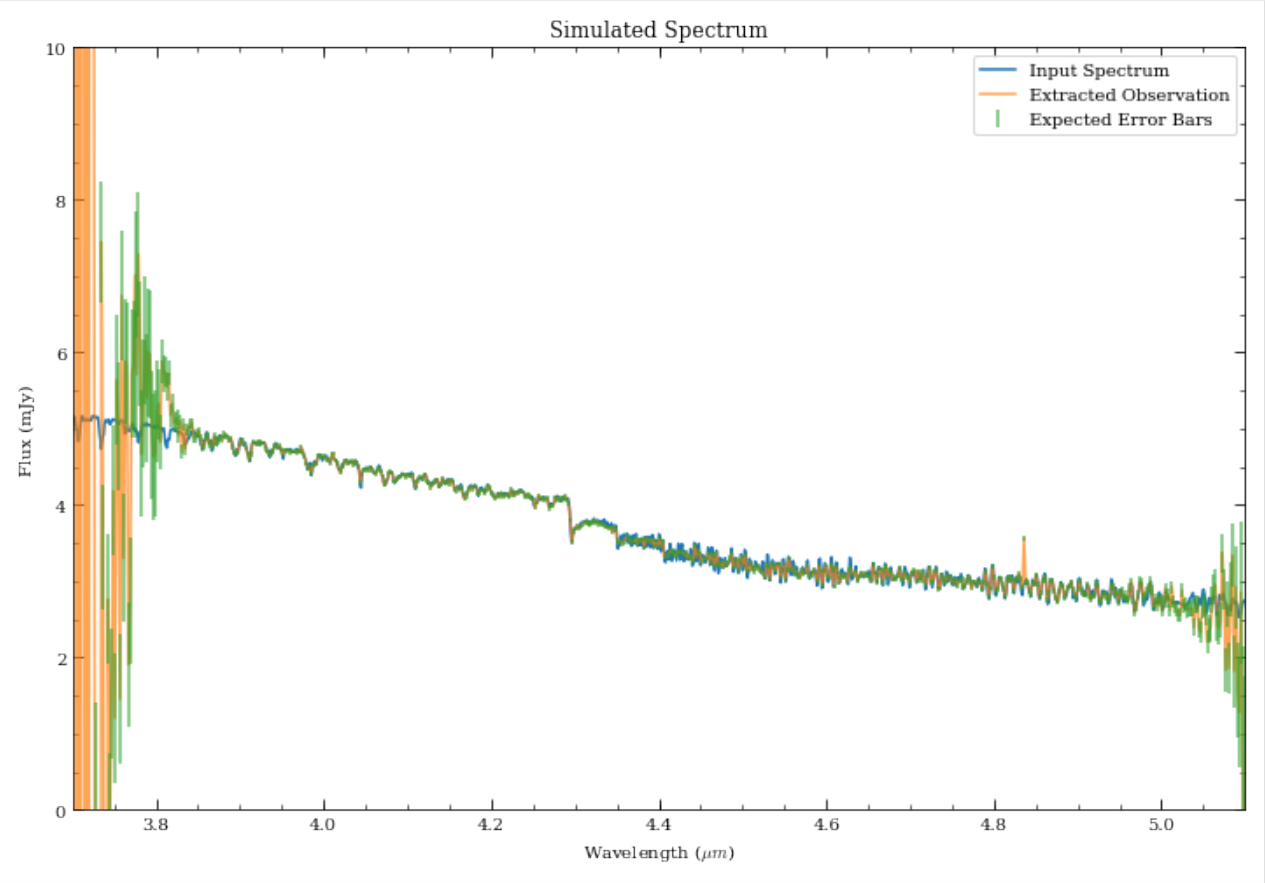
```
    slope_list.append(cf[1])

# Create a master averaged slope image
slopes_all = np.array(slope_list)
slope_sim = slopes_all.mean(axis=0) * nrc.Detectors[0].gain
```

[23]:
```
# Expand wspec to nx (fill value of 0)
# Then shrink to a size excluding wspec=0
# This assumes simulated spectrum is centered
wspec = nrc_utils.pad_or_cut_to_size(wspec, nx)
ind = wspec>0

# Estimate background emission and subtract from slope_sim
bg = np.median(slope_sim[:,~ind])
slope_sim = slope_sim[:,ind] - bg
wspec = wspec[ind]
```

[24]:
```
# Extract 2 spectral x 5 spatial pixels

# First, cut out the central 5 pixels
slope_sub = nrc_utils.pad_or_cut_to_size(slope_sim, (5,slope_sim.shape[1]))
slope_sub_ideal = nrc_utils.pad_or_cut_to_size(imspec, (5,imspec.shape[1]))

# Sum along the spatial axis
spec = slope_sub.sum(axis=0)
spec_ideal = slope_sub_ideal.sum(axis=0)
spec_ideal_rebin = nrc_utils.frebin(spec_ideal, scale=0.5, total=False)

# Build a quick RSRF from extracted ideal spectral slope
sp_M9V.convert('mjy')
rsrf = spec_ideal / sp_M9V.sample(wspec*1e4)

# Rebin along spectral direction
wspec_rebin = nrc_utils.frebin(wspec, scale=0.5, total=False)
spec_rebin_cal = nrc_utils.frebin(spec/rsrf, scale=0.5, total=False)
```

[25]:
```
# Expected noise per extraction element
snr_interp = np.interp(wspec_rebin, snr_dict['wave'], snr_dict['snr'])
_spec_rebin = spec_ideal_rebin / snr_interp
_spec_rebin_cal = _spec_rebin / nrc_utils.frebin(rsrf, scale=0.5, total=False)
```

[26]:
```
fig, ax = plt.subplots(1,1, figsize=(12,8))
ax.plot(sp_M9V.wave/1e4, sp_M9V.flux, label='Input Spectrum')
ax.plot(wspec_rebin, spec_rebin_cal, alpha=0.7, label='Extracted Observation')
ax.errorbar(wspec_rebin, spec_rebin_cal, yerr=_spec_rebin_cal, zorder=3,
            fmt='none', label='Expected Error Bars', alpha=0.7)

ax.set_ylim([0,10])
ax.set_xlim([3.7,5.1])

ax.set_xlabel('Wavelength ($\mu m$)')
ax.set_ylabel('Flux (mJy)')
ax.set_title('Simulated Spectrum')

ax.legend(loc='upper right');
```



1.5.4 Example 4: Exoplanet Transit Spectroscopy

Let's say we want to observe an exoplanet transit using NIRCam grisms in the F322W2 filter.

We assume a 2.1-hour transit duration for a K6V star (K=8.4 mag).

[27]:
```
nrc = pynrc.NIRCam('F322W2', pupil='GRISM0', wind_mode='STRIPE', ypix=64)
```

[28]:
```
# K6V star at K=8.4 mags
bp_k = S.ObsBandpass('k')
sp_K6V = pynrc.stellar_spectrum('K6V', 8.4, 'vegamag', bp_k)
```

[29]:
```
# Constraints
well      = 0.5         # Keep well below 50% full
tacq      = 2.1*3600.   # 2.1 hour transit duration
ng_max    = 30          # Transit spectroscopy allows for up to 30 groups per␣
↪integrations
nint_max = int(1e6)    # Effectively no limit on number of integrations

# Let's bin the spectrum to R~100
# dw_bin is a passable parameter for specifiying spectral bin sizes
R = 100
dw_bin = (nrc.bandpass.avgwave() / 10000) / R
```

[35]:
```
res = nrc.ramp_optimize(sp_K6V, tacq_max=tacq, nint_max=nint_max,
                        ng_min=10, ng_max=ng_max, well_frac_max=well,
                        dw_bin=dw_bin, verbose=True)
```

```
BRIGHT1
BRIGHT2
DEEP2
DEEP8
MEDIUM2
MEDIUM8
RAPID
SHALLOW2
SHALLOW4
 Pattern   NGRP NINT   t_int     t_exp     t_acq     SNR      Well     eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
BRIGHT1      25  442     16.69   7376.93   7529.80  30176.2    0.498  347.754
BRIGHT1      25  443     16.69   7393.62   7546.84  30210.3    0.498  347.754
BRIGHT1      25  444     16.69   7410.31   7563.87  30244.4    0.498  347.754
BRIGHT1      25  445     16.69   7427.00   7580.91  30278.5    0.498  347.754
BRIGHT1      25  446     16.69   7443.69   7597.95  30312.5    0.498  347.754
BRIGHT1      24  460     16.01   7363.99   7523.08  30132.7    0.478  347.408
BRIGHT2      24  451     16.35   7373.53   7529.51  29860.3    0.488  344.121
BRIGHT2      24  452     16.35   7389.87   7546.20  29893.4    0.488  344.121
BRIGHT2      24  453     16.35   7406.22   7562.90  29926.5    0.488  344.121
BRIGHT2      24  454     16.35   7422.57   7579.59  29959.5    0.488  344.121
...         ...  ...       ...       ...       ...      ...      ...      ...
SHALLOW2     10  464     16.01   7428.02   7588.50  29927.0    0.478  343.545
RAPID        30  713     10.22   7285.65   7532.24  29770.9    0.305  343.028
RAPID        30  714     10.22   7295.87   7542.81  29791.8    0.305  343.028
RAPID        30  715     10.22   7306.08   7553.37  29812.7    0.305  343.028
RAPID        30  716     10.22   7316.30   7563.94  29833.5    0.305  343.028
RAPID        30  717     10.22   7326.52   7574.50  29854.3    0.305  343.028
SHALLOW4     10  442     16.69   7376.93   7529.80  29235.0    0.498  336.907
SHALLOW4     10  443     16.69   7393.62   7546.84  29268.0    0.498  336.907
SHALLOW4     10  444     16.69   7410.31   7563.87  29301.1    0.498  336.907
SHALLOW4     10  445     16.69   7427.00   7580.91  29334.0    0.498  336.907
SHALLOW4     10  446     16.69   7443.69   7597.95  29367.0    0.498  336.907
Length = 27 rows
```

[31]:
```
# Print the Top 2 settings for each readout pattern
res2 = table_filter(res, 2)
print(res2)
```

```
 Pattern   NGRP NINT   t_int     t_exp     t_acq     SNR      Well     eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
RAPID        30  713     10.22   7285.65   7532.24  29770.8    0.305  343.026
RAPID        30  714     10.22   7295.87   7542.81  29791.6    0.305  343.026
BRIGHT1      25  442     16.69   7376.93   7529.80  30176.1    0.498  347.752
BRIGHT1      25  443     16.69   7393.62   7546.84  30210.2    0.498  347.752
BRIGHT2      24  451     16.35   7373.53   7529.51  29860.2    0.488  344.119
BRIGHT2      24  452     16.35   7389.87   7546.20  29893.2    0.488  344.119
SHALLOW2     10  460     16.01   7363.99   7523.08  29797.5    0.478  343.544
SHALLOW2     10  461     16.01   7380.00   7539.44  29829.9    0.478  343.544
SHALLOW4     10  442     16.69   7376.93   7529.80  29234.8    0.498  336.905
SHALLOW4     10  443     16.69   7393.62   7546.84  29267.9    0.498  336.905
```

[32]:
```
# Even though BRIGHT1 has a slight efficiency preference over RAPID
# and BRIGHT2, we decide to choose RAPID, because we are convinced
```

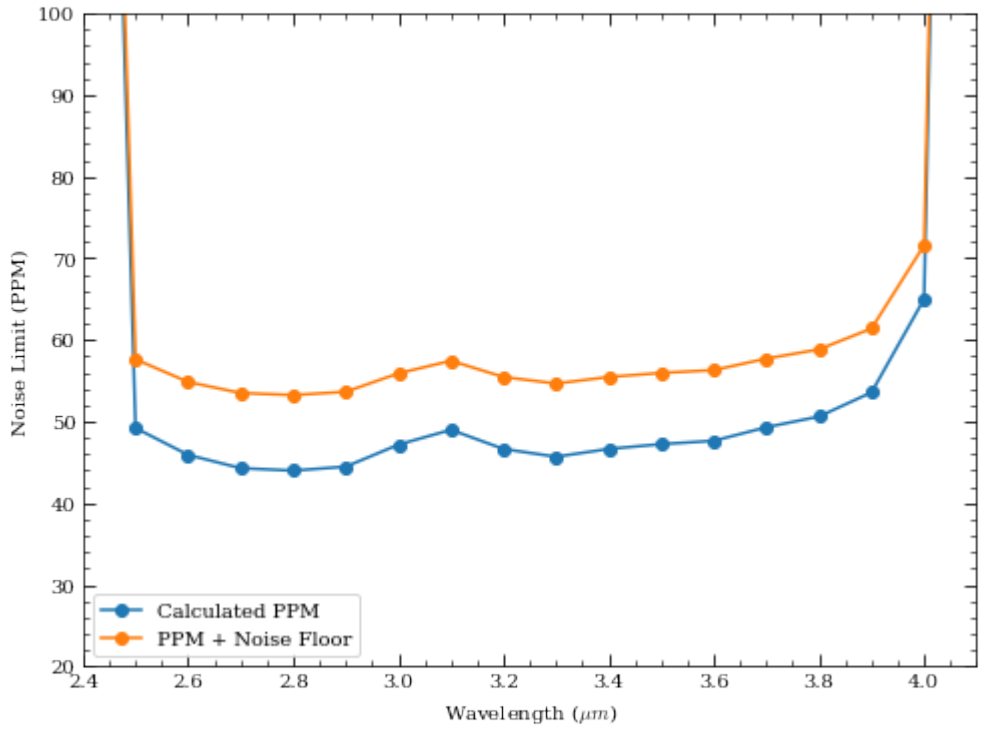
(continues on next page)

(continued from previous page)

```
# that saving all data (and no coadding) is a better option.
# If APT informs you that the data rates or total data shorage is
# an issue, you can select one of the other options.

# Update to RAPID, ngroup=30, nint=715 and plot PPM
nrc.update_detectors(read_mode='RAPID', ngroup=30, nint=715)
snr_dict = nrc.sensitivity(sp=sp_K6V, dw_bin=dw_bin, forwardSNR=True, units='Jy')
wave = np.array(snr_dict['wave'])
snr  = np.array(snr_dict['snr'])

# Let assume bg subtraction of something with similar noise
snr /= np.sqrt(2.)
ppm = 1e6 / snr

# NOTE: We have up until now neglected to include a "noise floor"
# which represents the expected minimum achievable ppm from
# unknown systematics. To first order, this can be added in
# quadrature to the calculated PPM.
noise_floor = 30 # in ppm
ppm_floor = np.sqrt(ppm**2 + noise_floor**2)
```

[33]:
```
plt.plot(wave, ppm, marker='o', label='Calculated PPM')
plt.plot(wave, ppm_floor, marker='o', label='PPM + Noise Floor')
plt.xlabel('Wavelength ($\mu m$)')
plt.ylabel('Noise Limit (PPM)')
plt.xlim([2.4,4.1])
plt.ylim([20,100])
plt.legend()
```

[33]:
```
<matplotlib.legend.Legend at 0x1c23656f28>
```



1.5.5 Example 5: Extended Souce

Expect some faint galaxies of 25 ABMag/arcsec^2 in our field. What is the best we can do with 10,000 seconds of acquisition time?

[34]:
```
# Detection bandpass is F200W
nrc = pynrc.NIRCam('F200W')

# Flat spectrum (in photlam) with ABMag = 25 in the NIRCam bandpass
sp = pynrc.stellar_spectrum('flat', 25, 'abmag', nrc.bandpass)
```

[35]:
```
res = nrc.ramp_optimize(sp, is_extended=True, tacq_max=10000, tacq_frac=0.05,␣
↪verbose=True)
```

```
BRIGHT1
BRIGHT2
DEEP2
DEEP8
MEDIUM2
MEDIUM8
RAPID
SHALLOW2
SHALLOW4
 Pattern    NGRP NINT   t_int      t_exp      t_acq      SNR       Well      eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
MEDIUM8      10    8   1052.20   8417.63   8503.53      9.8    0.019    0.106
MEDIUM8       9    9    944.84   8503.52   8600.16      9.8    0.017    0.106
MEDIUM8       9   10    944.84   9448.36   9555.73     10.4    0.017    0.106
MEDIUM8      10    9   1052.20   9469.83   9566.47     10.4    0.019    0.106
MEDIUM8       9   11    944.84  10393.19  10511.30     10.9    0.017    0.106
MEDIUM8      10   10   1052.20  10522.03  10629.41     11.0    0.019    0.106
DEEP8         8    5   1589.04   7945.21   7998.90      9.4    0.028    0.105
DEEP8         7    6   1374.31   8245.84   8310.27      9.6    0.024    0.105
MEDIUM8       8   11    837.47   9212.15   9330.26     10.2    0.015    0.105
DEEP8         6    8   1159.57   9276.57   9362.47     10.2    0.021    0.105
...         ...  ...       ...       ...       ...      ...      ...      ...
BRIGHT1      10   46    204.00   9383.94   9877.83      6.4    0.004    0.063
BRIGHT1      10   47    204.00   9587.94  10092.57      6.4    0.004    0.063
BRIGHT1      10   48    204.00   9791.93  10307.30      6.5    0.004    0.063
BRIGHT1      10   49    204.00   9995.93  10522.04      6.6    0.004    0.063
BRIGHT1       8   56    161.05   9018.89   9620.15      5.3    0.003    0.054
RAPID        10   83    107.37   8911.52   9802.68      4.7    0.002    0.046
RAPID        10   84    107.37   9018.89   9920.78      4.7    0.002    0.046
RAPID        10   85    107.37   9126.25  10038.89      4.7    0.002    0.046
RAPID        10   86    107.37   9233.62  10156.99      4.7    0.002    0.046
RAPID        10   87    107.37   9340.99  10275.09      4.8    0.002    0.046
RAPID         9   91     96.63   8793.41   9770.47      4.2    0.002    0.042
Length = 71 rows
```

[36]:
```
# Print the Top 2 settings for each readout pattern
res2 = table_filter(res, 2)
print(res2)
```

```
 Pattern    NGRP NINT   t_int      t_exp      t_acq      SNR       Well      eff
---------- ---- ---- --------- --------- --------- -------- -------- --------
RAPID        10   83    107.37   8911.52   9802.68      4.7    0.002    0.046
RAPID        10   84    107.37   9018.89   9920.78      4.7    0.002    0.046
BRIGHT1      10   45    204.00   9179.94   9663.10      6.3    0.004    0.063
```

(continues on next page)

(continued from previous page)

```
BRIGHT1      10   46    204.00   9383.94   9877.83      6.4    0.004    0.063
BRIGHT2      10   43    214.74   9233.62   9695.31      7.5    0.004    0.076
BRIGHT2      10   44    214.74   9448.36   9920.78      7.6    0.004    0.076
SHALLOW2     10   18    504.63   9083.31   9276.57      9.2    0.009    0.095
SHALLOW2     10   19    504.63   9587.94   9791.94      9.5    0.009    0.095
SHALLOW4     10   17    526.10   8943.73   9126.26      9.7    0.009    0.101
SHALLOW4     10   18    526.10   9469.83   9663.10     10.0    0.009    0.101
MEDIUM2      10    9    987.78   8890.05   8986.68      9.8    0.018    0.102
MEDIUM2      10   10    987.78   9877.83   9985.20     10.3    0.018    0.102
MEDIUM8      10    8   1052.20   8417.63   8503.53      9.8    0.019    0.106
MEDIUM8       9    9    944.84   8503.52   8600.16      9.8    0.017    0.106
DEEP2         9    5   1739.36   8696.78   8750.47      9.5    0.031    0.102
DEEP2         8    6   1524.62   9147.73   9212.15      9.8    0.027    0.102
DEEP8         8    5   1589.04   7945.21   7998.90      9.4    0.028    0.105
DEEP8         7    6   1374.31   8245.84   8310.27      9.6    0.024    0.105
```

[37]:
```
# MEDIUM8 10 10 looks like a good option
nrc.update_detectors(read_mode='MEDIUM8', ngroup=10, nint=10, verbose=True)
```

```
New Ramp Settings:
  read_mode :  MEDIUM8
  nf        :         8
  nd2       :         2
  ngroup    :        10
  nint      :        10
New Detector Settings
  wind_mode :      FULL
  xpix      :      2048
  ypix      :      2048
  x0        :         0
  y0        :         0
New Ramp Times
  t_group   :   107.368
  t_frame   :    10.737
  t_int     :  1052.203
  t_int_tot :  1062.940
  t_exp     : 10522.035
  t_acq     : 10629.408
```

[38]:
```
# Calculate flux/mag for various nsigma detection limits
tbl = Table(names=('Sigma', 'Point (nJy)',     'Extended (nJy/asec^2)',
                           'Point (AB Mag)', 'Extended (AB Mag/asec^2)'))
tbl['Sigma'].format = '.0f'
for k in tbl.keys()[1:]:
    tbl[k].format = '.2f'

for sig in [1,3,5,10]:
    snr_dict1 = nrc.sensitivity(nsig=sig, units='nJy', verbose=False)
    snr_dict2 = nrc.sensitivity(nsig=sig, units='abmag', verbose=False)
    tbl.add_row([sig, snr_dict1[0]['sensitivity'], snr_dict1[1]['sensitivity'],
                 snr_dict2[0]['sensitivity'], snr_dict2[1]['sensitivity']])
```

[39]:
```
tbl
```

[39]:
```
<Table length=4>
 Sigma  Point (nJy) ... Point (AB Mag) Extended (AB Mag/asec^2)
```

(continues on next page)

(continued from previous page)

```
float64   float64   ...    float64             float64
------- ----------- ... -------------- ------------------------
      1        1.18 ...          31.22                    27.63
      3        3.55 ...          30.02                    26.43
      5        5.94 ...          29.47                    25.87
     10       12.00 ...          28.70                    25.10
```

[ ]:

1.6 Coronagraph Basics

This set of exercises guides the user through a step-by-step process of simulating NIRCam coronagraphic observations of the HR 8799 exoplanetary system. The goal is to familiarize the user with basic `pynrc` classes and functions relevant to coronagraphy.

[1]:
```
# If running Python 2.x, makes print and division act like Python 3
from __future__ import print_function, division

# Import the usual libraries
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

# Enable inline plotting at lower left
%matplotlib inline

from IPython.display import display, Latex, clear_output
```

We will start by first importing `pynrc` along with the `obs_hci` (High Contrast Imaging) class, which lives in the `pynrc.obs_nircam` module.

[2]:
```
import pynrc
from pynrc import nrc_utils          # Variety of useful functions and classes
from pynrc.obs_nircam import obs_hci # High-contrast imaging observation class

# Disable informational messages and only include warnings and higher
pynrc.setup_logging(level='WARN')
```

```
pyNRC log messages of level WARN and above will be shown.
pyNRC log outputs will be directed to the screen.
```

1.6.1 Source Definitions

The `obs_hci` class first requires two arguments describing the spectra of the science and reference sources (`sp_sci` and `sp_ref`, respectively. Each argument should be a Pysynphot spectrum already normalized to some known flux. `pynrc` includes built-in functions for generating spectra. The user may use either of these or should feel free to supply their own as long as it meets the requirements.

1. The `pynrc.stellar_spectrum` function provides the simplest way to define a new spectrum:

```
bp_k = pynrc.bp_2mass('k') # Define bandpass to normalize spectrum
sp_sci = pynrc.stellar_spectrum('F0V', 5.24, 'vegamag', bp_k)
```

You can also be more specific about the stellar properties with `Teff`, `metallicity`, and `log_g` keywords.

```
sp_sci = pynrc.stellar_spectrum('F0V', 5.24, 'vegamag', bp_k,
                                Teff=7430, metallicity=-0.47, log_g=4.35)
```

2. Alternatively, the `pynrc.source_spectrum` class ingests spectral information of a given target and generates a model fit to the known photometric SED. Two model routines can be fit. The first is a very simple scale factor that is applied to the input spectrum, while the second takes the input spectrum and adds an IR excess modeled as a modified blackbody function. The user can find the relevant photometric data at http://vizier.u-strasbg.fr/vizier/sed/ and click download data as a VOTable.

[3]:
```
# Define 2MASS Ks bandpass and source information
bp_k = pynrc.bp_2mass('k')

# Science         source,  dist, age, sptype, Teff, [Fe/H], log_g, mag, band
args_sources = [('HR 8799', 39.0, 30,  'F0V', 7430, -0.47, 4.35, 5.24, bp_k)]

# References     source,     sptype, Teff, [Fe/H], log_g, mag, band
ref_sources = [('HD 220657', 'F8III', 5888, -0.01, 3.22, 3.04, bp_k)]
```

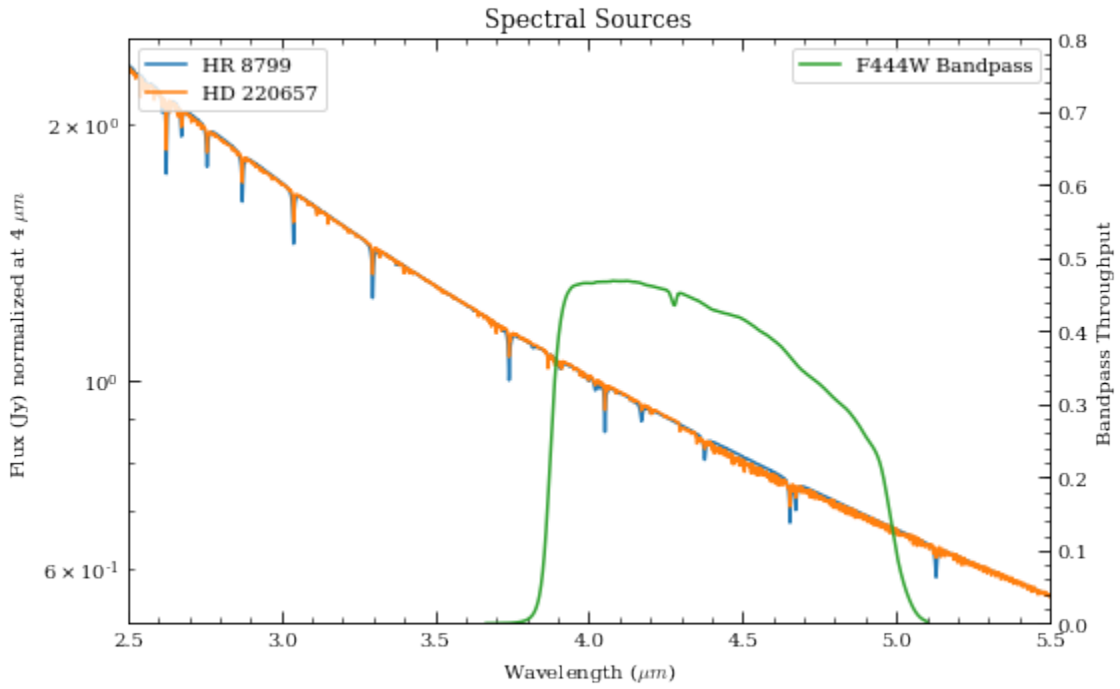
[4]:
```
name_sci, dist_sci, age, spt_sci, Teff_sci, feh_sci, logg_sci, mag_sci, bp_sci = args_
↪sources[0]
name_ref, spt_ref, Teff_ref, feh_ref, logg_ref, mag_ref, bp_ref = ref_sources[0]

# For the purposes of simplicity, we will use pynrc.stellar_spectrum()
sp_sci = pynrc.stellar_spectrum(spt_sci, mag_sci, 'vegamag', bp_sci,
                                Teff=Teff_sci, metallicity=feh_sci, log_g=logg_sci)
sp_sci.name = name_sci

# And the refernece source
sp_ref = pynrc.stellar_spectrum(spt_ref, mag_ref, 'vegamag', bp_ref,
                                Teff=Teff_ref, metallicity=feh_ref, log_g=logg_ref)
sp_ref.name = name_ref
```

[5]:
```
# Plot the two spectra
fig, ax = plt.subplots(1,1, figsize=(8,5))

xr = [2.5,5.5]

for sp in [sp_sci, sp_ref]:
    w = sp.wave / 1e4
    ind = (w>=xr[0]) & (w<=xr[1])
    sp.convert('Jy')
    f = sp.flux / np.interp(4.0, w, sp.flux)
    ax.semilogy(w[ind], f[ind], lw=1.5, label=sp.name)
    ax.set_ylabel('Flux (Jy) normalized at 4 $\mu m$')
    sp.convert('flam')

ax.set_xlim(xr)
ax.set_xlabel(r'Wavelength ($\mu m$)')
ax.set_title('Spectral Sources')

# Overplot Filter Bandpass
bp = pynrc.read_filter('F444W', 'CIRCLYOT', 'MASK430R')
ax2 = ax.twinx()
ax2.plot(bp.wave/1e4, bp.throughput, color='C2', label=bp.name+' Bandpass')
ax2.set_ylim([0,0.8])
```

(continues on next page)

(continued from previous page)

```
ax2.set_xlim(xr)
ax2.set_ylabel('Bandpass Throughput')

ax.legend(loc='upper left')
ax2.legend(loc='upper right')

fig.tight_layout()
```



1.6.2 Initialize Observation

Now we will initialize the high-contrast imaging class `pynrc.obs_hci` using the spectral objects and various other settings. The `obs_hci` object is a subclass of the more generalized `NIRCam` class. It implements new settings and functions specific to high-contrast imaging observations for corongraphy and direct imaging.

For this tutorial, we want to observe these targets using the `MASK430R` coronagraph in the `F444W` filter. All circular coronagraphic masks such as the `430R` (R=round) should be paired with the `CIRCLYOT` pupil element, whereas wedge/bar masks are paired with `WEDGELYOT` pupil. Observations in the LW channel are most commonly observed in `WINDOW` mode with a 320x320 detector subarray size. Full detector sizes are also available.

The PSF simulation size (`fov_pix` keyword) should also be of similar size as the subarray window (recommend avoiding anything above `fov_pix=1024` due to computation time and memory usage). Use odd numbers to center the PSF in the middle of the pixel. If `fov_pix` is specified as even, then PSFs get centered at the corners. This distinction really only matter for unocculted observations, (ie., where the PSF flux is concentrated in a tight central core).

We also need to specify a WFE drift value (`wfe_ref_drift` parameter), which defines the anticipated drift in nm between the science and reference sources. For the moment, let's intialize with a value of 0nm. This prevents an initially long process by which `pynrc` calculates changes made to the PSF over a wide range of drift values.

Extended disk models can also be specified upon initialization using the `disk_hdu` keyword.

[6]:
```
filt, mask, pupil = ('F444W',  'MASK430R', 'CIRCLYOT')
wind_mode, subsize = ('WINDOW', 320)
fov_pix, oversample = (320, 2)

wfe_ref_drift = 0
obs = pynrc.obs_hci(sp_sci, sp_ref, dist_sci, filter=filt, mask=mask, pupil=pupil,
                    wfe_ref_drift=wfe_ref_drift, fov_pix=fov_pix,␣
↪oversample=oversample,
                    wind_mode=wind_mode, xpix=subsize, ypix=subsize, verbose=True)
```

```
Generating background PSF coefficients...
Generating oversampled PSFs...
Updating NIRCam reference coefficients...
Creating NIRCam reference class...
Finished.
```

All information for the reference observation is stored in the attribute `obs.nrc_ref`, which is simply it's own isolated `NIRCam` (`nrc_hci`) class. After initialization, any updates made to the primary `obs` instrument configuration (e.g., filters, detector size, etc.) must also be made inside the `obs.nrc_ref` class as well. That is to say, it does not automatically propogate. In many ways, it's best to think of these as two separate classes,

```
obs_sci = obs
obs_ref = obs.nrc_ref
```

with some linked references between the two.

Now that we've succesffully initialized the obs_hci observations, let's specify the `wfe_ref_drift`. If this is your first time, then the `nrc_utils.wfed_coeff` function is called to determine a relationship between PSFs in the presense of WFE drift. This relationship is saved to disk in the `PYNRC_DATA` directory as a set of polynomial coefficients. Future calculations utilize these coefficients to quickly generate a new PSF for any arbitary drift value.

[7]:
```
# WFE drift amount between rolls
# This only gets called during gen_roll_image()
# and temporarily updates obs.wfe_drift to create
# a new PSF.
obs.wfe_roll_drift = 2

# Drift amount between Roll 1 and reference
# This is simply a link to obs.nrc_ref.wfe_drift
obs.wfe_ref_drift = 10
```

1.6.3 Exposure Settings

Optimization of exposure settings are demonstrated in another tutorial, so we will not repeat that process here. We can assume the optimization process was performed elsewhere to choose the `DEEP8` pattern with 16 groups and 5 total integrations. These settings apply to each roll position of the science observation as well as the for the reference observation.

[8]:
```
# Update both the science and reference observations
obs.update_detectors(read_mode='DEEP8', ngroup=16, nint=5, verbose=True)
obs.nrc_ref.update_detectors(read_mode='DEEP8', ngroup=16, nint=5)
```

```
New Ramp Settings:
  read_mode :    DEEP8
  nf        :        8
  nd2       :       12
```

(continues on next page)

(continued from previous page)

```
  ngroup    :       16
  nint      :        5
New Detector Settings
  wind_mode :   WINDOW
  xpix      :      320
  ypix      :      320
  x0        :      914
  y0        :     1513
New Ramp Times
  t_group   :   21.381
  t_frame   :    1.069
  t_int     :  329.264
  t_int_tot :  330.353
  t_exp     : 1646.322
  t_acq     : 1651.766
```

1.6.4 Add Planets

There are four known giant planets orbiting HR 8799 at various locations. Ideally, we would like to position them at their predicted locations on the anticipated observation date. For this case, we choose a plausible observation date of November 1, 2019. To convert between (x, y) and (r, θ), use the `nrc_utils.xy_to_rtheta` and `nrc_utils.rtheta_to_xy` functions.

When adding the planets, it doesn't matter too much which exoplanet model spectrum we decide to use since the spectra are still fairly unconstrained at these wavelengths. We do know roughly the planets' luminosities, so we can simply choose some reasonable model and renormalize it to the appropriate filter brightness. Currently, the only exoplanet spectral models available to `pynrc` are those from Spiegel & Burrows (2012).

[9]:
```
# Projected locations for date 11/01/2019
# These are prelimary positions, but within constrained orbital parameters
loc_list = [(-1.57, 0.64), (0.42, 0.87), (0.5, -0.45), (0.35,  0.20)]

# Estimated magnitudes within F444W filter
pmags = [16.0, 15.0, 14.6, 14.7]
```

[10]:
```
# Add planet information to observation class.
# These are stored in obs.planets.
# Can be cleared using obs.kill_planets().
obs.kill_planets()
for i, loc in enumerate(loc_list):
    obs.add_planet(mass=10, entropy=13, age=age, xy=loc, runits='arcsec',
                   renorm_args=(pmags[i], 'vegamag', obs.bandpass))
```

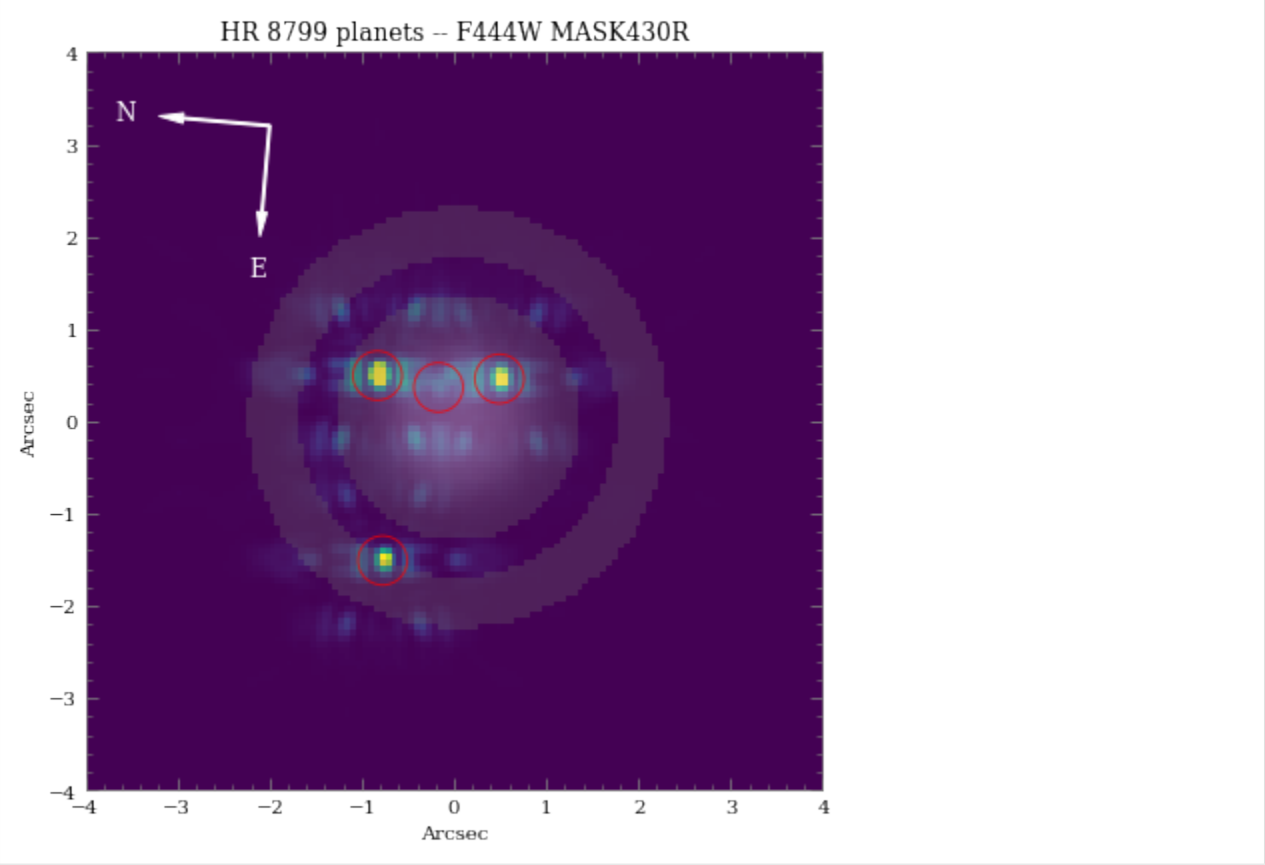
[11]:
```
# Generate and plot a noiseless slope image to make sure things look right
PA1 = 85
im_planets = obs.gen_planets_image(PA_offset=PA1)
```

[12]:
```
from matplotlib.patches import Circle
from pynrc.nrc_utils import (coron_ap_locs, build_mask_detid, fshift, pad_or_cut_to_
↪size)

fig, ax = plt.subplots(figsize=(6,6))

xasec = obs.det_info['xpix'] * obs.pix_scale
```

(continues on next page)

(continued from previous page)

```
yasec = obs.det_info['ypix'] * obs.pix_scale
extent = [-xasec/2, xasec/2, -yasec/2, yasec/2]
xylim = 4

vmin = 0
vmax = 0.5*im_planets.max()
ax.imshow(im_planets, extent=extent, vmin=vmin, vmax=vmax)

# Overlay the coronagraphic mask
detid = obs.Detectors[0].detid
im_mask = obs.mask_images[detid]
# Do some masked transparency overlays
masked = np.ma.masked_where(im_mask>0.99, im_mask)
#ax.imshow(1-masked, extent=extent, alpha=0.5)
ax.imshow(1-masked, extent=extent, alpha=0.3, cmap='Greys_r', vmin=-0.5)

xc_off = obs.bar_offset
for loc in loc_list:
    xc, yc = loc
    xc, yc = nrc_utils.xy_rot(xc, yc, PA1)
    xc += xc_off
    circle = Circle((xc,yc), radius=xylim/15., alpha=0.7, lw=1, edgecolor='red',␣
↪facecolor='none')
    ax.add_artist(circle)

xlim = ylim = np.array([-1,1])*xylim
xlim = xlim + xc_off
ax.set_xlim(xlim)
ax.set_ylim(ylim)

ax.set_xlabel('Arcsec')
ax.set_ylabel('Arcsec')

ax.set_title('{} planets -- {} {}'.format(sp_sci.name, obs.filter, obs.mask))

color = 'grey'
ax.tick_params(axis='both', color=color, which='both')
for k in ax.spines.keys():
    ax.spines[k].set_color(color)

nrc_utils.plotAxes(ax, width=1, headwidth=5, alength=0.15, angle=PA1,
                   position=(0.25,0.9), label1='E', label2='N')

fig.tight_layout()
```



As we can see, even with "perfect PSF subtraction" and no noise, it's difficult to make out planet e. This is primarily due to its location relative to the occulting mask reducing throughput along with confusion of bright diffraction spots from nearby sources.

1.6.5 Estimated Performance

Now we are ready to determine contrast performance and sensitivites as a function of distance from the star.

1. Roll-Subtracted Images

First, we will create a quick simulated roll-subtracted image using the in `gen_roll_image` method. For the selected observation date of 11/1/2019, APT shows a PA range of 84° to 96°. So, we'll assume Roll 1 has PA1=85, while Roll 2 has PA2=95. In this case, "roll subtraction" simply creates two science images observed at different parallactic angles, then subtracts the same reference observation from each. The two results are then de-rotated to a common PA=0 and averaged.

There is also the option to create ADI images, where the other roll position becomes the reference star by setting `no_ref=True`.

[13]:
```
# Cycle through a few WFE drift values
wfe_list = [0,5,10]

# PA values for each roll
PA1, PA2 = (85, 95)
```

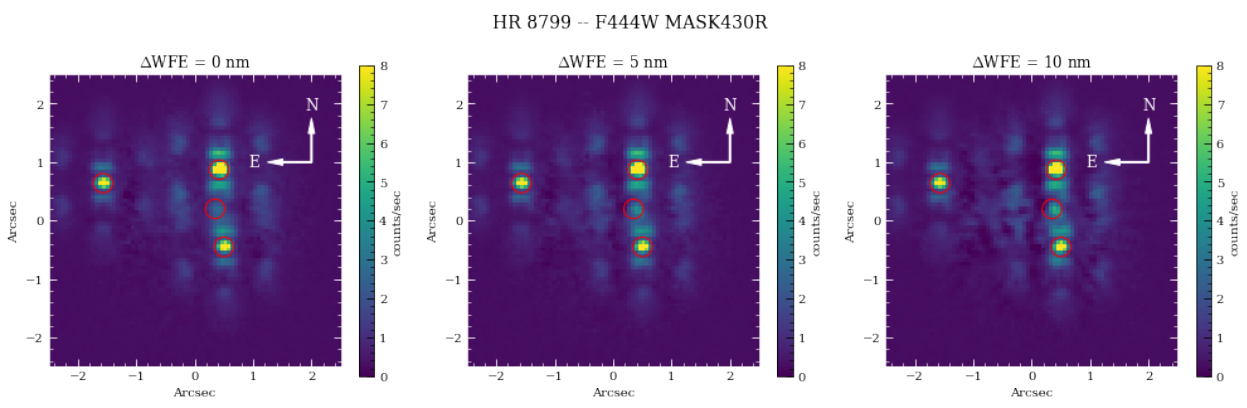
(continues on next page)

(continued from previous page)

```
# A dictionary of HDULists
hdul_dict = {}
for i, wfe_drift in enumerate(wfe_list):
    print(wfe_drift)
    # Upate WFE reference drift value
    obs.wfe_ref_drift = wfe_drift

    # Set the final output image to be oversampled
    hdulist = obs.gen_roll_image(PA1=PA1, PA2=PA2)
    hdul_dict[wfe_drift] = hdulist
```

```
0
5
10
```

[16]:
```
from pynrc.obs_nircam import plot_hdulist
from matplotlib.patches import Circle

fig, axes = plt.subplots(1,3, figsize=(14,4.3))
xylim = 2.5
xlim = ylim = np.array([-1,1])*xylim

for j, wfe_drift in enumerate(wfe_list):
    ax = axes[j]
    hdul = hdul_dict[wfe_drift]

    plot_hdulist(hdul, xr=xlim, yr=ylim, ax=ax, vmin=0, vmax=8)

    # Location of planet
    for loc in loc_list:
        circle = Circle(loc, radius=xylim/15., lw=1, edgecolor='red', facecolor='none
↪')
        ax.add_artist(circle)

    ax.set_title('$\Delta$WFE = {:.0f} nm'.format(wfe_drift))

    nrc_utils.plotAxes(ax, width=1, headwidth=5, alength=0.15, position=(0.9,0.7),␣
↪label1='E', label2='N')

fig.suptitle('{} -- {} {}'.format(name_sci, obs.filter, obs.mask), fontsize=14)
fig.tight_layout()
fig.subplots_adjust(top=0.85)
```



Note: At first glance, it appears as if the innermost Planet e is getting brighter with increased WFE drift, which would be understandably confusing. However, upon further investigation, there just happens to be a bright residual speckle that lines up well with Planet e when observed at this specific parallactic angle. This was verified by adjusting the observed PA as well as removing the planets from the simulations.

2. Contrast Curves

Next, we will cycle through a few WFE drift values to get an idea of potential predicted sensitivity curves. The `calc_contrast` method returns a tuple of three arrays: 1. The radius in arcsec. 2. The n-sigma contrast. 3. The n-sigma magnitude sensitivity limit (vega mag).

[17]:
```
# Cycle through varying levels of WFE drift and calculate contrasts
wfe_list = [0,5,10]
nsig = 5

# PA values for each roll
PA1, PA2 = (85, 95)
roll_angle = np.abs(PA2 - PA1)

curves = []
for i, wfe_drift in enumerate(wfe_list):
    print(wfe_drift)
    # Generate series of observations for each filter
    obs.wfe_ref_drift = wfe_drift

    # Generate contrast curves
    result = obs.calc_contrast(roll_angle=roll_angle, nsig=nsig)
    curves.append(result)
```

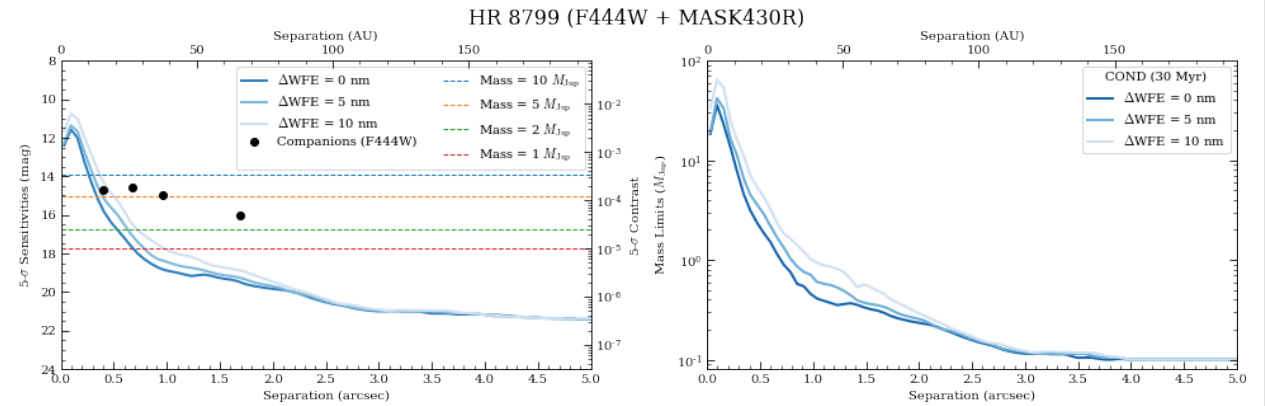
```
0
5
10
```

[18]:
```
from pynrc.obs_nircam import plot_contrasts, plot_planet_patches, plot_contrasts_mjup
import matplotlib.patches as mpatches

# fig, ax = plt.subplots(figsize=(8,5))
fig, axes = plt.subplots(1,2, figsize=(14,4.5))
xr=[0,5]
yr=[24,8]

# 1a. Plot contrast curves and set x/y limits
ax = axes[0]
ax, ax2, ax3 = plot_contrasts(curves, nsig, wfe_list, obs=obs,
                              xr=xr, yr=yr, ax=ax, return_axes=True)
# 1b. Plot the locations of exoplanet companions
label = 'Companions ({})'.format(filt)
planet_dist = [np.sqrt(x**2+y**2) for x,y in loc_list]
ax.plot(planet_dist, pmags, marker='o', ls='None', label=label, color='k', zorder=10)

# 1c. Plot Spiegel & Burrows (2012) exoplanet fluxes (Hot Start)
plot_planet_patches(ax, obs, age=age, entropy=13, av_vals=None)
ax.legend(ncol=2)

# 2. Plot in terms of MJup using COND models
ax = axes[1]
```

(continues on next page)

(continued from previous page)

```
plot_contrasts_mjup(curves, nsig, wfe_list, obs=obs, age=age,
                    ax=ax, twin_ax=True, xr=xr, yr=None)
ax.set_yscale('log')
ax.set_ylim([0.08,100])
ax.legend(loc='upper right', title='COND ({:.0f} Myr)'.format(age))

fig.suptitle('{} ({} + {})'.format(name_sci, obs.filter, obs.mask), fontsize=16)

fig.tight_layout()
fig.subplots_adjust(top=0.85, bottom=0.1 , left=0.05, right=0.97)
```



The innermost Planet e is right on the edge of the detection threshold as suggested by the simulated images.

3. Saturation Levels

Create an image showing level of saturation for each pixel. For NIRCam, saturation is important to track for purposes of accurate slope fits and persistence correction. In this case, we will plot the saturation levels both at `NGROUP=2` and `NGROUP=obs.det_info['ngroup']`. Saturation is defined at 80% well level, but can be modified using the `well_fill` keyword.

We want to perform this analysis for both science and reference targets.

[19]:
```
# Saturation limits
ng_max = obs.det_info['ngroup']
sp_flat = pynrc.stellar_spectrum('flat')

print('NGROUP=2')
_ = obs.sat_limits(sp=sp_flat,ngroup=2,verbose=True)

print('')
print('NGROUP={}'.format(ng_max))
_ = obs.sat_limits(sp=sp_flat,ngroup=ng_max,verbose=True)

mag_sci = obs.star_flux('vegamag')
mag_ref = obs.star_flux('vegamag', sp=obs.sp_ref)
print('')
print('{} flux at {}: {:0.2f} mags'.format(obs.sp_sci.name, obs.filter, mag_sci))
print('{} flux at {}: {:0.2f} mags'.format(obs.sp_ref.name, obs.filter, mag_ref))
```

```
NGROUP=2
F444W Saturation Limit assuming Flat spectrum in photlam source: 2.35 vegamag
```

(continues on next page)

(continued from previous page)

```
NGROUP=16
F444W Saturation Limit assuming Flat spectrum in photlam source: 4.95 vegamag

HR 8799 flux at F444W: 5.24 mags
HD 220657 flux at F444W: 3.03 mags
```

In this case, we don't expect HR 8799 to saturated. However, the reference source should have some saturated pixels before the end of an integration.

[20]:
```
# Well level of each pixel for science source
sci_levels1 = obs.saturation_levels(ngroup=2)
sci_levels2 = obs.saturation_levels(ngroup=ng_max)

# Which pixels are saturated?
sci_mask1 = sci_levels1 > 0.8
sci_mask2 = sci_levels2 > 0.8
```

[21]:
```
# Well level of each pixel for reference source
ref_levels1 = obs.saturation_levels(ngroup=2, do_ref=True)
ref_levels2 = obs.saturation_levels(ngroup=ng_max, do_ref=True)

# Which pixels are saturated?
ref_mask1 = ref_levels1 > 0.8
ref_mask2 = ref_levels2 > 0.8
```

[22]:
```
# How many saturated pixels?
nsat1_sci = len(sci_levels1[sci_mask1])
nsat2_sci = len(sci_levels2[sci_mask2])

print(obs.sp_sci.name)
print('{} saturated pixel at NGROUP=2'.format(nsat1_sci))
print('{} saturated pixel at NGROUP={}'.format(nsat2_sci,ng_max))

# How many saturated pixels?
nsat1_ref = len(ref_levels1[ref_mask1])
nsat2_ref = len(ref_levels2[ref_mask2])

print('')
print(obs.sp_ref.name)
print('{} saturated pixel at NGROUP=2'.format(nsat1_ref))
print('{} saturated pixel at NGROUP={}'.format(nsat2_ref,ng_max))
```

```
HR 8799
0 saturated pixel at NGROUP=2
0 saturated pixel at NGROUP=16

HD 220657
0 saturated pixel at NGROUP=2
719 saturated pixel at NGROUP=16
```

[23]:
```
# Saturation Mask for science target

nsat1, nsat2 = (nsat1_sci, nsat2_sci)
sat_mask1, sat_mask2 = (sci_mask1, sci_mask2)
sp = obs.sp_sci
```

(continues on next page)

(continued from previous page)

```
nrc = obs

# Only display saturation masks if there are saturated pixels
if nsat2 > 0:
    fig, axes = plt.subplots(1,2, figsize=(10,5))

    xasec = nrc.det_info['xpix'] * nrc.pix_scale
    yasec = nrc.det_info['ypix'] * nrc.pix_scale
    extent = [-xasec/2, xasec/2, -yasec/2, yasec/2]

    axes[0].imshow(sat_mask1, extent=extent)
    axes[1].imshow(sat_mask2, extent=extent)

    axes[0].set_title('{} Saturation (NGROUP=2)'.format(sp.name))
    axes[1].set_title('{} Saturation (NGROUP={})'.format(sp.name,ng_max))

    for ax in axes:
        ax.set_xlabel('Arcsec')
        ax.set_ylabel('Arcsec')

        ax.tick_params(axis='both', color='white', which='both')
        for k in ax.spines.keys():
            ax.spines[k].set_color('white')

    fig.tight_layout()
else:
    print('No saturation detected.')
```

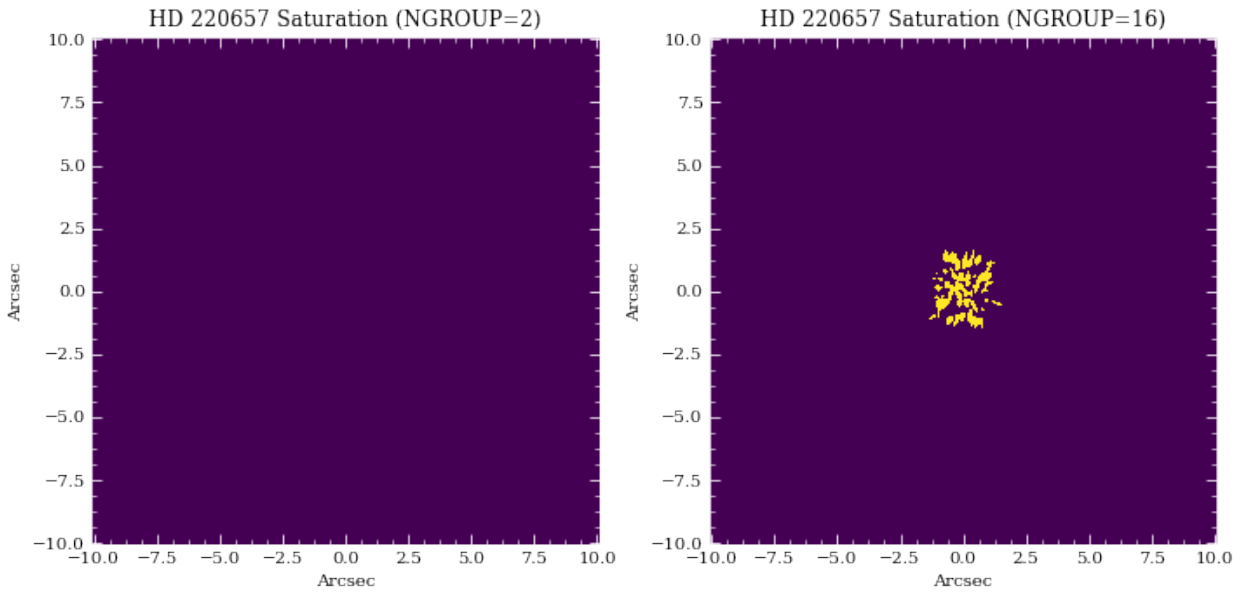
```
No saturation detected.
```

[24]:
```
# Saturation Mask for reference

nsat1, nsat2 = (nsat1_ref, nsat2_ref)
sat_mask1, sat_mask2 = (ref_mask1, ref_mask2)
sp = obs.sp_ref
nrc = obs.nrc_ref

# Only display saturation masks if there are saturated pixels
if nsat2 > 0:
    fig, axes = plt.subplots(1,2, figsize=(10,5))

    xasec = nrc.det_info['xpix'] * nrc.pix_scale
    yasec = nrc.det_info['ypix'] * nrc.pix_scale
    extent = [-xasec/2, xasec/2, -yasec/2, yasec/2]

    axes[0].imshow(sat_mask1, extent=extent)
    axes[1].imshow(sat_mask2, extent=extent)

    axes[0].set_title('{} Saturation (NGROUP=2)'.format(sp.name))
    axes[1].set_title('{} Saturation (NGROUP={})'.format(sp.name,ng_max))

    for ax in axes:
        ax.set_xlabel('Arcsec')
        ax.set_ylabel('Arcsec')

        ax.tick_params(axis='both', color='white', which='both')
        for k in ax.spines.keys():
```

(continues on next page)

(continued from previous page)

```
            ax.spines[k].set_color('white')

    fig.tight_layout()
else:
    print('No saturation detected.')
```



[ ]:

1.7 Coronagraph Wedge Masks

The notebook builds on the concepts introduced in `Coronagraph_Basics.ipynb`. Specifically, we concentrate on the complexities involved in simulating the wedge coronagraphs.

[1]:
```
# If running Python 2.x, makes print and division act like Python 3
from __future__ import print_function, division

# Import the usual libraries
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

# Enable inline plotting at lower left
%matplotlib inline

from IPython.display import display, Latex, clear_output
```

We will start by first importing `pynrc` along with the `obs_hci` (High Contrast Imaging) class, which lives in the `pynrc.obs_nircam` module.

[2]:
```
import pynrc
from pynrc import nrc_utils           # Variety of useful functions and classes
from pynrc.obs_nircam import obs_hci # High-contrast imaging observation class
```

(continues on next page)

(continued from previous page)

```
# Disable informational messages and only include warnings and higher
pynrc.setup_logging(level='WARN')
```

```
pyNRC log messages of level WARN and above will be shown.
pyNRC log outputs will be directed to the screen.
```

1.7.1 Source Definitions

In the previous notebook, we simply used the `stellar_spectrum` functions to create sources normalized at their observed K-Band magnitude. This time, we will utilize the `source_spectrum` class to generate a model fit to the known spectrophotometry. The user can find the relevant photometric data at http://vizier.u-strasbg.fr/vizier/sed/ and click download data as a VOTable.

[3]:
```
# Define 2MASS Ks bandpass and source information
bp_k = pynrc.bp_2mass('k')

# Science         source,  dist, age, sptype, Teff, [Fe/H], log_g, mag, band
args_sources = [('HR 8799', 39.0, 30,  'F0V', 7430, -0.47, 4.35, 5.24, bp_k)]

# References     source,     sptype, Teff, [Fe/H], log_g, mag, band
ref_sources = [('HD 220657', 'F8III', 5888, -0.01, 3.22, 3.04, bp_k)]

# Directory housing VOTables
# http://vizier.u-strasbg.fr/vizier/sed/
votdir = 'GTO/votables/'
```

[4]:
```
# Fit spectrum to SED photometry
i=0
name_sci, dist_sci, age, spt_sci, Teff_sci, feh_sci, logg_sci, mag_sci, bp_sci = args_
↪sources[i]
vot = votdir + name_sci.replace(' ' ,'') + '.vot'

args = (name_sci, spt_sci, mag_sci, bp_sci, vot)
kwargs = {'Teff':Teff_sci, 'metallicity':feh_sci, 'log_g':logg_sci}
src = nrc_utils.source_spectrum(*args, **kwargs)

src.fit_SED(use_err=True, robust=True)

# Final source spectrum
sp_sci = src.sp_model
```

```
[0.98216073]
```

[5]:
```
# Do the same for the reference source
name_ref, spt_ref, Teff_ref, feh_ref, logg_ref, mag_ref, bp_ref = ref_sources[i]
vot = votdir + name_ref.replace(' ' ,'') + '.vot'

args = (name_ref, spt_ref, mag_ref, bp_ref, vot)
kwargs = {'Teff':Teff_ref, 'metallicity':feh_ref, 'log_g':logg_ref}
ref = nrc_utils.source_spectrum(*args, **kwargs)

ref.fit_SED(use_err=False, robust=False, wlim=[0.5,5])
```

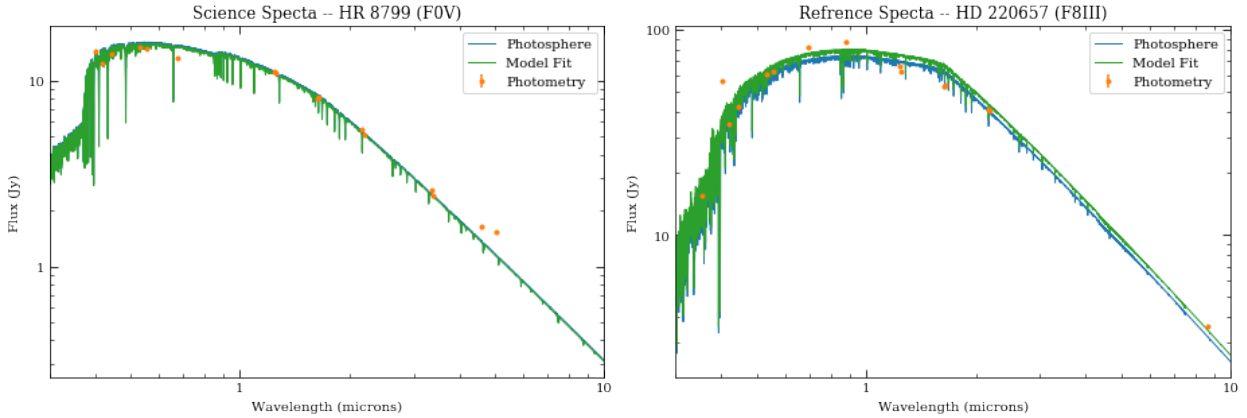
(continues on next page)

(continued from previous page)

```
# Final reference spectrum
sp_ref = ref.sp_model
```

```
[1.07580856]
```

[6]:
```
# Plot spectra
fig, axes = plt.subplots(1,2, figsize=(13,4.5))
src.plot_SED(xr=[0.3,10], ax=axes[0])
ref.plot_SED(xr=[0.3,10], ax=axes[1])

axes[0].set_title('Science Specta -- {} ({})'.format(src.name, spt_sci))
axes[1].set_title('Refrence Specta -- {} ({})'.format(ref.name, spt_ref))

fig.tight_layout()
```

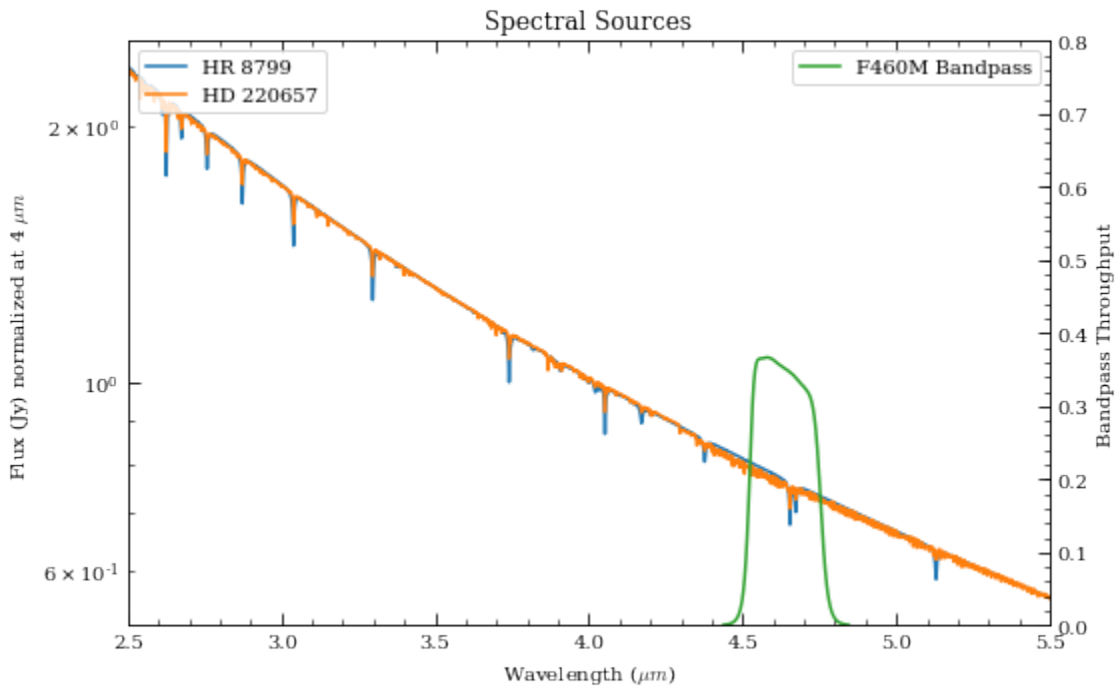


[7]:
```
# Plot the two spectra
fig, ax = plt.subplots(1,1, figsize=(8,5))

xr = [2.5,5.5]

for sp in [sp_sci, sp_ref]:
    w = sp.wave / 1e4
    ind = (w>=xr[0]) & (w<=xr[1])
    sp.convert('Jy')
    f = sp.flux / np.interp(4.0, w, sp.flux)
    ax.semilogy(w[ind], f[ind], lw=1.5, label=sp.name)
    ax.set_ylabel('Flux (Jy) normalized at 4 $\mu m$')
    sp.convert('flam')

ax.set_xlim(xr)
ax.set_xlabel(r'Wavelength ($\mu m$)')
ax.set_title('Spectral Sources')

# Overplot Filter Bandpass
bp = pynrc.read_filter('F460M', 'WEDGELYOT', 'MASKLWB')
ax2 = ax.twinx()
ax2.plot(bp.wave/1e4, bp.throughput, color='C2', label=bp.name+' Bandpass')
ax2.set_ylim([0,0.8])
ax2.set_xlim(xr)
ax2.set_ylabel('Bandpass Throughput')
```

(continues on next page)

(continued from previous page)

```
ax.legend(loc='upper left')
ax2.legend(loc='upper right')

fig.tight_layout()
```



1.7.2 Initialize Observation

Now we will initialize the high-contrast imaging class `pynrc.obs_hci` using the spectral objects and various other settings. The `obs_hci` object is a subclass of the more generalized `NIRCam` class. It implements new settings and functions specific to high-contrast imaging observations for corongraphy and direct imaging.

For this tutorial, we want to observe these targets using the `MASKLWB` coronagraph in the `F460M` filter. All wedge coronagraphic masks such as the `MASKLWB` (B=bar) should be paired with the `WEDGELYOT` pupil element. Observations in the LW channel are most commonly observed in `WINDOW` mode with a 320x320 detector subarray size. Full detector sizes are also available.

The wedge coronagraphs have an additional option to specify the location along the wedge to place your point source via the `bar_offset` keyword. If not specified, the location is automatically chosen based on the filter. A positive value will move the source to the right when viewing in V2/V3 coordiante convention (V2 to the left and V3 up). Specifying the location is a non-standard mode.

In this case, we're going to place our PSF at the narrow end of the LW bar, located `bar_offset=8` arcsec from the bar center.

[8]:
```
filt, mask, pupil = ('F460M',  'MASKLWB', 'WEDGELYOT')
wind_mode, subsize = ('WINDOW', 320)
fov_pix, oversample = (320, 2)

wfe_ref_drift = 0
obs = pynrc.obs_hci(sp_sci, sp_ref, dist_sci, filter=filt, mask=mask, pupil=pupil,
```

(continues on next page)

(continued from previous page)

```
                    wfe_ref_drift=wfe_ref_drift, fov_pix=fov_pix,␣
↪oversample=oversample,
                    wind_mode=wind_mode, xpix=subsize, ypix=subsize, verbose=True,
                    bar_offset=8)
```

```
Generating background PSF coefficients...
Generating oversampled PSFs...
Updating NIRCam reference coefficients...
Creating NIRCam reference class...
Finished.
```

Just as a reminder, all information for the reference observation is stored in the attribute `obs.nrc_ref`, which is simply it's own isolated `nrc_hci` (`NIRCam`) class. The `bar_offset` value is initialized to be the same as the science observation.

Now that we've succesffully initialized the obs_hci observations, let's specify the `wfe_ref_drift`. If this is your first time, then the `nrc_utils.wfed_coeff` function is called to determine a relationship between PSFs in the presense of WFE drift. This relationship is saved in the `PYNRC_DATA` directory as a set of polynomial coefficients. Future calculations utilize these coefficients to quickly generate a new PSF for any arbitary drift value.

[9]:
```
# WFE drift amount between rolls
# This only gets called during gen_roll_image()
# and temporarily updates obs.wfe_drift to create
# a new PSF.
obs.wfe_roll_drift = 2

# Drift amount of reference relative to nominal.
# This is simply a link to obs.nrc_ref.wfe_drift
obs.wfe_ref_drift = 10
```

1.7.3 Exposure Settings

Optimization of exposure settings are demonstrated in another tutorial, so we will not repeat that process here. We can assume that process was performed elsewhere to choose the `BRIGHT2` pattern with 10 groups and 40 total integrations. These settings apply to each roll position of the science observation as well as the for the reference observation.

[10]:
```
# Update both the science and reference observations
# These numbers come from GTO Proposal 1194
obs.update_detectors(read_mode='BRIGHT2', ngroup=10, nint=40, verbose=True)
obs.nrc_ref.update_detectors(read_mode='BRIGHT2', ngroup=4, nint=90)
```

```
New Ramp Settings:
  read_mode :  BRIGHT2
  nf        :        2
  nd2       :        0
  ngroup    :       10
  nint      :       40
New Detector Settings
  wind_mode :   WINDOW
  xpix      :      320
  ypix      :      320
  x0        :      276
  y0        :     1523
New Ramp Times
  t_group   :    2.138
```

(continues on next page)

(continued from previous page)

```
  t_frame   :    1.069
  t_int     :   21.381
  t_int_tot :   22.470
  t_exp     :  855.232
  t_acq     :  898.790
```

1.7.4 Add Planets

There are four known giant planets orbiting HR 8799 at various locations. Ideally, we would like to position them at their predicted locations on the anticipated observation date. For this case, we choose a plausible observation date of November 1, 2019. To convert between (x, y) and (r, θ), use the `nrc_utils.xy_to_rtheta` and `nrc_utils.rtheta_to_xy` functions.

When adding the planets, it doesn't matter too much which exoplanet model spectrum we decide to use since the spectra are still fairly unconstrained at these wavelengths. We do know roughly the planets' luminosities, so we can simply choose some reasonable model and renormalize it to the appropriate filter brightness. Currently, the only exoplanet spectral models available to `pynrc` are those from Spiegel & Burrows (2012).

[11]:
```
# Projected locations for date 11/01/2019
# These are prelimary positions, but within constrained orbital parameters
loc_list = [(-1.57, 0.64), (0.42, 0.87), (0.5, -0.45), (0.35,  0.20)]

# Estimated magnitudes within F460M filter
pmags = [16.0, 15.0, 14.6, 14.7]
```

[12]:
```
# Add planet information to observation class.
# These are stored in obs.planets.
# Can be cleared using obs.kill_planets().
obs.kill_planets()
for i, loc in enumerate(loc_list):
    obs.add_planet(mass=10, entropy=13, age=age, xy=loc, runits='arcsec',
                   renorm_args=(pmags[i], 'vegamag', obs.bandpass))
```

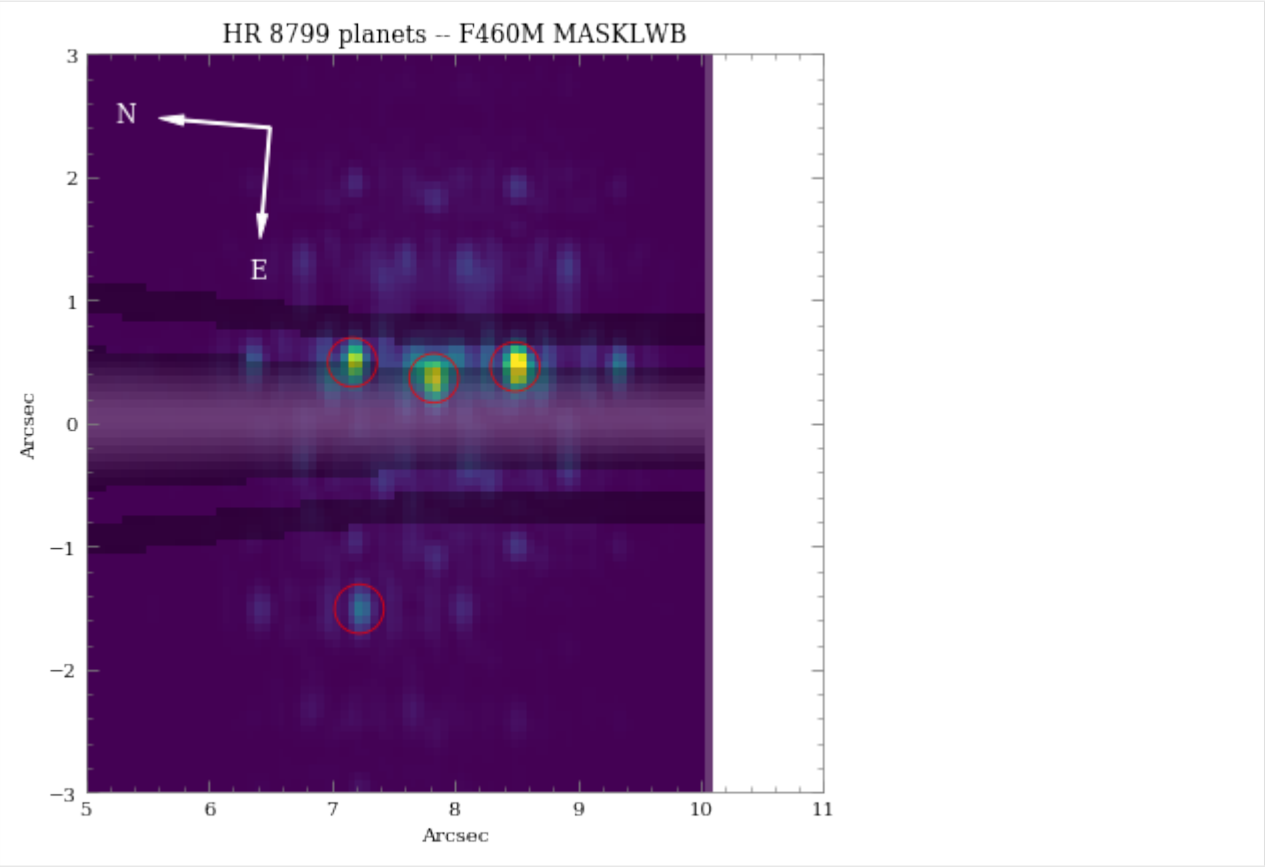
[13]:
```
# Generate and plot a noiseless slope image to make sure things look right
PA1 = 85
im_planets = obs.gen_planets_image(PA_offset=PA1)
```

[14]:
```
from matplotlib.patches import Circle
from pynrc.nrc_utils import (coron_ap_locs, build_mask_detid, fshift, pad_or_cut_to_
↪size)

fig, ax = plt.subplots(figsize=(6,6))

xasec = obs.det_info['xpix'] * obs.pix_scale
yasec = obs.det_info['ypix'] * obs.pix_scale
extent = [-xasec/2, xasec/2, -yasec/2, yasec/2]
xylim = 3

vmin = 0
vmax = 0.75*im_planets.max()
ax.imshow(im_planets, extent=extent, vmin=vmin, vmax=vmax)

# Overlay the coronagraphic mask
detid = obs.Detectors[0].detid
```

(continues on next page)

(continued from previous page)

```
im_mask = obs.mask_images[detid]
# Do some masked transparency overlays
masked = np.ma.masked_where(im_mask>0.97, im_mask)
#ax.imshow(1-masked, extent=extent, alpha=0.5)
ax.imshow(masked, extent=extent, alpha=0.3, cmap='Greys', vmin=-0.5)

xc_off = obs.bar_offset
for loc in loc_list:
    xc, yc = loc
    xc, yc = nrc_utils.xy_rot(xc, yc, PA1)
    xc += xc_off
    circle = Circle((xc,yc), radius=xylim/15., alpha=0.7, lw=1, edgecolor='red',␣
↪facecolor='none')
    ax.add_artist(circle)

xlim = ylim = np.array([-1,1])*xylim
xlim = xlim + xc_off
ax.set_xlim(xlim)
ax.set_ylim(ylim)

ax.set_xlabel('Arcsec')
ax.set_ylabel('Arcsec')

ax.set_title('{} planets -- {} {}'.format(sp_sci.name, obs.filter, obs.mask))

color = 'grey'
ax.tick_params(axis='both', color=color, which='both')
for k in ax.spines.keys():
    ax.spines[k].set_color(color)

nrc_utils.plotAxes(ax, width=1, headwidth=5, alength=0.15, angle=PA1,
                   position=(0.25,0.9), label1='E', label2='N')

fig.tight_layout()
```



As we can see, even with "perfect PSF subtraction" and no noise, it's difficult to make out planet e. This is primarily due to its location relative to the occulting mask reducing throughput combined with confusion of bright diffraction spots from nearby sources.

1.7.5 Estimated Performance

Now we are ready to determine contrast performance and sensitivites as a function of distance from the star.

Roll-Subtracted Images

First, we will create a quick simulated roll-subtracted image using the in `gen_roll_image` method. For the selected observation date of 11/1/2019, APT shows a PA range of 84° to 96°. So, we'll assume Roll 1 has PA1=85, while Roll 2 has PA2=95. In this case, "roll subtraction" simply creates two science observations at two different parallactic angles and subtracts the same reference observation from each. The two results are then de-rotated to a common PA=0 and averaged.

There is also the option to create ADI images, where the other roll position becomes the reference star by setting `no_ref=True`.

Contrast Curves

Next, we will cycle through a few WFE drift values to get an idea of potential predicted sensitivity curves. The `calc_contrast` method returns a tuple of three arrays: 1. The radius in arcsec. 2. The n-sigma contrast. 3. The n-sigma magnitude sensitivity limit (vega mag).

[15]:
```
# Cycle through a few WFE drift values
wfe_list = [0,5,10]
nsig = 5

# PA values for each roll
PA1, PA2 = (85, 95)
roll_angle = np.abs(PA2 - PA1)

# A dictionary of HDULists
hdul_dict = {}
hdul_dict2 = {}
curves = []
for i, wfe_drift in enumerate(wfe_list):
    print(wfe_drift)
    # Upate WFE reference drift value
    obs.wfe_ref_drift = wfe_drift

    # Set the final output image to be oversampled
    hdulist = obs.gen_roll_image(PA1=PA1, PA2=PA2)
    hdul_dict[wfe_drift] = hdulist

    # Generate contrast curves
    result = obs.calc_contrast(roll_angle=roll_angle, nsig=nsig)
    curves.append(result)
```

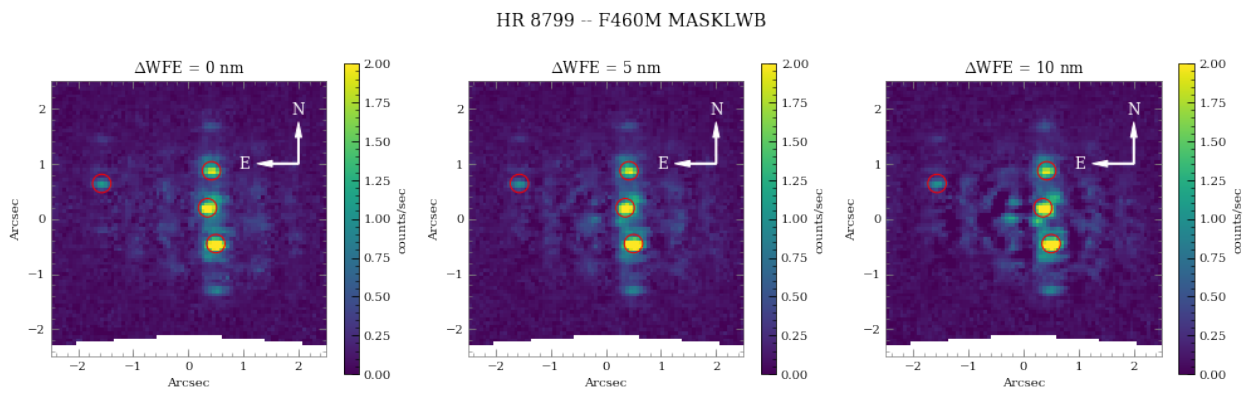
```
0
5
10
```

[16]:
```
from pynrc.obs_nircam import plot_hdulist
from matplotlib.patches import Circle

fig, axes = plt.subplots(1,3, figsize=(14,4.3))
xylim = 2.5
xlim = ylim = np.array([-1,1])*xylim

for j, wfe_drift in enumerate(wfe_list):
    ax = axes[j]
    hdul = hdul_dict[wfe_drift]

    plot_hdulist(hdul, xr=xlim, yr=ylim, ax=ax, vmin=0, vmax=2, axes_color='grey')

    # Location of planet
    for loc in loc_list:
        circle = Circle(loc, radius=xylim/15., lw=1, edgecolor='red', facecolor='none
↪')
        ax.add_artist(circle)

    ax.set_title('$\Delta$WFE = {:.0f} nm'.format(wfe_drift))

    nrc_utils.plotAxes(ax, width=1, headwidth=5, alength=0.15, position=(0.9,0.7),␣
↪label1='E', label2='N')
```

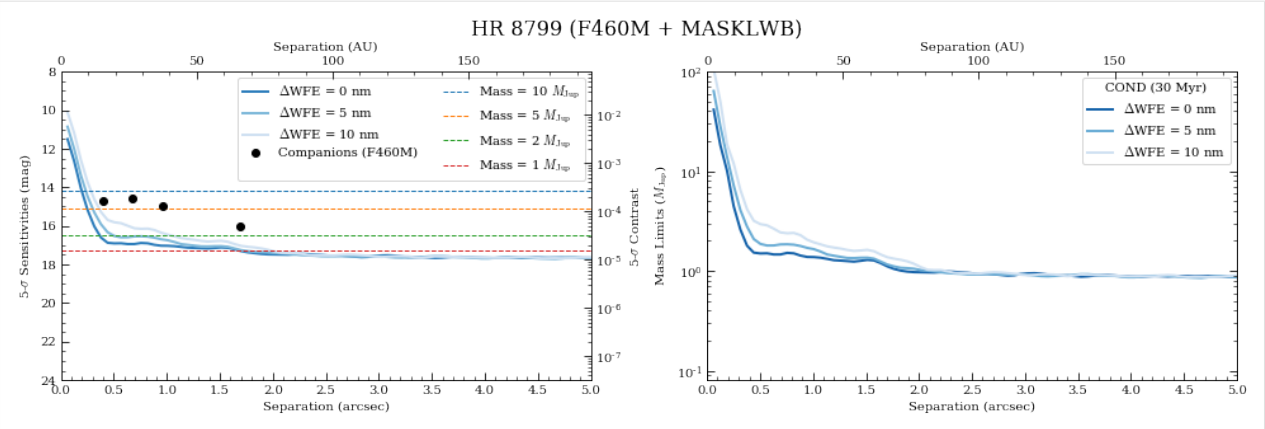
(continues on next page)

(continued from previous page)

```
fig.suptitle('{} -- {} {}'.format(name_sci, obs.filter, obs.mask), fontsize=14)
fig.tight_layout()
fig.subplots_adjust(top=0.85)
```



[17]:
```
from pynrc.obs_nircam import plot_contrasts, plot_planet_patches, plot_contrasts_mjup
import matplotlib.patches as mpatches

# fig, ax = plt.subplots(figsize=(8,5))
fig, axes = plt.subplots(1,2, figsize=(14,4.5))
xr=[0,5]
yr=[24,8]

# 1a. Plot contrast curves and set x/y limits
ax = axes[0]
ax, ax2, ax3 = plot_contrasts(curves, nsig, wfe_list, obs=obs,
                              xr=xr, yr=yr, ax=ax, return_axes=True)
# 1b. Plot the locations of exoplanet companions
label = 'Companions ({})'.format(filt)
planet_dist = [np.sqrt(x**2+y**2) for x,y in loc_list]
ax.plot(planet_dist, pmags, marker='o', ls='None', label=label, color='k', zorder=10)

# 1c. Plot Spiegel & Burrows (2012) exoplanet fluxes (Hot Start)
plot_planet_patches(ax, obs, age=age, entropy=13, av_vals=None)
ax.legend(ncol=2)

# 2. Plot in terms of MJup using COND models
ax = axes[1]
plot_contrasts_mjup(curves, nsig, wfe_list, obs=obs, age=age,
                    ax=ax, twin_ax=True, xr=xr, yr=None)
ax.set_yscale('log')
ax.set_ylim([0.08,100])
ax.legend(loc='upper right', title='COND ({:.0f} Myr)'.format(age))

fig.suptitle('{} ({} + {})'.format(name_sci, obs.filter, obs.mask), fontsize=16)

fig.tight_layout()
fig.subplots_adjust(top=0.85, bottom=0.1 , left=0.05, right=0.97)
```



The innermost Planet e is above the detection threshold as suggested by the simulated images.

[ ]:

1.8 Detailed API

1.8.1 Detector Classes

pynrc.DetectorOps

class pynrc.**DetectorOps**(*detector=481*, *wind_mode='FULL'*, *xpix=2048*, *ypix=2048*, *x0=0*, *y0=0*, *nff=None*, ***kwargs*)

Bases: pynrc.detops.det_timing

Class to hold detector operations information. Includes SCA attributes such as detector names and IDs as well as *multiaccum* class for ramp settings.

Parameters

- **detector** (*int, str*) – NIRCam detector ID (481-490) or SCA ID (A1-B5).
- **wind_mode** (*str*) – Window mode type 'FULL', 'STRIPE', 'WINDOW'.
- **xpix** (*int*) – Size of window in x-pixels for frame time calculation.
- **ypix** (*int*) – Size of window in y-pixels for frame time calculation.
- **x0** (*int*) – Lower-left x-coord position of detector window.
- **y0** (*int*) – Lower-left y-coord position of detector window.
- **nff** (*int*) – Number of fast row resets.

Keyword Arguments

- **read_mode** (*str*) – NIRCam Ramp Readout mode such as 'RAPID', 'BRIGHT1', etc.
- **nint** (*int*) – Number of integrations (ramps).
- **ngroup** (*int*) – Number of groups in a integration.
- **nf** (*int*) – Number of frames per group.
- **nd1** (*int*) – Number of drop frame after reset (before first group read).

- **nd2** (*int*) – Number of drop frames within a group (ie., groupgap).
- **nd3** (*int*) – Number of drop frames after final read frame in ramp.

Examples

Use kwargs functionality to pass keywords to the multiaccum class.

Send via a dictionary of keywords and values:

```
>>> kwargs = {'read_mode':'RAPID', 'nint':5, 'ngroup':10}
>>> d = DetectorOps(**kwargs)
```

Set the keywords directly:

```
>>> d = DetectorOps(read_mode='RAPID', nint=5, ngroup=10)
```

Attributes Summary

channel	Detector channel 'SW' or 'LW' (inferred from detector ID)
chsize	
detid	Selected Detector ID from detectors in the *detid_list* attribute.
detid_list	Allowed Detector IDs
detname	Selected Detector ID from detectors in the *scaid_list* attribute.
module	NIRCam modules A or B (inferred from detector ID)
nff	Number of fast row resets that occur before Reset Frame
nout	Number of simultaenous detector output channels stripes
ref_info	Array of reference pixel borders being read out [lower, upper, left, right].
scaid	Selected SCA ID from detectors in the *scaid_list* attribute.
scaid_list	Allowed SCA IDs
time_exp	Total photon collection time for all ramps.
time_frame	Determine frame times based on xpix, ypix, and wind_mode.
time_group	Time per group based on time_frame, nf, and nd2.
time_int	Same as time_ramp, except that 'int' follows the JWST nomenclature
time_ramp	Photon collection time for a single ramp.
time_row_reset	NFF Row Resets time per integration
time_total	Total exposure acquisition time
DetectorOps.time_total_int	

Methods Summary

make_header([filter, pupil, obs_time])	Create a generic NIRCam FITS header.
pixel_noise([fsrc, fzodi, fbg, rn, ktc, ...])	Noise values per pixel.
DetectorOps.pixel_timing_map	
times_to_dict([verbose])	Export ramp times as dictionary with option to print output to terminal.
to_dict([verbose])	Export detector settings to a dictionary.

Methods Documentation

make_header (*filter=None*, *pupil=None*, *obs_time=None*, ***kwargs*)
Create a generic NIRCam FITS header.

Parameters

- **filter** (*str*) – Name of filter element.
- **pupil** (*str*) – Name of pupil element.
- **obs_time** (*datetime*) – Specifies when the observation was considered to be executed. If not specified, then it will choose the current time. This must be a datetime object:

```
>>> datetime.datetime(2016, 5, 9, 11, 57, 5, 796686)
```

pixel_noise (*fsrc=0.0*, *fzodi=0.0*, *fbg=0.0*, *rn=None*, *ktc=None*, *idark=None*, *p_excess=None*, *ng=None*, *nf=None*, *verbose=False*, ***kwargs*)
Noise values per pixel.

Return theoretical noise calculation for the specified MULTIACCUM exposure in terms of e-/sec. This uses the pre-defined detector-specific noise properties. Can specify flux of a source as well as background and zodiacal light (in e-/sec/pix). After getting the noise per pixel per ramp (integration), value(s) are divided by the sqrt(NINT) to return the final noise

Parameters

- **fsrc** (*float or image*) – Flux of source in e-/sec/pix
- **fzodi** (*float or image*) – Flux of the zodiacal background in e-/sec/pix
- **fbg** (*float or image*) – Flux of telescope background in e-/sec/pix
- **idark** (*float or image*) – Option to specify dark current in e-/sec/pix.
- **rn** (*float*) – Option to specify Read Noise per pixel (e-).
- **ktc** (*float*) – Option to specify kTC noise (in e-). Only valid for single frame (n=1)
- **p_excess** (*array-like*) – Optional. An array or list of two elements that holds the parameters describing the excess variance observed in effective noise plots. By default these are both 0. For NIRCam detectors, recommended values are [1.0,5.0] for SW and [1.5,10.0] for LW.
- **ng** (*None or int or image*) – Option to explicitly states number of groups. This is specifically used to enable the ability of only calculating pixel noise for unsaturated groups for each pixel. If a numpy array, then it should be the same shape as *fsrc* image. By default will use *self.ngroup*.
- **verbose** (*bool*) – Print out results at the end.

Keyword Arguments **ideal_Poisson** (*bool*) – If set to True, use total signal for noise estimate, otherwise MULTIACCUM equation is used.

Notes

fsrc, fzodi, and fbg are functionally the same as they are immediately summed. They can also be single values or multiple elements (list, array, tuple, etc.). If multiple inputs are arrays, make sure their array sizes match.

times_to_dict(*verbose=False*)
Export ramp times as dictionary with option to print output to terminal.

to_dict(*verbose=False*)
Export detector settings to a dictionary.

pynrc.multiaccum

class pynrc.**multiaccum**(*read_mode='RAPID'*, *nint=1*, *ngroup=1*, *nf=1*, *nd1=0*, *nd2=0*, *nd3=0*, *nr1=1*, *nr2=1*, ***kwargs*)
Bases: object

A class for defining MULTIACCUM ramp settings. See NIRCam MULTIACCUM documentation for more details.

Parameters

- **read_mode** (*str*) – NIRCam Ramp Readout mode such as 'RAPID', 'BRIGHT1', 'DEEP8', etc., or 'CUSTOM'
- **nint** (*int*) – Number of integrations (ramps).
- **ngroup** (*int*) – Number of groups in a integration.
- **nf** (*int*) – Number of frames per group.
- **nd1** (*int*) – Number of drop frame after reset (before first group read). Default=0.
- **nd2** (*int*) – Number of drop frames within a group (ie., groupgap).
- **nd3** (*int*) – Number of drop frames after final read frame in ramp. Default=1.
- **nr1** (*int*) – Number of reset frames within first ramp. Default=0.
- **nr2** (*int*) – Number of reset frames for subsequent ramps. Default=1.

Notes

NIRCam-specific readout modes

Pattern	NF	ND2
RAPID	1	0
BRIGHT1	1	1
BRIGHT2	2	0
SHALLOW2	2	3
SHALLOW4	4	1
MEDIUM2	2	8
MEDIUM8	8	2
DEEP2	2	18
DEEP8	8	12

Attributes Summary

`nd1`	Number of drop frame after reset (before first group read).
`nd2`	Number of drop frames within a group (aka, group-gap).
`nd3`	Number of drop frames after final read frame in ramp.
`nf`	Number of frames per group.
`ngroup`	Number of groups in a ramp (integration).
`nint`	Number of ramps (integrations) in an exposure.
`patterns_list`	Allowed NIRCam MULTIACCUM patterns
`read_mode`	Selected Read Mode in the *patterns_list* attribute.

Methods Summary

to_dict([verbose])	Export ramp settings to a dictionary.

Methods Documentation

to_dict(*verbose=False*)
Export ramp settings to a dictionary.

1.8.2 Observation Classes

pynrc.NIRCam

class pynrc.**NIRCam**(*filter='F210M'*, *pupil=None*, *mask=None*, *module='A'*, *ND_acq=False*, *apname=None*, ***kwargs*)

Bases: object

NIRCam base instrument class

Creates a NIRCam instrument class that holds all the information pertinent to an observation using a given channel/module (SWA, SWB, LWA, LWB).

The user merely inputs the filter name, pupil element, coronagraphic mask, and module A or B. Based on these settings, a sequence of detector objects will be created that spans the correct channel. For instance, a filter value

of 'F210M' paired with module='A' will generate four detector objects, one for each of the four SWA SCAs 481-484 (A1-A4, if you prefer).

A number of other options are passed via the kwargs parameter that allow setting of the detector readout mode, MULTIACCUM settings, and settings for the PSF calculation to be passed to WebbPSF.

Parameters

- **filter** (*str*) – Name of NIRCam filter.
- **pupil** (*str, None*) – NIRCam pupil elements such as grisms or lyot stops (default: None).
- **mask** (*str, None*) – Specify which coronagraphic occulter (default: None).
- **module** (*str*) – NIRCam Module 'A' or 'B' (default: 'A').
- **wfe_drift** (*bool*) – Option to calculate and use WFE drift PSF modifications. Can be turned on later with the `wfe_drift` attribute.
- **wfe_field** (*bool*) – Option to use measured field-dependent SI WFE. Can be turned on later with the `wfe_field` attribute.
- **apname** (*str*) – Use SIAF aperture name insted of pupil, mask, module, etc.
- **ND_acq** (*bool*) – ND square acquisition (default: False).
- **ice_scale** (*float*) – Add in additional OTE H2O absorption. This is a scale factor relative to 0.0131 um thickness. No ice contributions by default.
- **nvr_scale** (*float*) – Add in additional NIRCam non-volatile residue. This is a scale factor relative to 0.280 um thickness. If set to None, then assumes a scale factor of 1.0 as is contained in the NIRCam filter curves. Setting nvr_scale=0 will remove these contributions.

Notes

Detector Settings

The following keyword arguments will be passed to both the *DetectorOps* and *multiaccum* instances. These can be set directly upon initialization of the of the NIRCam instances or updated later using the *update_detectors()* method. All detector instances will be considered to operate in this mode.

Keyword Arguments

- **det_list** (*list, None*) – List of detector names (481, 482, etc.) to consider. If not set, then defaults are chosen based on observing mode.
- **wind_mode** (*str*) – Window mode type 'FULL', 'STRIPE', 'WINDOW'.
- **xpix** (*int*) – Size of window in x-pixels for frame time calculation.
- **ypix** (*int*) – Size of window in y-pixels for frame time calculation.
- **x0** (*int*) – Lower-left x-coord position of detector window.
- **y0** (*int*) – Lower-left y-coord position of detector window.
- **read_mode** (*str*) – NIRCam Ramp Readout mode such as 'RAPID', 'BRIGHT1', etc.
- **nint** (*int*) – Number of integrations (ramps).
- **ngroup** (*int*) – Number of groups in a integration.
- **nf** (*int*) – Number of frames per group.
- **nd1** (*int*) – Number of drop frame after reset (before first group read).

- **nd2** (*int*) – Number of drop frames within a group (ie., groupgap).
- **nd3** (*int*) – Number of drop frames after final read frame in ramp.

Notes

PSF Settings

The following keyword arguments will be passed to the PSF generation function `gen_psf_coeff()` which calls `webbpsf`. These can be set directly upon initialization of the of the NIRCam instances or updated later using the *`update_psf_coeff()`* method.

Keyword Arguments

- **fov_pix** (*int*) – Size of the PSF FoV in pixels (real SW or LW pixels). The defaults depend on the type of observation. Odd number place the PSF on the center of the pixel, whereas an even number centers it on the "crosshairs."
- **oversample** (*int*) – Factor to oversample during WebbPSF calculations. Default 2 for coronagraphy and 4 otherwise.
- **offset_r** (*float*) – Radial offset from the center in arcsec.
- **offset_theta** (*float*) – Position angle for radial offset, in degrees CCW.
- **opd** (*tuple or HDUList*) – Tuple (file, slice) or filename or HDUList specifying OPD.
- **include_si_wfe** (*bool*) – Include SI WFE measurements? Default=True. if `wfe_field` is *True*, then `include_si_wfe=True`.
- **tel_pupil** (*str*) – File name or HDUList specifying telescope entrance pupil.
- **jitter** (*str or None*) – Currently either 'gaussian' or None.
- **jitter_sigma** (*float*) – If `jitter = 'gaussian'`, then this is the size of the blurring effect.
- **save** (*bool*) – Save the resulting PSF coefficients to a file? (default: True)
- **force** (*bool*) – Forces a recalcuation of PSF even if saved PSF exists. (default: False)
- **quick** (*bool*) – Only perform a fit over the filter bandpass with a lower default polynomial degree fit.
- **use_legendre** (*bool*) – Fit with Legendre polynomials, an orthonormal basis set.

Examples

Basic example of generating a full frame NIRCam observation:

```
>>> inst = pynrc.NIRCam('F210M', module='A', read_mode='DEEP8', nint=10, ngroup=5)
```

is the same as:

```
>>> inst = pynrc.NIRCam('F210M', module='A')
>>> inst.update_detectors(read_mode='DEEP8', nint=10, ngroup=5)
```

Attributes Summary

ND_acq	Use Coronagraphic ND acquisition square?
bandpass	The wavelength dependent bandpass.
NIRCam.bar_offset	
channel	NIRCam wavelength channel ('SW' or 'LW').
det_info	Dictionary housing detector info parameters and keywords.
filter	Name of filter bandpass
filter_list	List of allowable filters.
mask	Name of coronagraphic mask element
mask_list	List of allowable coronagraphic mask values.
module	NIRCam modules A or B
multiaccum	*multiaccum* object
multiaccum_times	Exposure timings in dictionary
pixelscale	Pixel scale in arcsec/pixel
psf_coeff	Cube of polynomial coefficients used to generate a PSF.
psf_coeff_bg	Cube of polynomial coefficients used to generate a PSF.
psf_info	Info used to create psf_coeff.
psf_info_bg	Info used to create psf_coeff for faint background sources.
pupil	Name of pupil element
pupil_list	List of allowable pupil mask values.
well_level	Detector well level in units of electrons
wfe_drift	Enable/Disable WFE Drift calculation

Methods Summary

bg_zodi([zfact])	Zodiacal background flux.
gen_exposures([sp, im_slope, file_out, ...])	Generate raw mock data.
gen_psf([sp, return_oversample, use_bg_psf, ...])	PSF image
plot_bandpass([ax, color, title])	Plot the instrument bandpass on a selected axis.
ramp_optimize(sp[, sp_bright, is_extended, ...])	Optimize ramp settings.
sat_limits([sp, bp_lim, units, well_frac, ...])	Saturation limits.
saturation_levels(sp[, full_size, ngroup, image])	Saturation levels.
sensitivity([nsig, units, sp, verbose])	Sensitivity limits.
update_detectors([verbose, det_list])	Generates a list of detector objects depending on module and channel.
update_psf_coeff([fov_pix, oversample, ...])	Create new PSF coefficients.

Methods Documentation

bg_zodi (*zfact=None*, ***kwargs*)
Zodiacal background flux.

There are options to call *jwst_backgrounds* to obtain better predictions of the background. Specify keywords *ra*, *dec*, and *thisday* to use *jwst_backgrounds*.

Returned values are in units of e-/sec/pixel

Parameters **zfact** (*float*) – Factor to scale Zodiacal spectrum (default 2.5)

Keyword Arguments

- **ra** (*float*) – Right ascension in decimal degrees
- **dec** (*float*) – Declination in decimal degrees
- **thisday** (*int*) – Calendar day to use for background calculation. If not given, will use the average of visible calendar days.

Notes

Representative values for zfact:

- 0.0 - No zodiacal emission
- 1.0 - Minimum zodiacal emission from JWST-CALC-003894
- 1.2 - Required NIRCam performance
- 2.5 - Average (default)
- 5.0 - High
- 10.0 - Maximum

gen_exposures (*sp=None*, *im_slope=None*, *file_out=None*, *return_results=None*, *targ_name=None*, *timeFileNames=False*, *DMS=True*, *det_name=None*, *dark=True*, *bias=True*, *det_noise=True*, *nproc=None*, ***kwargs*)
Generate raw mock data.

Create a series of ramp integration saved to FITS files based on the current NIRCam settings. This method calls the `gen_fits()` function, which in turn calls the detector noise generator `ngNRC`.

Only works on one detector at a time.

Currently, this image simulator does NOT take into account:

- QE variations across a pixel's surface
- Intrapixel Capacitance (IPC)
- Post-pixel Coupling (PPC) due to ADC "smearing"
- Pixel non-linearity
- Persistence/latent image
- Optical distortions
- Zodiacal background roll off for grism edges
- Cosmic Rays

TODO: Double-check the output for grism data w.r.t sci_to_det().

Parameters

- **im_slope** (*numpy array, None*) – Pass the slope image directly. If not set, then a slope image will be created from the input spectrum keyword. This should include zodiacal light emission, but not dark current. Make sure this array is in detector coordinates.
- **sp** (`pysynphot.spectrum`, None) – A pysynphot spectral object. If not specified, then it is assumed that we're looking at blank sky.
- **targ_name** (*str, None*) – A target name for the exposure file's header.
- **file_out** (*str, None*) – Path and name of output FITs files. Time stamps will be automatically inserted for unique file names if `timeFileNames=True`.
- **timeFileNames** (*bool*) – Save the exposure times in the file name? This is useful to see the timing, but also makes it a little harder to combine INTs later for DMS simulations.
- **DMS** (*bool*) – Create DMS data file and header format? Otherwise, the output is more similar to ISIM CV2/3 and OTIS campaigns.
- **return_results** (*bool, None*) – By default, we return results if file_out is not set and the results are not returned if file_out is set. This decision is based on the large amount of data and memory usage this method may incur if the results were always returned by default. Instead, it's better to save the FITs to disk, especially if NINTs is large. We include the return_results keyword if the user would like to do both (or neither).
- **det_name** (*str, None*) – Name of detector (A1-B5). If not found or set to None, then the first detector in *self.det_list* and *self.Detectors* is used.
- **dark** (*bool*) – Include the dark current?
- **bias** (*bool*) – Include the bias frame?
- **nproc** (*int*) – Advanced usage to explicitly specify the number of processors to use. Otherwise, a function is called to determine the optimum number of processes based on available memory and number of ramps being generated.
- **det_noise** (*bool*) – Include detector noise components? If set to False, then only perform Poisson noise. Darks and biases are also excluded.

Keyword Arguments

- **zfact** (*float*) – Factor to scale Zodiacal spectrum (default 2.5)
- **ra** (*float*) – Right ascension in decimal degrees
- **dec** (*float*) – Declination in decimal degrees
- **thisday** (*int*) – Calendar day to use for background calculation. If not given, will use the average of visible calendar days.

gen_psf(*sp=None*, *return_oversample=False*, *use_bg_psf=False*, *wfe_drift=None*, *coord_vals=None*, *coord_frame='tel'*, *bar_offset=None*, *return_hdul=False*, ***kwargs*)

PSF image

Create a PSF image from instrument settings. The image is noiseless and doesn't take into account any non-linearity or saturation effects, but is convolved with the instrument throughput. Pixel values are in counts/sec. The result is effectively an idealized slope image (no background).

If no spectral dispersers (grisms or DHS), then this returns a single image or list of images if sp is a list of spectra. By default, it returns only the detector-sampled PSF, but setting return_oversample=True will also return a set of oversampled images as a second output.

Parameters

- **sp** (pysynphot.spectrum) – If not specified, the default is flat in phot lam (equal number of photons per spectral bin). The default is normalized to produce 1 count/sec within that bandpass, assuming the telescope collecting area and instrument bandpass. Coronagraphic PSFs will further decrease this due to the smaller pupil size and coronagraphic spot.
- **return_oversample** (*bool*) – If True, then also returns the oversampled version of the PSF
- **use_bg_psf** (*bool*) – If a coronagraphic observation, off-center PSF is different.
- **wfe_drift** (*float or None*) – Wavefront error drift amplitude in nm. The attribute wfe_drift needs to be set to True.
- **coord_vals** (*tuple or None*) – Coordinates (in arcsec or pixels) to calculate field-dependent PSF. The attribute wfe_field must be True.
- **coord_frame** (*str*) –

  Type of desired output coordinates.

  - 'tel': arcsecs V2,V3
  - 'sci': pixels, in conventional DMS axes orientation
  - 'det': pixels, in raw detector read out axes orientation
  - 'idl': arcsecs relative to aperture reference location.
- **bar_offset** (*float or None*) – For bar masks, the position along the bar to place the PSF (arcsec). Use *offset_bar()* for filter-dependent locations. If both set to None, then decide location based on selected filter. A positive value will move the source to the right when viewing V2 to the left and V3 up.
- **return_hdul** (*bool*) – TODO: Return PSFs in an HDUList rather than set of arrays

plot_bandpass(*ax=None*, *color=None*, *title=None*, ***kwargs*)

Plot the instrument bandpass on a selected axis. Can pass various keywords to matplotlib.plot function.

Parameters

- **ax** (*matplotlib.axes, optional*) – Axes on which to plot bandpass.
- **color** – Color of bandpass curve.
- **title** (*str*) – Update plot title.

Returns *matplotlib.axes* – Updated axes

ramp_optimize(*sp*, *sp_bright=None*, *is_extended=False*, *patterns=None*, *snr_goal=None*, *snr_frac=0.02*, *tacq_max=None*, *tacq_frac=0.1*, *well_frac_max=0.8*, *nint_min=1*, *nint_max=5000*, *ng_min=2*, *ng_max=None*, *return_full_table=False*, *even_nints=False*, *verbose=False*, ***kwargs*)

Optimize ramp settings.

Find the optimal ramp settings to observe a spectrum based on input constraints. This function quickly runs through each detector readout pattern and calculates the acquisition time and SNR for all possible settings of NINT and NGROUP that fulfill the SNR requirement (and other constraints).

The final output table is then filtered, removing those exposure settings that have the same exact acquisition times but worse SNR. Further "obvious" comparisons are done that exclude settings where there is another setting that has both better SNR and less acquisition time. The best results are then sorted by an efficiency metric (SNR / sqrt(acq_time)). To skip filtering of results, set return_full_table=True.

The result is an AstroPy Table.

Parameters

- **sp** (`pysynphot.spectrum`) – A pysynphot spectral object to calculate SNR.
- **sp_bright** (`pysynphot.spectrum`, None) – Same as sp, but optionally used to calculate the saturation limit (treated as brightest source in field). If a coronagraphic mask observation, then this source is assumed to be occulted and sp is fully unocculted.
- **is_extended** (*bool*) – Treat sp source as extended object, then in units/arcsec^2
- **snr_goal** (*float*) – Minimum required SNR for source. For grism, this is the average SNR for all wavelength.
- **snr_frac** (*float*) – Give fractional buffer room rather than strict SNR cut-off.
- **tacq_max** (*float*) – Maximum amount of acquisition time in seconds to consider.
- **tacq_frac** (*float*) – Fractional amount of time to consider exceeding tacq_max.
- **patterns** (*numpy array*) – Subset of MULTIACCUM patterns to check, otherwise check all.
- **nint_min/max** (*int*) – Min/max number of desired integrations.
- **ng_min/max** (*int*) – Min/max number of desired groups in a ramp.
- **well_frac_max** (*float*) – Maximum level that the pixel well is allowed to be filled. Fractions greater than 1 imply hard saturation, but the reported SNR will not be aware of any saturation that may occur to sp.
- **even_nints** (*bool*) – Return only the even NINTS
- **return_full_table** (*bool*) – Don't filter or sort the final results (ingores event_ints).
- **verbose** (*bool*) – Prints out top 10 results.

Keyword Arguments

- **zfact** (`float`) – Factor to scale Zodiacal spectrum (default 2.5)
- **ra** (`float`) – Right ascension in decimal degrees
- **dec** (`float`) – Declination in decimal degrees
- **thisday** (`int`) – Calendar day to use for background calculation. If not given, will use the average of visible calendar days.
- **ideal_Poisson** (`bool`) – Use total signal for noise estimate? Otherwise MULTIACCUM equation is used. Default = True
- **rad_EE** (`int`) – Extraction aperture radius (in pixels) for imaging mode.
- **dw_bin** (`float`) – Delta wavelength to calculate spectral sensitivities for grisms and DHS.
- **ap_spec** (`float, int`) – Instead of dw_bin, specify the spectral extraction aperture in pixels. Takes priority over dw_bin. Value will get rounded up to nearest int.

Note: The keyword arguments ra, dec, thisday are not recommended for use given the amount of time it takes to query the web server. Instead, use `bg_zodi()` to match a zfact estimate.

Returns *astropy table* – A sorted and filtered table of ramp options.

sat_limits(*sp=None*, *bp_lim=None*, *units='vegamag'*, *well_frac=0.8*, *ngroup=None*, *trim_psf=33*, *verbose=False*, ***kwargs*)

Saturation limits.

Generate the limiting magnitude (80% saturation) with the current instrument parameters (filter and ramp settings) assuming some spectrum. If no spectrum is defined, then a G2V star is assumed.

The user can also define a separate bandpass in which to determine the limiting magnitude that will cause the current NIRCam bandpass to saturate.

Parameters

- **sp** (`pysynphot.spectrum`) – Spectrum to determine saturation limit.
- **bp_lim** (`pysynphot.obsbandpass`) – Bandpass to report limiting magnitude.
- **units** (*str*) – Output units (defaults to vegamag).
- **well_frac** (*float*) – Fraction of full well to consider 'saturated'.
- **ngroup** (*int, None*) – Option to specify the number of groups to determine integration time. If not set, then the default is to use those specified in the Detectors class. Can set ngroup=0 for the so-called Zero Frame in the event there are multiple reads per group.
- **trim_psf** (*int, None*) – Option to crop the PSF coefficient around the brightest pixel. For PSFs with large *fov_pix* values, this option helps speed up the saturation limit calculation. Afterall, we're usually only interested in the brightest pixel when calculating saturation limits. Set to *None* to use the 'fov_pix' value. Default = 33 (detector pixels).
- **verbose** (*bool*) – Print result details.

Example

```
>>> nrc = pynrc.NIRCam('F430M') # Initiate NIRCam observation
>>> sp_A0V = pynrc.stellar_spectrum('A0V') # Define stellar spectral type
>>> bp_k = S.ObsBandpass('steward,k') # Pysynphot K-Band bandpass
>>> bp_k.name = 'K-Band'
>>> mag_lim = nrc.sat_limits(sp_A0V, bp_k, verbose=True)
```

Returns K-Band Limiting Magnitude for F430M assuming A0V source.

saturation_levels(*sp*, *full_size=True*, *ngroup=2*, *image=None*, ***kwargs*)

Saturation levels.

Create image showing level of saturation for each pixel. Can either show the saturation after one frame (default) or after the ramp has finished integrating (ramp_sat=True).

Parameters

- **sp** (`pysynphot.spectrum`) – A pysynphot spectral object (normalized).
- **full_size** (*bool*) – Expand (or contract) to size of detector array? If False, use fov_pix size.
- **ngroup** (*int*) – How many group times to determine saturation level? If this number is higher than the total groups in ramp, then a warning is produced. The default is ngroup=2, A value of 0 corresponds to the so-called "zero-frame," which is the very first frame that is read-out and saved separately. This is the equivalent to ngroup=1 for RAPID and BRIGHT1 observations.
- **image** (*ndarray*) – Rather than generating an image on the fly, pass a pre-computed slope image. Overrides *sp* and *full_size*

sensitivity (*nsig=10*, *units=None*, *sp=None*, *verbose=False*, ***kwargs*)
Sensitivity limits.

Convenience function for returning the point source (and surface brightness) sensitivity for the given instrument setup. See bg_sensitivity() for more details.

Parameters

- **sp** (`pysynphot.spectrum`) – Input spectrum to use for determining sensitivity. Only the spectral shape matters, unless `forwardSNR=True`.
- **nsig** (*int, float*) – Desired nsigma sensitivity (default 10).
- **units** (*str*) – Output units (defaults to uJy for grisms, nJy for imaging).
- **verbose** (*bool*) – Print result details.

Keyword Arguments

- **forwardSNR** (*bool*) – Find the SNR of the input spectrum instead of sensitivity.
- **zfact** (*float*) – Factor to scale Zodiacal spectrum (default 2.5)
- **ideal_Poisson** (*bool*) – If set to True, use total signal for noise estimate, otherwise MULTIACCUM equation is used.
- **rad_EE** (*float*) – Extraction aperture radius (in pixels) for imaging mode.
- **dw_bin** (*float*) – Delta wavelength for spectral sensitivities (grisms & DHS).
- **ap_spec** (*int, float*) – Instead of dw_bin, specify the spectral extraction aperture in pixels. Takes priority over dw_bin. Value will get rounded up to nearest int.

update_detectors (*verbose=False*, *det_list=None*, ***kwargs*)
Generates a list of detector objects depending on module and channel. This function will get called any time a filter, pupil, mask, or module is modified by the user directly:

```
>>> nrc = pynrc.NIRCam('F430M', ngroup=10, nint=5)
>>> nrc.filter = 'F444W'
```

If the user wishes to change any properties of the multiaccum ramp or detector readout mode, pass those arguments through this function rather than creating a whole new NIRCam() instance. For example:

```
>>> nrc = pynrc.NIRCam('F430M', ngroup=10, nint=5)
>>> nrc.update_detectors(ngroup=2, nint=10, wind_mode='STRIPE', ypix=64)
```

A dictionary of the keyword settings can be referenced in `det_info`. This dictionary cannot be modified directly.

Parameters

- **det_list** (*list, None*) – List of detector names (481, 482, etc.) to consider. If not set, then defaults are chosen based on observing mode.
- **verbose** (*bool*) – Print out ramp and detector settings.

Keyword Arguments

- **wind_mode** (*str*) – Window mode type ‘FULL’, ‘STRIPE’, ‘WINDOW’.
- **xpix** (*int*) – Size of window in x-pixels for frame time calculation.
- **ypix** (*int*) – Size of window in y-pixels for frame time calculation.
- **x0** (*int*) – Lower-left x-coord position of detector window.

- **y0** (*int*) – Lower-left y-coord position of detector window.
- **read_mode** (*str*) – NIRCam Ramp Readout mode such as 'RAPID', 'BRIGHT1', etc.
- **nint** (*int*) – Number of integrations (ramps).
- **ngroup** (*int*) – Number of groups in a integration.
- **nf** (*int*) – Number of frames per group.
- **nd1** (*int*) – Number of drop frame after reset (before first group read).
- **nd2** (*int*) – Number of drop frames within a group (ie., groupgap).
- **nd3** (*int*) – Number of drop frames after final read frame in ramp.
- **nr1** (*int*) – Number of reset frames within first ramp.
- **nr2** (*int*) – Number of reset frames for subsequent ramps.

update_psf_coeff(*fov_pix=None*, *oversample=None*, *offset_r=None*, *offset_theta=None*, *tel_pupil=None*, *opd=None*, *include_si_wfe=None*, *jitter=None*, *jitter_sigma=None*, *bar_offset=None*, *save=None*, *force=False*, *use_legendre=None*, *quick=None*, ***kwargs*)

Create new PSF coefficients.

Generates a set of PSF coefficients from a sequence of WebbPSF images. These coefficients can then be used to generate a sequence of monochromatic PSFs (useful if you need to make hundreds of PSFs for slitless grism or DHS observations) that are subsequenty convolved with the the instrument throughput curves and stellar spectrum. The coefficients are stored in `psf_coeff` attribute.

A corresponding dictionary `psf_info` is also saved that contains the size of the FoV (in detector pixels) and oversampling factor that was used to generate the coefficients.

While originally created for coronagraphy, grism, and DHS observations, this method is actually pretty quick, so it has become the default for imaging as well.

Parameters

- **fov_pix** (*int*) – Size of the PSF FoV in pixels (real SW or LW pixels). The defaults depend on the type of observation. Odd number place the PSF on the center of the pixel, whereas an even number centers it on the "crosshairs."
- **oversample** (*int*) – Factor to oversample during WebbPSF calculations. Default 2 for coronagraphy and 4 otherwise.
- **offset_r** (*float*) – Radial offset from the center in arcsec.
- **offset_theta** (*float*) – Position angle for radial offset, in degrees CCW.
- **opd** (*tuple, str, or HDUList*) – Tuple (file, slice) or filename or HDUList specifying OPD.
- **include_si_wfe** (*bool*) – Include SI WFE measurements? Default=True.
- **tel_pupil** (*str or HDUList*) – File name or HDUList specifying telescope entrance pupil.
- **jitter** (*str*) – Either 'gaussian' or 'none'.
- **jitter_sigma** (*float*) – If `jitter = 'gaussian'`, then this is the size of the blurring effect.
- **save** (*bool*) – Save the resulting PSF coefficients to a file? (default: True)
- **force** (*bool*) – Forces a recalcuation of PSF even if saved PSF exists. (default: False)
- **quick** (*bool*) – Only perform a fit over the filter bandpass with a lower default polynomial degree fit. Default is True for narroband, False otherwise.

- **use_legendre** (*bool*) – Fit with Legendre polynomials, an orthonormal basis set.

pynrc.nrc_hci

class pynrc.**nrc_hci**(*wind_mode='WINDOW'*, *xpix=320*, *ypix=320*, *wfe_drift=True*, *verbose=False*, ***kwargs*)

Bases: pynrc.pynrc_core.NIRCam

NIRCam coronagraphy (and direct imaging)

Subclass of the *NIRCam* instrument class with updates for PSF generation of off-axis PSFs. If a coronagraph is not present, then this is effetively the same as the *NIRCam* class.

Parameters

- **wind_mode** (*str*) – 'FULL', 'STRIPE', or 'WINDOW'
- **xpix** (*int*) – Size of the detector readout along the x-axis. The detector is assumed to be in window mode unless the user explicitly sets wind_mode='FULL'.
- **ypix** (*int*) – Size of the detector readout along the y-axis. The detector is assumed to be in window mode unless the user explicitly sets wind_mode='FULL'.

Methods Summary

gen_offset_psf(offset_r, offset_theta[, sp, ...])	Create a PSF offset from center FoV

Methods Documentation

gen_offset_psf(*offset_r*, *offset_theta*, *sp=None*, *return_oversample=False*)

Create a PSF offset from center FoV

Generate some off-axis PSF at a given (r,theta) offset from center. This function is mainly for coronagraphic observations where the off-axis PSF varies w.r.t. position. The PSF is centered in the resulting image.

Parameters

- **offset_r** (*float*) – Radial offset of the target from center in arcsec.
- **offset_theta** (*float*) – Position angle for that offset, in degrees CCW (+Y).

Keyword Arguments

- **sp** (None, pysynphot.spectrum) – If not specified, the default is flat in phot lam (equal number of photons per spectral bin).
- **return_oversample** (*bool*) – Return either the pixel-sampled or oversampled PSF.

pynrc.obs_hci

class pynrc.**obs_hci**(*sp_sci*, *sp_ref*, *distance*, *wfe_ref_drift=10*, *wfe_roll_drift=0*, *offset_list=None*, *wind_mode='WINDOW'*, *xpix=320*, *ypix=320*, *disk_hdu=None*, *verbose=False*, ***kwargs*)

Bases: pynrc.obs_nircam.nrc_hci

NIRCam coronagraphic observations

Subclass of the *nrc_hci* instrument class used to observe stars (plus exoplanets and disks) with either a coronagraph or direct imaging.

The main concept is to generate a science target of the primary source along with a simulated disk structure. Planets are further added to the astronomical scene. A separate reference source is also defined for PSF subtraction, which contains a specified WFE. A variety of methods exist to generate slope images and analyze the PSF-subtracted results via images and contrast curves.

Parameters

- **sp_sci** (pysynphot.spectrum) – A pysynphot spectrum of science target (e.g., central star). Should already be normalized to the apparent flux.
- **sp_ref** (pysynphot.spectrum or None) – A pysynphot spectrum of reference target. Should already be normalized to the apparent flux.
- **distance** (*float*) – Distance in parsecs to the science target. This is used for flux normalization of the planets and disk.
- **wfe_ref_drift** (*float*) – WFE drift in nm between the science and reference targets. Expected values are between ~3-10 nm.
- **wfe_roll_drift** (*float*) – WFE drift in nm between science roll angles. Default=0.
- **wind_mode** (*str*) – 'FULL', 'STRIPE', or 'WINDOW'
- **xpix** (*int*) – Size of the detector readout along the x-axis. The detector is assumed to be in window mode unless the user explicitly sets wind_mode='FULL'.
- **ypix** (*int*) – Size of the detector readout along the y-axis. The detector is assumed to be in window mode unless the user explicitly sets wind_mode='FULL'.
- **disk_hdu** (*HDUList*) – A model of the disk in photons/sec. This requires header keywords PIXSCALE (in arcsec/pixel) and DISTANCE (in pc).

Attributes Summary

planets	Planet info (if any exists)
wfe_ref_drift	WFE drift (nm) of ref obs relative to sci obs

Methods Summary

add_planet([atmo, mass, age, entropy, xy, ...])	Insert a planet into observation.
kill_planets()	Remove planet info
calc_contrast([hdu_diff, roll_angle, nsig, ...])	Create contrast curve.
gen_disk_image([PA_offset, use_cmask])	Create image of just disk.
gen_planets_image([PA_offset, quick_PSF, ...])	Create image of just planets.
gen_roll_image([PA1, PA2, zfact, ...])	Make roll-subtracted image.
planet_spec([Av])	Exoplanet spectrum.
saturation_levels([ngroup, do_ref, image])	Saturation levels
star_flux([fluxunit, sp])	Stellar flux.

Methods Documentation

add_planet(*atmo='hy3s'*, *mass=10*, *age=100*, *entropy=10*, *xy=None*, *rtheta=None*, *runits='AU'*, *Av=0*, *renorm_args=None*, *sptype=None*, *accr=False*, *mmdot=None*, *mdot=None*, *accr_rin=2*, *truncated=False*)

Insert a planet into observation.

Add exoplanet information that will be used to generate a point source image using a spectrum from Spiegel & Burrows (2012). Use self.kill_planets() to delete them.

Coordinate convention is for +N up and +E to left.

Parameters

- **atmo** (*str*) – A string consisting of one of four atmosphere types: ['hy1s', 'hy3s', 'cf1s', 'cf3s'].
- **mass** (*int*) – Number 1 to 15 Jupiter masses.
- **age** (*float*) – Age in millions of years (1-1000).
- **entropy** (*float*) – Initial entropy (8.0-13.0) in increments of 0.25
- **sptype** (*str*) – Instead of using a exoplanet spectrum, specify a stellar type.
- **renorm_args** (*dict*) – Pysynphot renormalization arguments in case you want very specific luminosity in some bandpass. Includes (value, units, bandpass).
- **Av** (*float*) – Extinction magnitude (assumes Rv=4.0) of the exoplanet due to being embedded in a disk.
- **xy** (*tuple, None*) – (X,Y) position in sky coordinates of companion (N up, E left).
- **rtheta** (*tuple, None*) – Radius and position angle relative to stellar position. Alternative to xy keyword
- **runits** (*str*) – What are the spatial units? Valid values are 'AU', 'asec', or 'pix'.
- **accr** (*bool*) – Include accretion? default: False
- **mmdot** (*float*) – From Zhu et al. (2015), the Mjup^2/yr value. If set to None then calculated from age and mass.
- **mdot** (*float*) – Or use mdot (Mjup/yr) instead of mmdot.
- **accr_rin** (*float*) – Inner radius of accretion disk (units of RJup; default: 2)
- **truncated** (*bool*) – Full disk or truncated (ie., MRI; default: False)?

kill_planets()
Remove planet info

calc_contrast(*hdu_diff=None*, *roll_angle=10*, *nsig=1*, *exclude_disk=True*, *exclude_planets=True*, *no_ref=False*, *func_std=<function std>*, ***kwargs*)
Create contrast curve.

Generate n-sigma contrast curve for the current observation settings. Make sure that MULTIACCUM parameters are set for both the main class (`self.update_detectors()`) as well as the reference target class (`self.nrc_ref.update_detectors()`).

Parameters

- **hdu_diff** (*HDUList*) – Option to pass an already pre-made difference image.
- **roll_angle** (*float*) – Telescope roll angle (deg) between two observations. If set to 0 or None, then only one roll will be performed. If value is >0, then two rolls are performed, each using the specified MULTIACCUM settings (doubling the effective exposure time).
- **nsig** (*float*) – n-sigma contrast curve.
- **exclude_disk** (*bool*) – Ignore disk when generating image?
- **exclude_planets** (*bool*) – Ignore planets when generating image?
- **no_ref** (*bool*) – Exclude reference observation. Subtraction is then Roll1-Roll2.
- **func_std** (*func*) – The function to use for calculating the radial standard deviation.

Keyword Arguments

- **zfact** (*float*) – Zodiacal background factor (default=2.5)
- **exclude_noise** (*bool*) – Don't add random Gaussian noise (detector+photon)?
- **ideal_Poisson** (*bool*) – If set to True, use total signal for noise estimate, otherwise MULTIACCUM equation is used.
- **opt_diff** (*bool*) – Optimal reference differencing (scaling only on the inner regions)

Returns *tuple* – Three arrays in a tuple: the radius in arcsec, n-sigma contrast, and n-sigma magnitude sensitivity limit (vega mag).

gen_disk_image(*PA_offset=0*, *use_cmask=False*, ***kwargs*)
Create image of just disk.

Generate a (noiseless) convolved image of the disk at some PA offset. The PA offset value will rotate the image CCW. Image units of e-/sec.

Coordinate convention is for N up and E to left.

Parameters

- **PA_offset** (*float*) – Rotate entire scene by some position angle. Positive values are counter-clockwise from +Y direction. Corresponds to instrument aperture PA.
- **use_cmask** (*bool*) – Use the coronagraphic mask image to attenuate disk regions getting obscurred by a corongraphic mask feature

gen_planets_image(*PA_offset=0*, *quick_PSF=True*, *use_cmask=False*, ***kwargs*)
Create image of just planets.

Use info stored in self.planets to create a noiseless slope image of just the exoplanets (no star).

Coordinate convention is for +N up and +E to left.

Parameters

- **PA_offset** (*float*) – Rotate entire scene by some position angle. Positive values are counter-clockwise from +Y direction. Corresponds to instrument aperture PA.
- **quick_PSF** (*bool*) – Rather than generate a spectrum-weighted PSF, use the cached PSF scaled by the photon count through the bandpass. Resulting PSFs are less accurate, but generation is much faster. Default is True.
- **use_cmask** (*bool*) – Use the coronagraphic mask image to determine if any planet is getting obscurred by a corongraphic mask feature

gen_roll_image (*PA1=0*, *PA2=10*, *zfact=None*, *oversample=None*, *no_ref=False*, *opt_diff=True*, *fix_sat=False*, *ref_scale_all=False*, ***kwargs*)

Make roll-subtracted image.

Create a final roll-subtracted slope image based on current observation settings. Coordinate convention is for N up and E to left.

Procedure:

- Create Roll 1 and Roll 2 slope images (star+exoplanets+disk)
- Create Reference Star slope image
- Add noise to all images
- Scale ref image
- Subtract ref image from both rolls
- De-rotate Roll 1 and Roll 2 to common coordinates
- Average Roll 1 and Roll 2

Returns an HDUList of final image (N rotated upwards).

Parameters

- **PA1** (*float*) – Position angle of first roll position (clockwise, from East to West)
- **PA2** (*float, None*) – Position angle of second roll position. If set equal to PA1 (or to None), then only one roll will be performed. Otherwise, two rolls are performed, each using the specified MULTIACCUM settings (doubling the effective exposure time).
- **oversample** (*float*) – Set oversampling of final image.
- **no_ref** (*bool*) – Exclude reference observation. Subtraction is then Roll1-Roll2.
- **opt_diff** (*bool*) – Optimal reference differencing (scaling only on the inner regions)
- **fix_sat** (*bool*) – Calculate saturated regions and fix with median of nearby data.
- **ref_scale_all** (*bool*) – Normally we just use the science and reference PSFs to calculate scaling. However, if there is an unresolved companion or disk emission close to the star, then we won't get the correct scale factor for optimal reference subtraction. Instead, this option inludes disk and companions for calculating the reference scale factor.

Keyword Arguments

- **exclude_disk** (*bool*) – Do not include disk in final image (for radial contrast), but still add Poisson noise from disk.
- **exclude_planets** (*bool*) – Do not include planets in final image (for radial contrast), but still add Poisson noise from disk.
- **exclude_noise** (*bool*) – Don't add random Gaussian noise (detector+photon)

- **ideal_Poisson** (*bool*) – If set to True, use total signal for noise estimate, otherwise MULTIACCUM equation is used.
- **quick_PSF** (*bool*) – Rather than generate a spectrum-weighted PSF for planets, use the cached PSF scaled by the photon count through the bandpass. Resulting PSFs are slightly less accurate, but much faster. Default is True.
- **use_cmask** (*bool*) – Use the coronagraphic mask image to attenuate planets or disk obscurred by a corongraphic mask feature.
- **zfact** (*float*) – Zodiacal background factor (default=2.5)
- **ra** (*float*) – Right ascension in decimal degrees
- **dec** (*float*) – Declination in decimal degrees
- **thisday** (*int*) – Calendar day to use for background calculation. If not given, will use the average of visible calendar days.

planet_spec (*Av=0*, ***kwargs*)

Exoplanet spectrum.

Return the planet spectrum from Spiegel & Burrows (2012) normalized to distance of current target. Output is a `pysynphot.spectrum`.

Parameters **Av** (*float*) – Extinction magnitude (assumes Rv=4.0).

Keyword Arguments

- **atmo** (*str*) – A string consisting of one of four atmosphere types: ['hy1s', 'hy3s', 'cf1s', 'cf3s'].
- **mass** (*float*) – Number 1 to 15 Jupiter masses.
- **age** (*float*) – Age in millions of years (1-1000).
- **entropy** (*float*) – Initial entropy (8.0-13.0) in increments of 0.25
- **accr** (*bool*) – Include accretion? Default: False.
- **mmdot** (*float*) – From Zhu et al. (2015), the Mjup^2/yr value. If set to None then calculated from age and mass.
- **mdot** (*float*) – Or use mdot (Mjup/yr) instead of mmdot.
- **accr_rin** (*float*) – Inner radius of accretion disk (units of RJup; default: 2)
- **truncated** (*bool*) – Full disk or truncated (ie., MRI; default: False)?

saturation_levels (*ngroup=2*, *do_ref=False*, *image=None*, ***kwargs*)

Saturation levels

Create image showing level of saturation for each pixel. Saturation at different number of groups is possible with ngroup keyword. Returns an array the same shape as *det_info*.

Parameters

- **ngroup** (*int*) – How many group times to determine saturation level? If this number is higher than the total groups in ramp, then a warning is produced. The default is ngroup=2, A value of 0 corresponds to the so-called "zero-frame," which is the very first frame that is read-out and saved separately. This is the equivalent to ngroup=1 for RAPID and BRIGHT1 observations.
- **do_ref** (*bool*) – Get saturation levels for reference soure instead of science

- **image** (*ndarray*) – Rather than generating an image on the fly, pass a pre-computed slope image.

Keyword Arguments

- **quick_PSF** (*bool*) – Rather than generate a spectrum-weighted PSF for planets, use the cached PSF scaled by the photon count through the bandpass. Resulting PSFs are slightly less accurate, but much faster. Default is True.
- **use_cmask** (*bool*) – Use the coronagraphic mask image to attenuate planet or disk that is obscurred by a corongraphic mask feature.
- **zfact** (*float*) – Zodiacal background factor (default=2.5)
- **ra** (*float*) – Right ascension in decimal degrees
- **dec** (*float*) – Declination in decimal degrees
- **thisday** (*int*) – Calendar day to use for background calculation. If not given, will use the average of visible calendar days.

star_flux (*fluxunit='counts'*, *sp=None*)
Stellar flux.

Return the stellar flux in whatever units, such as vegamag, counts, or Jy.

Parameters

- **fluxunits** (*str*) – Desired output units, such as counts, vegamag, Jy, etc. Must be a Pysynphot supported unit string.
- **sp** (pysynphot.spectrum) – Normalized Pysynphot spectrum.

1.8.3 Simulations

nghxrg

class pynrc.simul.nghxrg.**HXRGNoise**(*naxis1=None*, *naxis2=None*, *naxis3=None*, *n_out=None*, *dt=None*, *nroh=None*, *nfoh=None*, *nfoh_pix=None*, *dark_file=None*, *bias_file=None*, *verbose=False*, *reverse_scan_direction=False*, *reference_pixel_border_width=None*, *wind_mode='FULL'*, *x0=0*, *y0=0*, *det_size=None*, *use_fftw=False*, *ncores=None*)

Bases: object

Simulate Teledyne HxRG + SIDECAR ASIC noise

HXRGNoise is a class for making realistic Teledyne HxRG system noise. The noise model includes correlated, uncorrelated, stationary, and non-stationary components. The default parameters make noise that resembles Channel 1 of JWST NIRSpec. NIRSpec uses H2RG detectors. They are read out using four video outputs at 1.e+5 pix/s/output.

Parameters

- **naxis1** (*int*) – X-dimension of the FITS cube.
- **naxis2** (*int*) – Y-dimension of the FITS cube.
- **naxis3** (*int*) – Z-dimension of the FITS cube (number of up-the-ramp samples).
- **n_out** (*int*) – Number of detector amplifiers/channels/outputs.

- **nroh** (*int*) – New row overhead in pixels. This allows for a short wait at the end of a row before starting the next one.
- **nfoh** (*int*) – New frame overhead in rows. This allows for a short wait at the end of a frame before starting the next one.
- **nfoh_pix(TBD)** (*int*) – New frame overhead in pixels. This allows for a short wait at the end of a frame before starting the next one. Generally a single pix offset for full frame and stripe for JWST ASIC systems.
- **dt** (*float*) – Pixel dwell time in seconds (10e-6 sec, for instance).
- **bias_file** (*str*) – Name of a FITS file that contains bias pattern, also used for PCA-zero.
- **dark_file** (*str*) – Name of a FITS file that contains dark current values per pixel.
- **verbose** (*bool*) – Enable this to provide status reporting.
- **wind_mode** (*str*) – 'FULL', 'STRIPE', or 'WINDOW'.
- **x0/y0** (*int*) – Pixel positions of subarray mode.
- **det_size** (*int*) – Pixel dimension of full detector (square).
- **reference_pixel_border_width** (*int*) – Width of reference pixel border around image area.
- **reverse_scan_direction** (*bool*) – Enable this to reverse the fast scanner readout directions. This capability was added to support Teledyne's programmable fast scan readout directions. The default setting of False corresponds to what HxRG detectors default to upon power up.
- **use_fftw** (*bool*) – If pyFFTW is installed, you can use this in place of np.fft.
- **ncores** (*int*) – Specify number of cores (threads, actually) to use for pyFFTW.

Methods Summary

`message`(message_text)	Used for status reporting
`mknoise`([o_file, gain, rd_noise, c_pink, ...])	Create FITS cube containing only noise
`pink_noise`(mode[, fmin])	Generate a vector of non-periodic pink noise.
`white_noise`([nstep])	Gaussian noise

Methods Documentation

message (*message_text*)
Used for status reporting

mknoise (*o_file=None*, *gain=None*, *rd_noise=None*, *c_pink=None*, *u_pink=None*, *acn=None*, *aco_a=None*, *aco_b=None*, *pca0_amp=None*, *reference_pixel_noise_ratio=None*, *ktc_noise=None*, *bias_off_avg=None*, *bias_off_sig=None*, *bias_amp=None*, *ch_off=None*, *ref_f2f_corr=None*, *ref_f2f_ucorr=None*, *ref_inst=None*, *out_ADU=True*)
Create FITS cube containing only noise

Parameters

- **o_file** (*str, None*) – Output filename. If None, then no output.
- **gain** (*float*) – Gain in e/ADU. Defaults to 1.0.
- **ktc_noise** (*float*) – kTC noise in electrons. Set this equal to sqrt(k*T*C_pixel)/q_e, where k is Boltzmann's constant, T is detector temperature, and C_pixel is pixel capacitance. For

an H2RG, the pixel capacitance is typically about 40 fF.

- **rd_noise** (*float*) – Standard deviation of read noise in electrons. Can be an array for individual amplifiers.
- **c_pink** (*float*) – Standard deviation of correlated pink noise in electrons.
- **u_pink** (*float*) – Standard deviation of uncorrelated pink noise in electrons. Can be an array for individual amplifiers.
- **acn** (*float*) – Standard deviation of alterating column noise in electrons
- **pca0_amp** (*float*) – Standard deviation of pca0 in electrons
- **reference_pixel_noise_ratio** (*float*) – Ratio of the standard deviation of the reference pixels to the science pixels. Reference pixels are usually a little lower noise.
- **bias_off_avg** (*float*) – On average, integrations start here in electrons. Set this so that all pixels are in range.
- **bias_off_sig** (*float*) – bias_off_avg has some variation. This is its std dev.
- **bias_amp** (*float*) – A multiplicative factor that we multiply bias_image by to simulate a bias pattern. This is completely independent from adding in "picture frame" noise. Set to 0.0 remove bias pattern. For NIRCam, default is 1.0.
- **ch_off** (*float*) – Offset of each channel relative to bias_off_avg. Can be an array for individual amplifiers.
- **ref_f2f_corr** (*float*) – Random frame-to-frame reference offsets due to PA reset, correlated between channels.
- **ref_f2f_ucorr** (*float*) – Random frame-to-frame reference offsets due to PA reset, per channel. Can be an array for individual amplifiers.
- **aco_a** (*float*) – Relative offsets of altnernating columns "a". Can be an array for individual amplifiers.
- **aco_b** (*float*) – Relative offsets of altnernating columns "b". Can be an array for individual amplifiers.
- **ref_inst** (*float*) – Reference instability relative to active pixels.
- **out_ADU** (*bool*) – Return as converted to ADU (True) or raw electrons?

Notes

Because of the noise correlations, there is no simple way to predict the noise of the simulated images. However, to a crude first approximation, these components add in quadrature.

The units in the above are mostly "electrons". This follows convention in the astronomical community. From a physics perspective, holes are actually the physical entity that is collected in Teledyne's p-on-n (p-type implants in n-type bulk) HgCdTe architecture.

pink_noise(*mode*, *fmin=None*)

Generate a vector of non-periodic pink noise.

Parameters

- **mode** (*str*) – Selected from 'pink', 'acn', or 'ref_inst'.
- **fmin** (*float, optional*) – Low-frequency cutoff. A value of 0 means no cut-off.

white_noise(*nstep=None*)
Gaussian noise

Generate white noise for an HxRG including all time steps (actual pixels and overheads).

Parameters **nstep** (*int*) – Length of vector returned

ngNRC

Summary

SCAnoise([det, scaid, params, caldir, ...])	NIRCam SCA noise generator
slope_to_ramp(det[, im_slope, out_ADU, ...])	Convert slope image to simulated ramp

Documentation

pynrc.simul.ngNRC.**SCAnoise**(*det=None*, *scaid=None*, *params=None*, *caldir=None*, *file_out=None*, *dark=True*, *bias=True*, *out_ADU=False*, *verbose=False*, *use_fftw=False*, *ncores=None*, ***kwargs*)
NIRCam SCA noise generator

Create a data cube consisting of realistic NIRCam detector noise.

This is essentially a wrapper for nghxrg.py that selects appropriate values for a specified SCA in order to reproduce realistic noise properties similiar to those measured during ISIM CV3.

Parameters

- **det** (*object*) – Option to specify already existing NIRCam detector object class Otherwise, use scaid and params
- **scaid** (*int*) – NIRCam SCA number (481, 482, ..., 490)
- **params** (*dictionary*) – A set of MULTIACCUM parameters such as:

```
>>> params = {'ngroup': 2, 'wind_mode': 'FULL',
>>>           'xpix': 2048, 'ypix': 2048, 'x0':0, 'y0':0}
```

  wind_mode can be FULL, STRIPE, or WINDOW
- **file_out** (*str*) – Folder name and destination to place optional FITS output. A timestamp will be appended to the end of the file name (and before .fits').
- **caldir** (*str*) – Directory location housing the super bias and super darks for each SCA.
- **dark** (*bool*) – Use super dark? If True, then reads in super dark slope image.
- **bias** (*bool*) – Use super bias? If True, then reads in super bias image.
- **out_ADU** (*bool*) – Noise values are calculated in terms of equivalent electrons. This gives the option of converting to ADU (True) or keeping in term of e- (False). ADU values are converted to 16-bit UINT. Keep in e- if applying to a ramp observation then convert combined data to ADU later.

Returns *Primary HDU with noise ramp in hud.data and header info in hdu.header.*

Examples

```
>>> import ngNRC
>>> params = {'ngroup': 108, 'wind_mode': 'FULL',
>>>           'xpix': 2048, 'ypix': 2048, 'x0':0, 'y0':0}
```

Output to a file:

```
>>> scaid = 481
>>> caldir = '/data/darks_sim/nghxrg/sca_images/'
>>> file_out = '/data/darks_sim/dark_sim_481.fits'
>>> hdu = ngNRC.SCAnoise(scaid, params, file_out=file_out, caldir=caldir,      >>>␣
↪   dark=True, bias=True, out_ADU=True, use_fftw=False, ncores=None,␣
↪verbose=False)
```

Don't save file, but keep hdu in e- for adding to simulated observation ramp:

```
>>> scaid = 481
>>> caldir = '/data/darks_sim/nghxrg/sca_images/'
>>> hdu = ngNRC.SCAnoise(scaid, params, file_out=None, caldir=caldir,      >>>
↪dark=True, bias=True, out_ADU=False, use_fftw=False, ncores=None, verbose=False)
```

pynrc.simul.ngNRC.**slope_to_ramp**(*det*, *im_slope=None*, *out_ADU=False*, *file_out=None*, *filter=None*, *pupil=None*, *obs_time=None*, *targ_name=None*, *DMS=True*, *dark=True*, *bias=True*, *det_noise=True*, *return_results=True*)

Convert slope image to simulated ramp

For a given detector operations class and slope image, create a ramp integration using Poisson noise and detector noise.

Currently, this image simulator does NOT take into account:

- QE variations across a pixel's surface
- Intrapixel Capacitance (IPC)
- Post-pixel Coupling (PPC) due to ADC "smearing"
- Pixel non-linearity
- Persistence/latent image
- Optical distortions
- Zodiacal background roll off for grism edges
- Telescope jitter
- Cosmic Rays

Parameters

- **det** (*object*) – Detector operations object.
- **im_slope** (*ndarray*) – Idealized slope image.
- **out_ADU** (*bool*) – If true, divide by gain and convert to 16-bit UINT.
- **file_out** (*str*) – Name (including directory) to save FITS file
- **filter** (*str*) – Name of filter element for header
- **pupil** (*str*) – Name of pupil element for header

- **obs_time** (*datetime*) – Specifies when the observation was considered to be executed. If not specified, then it will choose the current time. This must be a datetime object:

```
>>> datetime.datetime(2016, 5, 9, 11, 57, 5, 796686)
```

  This information is added to the header.

- **targ_name** (*str*) – Target name (optional)
- **DMS** (*bool*) – Package the data in the format used by DMS?
- **dark** (*bool*) – Include the dark current?
- **bias** (*bool*) – Include the bias frame?
- **det_noise** (*bool*) – Include detector noise components? If set to False, then only perform Poisson noise. Darks and biases are also excluded.

Summary

pynrc.simul.nghxrg.HXRGNoise([naxis1, ...])	Simulate Teledyne HxRG + SIDECAR ASIC noise
pynrc.simul.ngNRC.SCAnoise([det, scaid, ...])	NIRCam SCA noise generator
pynrc.simul.ngNRC.slope_to_ramp(det[, ...])	Convert slope image to simulated ramp

1.8.4 Data Reduction

Ref. Pixel Correction

Summary

NRC_refs(data, header[, DMS, altcol, do_all])	Reference pixel correction object
reffix_hxrg(cube[, nchans, in_place, fixcol])	Reference pixel correction function
reffix_amps(cube[, nchans, in_place, ...])	Correct amplifier offsets
ref_filter(cube[, nchans, in_place, ...])	Optimal Smoothing
calc_avg_amps(refs_all, data_shape[, ...])	Calculate amplifier averages
calc_avg_cols([refs_left, refs_right, ...])	Calculate average of column references
calc_col_smooth(refvals, data_shape[, ...])	Perform optimal smoothing of side ref pix
smooth_fft(data, delt[, first_deriv, ...])	Optimal smoothing algorithm

Class Documentation

class pynrc.reduce.ref_pixels.**NRC_refs**(*data*, *header*, *DMS=False*, *altcol=True*, *do_all=False*, ***kwargs*)

Bases: object

Reference pixel correction object

Object class for reference pixel correction of NIRCam data (single integration). Specify the data cube, header, and whether or not the header is in DMS format.

General usage of functions:

1. Create instance: `ref = NRC_refs(data, header)`
2. Determine reference offset values: `ref.calc_avg_amps()`. Stored at `ref.refs_amps_avg`.
3. Fix amplifier offsets: `ref.correct_amp_refs()`. Removes offsets that are stored at `ref.refs_amps_avg`.
4. Determine average of column references tracking 1/f noise: `ref.calc_avg_cols()`. Reference values offset for a mean value of 0. Averages are stored at `ref.refs_side_avg`.
5. Optimal smoothing of side reference values: `ref.calc_col_smooth()`. Stores smoothed version at `ref.refs_side_smth`.
6. Remove approximation of 1/f noise: `ref.correct_col_refs()`.

Parameters

- **data** (*ndarray*) – Input datacube. Can be two or three dimensions (nz,ny,nx).
- **header** (*obj*) – NIRCam Header associated with data.
- **DMS** (*bool*) – Is the header in DMS format?
- **altcol** (*bool*) – Calculate separate reference values for even/odd columns? Default=True.
- **do_all** (*bool*) – Perform the default pixel correction procedures.

Attributes Summary

multiaccum	A *multiaccum* object
multiaccum_times	Exposure timings in dictionary
refs_bot	Return raw bottom reference values
refs_top	Return raw top reference values
refs_right	Return raw right reference values
refs_left	Return raw left reference values

Methods Summary

`calc_avg_amps`([top_ref, bot_ref])	Calculate amplifier averages
`calc_avg_cols`([left_ref, right_ref, avg_type])	Calculate average of column references
`calc_col_smooth`([perint, edge_wrap, savgol])	Optimal smoothing of side reference pixels
`correct_amp_refs`([supermean])	Correct amplifier offsets
`correct_col_refs`()	Remove 1/f noise from data

Methods Documentation

calc_avg_amps (*top_ref=True*, *bot_ref=True*)
Calculate amplifier averages

Save the average reference value for each amplifier in each frame. Each array has a size of (namp, ngroup). Average values are saved at `self.refs_amps_avg`.

Parameters

- **top_ref** (*bool*) – Include top reference rows when correcting channel offsets.
- **bot_ref** (*bool*) – Include bottom reference rows when correcting channel offsets.

calc_avg_cols (*left_ref=True*, *right_ref=True*, *avg_type='frame'*, ***kwargs*)
Calculate average of column references

Create a copy of the left and right reference pixels, removing the average value of the reference pixels on an int, frame, or pixel basis. Do this after correcting the amplifier offsets with `correct_amp_refs()`. Averages are stored in `self.refs_side_avg`.

Parameters

- **left_ref** (*bool*) – Include left reference cols when correcting 1/f noise.
- **right_ref** (*bool*) – Include right reference cols when correcting 1/f noise.
- **avg_type** (*str*) – Type of ref col averaging to perform. Allowed values are 'pixel', 'frame', or 'int'.
- **mean_func** (*func*) – Function to use to calculate averages of reference columns

calc_col_smooth (*perint=False*, *edge_wrap=False*, *savgol=False*, ***kwargs*)
Optimal smoothing of side reference pixels

Geneated smoothed version of column reference values. Uses *calc_avg_cols()* to determine approx 1/f noise in data and store in `self.refs_side_smth`.

Parameters

- **perint** (*bool*) – Smooth side reference pixel per int, otherwise per frame.
- **edge_wrap** (*bool*) – Add a partial frames to the beginning and end of each averaged time series pixels in order to get rid of edge effects.

correct_amp_refs (*supermean=False*)
Correct amplifier offsets

Use values in `self.refs_amps_avg` to correct amplifier offsets.

Parameters **supermean** (*bool*) – Add back the overall mean of the reference pixels.

correct_col_refs()
Remove 1/f noise from data

Correct 1/f noise using the approximation stored in `self.refs_side_smth`.

Functions Documentation

pynrc.reduce.ref_pixels.**reffix_hxrg**(*cube*, *nchans=4*, *in_place=True*, *fixcol=False*, ***kwargs*)

Reference pixel correction function

This function performs a reference pixel correction on HAWAII-[1,2,4]RG detector data read out using N outputs. Top and bottom reference pixels are used first to remove channel offsets.

Parameters

- **cube** (*ndarray*) – Input datacube. Can be two or three dimensions (nz,ny,nx).
- **in_place** (*bool*) – Perform calculations in place. Input array is overwritten.
- **nchans** (*int*) – Number of output amplifier channels in the detector. Default=4.
- **fixcol** (*bool*) – Perform reference column corrections?

Keyword Arguments

- **altcol** (*bool*) – Calculate separate reference values for even/odd columns.
- **supermean** (*bool*) – Add back the overall mean of the reference pixels.
- **top_ref** (*bool*) – Include top reference rows when correcting channel offsets.
- **bot_ref** (*bool*) – Include bottom reference rows when correcting channel offsets.
- **ntop** (*int*) – Specify the number of top reference rows.
- **nbot** (*int*) – Specify the number of bottom reference rows.
- **left_ref** (*bool*) – Include left reference cols when correcting 1/f noise.
- **right_ref** (*bool*) – Include right reference cols when correcting 1/f noise.
- **nleft** (*int*) – Specify the number of left reference columns.
- **nright** (*int*) – Specify the number of right reference columns.
- **perint** (*bool*) – Smooth side reference pixel per integration, otherwise do frame-by-frame.
- **avg_type** (*str*) – Type of side column averaging to perform to determine ref pixel drift. Allowed values are ‘pixel’, ‘frame’, or ‘int’:
  - ’int’ : Subtract the avg value of all side ref pixels in ramp.
  - ’frame’ : For each frame, get avg of side ref pixels and subtract framewise.
  - ’pixel’ : For each ref pixel, subtract its avg value from all frames.
- **savgol** (*bool*) – Using Savitsky-Golay filter method rather than FFT.
- **winsize** (*int*) – Size of the window filter.
- **order** (*int*) – Order of the polynomial used to fit the samples.

pynrc.reduce.ref_pixels.**reffix_amps**(*cube*, *nchans=4*, *in_place=True*, *altcol=True*, *supermean=False*, *top_ref=True*, *bot_ref=True*, *ntop=4*, *nbot=4*, ***kwargs*)

Correct amplifier offsets

Matches all amplifier outputs of the detector to a common level.

This routine subtracts the average of the top and bottom reference rows for each amplifier and frame individually.

By default, reference pixel corrections are performed in place since it's faster and consumes less memory.

Parameters

- **cube** (*ndarray*) – Input datacube. Can be two or three dimensions (nz,ny,nx).
- **nchans** (*int*) – Number of output amplifier channels in the detector. Default=4.
- **altcol** (*bool*) – Calculate separate reference values for even/odd columns.
- **supermean** (*bool*) – Add back the overall mean of the reference pixels.
- **in_place** (*bool*) – Perform calculations in place. Input array is overwritten.
- **top_ref** (*bool*) – Include top reference rows when correcting channel offsets.
- **bot_ref** (*bool*) – Include bottom reference rows when correcting channel offsets.
- **ntop** (*int*) – Specify the number of top reference rows.
- **nbot** (*int*) – Specify the number of bottom reference rows.

pynrc.reduce.ref_pixels.**ref_filter**(*cube*, *nchans=4*, *in_place=True*, *avg_type='frame'*, *perint=False*, *edge_wrap=False*, *left_ref=True*, *right_ref=True*, *nleft=4*, *nright=4*, ***kwargs*)

Optimal Smoothing

Performs an optimal filtering of the vertical reference pixel to reduce 1/f noise (horizontal stripes).

Adapted from M. Robberto IDL code: http://www.stsci.edu/~robberto/Main/Software/IDL4pipeline/

Parameters

- **cube** (*ndarray*) – Input datacube. Can be two or three dimensions (nz,ny,nx).
- **nchans** (*int*) – Number of output amplifier channels in the detector. Default=4.
- **in_place** (*bool*) – Perform calculations in place. Input array is overwritten.
- **perint** (*bool*) – Smooth side reference pixel per integration, otherwise do frame-by-frame.
- **avg_type** (*str*) – Type of ref col averaging to perform. Allowed values are 'pixel', 'frame', or 'int'.
- **left_ref** (*bool*) – Include left reference cols when correcting 1/f noise.
- **right_ref** (*bool*) – Include right reference cols when correcting 1/f noise.
- **nleft** (*int*) – Specify the number of left reference columns.
- **nright** (*int*) – Specify the number of right reference columns.

Keyword Arguments

- **savgol** (*bool*) – Using Savitsky-Golay filter method rather than FFT.
- **winsize** (*int*) – Size of the window filter.
- **order** (*int*) – Order of the polynomial used to fit the samples.
- **mean_func** (*func*) – Function to use to calculate averages of reference columns.

pynrc.reduce.ref_pixels.**calc_avg_amps**(*refs_all*, *data_shape*, *nchans=4*, *altcol=True*)
Calculate amplifier averages

Save the average reference value for each amplifier in each frame. Assume by default that alternating columns are offset from each other, so we save two arrays: self.refs_amps_avg1 and self.refs_amps_avg2. Each array has a size of (namp, ngroup).

Parameters

- **refs_all** (*ndarray*) – The top and/or bottom references pixels order in a shape (nz, nref_rows, nx)
- **data_shape** (*tuple*) – Shape of the data array: (nz, ny, nx).
- **nchans** (*int*) – Number of amplifier output channels.
- **altcol** (*bool*) – Calculate separate reference values for even/odd columns? Default=True.

pynrc.reduce.ref_pixels.**calc_avg_cols**(*refs_left=None*, *refs_right=None*, *avg_type='frame'*, *mean_func=<function median>*, ***kwargs*)
Calculate average of column references

Determine the average values for the column references, which is subsequently used to estimate the 1/f noise contribution.

Parameters

- **refs_left** (*ndarray*) – Left reference columns.
- **refs_right** (*ndarray*) – Right reference columns.
- **avg_type** (*str*) – Type of ref column averaging to perform to determine ref pixel variation. Allowed values are 'pixel', 'frame', or 'int'. 'pixel' : For each ref pixel, subtract its avg value from all frames. 'frame' : For each frame, get avg ref pixel values and subtract framewise. 'int' : Calculate avg of all ref pixels within the ramp and subtract.
- **mean_func** (*func*) – Function to use to calculate averages of reference columns

pynrc.reduce.ref_pixels.**calc_col_smooth**(*refvals*, *data_shape*, *perint=False*, *edge_wrap=False*, *delt=0.000524*, *savgol=False*, *winsize=31*, *order=3*, ***kwargs*)
Perform optimal smoothing of side ref pix

Geneated smoothed version of column reference values. Smooths values from calc_avg_cols() via FFT.

Parameters

- **refvals** (*ndarray*) – Averaged column reference pixels
- **data_shape** (*tuple*) – Shape of original data (nz,ny,nx)

Keyword Arguments

- **perint** (*bool*) – Smooth side reference pixel per int, otherwise per frame.
- **edge_wrap** (*bool*) – Add a partial frames to the beginning and end of each averaged time series pixels in order to get rid of edge effects.
- **delt** (*float*) – Time between reference pixel samples.
- **savgol** (*bool*) – Using Savitsky-Golay filter method rather than FFT.
- **winsize** (*int*) – Size of the window filter.
- **order** (*int*) – Order of the polynomial used to fit the samples.

pynrc.reduce.ref_pixels.**smooth_fft**(*data*, *delt*, *first_deriv=False*, *second_deriv=False*)
Optimal smoothing algorithm

Smoothing algorithm to perform optimal filtering of the vertical reference pixel to reduce 1/f noise (horizontal stripes), based on the Kosarev & Pantos algorithm. This assumes that the data to be filtered/smoothed has been sampled evenly.

If first_deriv is set, then returns two results if second_deriv is set, then returns three results.

Adapted from M. Robberto IDL code: http://www.stsci.edu/~robberto/Main/Software/IDL4pipeline/

Parameters

- **data** (*ndarray*) – Signal to be filtered.
- **delt** (*float*) – Delta time between samples.
- **first_deriv** (*bool*) – Return the first derivative.
- **second_deriv** (*bool*) – Return the second derivative (along with first).

Summary

pynrc.reduce.ref_pixels.NRC_refs(data, header)	Reference pixel correction object
pynrc.reduce.ref_pixels.reffix_hxrg(cube[, ...])	Reference pixel correction function
pynrc.reduce.ref_pixels.reffix_amps(cube[, ...])	Correct amplifier offsets
pynrc.reduce.ref_pixels.ref_filter(cube[, ...])	Optimal Smoothing
pynrc.reduce.ref_pixels.calc_avg_amps(...[, ...])	Calculate amplifier averages
pynrc.reduce.ref_pixels.calc_avg_cols([...])	Calculate average of column references
pynrc.reduce.ref_pixels.calc_col_smooth(...)	Perform optimal smoothing of side ref pix
pynrc.reduce.ref_pixels.smooth_fft(data, delt)	Optimal smoothing algorithm

1.8.5 Utilities

Math Tools

Coordinates Summary

dist_image(image[, pixscale, center, ...])	Pixel distances
xy_to_rtheta(x, y)	Convert (x,y) to (r,theta)
rtheta_to_xy(r, theta)	Convert (r,theta) to (x,y)
xy_rot(x, y, ang)	Rotate (x,y) positions to new coords
Tel2Sci_info(channel, coords[, pupil, ...])	Telescope coords converted to Science coords
det_to_V2V3(image, detid)	Same as *det_to_sci*
V2V3_to_det(image, detid)	Same as *sci_to_det*

continues on next page

Table 17 – continued from previous page

plotAxes(ax[, position, label1, label2, ...])	Compass arrows

Image Manipulation Summary

hist_indices(values[, bins, return_more])	Histogram indices
binned_statistic(x, values[, func, bins])	Binned statistic
frebin(image[, dimensions, scale, total])	Fractional rebin
fshift(image[, delx, dely, pad, cval])	Fractional image shift
fourier_imshift(image, xshift, yshift[, ...])	Fourier shift image
shift_subtract(params, reference, target[, ...])	Shift and subtract image
align_LSQ(reference, target[, mask, pad, ...])	Find best shift value
scale_ref_image(im1, im2[, mask, ...])	Reference image scaling
optimal_difference(im_sci, im_ref, scale[, ...])	Optimize subtraction of ref PSF
pad_or_cut_to_size(array, new_shape[, fill_val])	Resize an array to a new shape by either padding with zeros or trimming off rows and/or columns.
fix_nans_with_med(im[, niter_max, verbose])	Iteratively fix NaNs with surrounding Real data
rotate_offset(data, angle[, cen, cval, ...])	Rotate and offset an array.

Polynomial Fitting Summary

jl_poly_fit(x, yvals[, deg, QR, robust_fit, ...])	Fast polynomial fitting
jl_poly(xvals, coeff[, dim_reorder, ...])	Evaluate polynomial

Robust Summary

biweightMean(inputData[, axis, dtype, iterMax])	Biweight Mean
checkfit(inputData, inputFit, epsilon, delta)	Determine the quality of a fit and biweights.
linefit(inputX, inputY[, iterMax, Bisector, ...])	Outlier resistance two-variable linear regression function.
mean(inputData[, Cut, axis, dtype, ...])	Robust Mean
medabsdev(data[, axis, keepdims, nan])	Median Absolute Deviation
mode(inputData[, axis, dtype])	Robust estimator of the mode of a data set using the half-sample mode.
polyfit(inputX, inputY, order[, iterMax])	Outlier resistance two-variable polynomial function fitter.
std(inputData[, Zero, axis, dtype, ...])	Robust Sigma

pynrc.maths.coords

pynrc.maths.coords.**dist_image**(*image*, *pixscale=None*, *center=None*, *return_theta=False*)
Pixel distances

Returns radial distance in units of pixels, unless pixscale is specified. Use the center keyword to specify the position (in pixels) to measure from. If not set, then the center of the image is used.

return_theta will also return the angular position of each pixel relative to the specified center

Parameters

- **image** (*ndarray*) – Input image to find pixel distances (and theta).
- **pixscale** (*int, None*) – Pixel scale (such as arcsec/pixel or AU/pixel) that dictates the units of the output distances. If None, then values are in units of pixels.
- **center** (*tuple*) – Location (x,y) in the array calculate distance. If set to None, then the default is the array center pixel.
- **return_theta** (*bool*) – Also return the angular positions as a 2nd element.

pynrc.maths.coords.**xy_to_rtheta**(*x*, *y*)
Convert (x,y) to (r,theta)

Input (x,y) coordinates and return polar cooridnates that use the WebbPSF convention (theta is CCW of +Y)

Input can either be a single value or numpy array.

Parameters

- **x** (*float or array*) – X location values
- **y** (*float or array*) – Y location values

pynrc.maths.coords.**rtheta_to_xy**(*r*, *theta*)
Convert (r,theta) to (x,y)

Input polar cooridnates (WebbPSF convention) and return Carteesian coords in the imaging coordinate system (as opposed to RA/DEC)

Input can either be a single value or numpy array.

Parameters

- **r** (*float or array*) – Radial offset from the center in pixels
- **theta** (*float or array*) – Position angle for offset in degrees CCW (+Y).

pynrc.maths.coords.**xy_rot**(*x*, *y*, *ang*)
Rotate (x,y) positions to new coords

Rotate (x,y) values by some angle. Positive ang values rotate counter-clockwise.

Parameters

- **x** (*float or array*) – X location values
- **y** (*float or array*) – Y location values
- **ang** (*float or array*) – Rotation angle in degrees CCW

pynrc.maths.coords.**Tel2Sci_info**(*channel*, *coords*, *pupil=None*, *output='sci'*, *return_apname=False*, ***kwargs*)
Telescope coords converted to Science coords

Returns the detector name and position associated with input coordinates. This is alway relative to a full frame detector aperture. The detector that is chosen is the one whose center is closest to the input coords.

Parameters

- **channel** (*str*) – 'SW' or 'LW'
- **coords** (*tuple*) – Telescope coordinates (V2,V3) in arcsec.
- **output** (*str*) –

  Type of desired output coordinates.

  – det: pixels, in raw detector read out axes orientation

  – sci: pixels, in conventional DMS axes orientation

  – idl: arcsecs relative to aperture reference location.

  – tel: arcsecs V2,V3

pynrc.maths.coords.**det_to_V2V3**(*image*, *detid*)
Same as *det_to_sci*

pynrc.maths.coords.**V2V3_to_det**(*image*, *detid*)
Same as *sci_to_det*

pynrc.maths.coords.**plotAxes**(*ax*, *position=0.9, 0.1*, *label1='V2'*, *label2='V3'*, *dir1=[- 1, 0]*, *dir2=[0, 1]*, *angle=0*, *alength=0.12*, *width=2*, *headwidth=8*, *color='w'*)
Compass arrows

Show V2/V3 coordinate axis on a plot. By default, this function will plot the compass arrows in the lower right position in sky-right coordinates (ie., North/V3 up, and East/V2 to the left).

Parameters

- **ax** (*axis*) – matplotlib axis to plot coordiante arrows.
- **position** (*tuple*) – XY-location of joined arrows as a fraction (0.0-1.0).
- **label1** (*str*) – Label string for horizontal axis (ie., 'E' or 'V2').
- **label2** (*str*) – Label string for vertical axis (ie, 'N' or 'V3').
- **dir1** (*array like*) – XY-direction values to point "horizontal" arrow.
- **dir2** (*array like*) – XY-direction values to point "vertical" arrow.
- **angle** (*float*) – Rotate coordinate axis by some angle. Positive values rotate counter-clockwise.
- **alength** (*float*) – Length of arrow vectors as fraction of plot axis.
- **width** (*float*) – Width of the arrow in points.
- **headwidth** (*float*) – Width of the base of the arrow head in points.
- **color** (*color*) – Self-explanatory.

pynrc.maths.image_manip

pynrc.maths.image_manip.**hist_indices**(*values*, *bins=10*, *return_more=False*)
Histogram indices

This function bins an input of values and returns the indices for each bin. This is similar to the reverse indices functionality of the IDL histogram routine. It's also much faster than doing a for loop and creating masks/indices at each iteration, because we utilize a sparse matrix constructor.

Returns a list of indices grouped together according to the bin. Only works for evenly spaced bins.

Parameters

- **values** (*ndarray*) – Input numpy array. Should be a single dimension.
- **bins** (*int or ndarray*) – If bins is an int, it defines the number of equal-width bins in the given range (10, by default). If bins is a sequence, it defines the bin edges, including the rightmost edge. In the latter case, the bins must encompass all values.
- **return_more** (*bool*) – Option to also return the values organized by bin and the value of the centers (igroups, vgroups, center_vals).

Example

Find the standard deviation at each radius of an image

```
>>> rho = dist_image(image)
>>> binsize = 1
>>> bins = np.arange(rho.min(), rho.max() + binsize, binsize)
>>> igroups, vgroups, center_vals = hist_indices(rho, bins, True)
>>> # Get the standard deviation of each bin in image
>>> std = binned_statistic(igroups, image, func=np.std)
```

pynrc.maths.image_manip.**binned_statistic**(*x*, *values*, *func=<function mean>*, *bins=10*)
Binned statistic

Compute a binned statistic for a set of data. Drop-in replacement for scipy.stats.binned_statistic.

Parameters

- **x** (*ndarray*) – A sequence of values to be binned. Or a list of binned indices from hist_indices().
- **values** (*ndarray*) – The values on which the statistic will be computed.
- **func** (*func*) – The function to use for calculating the statistic.
- **bins** (*int or ndarray*) – If bins is an int, it defines the number of equal-width bins in the given range (10, by default). If bins is a sequence, it defines the bin edges, including the rightmost edge. This doesn't do anything if x is a list of indices.

Example

Find the standard deviation at each radius of an image

```
>>> rho = dist_image(image)
>>> binsize = 1
>>> radial_bins = np.arange(rho.min(), rho.max() + binsize, binsize)
>>> radial_stds = binned_statistic(rho, image, func=np.std, bins=radial_bins)
```

pynrc.maths.image_manip.**frebin**(*image*, *dimensions=None*, *scale=None*, *total=True*)
Fractional rebin

Python port from the IDL frebin.pro Shrink or expand the size of a 1D or 2D array by an arbitary amount using bilinear interpolation. Conserves flux by ensuring that each input pixel is equally represented in the output array.

Parameters

- **image** (*ndarray*) – Input image, 1-d or 2-d ndarray.
- **dimensions** (*tuple or None*) – Desired size of output array (take priority over scale).
- **scale** (*tuple or None*) – Factor to scale output array size. A scale of 2 will increase the number of pixels by 2 (ie., finer pixel scale).
- **total** (*bool*) – Conserves the surface flux. If True, the output pixels will be the sum of pixels within the appropriate box of the input image. Otherwise, they will be the average.

Returns *ndarray* – The binned ndarray

pynrc.maths.image_manip.**fshift**(*image*, *delx=0*, *dely=0*, *pad=False*, *cval=0.0*)
Fractional image shift

Ported from IDL function fshift.pro. Routine to shift an image by non-integer values.

Parameters

- **image** (*ndarray*) – 1D or 2D array to be shifted
- **delx** (*float*) – shift in x (same direction as IDL SHIFT function)
- **dely** (*float*) – shift in y
- **pad** (*bool*) – Should we pad the array before shifting, then truncate? Otherwise, the image is wrapped.
- **cval** (*sequence or float, optional*) – The values to set the padded values for each axis. Default is 0. ((before_1, after_1), ... (before_N, after_N)) unique pad constants for each axis. ((before, after),) yields same before and after constants for each axis. (constant,) or int is a shortcut for before = after = constant for all axes.

Returns *ndarray* – Shifted image

pynrc.maths.image_manip.**fourier_imshift**(*image*, *xshift*, *yshift*, *pad=False*, *cval=0.0*)
Fourier shift image

Shift an image by use of Fourier shift theorem

Parameters

- **image** (*nd array*) – N x K image
- **xshift** (*float*) – Number of pixels to shift image in the x direction
- **yshift** (*float*) – Number of pixels to shift image in the y direction

- **pad** (*bool*) – Should we pad the array before shifting, then truncate? Otherwise, the image is wrapped.
- **cval** (*sequence or float, optional*) – The values to set the padded values for each axis. Default is 0. ((before_1, after_1), ... (before_N, after_N)) unique pad constants for each axis. ((before, after),) yields same before and after constants for each axis. (constant,) or int is a shortcut for before = after = constant for all axes.

Returns *ndarray* – Shifted image

pynrc.maths.image_manip.**shift_subtract**(*params*, *reference*, *target*, *mask=None*, *pad=False*, *shift_function=<function fshift>*)

Shift and subtract image

Use Fourier Shift theorem for subpixel shifts for input into least-square optimizer.

Parameters

- **params** (*tuple*) – xshift, yshift, beta
- **reference** (*ndarray*) – See align_fourierLSQ
- **target** (*ndarray*) – See align_fourierLSQ
- **mask** (*ndarray, optional*) – See align_fourierLSQ
- **pad** (*bool*) – Should we pad the array before shifting, then truncate? Otherwise, the image is wrapped.
- **shift_function** (*func*) – which function to use for sub-pixel shifting

Returns *ndarray* – 1D array of target-reference residual after applying shift and intensity fraction.

pynrc.maths.image_manip.**align_LSQ**(*reference*, *target*, *mask=None*, *pad=False*, *shift_function=<function fshift>*)

Find best shift value

LSQ optimization with option of shift alignment algorithm

Parameters

- **reference** (*ndarray*) – N x K image to be aligned to
- **target** (*ndarray*) – N x K image to align to reference
- **mask** (*ndarray, optional*) – N x K image indicating pixels to ignore when performing the minimization. The masks acts as a weighting function in performing the fit.
- **pad** (*bool*) – Should we pad the array before shifting, then truncate? Otherwise, the image is wrapped.
- **shift_function** (*func*) – which function to use for sub-pixel shifting. Options are fourier_imshift or fshift. fshift tends to be 3-5 times faster for similar results.

Returns *list* – (x, y, beta) values from LSQ optimization, where (x, y) are the misalignment of target from reference and beta is the fraction by which the target intensity must be reduced to match the intensity of the reference.

pynrc.maths.image_manip.**scale_ref_image**(*im1*, *im2*, *mask=None*, *smooth_imgs=False*, *return_shift_values=False*)

Reference image scaling

Find value to scale a reference image by minimizing residuals. Assumes everything is already aligned if return_shift_values=False.

Or simply turn on return_shift_values to return (dx,dy,scl). Then fshift(im2,dx,dy) to shift the reference image.

Parameters

- **im1** (*ndarray*) – Science star observation.
- **im2** (*ndarray*) – Reference star observation.
- **mask** (*bool array or None*) – Use this mask to exclude pixels for performing standard deviation. Boolean mask where True is included and False is excluded.
- **smooth_imgs** (*bool*) – Smooth the images with nearest neighbors to remove bad pixels?
- **return_shift_values** (*bool*) – Option to return x and y shift values

pynrc.maths.image_manip.**optimal_difference**(*im_sci*, *im_ref*, *scale*, *binsize=1*, *center=None*, *mask_good=None*, *sub_mean=True*, *std_func=<function std>*)

Optimize subtraction of ref PSF

Scale factors from scale_ref_image work great for subtracting a reference PSF from a science image where there are plenty of photons, but perform poorly in the noise-limited regime. If we simply perform a difference by scaling the reference image, then we also amplify the noise. In the background, it's better to simply subtract the unscaled reference pixels. This routine finds the radial cut-off of the dominant noise source.

Parameters

- **im_sci** (*ndarray*) – Science star observation.
- **im_ref** (*ndarray*) – Reference star observation.
- **scale** (*float*) – Scale factor from *scale_ref_image()*
- **binsize** (*int*) – Radial binsize (in pixels) to perform calculations
- **center** (*tuple or None*) – Location (x,y) to calculate radial distances. Default is center of image.
- **mask_good** (*bool array*) – Only perform operations on pixels where mask_good=True.
- **sub_mean** (*bool*) – Subtract mean (median, actually) of pixels in each radial bin? Basically a background subtraction.
- **std_func** (*func*) – What function do we want to use for calculating the standard deviation in each radial bin? After comparing the standard deviation between the two scaled differences in each radial bin, we only keep the better of the two.

pynrc.maths.image_manip.**pad_or_cut_to_size**(*array*, *new_shape*, *fill_val=0.0*)

Resize an array to a new shape by either padding with zeros or trimming off rows and/or columns. The ouput shape can be of any arbitrary amount.

Parameters

- **array** (*ndarray*) – A 1D or 2D array representing some image
- **padded_shape** (*tuple of 2 elements*) – Desired size for the output array. For 2D case, if a single value, then will create a 2-element tuple of the same value.
- **fill_val** (*scalar, optional*) – Value to pad borders. Default is 0.0

Returns **output** (*ndarray*) – An array of size new_shape that preserves the central information of the input array.

pynrc.maths.image_manip.**fix_nans_with_med**(*im*, *niter_max=5*, *verbose=False*, ***kwargs*)

Iteratively fix NaNs with surrounding Real data

pynrc.maths.image_manip.**rotate_offset**(*data*, *angle*, *cen=None*, *cval=0.0*, *order=1*, *reshape=True*, *recenter=True*, ***kwargs*)

Rotate and offset an array.

Same as *rotate* in *scipy.ndimage.interpolation* except that it rotates around a center point given by *cen* keyword. The array is rotated in the plane defined by the two axes given by the *axes* parameter using spline interpolation of the requested order.

Parameters

- **data** (*ndarray*) – The input array.
- **angle** (*float*) – The rotation angle in degrees (rotates in CW direction).
- **cen** (*tuple*) – Center location around which to rotate image. Values are expected to be *(xcen, ycen)*.
- **recenter** (*bool*) – Do we want to reposition so that *cen* is the image center?

Keyword Arguments

- **axes** (*tuple of 2 ints, optional*) – The two axes that define the plane of rotation. Default is the first two axes.
- **reshape** (*bool, optional*) – If *reshape* is True, the output shape is adapted so that the input array is contained completely in the output. Default is True.
- **order** (*int, optional*) – The order of the spline interpolation, default is 1. The order has to be in the range 0-5.
- **mode** (*str, optional*) – Points outside the boundaries of the input are filled according to the given mode ('constant', 'nearest', 'reflect', 'mirror' or 'wrap'). Default is 'constant'.
- **cval** (*scalar, optional*) – Value used for points outside the boundaries of the input if `mode='constant'`. Default is 0.0
- **prefilter** (*bool, optional*) – The parameter prefilter determines if the input is prefiltered with *spline_filter* before interpolation (necessary for spline interpolation of order > 1). If False, it is assumed that the input is already filtered. Default is True.

Returns **rotate** (*ndarray or None*) – The rotated data.

---

pynrc.maths.fast_poly

pynrc.maths.fast_poly.**jl_poly_fit**(*x*, *yvals*, *deg=1*, *QR=True*, *robust_fit=False*, *niter=25*, *use_legendre=False*, *lxmap=None*, ***kwargs*)

Fast polynomial fitting

Fit a polynomial to a function using linear least-squares. This function is particularly useful if you have a data cube and want to simultaneously fit a slope to all pixels in order to produce a slope image.

Gives the option of performing QR decomposition, which provides a considerable speed-up compared to simply using *np.linalg.lstsq()*. In addition to being fast, it has better numerical stability than linear regressions that involve matrix inversions (ie., *dot(x.T,x)*).

Returns the coefficients of the fit for each pixel.

Parameters

- **x** (*ndarray*) – X-values of the data array (1D).

- **yvals** (*ndarray*) – Y-values (1D, 2D, or 3D) where the first dimension must have equal length of x. For instance, if x is a time series of a data cube with size NZ, then the data cube must follow the Python convention (NZ,NY,NZ).

Keyword Arguments

- **deg** (*int*) – Degree of polynomial to fit to the data.
- **QR** (*bool*) – Perform QR decomposition? Default=True.
- **robust_fit** (*bool*) – Perform robust fitting, iteratively kicking out outliers until convergence.
- **niter** (*int*) – Maximum number of iterations for robust fitting. If convergence is attained first, iterations will stop.
- **use_legendre** (*bool*) – Fit with Legendre polynomials, an orthonormal basis set.
- **lxmap** (*ndarray or None*) – Legendre polynomials are normaly mapped to xvals of [-1,+1]. *lxmap* gives the option to supply the values for xval that should get mapped to [-1,+1]. If set to None, then assumes [xvals.min(),xvals.max()].

Example

Fit all pixels in a data cube to get slope image in terms of ADU/sec

```
>>> nz, ny, nx = cube.shape
>>> tvals = (np.arange(nz) + 1) * 10.737
>>> coeff = jl_poly_fit(tvals, cube, deg=1)
>>> bias = coeff[0]  # Bias image (y-intercept)
>>> slope = coeff[1] # Slope image (DN/sec)
```

pynrc.maths.fast_poly.**jl_poly**(*xvals*, *coeff*, *dim_reorder=False*, *use_legendre=False*, *lxmap=None*, ***kwargs*)

Evaluate polynomial

Replacement for *np.polynomial.polynomial.polyval(wgood, coeff)* to evaluate y-values given a set of xvals and coefficients. Uses matrix multiplication, which is much faster. Beware, the default output array shapes organization may differ from the polyval routine for 2D and 3D results.

Parameters

- **xvals** (*ndarray*) – 1D array (time, for instance)
- **coeff** (*ndarray*) – 1D, 2D, or 3D array of coefficients from a polynomial fit. The first dimension should have a number of elements equal to the polynomial degree + 1. Order such that lower degrees are first, and higher degrees are last.

Keyword Arguments

- **dim_reorder** (*bool*) – If true, then result to be ordered (nx,ny,nz), otherwise we use the Python preferred ordering (nz,ny,nx)
- **use_legendre** (*bool*) – Fit with Legendre polynomial, an orthonormal basis set.
- **lxmap** (*ndarray or None*) – Legendre polynomials are normaly mapped to xvals of [-1,+1]. *lxmap* gives the option to supply the values for xval that should get mapped to [-1,+1]. If set to None, then assumes [xvals.min(),xvals.max()].

Returns *float array* – An array of values where each xval has been evaluated at each set of supplied coefficients. The output shape has the first dimension equal to the number of xvals, and the final

dimensions correspond to coeff's latter dimensions. The result is flattened if there is either only one xval or one set of coeff (or both).

---

pynrc.maths.robust

Small collection of robust statistical estimators based on functions from Henry Freudenriech (Hughes STX) statistics library (called ROBLIB) that have been incorporated into the AstroIDL User's Library.

pynrc.maths.robust.**biweightMean**(*inputData*, *axis=None*, *dtype=None*, *iterMax=25*)
Biweight Mean

Calculate the mean of a data set using bisquare weighting.

Based on the biweight_mean routine from the AstroIDL User's Library.

Changed in version 1.0.3: Added the 'axis' and 'dtype' keywords to make this function more compatible with np.mean()

pynrc.maths.robust.**checkfit**(*inputData*, *inputFit*, *epsilon*, *delta*, *BisquareLimit=6.0*)
Determine the quality of a fit and biweights. Returns a tuple with elements:

0. Robust standard deviation analog
1. Fractional median absolute deviation of the residuals
2. Number of input points given non-zero weight in the calculation
3. Bisquare weights of the input points
4. Residual values scaled by sigma

This function is based on the rob_checkfit routine from the AstroIDL User's Library.

pynrc.maths.robust.**linefit**(*inputX*, *inputY*, *iterMax=25*, *Bisector=False*, *BisquareLimit=6.0*, *CloseFactor=0.03*)
Outlier resistance two-variable linear regression function.

Based on the robust_linefit routine in the AstroIDL User's Library.

pynrc.maths.robust.**mean**(*inputData*, *Cut=3.0*, *axis=None*, *dtype=None*, *keepdims=False*, *return_std=False*, *return_mask=False*)
Robust Mean

Robust estimator of the mean of a data set. Based on the *resistant_mean* function from the AstroIDL User's Library. NaN values are excluded.

This function trims away outliers using the median and the median absolute deviation. An approximation formula is used to correct for the truncation caused by trimming away outliers.

Parameters **inputData** (*ndarray*) – The input data.

Keyword Arguments

- **Cut** (*float*) – Sigma for rejection; default is 3.0.
- **axis** (*None or int or tuple of ints, optional*) – Axis or axes along which the deviation is computed. The default is to compute the deviation of the flattened array.

  If this is a tuple of ints, a standard deviation is performed over multiple axes, instead of a single axis or all the axes as before. This is the equivalent of reshaping the input data and then taking the standard devation.

- **keepdims** (*bool, optional*) – If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the original *arr*.
- **return_std** (*bool*) – Also return the std dev calculated using only the "good" data?
- **return_mask** (*bool*) – If set to True, then return only boolean array of good (1) and rejected (0) values.

pynrc.maths.robust.**medabsdev**(*data*, *axis=None*, *keepdims=False*, *nan=True*)
Median Absolute Deviation

A "robust" version of standard deviation.

Parameters

- **data** (*ndarray*) – The input data.
- **axis** (*None or int or tuple of ints, optional*) – Axis or axes along which the deviation is computed. The default is to compute the deviation of the flattened array.

  If this is a tuple of ints, a standard deviation is performed over multiple axes, instead of a single axis or all the axes as before. This is the equivalent of reshaping the input data and then taking the standard devation.
- **keepdims** (*bool, optional*) – If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the original *arr*.
- **nan** (*bool, optional*) – Ignore NaNs? Default is True.

pynrc.maths.robust.**mode**(*inputData*, *axis=None*, *dtype=None*)
Robust estimator of the mode of a data set using the half-sample mode.

pynrc.maths.robust.**polyfit**(*inputX*, *inputY*, *order*, *iterMax=25*)
Outlier resistance two-variable polynomial function fitter.

Based on the robust_poly_fit routine in the AstroIDL User's Library.

Unlike robust_poly_fit, two different polynomial fitters are used because np.polyfit does not support non-uniform weighting of the data. For the weighted fitting, the SciPy Orthogonal Distance Regression module (scipy.odr) is used.

pynrc.maths.robust.**std**(*inputData*, *Zero=False*, *axis=None*, *dtype=None*, *keepdims=False*, *return_mask=False*)
Robust Sigma

Based on the robust_sigma function from the AstroIDL User's Library.

Calculate a resistant estimate of the dispersion of a distribution.

Use the median absolute deviation as the initial estimate, then weight points using Tukey's Biweight. See, for example, "Understanding Robust and Exploratory Data Analysis," by Hoaglin, Mosteller and Tukey, John Wiley & Sons, 1983, or equation 9 in Beers et al. (1990, AJ, 100, 32).

Parameters **inputData** (*ndarray*) – The input data.

Keyword Arguments

- **axis** (*None or int or tuple of ints, optional*) – Axis or axes along which the deviation is computed. The default is to compute the deviation of the flattened array.

If this is a tuple of ints, a standard deviation is performed over multiple axes, instead of a single axis or all the axes as before. This is the equivalent of reshaping the input data and then taking the standard devation.

- **keepdims** (*bool, optional*) – If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the original *arr*.
- **return_mask** (*bool*) – If set to True, then only return boolean array of good (1) and rejected (0) values.

NIRCam Tools

Functions Summary

read_filter(filter[, pupil, mask, module, ...])	Read filter bandpass.
nrc_utils.psf_coeff	
wfed_coeff(filter_or_bp[, force, save, ...])	PSF Coefficient Mod for WFE Drift
gen_image_coeff(filter_or_bp[, pupil, mask, ...])	Generate PSF
bg_sensitivity(filter_or_bp[, pupil, mask, ...])	Sensitivity Estimates
sat_limit_webbpsf(filter_or_bp[, pupil, ...])	Saturation limits
pix_noise([ngroup, nf, nd2, tf, rn, ktc, ...])	Noise per pixel
channel_select(bp)	Select wavelength channel
grism_res([pupil, module, m])	Grism information
build_mask([module, pixscale])	Create coronagraphic mask image
build_mask_detid(detid[, oversample, ...])	Create mask image for a given detector
coron_trans(name[, module, pixscale, fov, ...])	Build a transmission image of a coronagraphic mask spanning the 20" coronagraphic FoV.
offset_bar(filt, mask)	Bar mask offset locations
nrc_utils.config2	
nrc_utils.create_detops	

Documentation

pynrc.nrc_utils.**read_filter**(*filter*, *pupil=None*, *mask=None*, *module=None*, *ND_acq=False*, *ice_scale=None*, *nvr_scale=None*, *ote_scale=None*, *nc_scale=None*, *grism_order=1*, *coron_substrate=False*, ***kwargs*)

Read filter bandpass.

Read in filter throughput curve from file generated by STScI. Includes: OTE, NRC mirrors, dichroic, filter curve, and detector QE.

To Do: Account for pupil size reduction for DHS and grism observations.

Parameters

- **filter** (*str*) – Name of a filter.
- **pupil** (*str, None*) – NIRCam pupil elements such as grisms or lyot stops.
- **mask** (*str, None*) – Specify the coronagraphic occulter (spots or bar).
- **module** (*str*) – Module 'A' or 'B'.
- **ND_acq** (*bool*) – ND acquisition square in coronagraphic mask.

- **ice_scale** (*float*) – Add in additional OTE H2O absorption. This is a scale factor relative to 0.0131 um thickness. Also includes about 0.0150 um of photolyzed Carbon.
- **nvr_scale** (*float*) – Modify NIRCam non-volatile residue. This is a scale factor relative to 0.280 um thickness already built into filter throughput curves. If set to None, then assumes a scale factor of 1.0. Setting nvr_scale=0 will remove these contributions.
- **ote_scale** (*float*) – Scale factor of OTE contaminants relative to End of Life model. This is the same as setting ice_scale. Will override ice_scale value.
- **nc_scale** (*float*) – Scale factor for NIRCam contaminants relative to End of Life model. This model assumes 0.189 um of NVR and 0.050 um of water ice on the NIRCam optical elements. Setting this keyword will remove all NVR contributions built into the NIRCam filter curves. Overrides nvr_scale value.
- **grism_order** (*int*) – Option to use 2nd order grism throughputs instead. Useful if someone wanted to overlay the 2nd order contributions onto a wide field observation.
- **coron_substrate** (*bool*) – Explicit option to include coronagraphic substrate transmission even if mask=None. Gives the option of using LYOT pupilt with or without coron substrate.

Returns `pysynphot.obsbandpass` – A Pysynphot bandpass object.

`pynrc.nrc_utils.`**`wfed_coeff`**(*filter_or_bp*, *force=False*, *save=True*, *save_name=None*, *nsplit=None*, ***kwargs*)

PSF Coefficient Mod for WFE Drift

This function finds a relationship between PSF coefficients in the presence of WFE drift. For a series of WFE drift values, we generate corresponding PSF coefficients and fit a polynomial relationship to the residual values. This allows us to quickly modify a nominal set of PSF image coefficients to generate a new PSF where the WFE has drifted by some amplitude.

Keyword Arguments match those in `gen_psf_coeff()`.

Parameters

- **filter_or_bp** (*str*) – Name of a filter.
- **force** (*bool*) – Forces a recalcuation of coefficients even if saved PSF already exists. (default: False)
- **save** (*bool*) – Save the resulting WFE drift coefficents to a file? (default: True)
- **save_name** (*str, None*) – Full path name of save file (.npy) to save/load. If None, then a name is automatically generated, matching the `gen_psf_coeff()` function.
- **nsplit** (*int*) – Number of processors to split over. There are checks to make sure you're not requesting more processors than the current machine has available.

Example

Generate PSF coefficient, WFE drift modifications, then create an undrifted and drifted PSF.

```
>>> from pynrc import nrc_utils
>>> fpix, osamp = (128, 4)
>>> coeff  = nrc_utils.gen_psf_coeff('F210M', fov_pix=fpix, oversample=osamp)
>>> wfe_cf = nrc_utils.wfed_coeff('F210M', fov_pix=fpix, oversample=osamp)
>>> psf0   = nrc_utils.gen_image_coeff('F210M', coeff=coeff, fov_pix=fpix,␣
↪oversample=osamp)
```

```
>>> # Drift the coefficients
>>> wfe_drift = 5    # nm
>>> cf_fit = wfe_cf.reshape([wfe_cf.shape[0], -1])
>>> cf_mod = nrc_utils.jl_poly(np.array([wfe_drift]), cf_fit).reshape(coeff.shape)
>>> cf_new = coeff + cf_mod
>>> psf5nm = nrc_utils.gen_image_coeff('F210M', coeff=cf_new, fov_pix=fpix,␣
↪oversample=osamp)
```

pynrc.nrc_utils.**gen_image_coeff**(*filter_or_bp*, *pupil=None*, *mask=None*, *module='A'*, *coeff=None*, *coeff_hdr=None*, *sp_norm=None*, *nwaves=None*, *fov_pix=11*, *oversample=4*, *return_oversample=False*, *use_sp_waveset=False*, ***kwargs*)

Generate PSF

Create an image (direct, coronagraphic, grism, or DHS) based on a set of instrument parameters and PSF coefficients. The image is noiseless and doesn't take into account any non-linearity or saturation effects, but is convolved with the instrument throughput. Pixel values are in counts/sec. The result is effectively an idealized slope image.

If no spectral dispersers (grisms or DHS), then this returns a single image or list of images if sp_norm is a list of spectra.

Parameters

- **filter_or_bp** (str, `pysynphot.obsbandpass`) – Either the name of a filter or a Pysynphot bandpass.
- **pupil** (*str, None*) – NIRCam pupil elements such as grisms or lyot stops.
- **mask** (*str, None*) – Specify the coronagraphic occulter (spots or bar).
- **module** (*str*) – Module 'A' or 'B'.
- **sp_norm** (`pysynphot.spectrum`) – A normalized Pysynphot spectrum to generate image. If not specified, the default is flat in phot lam (equal number of photons per spectral bin). The default is normalized to produce 1 count/sec within that bandpass, assuming the telescope collecting area. Coronagraphic PSFs will further decrease this flux.
- **coeff** (*ndarray*) – A cube of polynomial coefficients for generating PSFs. This is generally oversampled with a shape (fov_pix*oversamp, fov_pix*oversamp, deg). If not set, this will be calculated using the `gen_psf_coeff()` function.
- **coeff_hdr** (*FITS header*) – Header information saved while generating coefficients.
- **nwaves** (*int*) – Option to specify the number of evenly spaced wavelength bins to generate and sum over to make final PSF. Useful for wide band filters with large PSFs over continuum sorurce.
- **use_sp_waveset** (*bool*) – Set this option to use *sp_norm* waveset instead of bandpass waveset. Useful if user inputs a high-resolution spectrum with line emissions, so may wants to keep a grism PSF (for instance) at native resolution rather than blurred with the bandpass waveset. TODO: Test.
- **fov_pix** (*int*) – Number of detector pixels in the image coefficient and PSF.
- **oversample** (*int*) – Factor of oversampling of detector pixels.
- **return_oversample** (*bool*) – If True, then also returns the oversampled version of the PSF.

Keyword Arguments

- **grism_order** (*int*) – Grism spectral order (default=1).

- **npsf** (*int*) – Number of evenly-spaced (with wavelength) monochromatic PSFs to generate with webbPSF. If not specified, then the default is to produce 20 PSFs/um. The wavelength range is determined by choosing those wavelengths where throughput is >0.001.
- **ndeg** (*int*) – Polynomial degree for PSF fitting. read_filter - ND_acq
- **ND_acq** (*bool*) – ND acquisition square in coronagraphic mask.

pynrc.nrc_utils.**bg_sensitivity**(*filter_or_bp*, *pupil=None*, *mask=None*, *module='A'*, *pix_scale=None*, *sp=None*, *units=None*, *nsig=10*, *tf=10.737*, *ngroup=2*, *nf=1*, *nd2=0*, *nint=1*, *coeff=None*, *coeff_hdr=None*, *fov_pix=11*, *oversample=4*, *quiet=True*, *forwardSNR=False*, *offset_r=0*, *offset_theta=0*, *return_image=False*, *image=None*, *cr_noise=True*, *dw_bin=None*, *ap_spec=None*, *rad_EE=None*, ***kwargs*)

Sensitivity Estimates

Estimates the sensitivity for a set of instrument parameters. By default, a flat spectrum is convolved with the specified bandpass. For imaging, this function also returns the surface brightness sensitivity.

The number of photo-electrons are computed for a source at some magnitude as well as the noise from the detector readout and some average zodiacal background flux. Detector readout noise follows an analytical form that matches extensive long dark observations during cryo-vac testing.

This function returns the n-sigma background limit in units of uJy (unless otherwise specified; valid units can be found on the Pysynphot webpage at https://pysynphot.readthedocs.io/).

For imaging, a single value is given assuming aperture photometry with a radius of ~1 FWHM rounded to the next highest integer pixel (or 2.5 pixels, whichever is larger). For spectral observations, this function returns an array of sensitivities at 0.1um intervals with apertures corresponding to 2 spectral pixels and a number of spatial pixels equivalent to 1 FWHM rounded to the next highest integer (minimum of 5 spatial pixels).

Parameters

- **Instrument Settings**
- **————————-**
- **filter_or_bp** (*Either the name of the filter or pre-computed Pysynphot bandpass.*)
- **pupil** (*NIRCam pupil elements such as grisms or lyot stops*)
- **mask** (*Specify the coronagraphic occulter (spots or bar)*)
- **module** (*'A' or 'B'*)
- **pix_scale** (*Pixel scale in arcsec/pixel*)
- **Spectrum Settings**
- **————————-**
- **sp** (*A pysynphot spectral object to calculate sensitivity*) – (default: Flat spectrum in photlam)
- **nsig** (*Desired nsigma sensitivity*)
- **units** (*Output units (defaults to uJy for grisms, nJy for imaging)*)
- **forwardSNR** (*Find the SNR of the input spectrum instead of determining sensitivity.*)
- **Ramp Settings**
- **————————-**
- **tf** (*Time per frame*)
- **ngroup** (*Number of groups per integration*)

- **nf** (*Number of averaged frames per group*)
- **nd2** (*Number of dropped frames per group*)
- **nint** (*Number of integrations/ramps to consider*)
- **PSF Information**
- **————————-**
- **coeff** (*A cube of polynomial coefficients for generating PSFs. This is*) – generally oversampled with a shape (fov_pix*oversamp, fov_pix*oversamp, deg). If not set, this will be calculated using `gen_psf_coeff()`.
- **coeff_hdr** (*Header associated with coeff cube.*)
- **fov_pix** (*Number of detector pixels in the image coefficient and PSF.*)
- **oversample** (*Factor of oversampling of detector pixels.*)
- **offset_r** (*Radial offset of the target from center.*)
- **offset_theta** (*Position angle for that offset, in degrees CCW (+Y).*)
- **Misc.**
- **————————-**
- **image** (*Explicitly pass image data rather than calculating from coeff.*)
- **return_image** (*Instead of calculating sensitivity, return the image calced from coeff.*) – Useful if needing to calculate sensitivities for many different settings.
- **rad_EE** (*Extraction aperture radius (in pixels) for imaging mode.*)
- **dw_bin** (*Delta wavelength to calculate spectral sensitivities (grisms & DHS).*)
- **ap_spec** (*Instead of dw_bin, specify the spectral extraction aperture in pixels.*) – Takes priority over dw_bin. Value will get rounded up to nearest int.
- **cr_noise** (*Include noise from cosmic ray hits?*)

Keyword Arguments

- **– zfact, ra, dec, thisday, [locstr, year, day]** (`zodi_spec`) –
- **– rn, ktc, idark, and p_excess** (`pix_noise`) –
- **– npsf and ndeg** (*gen_psf_coeff*) –
- **– ND_acq** (`read_filter`) –

pynrc.nrc_utils.**sat_limit_webbpsf**(*filter_or_bp*, *pupil=None*, *mask=None*, *module='A'*, *pix_scale=None*, *sp=None*, *bp_lim=None*, *int_time=21.47354*, *full_well=81000.0*, *well_frac=0.8*, *coeff=None*, *coeff_hdr=None*, *fov_pix=11*, *oversample=4*, *quiet=True*, *units='vegamag'*, *offset_r=0*, *offset_theta=0*, ***kwargs*)

Saturation limits

Estimate the saturation limit of a point source for some bandpass. By default, it outputs the max K-Band magnitude assuming a G2V star, following the convention on the UA NIRCam webpage. This can be useful if one doesn't know how bright a source is in the selected NIRCam filter bandpass. However any user-defined bandpass (or user-defined spectrum) can be specifed. These must follow the Pysynphot conventions found here: http://pysynphot.readthedocs.org/en/latest/using_pysynphot.html

This function returns the saturation limit in Vega magnitudes by default, however, any flux unit supported by Pysynphot is possible via the 'units' keyword.

Parameters

- **Instrument Settings**
- **————————-**
- **filter_or_bp** (*Either the name of the filter or pre-computed Pysynphot bandpass.*)
- **pupil** (*NIRCam pupil elements such as grisms or lyot stops*)
- **mask** (*Specify the coronagraphic occulter (spots or bar)*)
- **module** (*'A' or 'B'*)
- **Spectrum Settings**
- **————————-**
- **sp** (*A Pysynphot spectrum to calculate saturation (default: G2V star)*)
- **bp_lim** (*A Pysynphot bandpass at which we report the magnitude that will*) – saturate the NIRCam band assuming some spectrum sp
- **units** (*Output units for saturation limit*)
- **Detector Settings**
- **————————-**
- **int_time** (*Integration time in seconds (default corresponds to 2 full frames)*)
- **full_well** (*Detector full well level in electrons.*)
- **well_frac** (*Fraction of full well to consider "saturated." 0.8 by default.*)
- **PSF Information**
- **————————-**
- **coeff** (*A cube of polynomial coefficients for generating PSFs. This is*) –

  generally oversampled and has the shape:

  [fov_pix*oversample, fov_pix*oversample, deg]

  If not set, this this will be calculated from fov_pix, oversample, and npsf by generating a number of webbPSF images within the bandpass and fitting a high-order polynomial.
- **fov_pix** (*Number of detector pixels in the image coefficient and PSF.*)
- **oversample** (*Factor of oversampling of detector pixels.*)
- **offset_r** (*Radial offset of the target from center.*)
- **offset_theta** (*Position angle for that offset, in degrees CCW (+Y).*)

Keyword Arguments

- **- npsf and ndeg** (*gen_psf_coeff*) –
- **- ND_acq** (read_filter) –

pynrc.nrc_utils.**pix_noise**(*ngroup=2*, *nf=1*, *nd2=0*, *tf=10.737*, *rn=15.0*, *ktc=29.0*, *p_excess=0, 0*, *fsrc=0.0*, *idark=0.003*, *fzodi=0*, *fbg=0*, *ideal_Poisson=False*, *ff_noise=False*, ***kwargs*)

Noise per pixel

Theoretical noise calculation of a generalized MULTIACCUM ramp in terms of e-/sec. Includes flat field errors from JWST-CALC-003894.

Parameters

- **n** (*int*) – Number of groups in integration rampl
- **m** (*int*) – Number of frames in each groupl
- **s** (*int*) – Number of dropped frames in each groupl
- **tf** (*float*) – Frame timel
- **rn** (*float*) – Read Noise per pixel (e-).
- **ktc** (*float*) – kTC noise (in e-). Only valid for single frame (n=1)l
- **p_excess** (*array-like*) – An array or list of two elements that holds the parameters describing the excess variance observed in effective noise plots. By default these are both 0. For NIRCam detectors, recommended values are [1.0,5.0] for SW and [1.5,10.0] for LW.
- **fsrc** (*float*) – Flux of source in e-/sec/pix.
- **idark** (*float*) – Dark current in e-/sec/pix.
- **fzodi** (*float*) – Zodiacal light emission in e-/sec/pix.
- **fbg** (*float*) – Any additional background (telescope emission or scattered light?)
- **ideal_Poisson** (*bool*) – If set to True, use total signal for noise estimate, otherwise MULTIACCUM equation is used?
- **ff_noise** (*bool*) – Include flat field errors in calculation? From JWST-CALC-003894. Default=False.

Notes

Various parameters can either be single values or numpy arrays. If multiple inputs are arrays, make sure their array sizes match. Variables that need to have the same array shapes (or a single value):

- n, m, s, & tf
- rn, idark, ktc, fsrc, fzodi, & fbg

Array broadcasting also works.

Example

```
>>> n = np.arange(50)+1  # An array of different ngroups to test out
```

```
>>> # Create 2D Gaussian PSF with FWHM = 3 pix
>>> npix = 20  # Number of pixels in x and y direction
>>> fwhm = 3.0
>>> x = np.arange(0, npix, 1, dtype=float)
>>> y = x[:,np.newaxis]
>>> x0 = y0 = npix // 2  # Center position
>>> fsrc = np.exp(-4*np.log(2.) * ((x-x0)**2 + (y-y0)**2) / fwhm**2)
>>> fsrc /= fsrc.max()
>>> fsrc *= 10  # 10 counts/sec in peak pixel
>>> fsrc = fsrc.reshape(npix,npix,1)  # Necessary for broadcasting
```

```
>>> # Represents pixel array w/ slightly different RN/pix
>>> rn = 15 + np.random.normal(loc=0, scale=0.5, size=(1,npix,npix))
>>> # Results is a 50x(20x20) showing the noise in e-/sec/pix at each group
>>> noise = pix_noise(ngroup=n, rn=rn, fsrc=fsrc)
```

pynrc.nrc_utils.**channel_select**(*bp*)
Select wavelength channel

Based on input bandpass, return the pixel scale, dark current, and excess read noise parameters. These values are typical for either a SW or LW NIRCam detector.

Parameters **bp** (pysynphot.obsbandpass) – NIRCam filter bandpass.

pynrc.nrc_utils.**grism_res**(*pupil='GRISM'*, *module='A'*, *m=1*)
Grism information

Based on the pupil input and module, return the spectral dispersion and resolution as a tuple (res, dw).

Parameters

- **pupil** (*str*) – 'GRISM0' or 'GRISM90', otherwise assume res=1000 pix/um
- **module** (*str*) – 'A' or 'B'
- **m** (*int*) – Spectral order (1 or 2).

pynrc.nrc_utils.**build_mask**(*module='A'*, *pixscale=0.03*)
Create coronagraphic mask image

Return a truncated image of the full coronagraphic mask layout for a given module.

+V3 is up, and +V2 is to the left.

pynrc.nrc_utils.**build_mask_detid**(*detid*, *oversample=1*, *ref_mask=None*, *pupil=None*)
Create mask image for a given detector

Return a full coronagraphic mask image as seen by a given SCA. +V3 is up, and +V2 is to the left.

Parameters

- **detid** (*str*) – Name of detector, 'A1', A2', ... 'A5' (or 'ALONG'), etc.
- **oversample** (*float*) – How much to oversample output mask relative to detector sampling.
- **ref_mask** (*str or None*) – Reference mask for placement of coronagraphic mask elements. If None, then defaults are chosen for each detector.
- **pupil** (*str or None*) – Which Lyot pupil stop is being used? If None, then defaults based on ref_mask.

pynrc.nrc_utils.**coron_trans**(*name*, *module='A'*, *pixscale=None*, *fov=20*, *nd_squares=True*)
Build a transmission image of a coronagraphic mask spanning the 20" coronagraphic FoV.

Pulled from WebbPSF

pynrc.nrc_utils.**offset_bar**(*filt*, *mask*)
Bar mask offset locations

Get the appropriate offset in the x-position to place a source on a bar mask. Each bar is 20" long with edges and centers corresponding to:

```
SWB: [1.03, 2.10, 3.10] (um) => [-10, 0, +10] (asec)
LWB: [2.30, 4.60, 6.90] (um) => [+10, 0, -10] (asec)
```

Spectral Tools

Class Summary

source_spectrum(name, sptype, mag_val, bp, ...)	Model source spectrum
planets_sb12([atmo, mass, age, entropy, ...])	Exoplanet spectrum from Spiegel & Burrows (2012)

Functions Summary

stellar_spectrum(sptype, *renorm_args, **kwargs)	Stellar spectrum
BOSZ_spectrum(Teff, metallicity, log_g[, ...])	BOSZ stellar atmospheres (Bohlin et al 2017).
sp_accr(mmdot[, rin, dist, truncated, ...])	Exoplanet accretion flux values (Zhu et al., 2015).
jupiter_spec([dist, waveout, fluxout, base_dir])	Jupiter as an Exoplanet
zodi_spec([zfact, ra, dec, thisday])	Zodiacal light spectrum.
zodi_euclid(locstr, year, day[, ...])	IPAC Euclid Background Model
bp_2mass(filter)	2MASS Bandpass
bin_spectrum(sp, wave[, waveunits])	Rebin spectrum

Documentation

class pynrc.nrc_utils.**source_spectrum**(*name*, *sptype*, *mag_val*, *bp*, *votable_file*, *Teff=None*, *metallicity=None*, *log_g=None*, *Av=None*, ***kwargs*)

Bases: `object`

Model source spectrum

The class ingests spectral information of a given target and generates `pysynphot.spectrum` model fit to the known photometric SED. Two model routines can fit. The first is a very simple scale factor that is applied to the input spectrum, while the second takes the input spectrum and adds an IR excess modeled as a modified blackbody function.

Parameters

- **name** (*string*) – Source name.
- **sptype** (*string*) – Assumed stellar spectral type. Not relevant if Teff, metallicity, and log_g are specified.
- **mag_val** (*float*) – Magnitude of input bandpass for initial scaling of spectrum.
- **bp** (`pysynphot.obsbandpass`) – Bandpass to apply initial mag_val scaling.
- **votable_file** (*string*) – VOTable name that holds the source's photometry. The user can find the relevant data at http://vizier.u-strasbg.fr/vizier/sed/ and click download data.

Keyword Arguments

- **Teff** (*float*) – Effective temperature ranging from 3500K to 30000K.
- **metallicity** (*float*) – Metallicity [Fe/H] value ranging from -2.5 to 0.5.
- **log_g** (*float*) – Surface gravity (log g) from 0 to 5.
- **catname** (*str*) – Catalog name, including 'bosz', 'ck04models', and 'phoenix'. Default is 'bosz', which comes from *BOSZ_spectrum()*.

- **res** (*str*) – Spectral resolution to use (200 or 2000 or 20000).
- **interpolate** (*bool*) – Interpolate spectrum using a weighted average of grid points surrounding the desired input parameters.

Example

Generate a source spectrum and fit photometric data

```
>>> import pynrc
>>> from pynrc.nrc_utils import source_spectrum
>>>
>>> name = 'HR8799'
>>> vot = 'votables/{}.vot'.format(name)
>>> bp_k = pynrc.bp_2mass('k')
>>>
>>> # Read in stellar spectrum model and normalize to Ks = 5.24
>>> src = source_spectrum(name, 'F0V', 5.24, bp_k, vot,
>>>                       Teff=7430, metallicity=-0.47, log_g=4.35)
>>> # Fit model to photometry from 0.1 - 30 micons
>>> # Saves pysynphot spectral object at src.sp_model
>>> src.fit_SED(wlim=[0.1,30])
>>> sp_sci = src.sp_model
```

Methods Summary

bb_jy(wave, T)	Blackbody function (Jy)
model_scale(x[, sp])	Simple model to scale stellar spectrum
model_IRexcess(x[, sp])	Model for stellar spectrum with IR excesss
func_resid(x[, IR_excess, wlim, use_err])	Calculate model residuals
fit_SED([x0, robust, use_err, IR_excess, ...])	Fit a model function to photometry
plot_SED([ax, return_figax, xr, yr, units])	

Methods Documentation

bb_jy (*wave*, *T*)
Blackbody function (Jy)

For a given wavelength set (in um) and a Temperature (K), return the blackbody curve in units of Jy.

Parameters

- **wave** (*array_like*) – Wavelength array in microns
- **T** (*float*) – Temperature of blackbody (K)

model_scale (*x*, *sp=None*)
Simple model to scale stellar spectrum

model_IRexcess (*x*, *sp=None*)
Model for stellar spectrum with IR excesss

Model of a stellar spectrum plus IR excess, where the excess is a modified blackbody. The final model follows the form:

$$x_0 BB(\lambda, x_1)\lambda^{x_2}$$

func_resid(*x*, *IR_excess=False*, *wlim=[0.1, 30]*, *use_err=True*)
Calculate model residuals

Parameters

- **x** (*array_like*) – Model parameters for either *model_scale* or *model_IRexcess*. See these two functions for more details.
- **IR_excess** (*bool*) – Include IR excess in model fit? This is a simple modified blackbody.
- **wlim** (*array_like*) – Min and max limits for wavelengths to consider (microns).
- **use_err** (*bool*) – Should we use the uncertainties in the SED photometry for weighting?

fit_SED(*x0=None*, *robust=True*, *use_err=True*, *IR_excess=False*, *wlim=[0.3, 10]*, *verbose=True*)
Fit a model function to photometry

Use `scipy.optimize.least_squares()` to find the best fit model to the observed photometric data. If not parameters passed, then defaults are set.

Keyword Arguments

- **x0** (*array_like*) – Initial guess of independent variables.
- **robust** (*bool*) – Perform an outlier-resistant fit.
- **use_err** (*bool*) – Should we use the uncertainties in the SED photometry for weighting?
- **IR_excess** (*bool*) – Include IR excess in model fit? This is a simple modified blackbody.
- **wlim** (*array_like*) – Min and max limits for wavelengths to consider (microns).
- **verbose** (*bool*) – Print out best-fit model parameters. Defalt is True.

plot_SED(*ax=None*, *return_figax=False*, *xr=[0.3, 30]*, *yr=None*, *units='Jy'*, ***kwargs*)

class pynrc.nrc_utils.**planets_sb12**(*atmo='hy1s'*, *mass=1*, *age=100*, *entropy=10.0*, *distance=10*, *accr=False*, *mmdot=None*, *mdot=None*, *accr_rin=2.0*, *truncated=False*, *base_dir=None*, ***kwargs*)

Bases: `object`

Exoplanet spectrum from Spiegel & Burrows (2012)

This contains 1680 files, one for each of 4 atmosphere types, each of 15 masses, and each of 28 ages. Wavelength range of 0.8 - 15.0 um at moderate resolution (R ~ 204).

The flux in the source files are at 10 pc. If the distance is specified, then the flux will be scaled accordingly. This is also true if the distance is changed by the user. All other properties (atmo, mass, age, entropy) are not adjustable once loaded.

Parameters

- **atmo** (*str*) –

  A string consisting of one of four atmosphere types:

  – 'hy1s' = hybrid clouds, solar abundances
  – 'hy3s' = hybrid clouds, 3x solar abundances
  – 'cf1s' = cloud-free, solar abundances
  – 'cf3s' = cloud-free, 3x solar abundances

- **mass** (*float*) – A number 1 to 15 Jupiter masses.

- **age** (*float*) – Age in millions of years (1-1000)
- **entropy** (*float*) – Initial entropy (8.0-13.0) in increments of 0.25
- **distance** (*float*) – Assumed distance in pc (default is 10pc)
- **accr** (*bool*) – Include accretion (default: False)?
- **mmdot** (*float*) – From Zhu et al. (2015), the Mjup^2/yr value. If set to None then calculated from age and mass.
- **mdot** (*float*) – Or use mdot (Mjup/yr) instead of mmdot.
- **accr_rin** (*float*) – Inner radius of accretion disk (units of RJup; default: 2)
- **truncated** (*bool*) – Full disk or truncated (ie., MRI; default: False)?
- **base_dir** (*str, None*) – Location of atmospheric model sub-directories.

Attributes Summary

`age`	Age in millions of years
`atmo`	Atmosphere type
`distance`	Assumed distance to source (pc)
`entropy`	Initial entropy (8.0-13.0)
`flux`	Spectral flux
`fluxunits`	Flux units
`mass`	Mass of planet (MJup)
`mdot`	Accretion rate in MJup/yr
`wave`	Wavelength of spectrum
`waveunits`	Wavelength units

Methods Summary

`export_pysynphot`([waveout, fluxout])	Output to `pysynphot.spectrum` object

Methods Documentation

`export_pysynphot`(*waveout='angstrom'*, *fluxout='flam'*)
Output to `pysynphot.spectrum` object

Export object settings to a `pysynphot.spectrum`.

Parameters

- **waveout** (*str*) – Wavelength units for output
- **fluxout** (*str*) – Flux units for output

`pynrc.nrc_utils.`**`stellar_spectrum`**(*sptype*, **renorm_args*, ***kwargs*)
Stellar spectrum

Similar to specFromSpectralType() in WebbPSF/Poppy, this function uses a dictionary of fiducial values to determine an appropriate spectral model. If the input spectral type is not found, this function interpolates the effective temperature, metallicity, and log g values .

You can also specify renormalization arguments to pass to `sp.renorm()`. The order (after `sptype`) should be (`value, units, bandpass`):

```
>>> sp = stellar_spectrum('G2V', 10, 'vegamag', bp)
```

Flat spectrum (in photlam) are also allowed via the 'flat' string.

Use `catname='bosz'` for BOSZ stellar atmosphere (ATLAS9) (default) Use `catname='ck04models'` keyword for ck04 models Use `catname='phoenix'` keyword for Phoenix models

Keywords exist to directly specify Teff, metallicity, an log_g rather than a spectral type.

Parameters

- **sptype** (*str*) – Spectral type, such as 'A0V' or 'K2III'.
- **renorm_args** (*tuple*) – Renormalization arguments to pass to `sp.renorm()`. The order (after `sptype`) should be (`value, units, bandpass`) Bandpass should be a `pysynphot.obsbandpass` type.

Keyword Arguments

- **catname** (*str*) – Catalog name, including 'bosz', 'ck04models', and 'phoenix'. Default is 'bosz', which comes from *BOSZ_spectrum()*.
- **Teff** (*float*) – Effective temperature ranging from 3500K to 30000K.
- **metallicity** (*float*) – Metallicity [Fe/H] value ranging from -2.5 to 0.5.
- **log_g** (*float*) – Surface gravity (log g) from 0 to 5.
- **res** (*str*) – BOSZ spectral resolution to use (200 or 2000 or 20000). Default: 2000.
- **interpolate** (*bool*) – Interpolate BOSZ spectrum using a weighted average of grid points surrounding the desired input parameters. Default is True. Default: True

pynrc.nrc_utils.**BOSZ_spectrum**(*Teff*, *metallicity*, *log_g*, *res=2000*, *interpolate=True*, ***kwargs*)

BOSZ stellar atmospheres (Bohlin et al 2017).

Read in a spectrum from the BOSZ stellar atmosphere models database. Returns a Pysynphot spectral object. Wavelength values range between 1000-32000 Angstroms. Teff range from 3500K to 36000K.

This function interoplates the model grid by reading in those models closest in temperature, metallicity, and log g to the desired parameters, then takes the weighted average of these models based on their relative offsets. Can also just read in the closest model by setting interpolate=False.

Different spectral resolutions can also be specified, currently only res=200 or 2000 or 20000.

Parameters

- **Teff** (*float*) – Effective temperature ranging from 3500K to 30000K.
- **metallicity** (*float*) – Metallicity [Fe/H] value ranging from -2.5 to 0.5.
- **log_g** (*float*) – Surface gravity (log g) from 0 to 5.

Keyword Arguments

- **res** (*str*) – Spectral resolution to use (200 or 2000 or 20000).
- **interpolate** (*bool*) – Interpolate spectrum using a weighted average of grid points surrounding the desired input parameters.

References

https://archive.stsci.edu/prepds/bosz/

pynrc.nrc_utils.**sp_accr**(*mmdot*, *rin=2*, *dist=10*, *truncated=False*, *waveout='angstrom'*, *fluxout='flam'*, *base_dir=None*)

Exoplanet accretion flux values (Zhu et al., 2015).

Calculated the wavelength-dependent flux of an exoplanet accretion disk/shock from Zhu et al. (2015). A

Note: This function only uses the table of photometric values to calculate photometric brightness from a source, so not very useful for simulating spectral observations.

Parameters

- **mmdot** (*float*) – Product of the exoplanet mass and mass accretion rate (MJup^2/yr). Values range from 1e-7 to 1e-2.
- **rin** (*float*) – Inner radius of accretion disk (units of RJup; default: 2).
- **dist** (*float*) – Distance to object (pc).
- **truncated** (*bool*) – If True, then the values are for a disk with Rout=50 RJup, otherwise, values were calculated for a full disk (Rout=1000 RJup). Accretion from a "tuncated disk" is due mainly to MRI. Luminosities for full and truncated disks are very similar.
- **waveout** (*str*) – Wavelength units for output
- **fluxout** (*str*) – Flux units for output
- **base_dir** (*str, None*) – Location of accretion model sub-directories.

pynrc.nrc_utils.**jupiter_spec**(*dist=10*, *waveout='angstrom'*, *fluxout='flam'*, *base_dir=None*)

Jupiter as an Exoplanet

Read in theoretical Jupiter spectrum from Irwin et al. 2014 and output as a `pysynphot.spectrum`.

Parameters

- **dist** (*float*) – Distance to Jupiter (pc).
- **waveout** (*str*) – Wavelength units for output.
- **fluxout** (*str*) – Flux units for output.
- **base_dir** (*str, None*) – Location of tabulated file irwin_2014_ref_spectra.txt.

pynrc.nrc_utils.**zodi_spec**(*zfact=None*, *ra=None*, *dec=None*, *thisday=None*, ***kwargs*)

Zodiacal light spectrum.

New: Use *ra*, *dec*, and *thisday* keywords to call *jwst_backgrounds* to obtain more accurate predictions of the background.

Creates a spectrum of the zodiacal light emission in order to estimate the in-band sky background flux. This is primarily the addition of two blackbodies at T=5300K (solar scattered light) and T=282K (thermal dust emission) that have been scaled to match literature flux values.

In reality, the intensity of the zodiacal dust emission varies as a function of viewing position. In this case, we have added the option to scale the zodiacal level (or each component individually) by some user-defined factor 'zfact'. The user can set zfact as a scalar in order to scale the entire spectrum. If defined as a list, tuple, or np array, then the each component gets scaled where T=5300K corresponds to the first elements and T=282K is the second element of the array.

The *zfact* parameter has no effect if *jwst_backgrounds* is called. Representative values for zfact:

- 0.0 - No zodiacal emission
- 1.0 - Minimum zodiacal emission from JWST-CALC-003894
- 1.2 - Required NIRCam performance
- 2.5 - Average (default)
- 5.0 - High
- 10.0 - Maximum

Parameters

- **zfact** (*float*) – Factor to scale Zodiacal spectrum (default 2.5).
- **ra** (*float*) – Right ascension in decimal degrees
- **dec** (*float*) – Declination in decimal degrees
- **thisday** (*int*) – Calendar day to use for background calculation. If not given, will use the average of visible calendar days.

Returns `pysynphot.spectrum` – Output is a Pysynphot spectrum with default units of flam (erg/s/cm^2/A/sr). Note: Pysynphot doesn't recognize that it's per steradian, but we must keep that in mind when integrating the flux per pixel.

Notes

Added the ability to query the Euclid background model using *`zodi_euclid()`* for a specific location and observing time. The two blackbodies will be scaled to the 1.0 and 5.5 um emission. This functionality is deprecated in favor of jwst_backgrounds.

Keyword Arguments

- **`locstr`** – Object name or RA/DEC (decimal degrees or sexigesimal). Queries the IPAC Euclid Background Model
- **`year`** (*`int`*) – Year of observation.
- **`day`** (*`float`*) – Day of observation.

`pynrc.nrc_utils.`**`zodi_euclid`**(*locstr*, *year*, *day*, *wavelengths=[1, 5.5]*, *ido_viewin=0*, ***kwargs*)

IPAC Euclid Background Model

Queries the IPAC Euclid Background Model in order to get date and position-specific zodiacal dust emission.

The program relies on `urllib2` to download the page in XML format. However, the website only allows single wavelength queries, so this program implements a multithreaded procedure to query multiple wavelengths simultaneously. However, due to the nature of `urllib2` library, only so many requests are allowed to go out at a time, so this process can take some time to complete. Testing shows about 500 wavelengths in 10 seconds as a rough ballpark.

Recommended to grab only a few wavelengths for normalization purposes.

References

See the Euclid Help Website for more details.

pynrc.nrc_utils.**bp_2mass**(*filter*)
2MASS Bandpass

Create a 2MASS J, H, or Ks filter bandpass used to generate synthetic photometry.

Parameters **filter** (*str*) – Filter 'j', 'h', or 'k'.

Returns pysynphot.obsbandpass – A Pysynphot bandpass object.

pynrc.nrc_utils.**bin_spectrum**(*sp*, *wave*, *waveunits='um'*)
Rebin spectrum

Rebin a pysynphot.spectrum to a lower wavelength grid. This function first converts the input spectrum to units of counts then combines the photon flux onto the specified wavelength grid.

Output spectrum units are the same as the input spectrum.

Parameters

- **sp** (pysynphot.spectrum) – Spectrum to rebin.
- **wave** (*array_like*) – Wavelength grid to rebin onto.
- **waveunits** (*str*) – Units of wave input. Must be recognizeable by Pysynphot.

Returns pysynphot.spectrum – Rebinned spectrum in same units as input spectrum.

Speckle Noise Tools

Disclaimer

These tools were created for investigations into speckle maps during development of pynrc. Documentation is relatively sparse and functions may not be maintained very well with respect to updated versions of Python and webbpsf. They are provided here for completeness, and may even prove useful to someone in the future, present, or past(?).

Class Summary

OPD_extract(opd, header[, seg_terms, verbose])	Decompose OPD to Zernike/Hexike coefficients

Functions Summary

opd_extract_mp(opds, header[, nproc])	Generate multiple OPD_extract objects
opd_sci_gen(opd)	OPD image for science observation
opd_sci_gen_mp(opds_all[, nproc])	Multiple OPD images for science observations
opd_ref_gen(args[, verbose])	Generate a drifted OPD image
ODP_drift_all(wfe_drift, opds_all, ...)	Drift list of OPDs
get_psf(opd, header[, filter, mask, pupil, ...])	Generate PSF
gen_psf_ref_all(opd_ref_list_all, ...[, verbose])	Calculate a series of reference PSFs
get_contrast(speckle_noise_image, planet_psf)	Output single contrast curve

continues on next page

Table 28 – continued from previous page

residual_speckle_image(psf_star_all, psf_ref_all)	Create residual speckle images
speckle_noise_image(psf_star_all, psf_ref_all)	Create a speckle noise image map
read_opd_file(opd_file[, opd_path, header])	Read in the OPD file data and header (optional)
read_opd_slice(pupilopd[, opd_path, header])	Get only a specified OPD slice from file.

Documentation

class pynrc.speckle_noise.**OPD_extract**(*opd*, *header*, *seg_terms=30*, *verbose=False*)

Bases: object

Decompose OPD to Zernike/Hexike coefficients

For a given JWST OPD image and header, extract the Zernike/Hexike components for the overall pupil and each mirror segment. Makes use of functions in webbpsf.

Parameters

- **opd** (*ndarray*) – OPD image map.
- **header** (*obj*) – Header extracted from OPD FITS files.
- **seg_terms** (*int*) – Number of Hexike terms to fit to each mirror segment.
- **verbose** (*bool*) – Print information as fittings are performed.

Attributes Summary

mask_pupil	Tricontagon mask
mask_opd	Mask generated from OPD map
coeff_pupil	Zernike coefficients for overall pupil
coeff_segs	Hexike coefficients for each segment

Methods Summary

mask_seg(i)	Return a sampled subsection of the analytic segment mask
opd_seg(i[, opd_pupil])	Return a subsection of some OPD image for the provided segment index
combine_opd_segs([opd_segs])	Combine list of OPD segments into a single image

Methods Documentation

mask_seg(*i*)

Return a sampled subsection of the analytic segment mask

Parameters **i** (*int*) – Segment index

opd_seg(*i*, *opd_pupil=None*)

Return a subsection of some OPD image for the provided segment index

Parameters

- **i** (*int*) – Segment index
- **opd_pupil** (*mask*) – Optional pupil mask to modify resulting segment OPD

combine_opd_segs (*opd_segs=None*)
Combine list of OPD segments into a single image

pynrc.speckle_noise.**opd_extract_mp** (*opds*, *header*, *nproc=None*)
Generate multiple OPD_extract objects

If you have multiple opd maps to perform Zernike/Hexike extractions on, this function calls Python's multiprocessor routines to perform calculations in parallel.

Parameters

- **opds** (*ndarray*) – Data cube of size (nopds, ny, nx)
- **header** (*obj*) – Header information
- **nproc** (*int, None*) – Number of simultaneous processes to perform. If not set, then:

```
>>> nproc = int(np.min([nopd,mp.cpu_count()*0.75]))
```

pynrc.speckle_noise.**opd_sci_gen** (*opd*)
OPD image for science observation

Function to go through an OPD class and generate a science OPD image.

Parameters **opd** (*obj*) – OPD_extract object

pynrc.speckle_noise.**opd_sci_gen_mp** (*opds_all*, *nproc=None*)
Multiple OPD images for science observations

Function to go through multiple OPD classes and generate science OPD images.

Parameters

- **opds_all** (*obj list*) – List of OPD_extract objects
- **nproc** (*int, None*) – Number of simultaneous processes to perform. If not set, then:

```
>>> nproc = int(np.min([nopd,mp.cpu_count()*0.75]))
```

pynrc.speckle_noise.**opd_ref_gen** (*args*, *verbose=False*)
Generate a drifted OPD image

pynrc.speckle_noise.**ODP_drift_all** (*wfe_drift*, *opds_all*, *pup_cf_std*, *seg_cf_std*, *opd_resid_list*)
Drift list of OPDs

Drift a list of OPDs by some RMS WFE (multiprocess function).

Parameters

- **wfe_drift** (*float*) – Single value in nm
- **opds_all** (*list*) – List of OPD objects (usually 10 from RevV OPDs)
- **pup_cf_std** (*float*) – Zernike sigma for overall pupil
- **seg_cf_std** (*float*) – Hexike sigma for segments
- **opd_resid_list** (*list*) – List of residual images (from opd_sci_gen) for each OPD

pynrc.speckle_noise.**get_psf** (*opd*, *header*, *filter='F410M'*, *mask=None*, *pupil=None*, *jitter=False*, *njit=10*, *jitter_sigma=0.005*, *r=None*, *theta=None*, ***kwargs*)
Generate PSF

Create a NIRCam PSF based on the input opd, header, and other info.

The jitter model recomputes the PSF at random positional offsets with a standard deviation equal to jitter_sigma. It does this njit times and averages the resulting PSFs.

pynrc.speckle_noise.**gen_psf_ref_all**(*opd_ref_list_all*, *opd_header*, *filt*, *mask*, *pupil*, *verbose=False*, ***kwargs*)

Calculate a series of reference PSFs

Given a list of reference OPD, calculate a series of PSFs. Assume that opd_ref_list_all is a nested list of drifted OPDs. For instance:

```
[[10 OPDs with 1nm drift], [10 OPDs with 5nm drift], [10 OPDs with 10nm drift]]
```

pynrc.speckle_noise.**get_contrast**(*speckle_noise_image*, *planet_psf*)

Output single contrast curve

Return the contrast for curve for a speckle noise map. Both inputs are HDULists.

pynrc.speckle_noise.**residual_speckle_image**(*psf_star_all*, *psf_ref_all*)

Create residual speckle images

For a list of science and reference PSFs, create residual speckle images

pynrc.speckle_noise.**speckle_noise_image**(*psf_star_all*, *psf_ref_all*)

Create a speckle noise image map

For a list of science and reference PSFs, create a speckle noise image map

pynrc.speckle_noise.**read_opd_file**(*opd_file*, *opd_path=None*, *header=True*)

Read in the OPD file data and header (optional)

pynrc.speckle_noise.**read_opd_slice**(*pupilopd*, *opd_path=None*, *header=True*)

Get only a specified OPD slice from file.

pupilopd is a tuple of form (filename, slice)

Coordinates Summary

dist_image(image[, pixscale, center, ...])	Pixel distances
xy_to_rtheta(x, y)	Convert (x,y) to (r,theta)
rtheta_to_xy(r, theta)	Convert (r,theta) to (x,y)
xy_rot(x, y, ang)	Rotate (x,y) positions to new coords
det_to_V2V3(image, detid)	Same as *det_to_sci*
V2V3_to_det(image, detid)	Same as *sci_to_det*
plotAxes(ax[, position, label1, label2, ...])	Compass arrows

Image Manipulation Summary

hist_indices(values[, bins, return_more])	Histogram indices
binned_statistic(x, values[, func, bins])	Binned statistic
frebin(image[, dimensions, scale, total])	Fractional rebin
fshift(image[, delx, dely, pad, cval])	Fractional image shift
fourier_imshift(image, xshift, yshift[, ...])	Fourier shift image
shift_subtract(params, reference, target[, ...])	Shift and subtract image

continues on next page

Table 32 – continued from previous page

align_LSQ(reference, target[, mask, pad, ...])	Find best shift value
scale_ref_image(im1, im2[, mask, ...])	Reference image scaling
optimal_difference(im_sci, im_ref, scale[, ...])	Optimize subtraction of ref PSF
pad_or_cut_to_size(array, new_shape[, fill_val])	Resize an array to a new shape by either padding with zeros or trimming off rows and/or columns.
fix_nans_with_med(im[, niter_max, verbose])	Iteratively fix NaNs with surrounding Real data

Polynomial Fitting Summary

jl_poly_fit(x, yvals[, deg, QR, robust_fit, ...])	Fast polynomial fitting
jl_poly(xvals, coeff[, dim_reorder, ...])	Evaluate polynomial

Robust Summary

biweightMean(inputData[, axis, dtype, iterMax])	Biweight Mean
checkfit(inputData, inputFit, epsilon, delta)	Determine the quality of a fit and biweights.
linefit(inputX, inputY[, iterMax, Bisector, ...])	Outlier resistance two-variable linear regression function.
mean(inputData[, Cut, axis, dtype, ...])	Robust Mean
medabsdev(data[, axis, keepdims, nan])	Median Absolute Deviation
mode(inputData[, axis, dtype])	Robust estimator of the mode of a data set using the half-sample mode.
polyfit(inputX, inputY, order[, iterMax])	Outlier resistance two-variable polynomial function fitter.
std(inputData[, Zero, axis, dtype, ...])	Robust Sigma

NIRCam Tools Summary

read_filter(filter[, pupil, mask, module, ...])	Read filter bandpass.
nrc_utils.psf_coeff	
gen_image_coeff(filter_or_bp[, pupil, mask, ...])	Generate PSF
bg_sensitivity(filter_or_bp[, pupil, mask, ...])	Sensitivity Estimates
sat_limit_webbpsf(filter_or_bp[, pupil, ...])	Saturation limits
pix_noise([ngroup, nf, nd2, tf, rn, ktc, ...])	Noise per pixel
channel_select(bp)	Select wavelength channel
grism_res([pupil, module, m])	Grism information

Spectral Tools Summary

stellar_spectrum(sptype, *renorm_args, **kwargs)	Stellar spectrum
BOSZ_spectrum(Teff, metallicity, log_g[, ...])	BOSZ stellar atmospheres (Bohlin et al 2017).
planets_sb12([atmo, mass, age, entropy, ...])	Exoplanet spectrum from Spiegel & Burrows (2012)
sp_accr(mmdot[, rin, dist, truncated, ...])	Exoplanet accretion flux values (Zhu et al., 2015).
zodi_spec([zfact, ra, dec, thisday])	Zodiacal light spectrum.
zodi_euclid(locstr, year, day[, ...])	IPAC Euclid Background Model
bp_2mass(filter)	2MASS Bandpass
bin_spectrum(sp, wave[, waveunits])	Rebin spectrum

Speckle Noise Summary

OPD_extract(opd, header[, seg_terms, verbose])	Decompose OPD to Zernike/Hexike coefficients
opd_extract_mp(opds, header[, nproc])	Generate multiple OPD_extract objects
opd_sci_gen(opd)	OPD image for science observation
opd_sci_gen_mp(opds_all[, nproc])	Multiple OPD images for science observations
opd_ref_gen(args[, verbose])	Generate a drifted OPD image
ODP_drift_all(wfe_drift, opds_all, ...)	Drift list of OPDs
get_psf(opd, header[, filter, mask, pupil, ...])	Generate PSF
gen_psf_ref_all(opd_ref_list_all, ...[, verbose])	Calculate a series of reference PSFs
get_contrast(speckle_noise_image, planet_psf)	Output single contrast curve
residual_speckle_image(psf_star_all, psf_ref_all)	Create residual speckle images
speckle_noise_image(psf_star_all, psf_ref_all)	Create a speckle noise image map
read_opd_file(opd_file[, opd_path, header])	Read in the OPD file data and header (optional)
read_opd_slice(pupilopd[, opd_path, header])	Get only a specified OPD slice from file.

1.9 Revision History

1.9.1 v0.8.0beta (ongoing)

- Release to work with WebbPSF 0.8.0.
- Phasing out support for Python 2
- Add info on saturation limits in terms of surface brightness
- Include option to create grism 2nd order
- Detector pixel timing bugs
- Field-dependent WFE extrapolated beyond FoV for better sampling diversity
- Included field-dependent WFE for coronagraphy
- Added wavelength dispersion of LW coronagraphic PSF

1.9.2 v0.7.0 (Jun 2018)

- Did not make it out of development before WebbPSF 0.8.0 release.
- Works with WebbPSF 0.7.0.
  - Field-dependent WFE
  - Image plane distortions
- Implemented `jwst_backgrounds` (not required)

1.9.3 v0.6.5 (Mar 2018)

- Fixed a critical bug where the off-axis PSF size was incorrect when performing WFE drift calculations.

1.9.4 v0.6.4 (Mar 2018)

- Off-axis PSFs now get drifted in the same way as their on-axis counterparts.
- Created an intermediate *nrc_hci* class to enable offsets of WFE drifted PSFs.

1.9.5 v0.6.3 (Mar 2018)

- First PyPI release.
- Effectively the same as 0.6.2, but better documentation of packaging and distributing.

1.9.6 v0.6.2 (Mar 2018)

- Implemented coronagraphic wedges, including arbitrary offsets along bar
- Renamed `obs_coronagraphy` to *obs_hci*
  - Faster modeling of off-axis PSFs
  - Include coronagraphic features (e.g.: ND squares) in slope images
  - Roll subtracted images include option to use Roll1-Roll2
  - Fixed bug that was slowing down PSF convolution of disks
- Can now generate docs directly from Jupyter notebooks using nbsphinx extension
- Coronagraphic tutorials for docs
- Create the `source_spectrum` class to fit spectra to observed photometry.

1.9.7 v0.6.0 (Dec 2017)

- Support for Python 3 (mostly `map`, `dict`, and index fixes)
- Updated code comments for `sphinx` and `readthedocs` documentation
- Create `setup.py` install file
- Modify grism PSF shapes due to aperture shape
- Detector frames times based on ASIC microcode build 10
- Headers for DMS data
- Three major changes to PSF coefficients
  - coefficients based on module (SWA, SWB, LWA, LWB), rather than filter
  - WFE drift coefficient relations
  - field-dependent coefficient relation

1.9.8 v0.5.0 (Feb 2017)

- Initial GitHub release
- Match version numbering to `WebbPSF` equivalent
- ND Acquisition mode
- Ramp settings optimizer
- Can now simulate ramps with detector noise
- Query Euclid's IPAC server for time/position-dependent Zodiacal emission
- Added example Jupyter notebooks

1.9.9 v0.1.2 (Jan 2017)

- Observations subclass for coronagraphs and direct imaging

1.9.10 v0.1.1 (Sep 2016)

- Add support for LW slitless grism
- Add support for extended sources

1.9.11 v0.1.0 (Aug 2016)

- Rewrite of `SimNRC` and rename `pynrc`
- Object oriented `multiaccum`, `DetectorOps`, and `NIRCam` classes
- Create separate detector instances in `NIRCam` class

1.9.12 Planned Updates

FoV aware positions

- Correct coronagraph field locations depending on Lyot optical wedge
- Background roll off at grism edges
- Filter location relative offsets
- SIAF info
- Subarray positions
- SCA Gaps and Module gaps

Detector updates in ngNRC.py

- Pixel non-linearity
- Intrapixel Capacitance (IPC)
- Post-pixel Coupling (PPC) due to ADC "smearing"
- Pixel glow based on subarray size
- Charge diffusion (esp for saturated pixels)
- Persistence/latent image
- Optical distortions
- QE variations across a pixel's surface
- RTN Noise
- Flat field variations

PSF Related

- Coronagraph target acquisition uncertainties
- Small grid dither strategy in coronagraphic observations
- Actual coronagraphic throughput from FITS files
- More PSF Jitter options
- PSF convolution based on geometric spot size

Observation Classes

- Photometric time series (incl. weak lens)
- Grism time series
- Wide-field grism
- Wide field imaging (esp. SW modules)

Miscellaneous

- Random cosmic ray hits in exposure simulator
- Ramp optimizer warning for large number of group loops?
- multi-thread ramp optimizer?
- DHS mode

1.10 License

1.11 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

1.11.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/JarronL/pynrc/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

pyNRC could always use more documentation, whether as part of the official pyNRC docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/JarronL/pynrc/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

1.11.2 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/JarronL/pynrc/pull_requests and make sure that the tests pass for all supported Python versions.

1.11.3 Tips

To run a subset of tests:

```
$ py.test tests.test_pynrc
```

1.11.4 Deploying

A reminder for the maintainers on how to deploy. First, make sure the following packages are installed:

```
$ pip install sphinx_automodapi
$ conda install sphinx_rtd_theme
$ conda install nbsphinx
$ conda install twine
```

1. Update version info in pynrc.version. Add entries to HISTORY.rst. Make sure all your changes are committed to git.
2. Generate documentation locally:

   ```
   $ make docs
   ```

3. Push all updates to github and make sure readthedocs generates correctly before actually submitting the release.
4. Package a distribution and test upload the release to TestPyPI:

   ```
   $ make release-test
   ```

5. If everything works without a hitch, then upload the release to PyPI:

   ```
   $ make release
   ```

   This command also tags the release on github. Make sure to have the command line token handy to enter as the requested password. Double-check *stable* release of readthedocs.

Todo:

6. Release code to conda-forge If you already have a conda-forge feedstock forked to your own GitHub account, first edit *recipe/meta.yaml* to update the version, hash, etc. To calculate the sha256 hash, run:

   `shell openssl dgst -sha256 path/to/package_name-0.1.1.tar.gz `

   Then, commit and push the yaml file to GitHub:

   `shell git pull upstream master git add --all git commit -m 'version bump to v0.1.1' git push -u origin master `

   Finally, issue a pull request to conda-forge.

---

7. At end of all this, double-check the build environments at https://readthedocs.org/projects/pynrc/builds/. For whatever reason, it is common for there to be an OSError and the build to fail. Resetting the environment at https://readthedocs.org/projects/pynrc/versions/ tends to fix this issue. Build times take about 5 minutes.

CHAPTER

TWO

LICENSE & ATTRIBUTION

pyNRC is free software made available under the MIT License. For details see *LICENSE*.

Attention: Citing pyNRC

If you make use of pyNRC in your work, please cite the following paper: *Leisenring et al., "pyNRC: A NIRCam ETC and Simulation Toolset"* (in prep).

INDEX

T

U

V

W

X

Z