pyNRC Documentation

Release 1.0.4

Jarron Leisenring

Jan 08, 2022

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 compatible with the JWST pipeline. This package works for a variety of NIRCam observing modes including direct imaging, coronagraphic imaging, slitless grism spectroscopy, and weak lens imaging. All PSFs are generated via WebbPSF and WebbPSF Extensions to reproduce realistic JWST images and spectra.

Developed by Jarron Leisenring and contributors at University of Arizona (2015 - 2021).

1.1 Overview

1.1.1 A JWST NIRCam ETC and Simulator

Authors: Jarron Leisenring (U. of Arizona, Steward Observatory)

Contributors: Everett Schlawin, Jonathan Fraine, Jonathan Aguilar

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 compatible with the JWST pipeline.

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 by WebbPSF as implemented by the WebbPSF Extensions package to reproduce realistic JWST images and spectra.

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

Similar to some of its dependencies, pyNRC requires 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.

Warning: 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 in flight. Check out JDocs for more information.

1.1.2 Simulating PSFs

Simulating PSFs with WebbPSF can become computationally expensive if generating new ones on the fly, especially considering JWST PSFs vary with respect to wavelength, field position, and time-dependent wavefront error drift. In an effort to speed up this process, pyNRC uses WebbPSF to generate a series of monochromatic PSF simulations, then produces polynomial fits to each oversampled pixel. Storing the coefficients rather than a library of PSFS allows for quick creation (via matrix multiplication) of PSF images for an arbitrary number of wavelengths (subject to hardware memory limitations, of course). The applications range from quickly creating PSFs for many different stellar types over wide bandpasses to generating a large number of monochromatic PSFs for spectral dispersion.

In addition, each science instrument PSF is dependent on the detector position due to field-dependent wavefront errors. Such changes are tracked in WebbPSF, but it becomes burdensome to generate new PSFs from scratch at each location, especially for large starfields. Instead, these changes can be stored by the fitting the residuals of the PSF coefficients across an instrument's field of view, then interpolating for an arbitrary location. A similar scheme can be achieved for coronagraphic occulters, where the PSF changes as the source position moves with respect to the mask.

JWST's thermal evolution (e.g., changing the angle of the sunshield after slewing to a new target) causes small but significant distortions to the telescope backplane. WebbPSF has tools to modify OPDs, but high-fidelity simulations take time to calculate. Since the change to the PSF coefficients varies smoothly with respect to WFE drift components, it's simple to parameterize the coefficient residuals in a fashion similar to the field-dependence.

1.2 Basic Installation

1.2.1 Requirements

pyNRC requires Python 3.7+ along with the following packages:

- Recent version of Numpy, Scipy, and matplotlib
- Astropy 4.2+
- Astroquery 0.4.3+
- pysynphot 2.0.0+
- WebbPSF 1.0.0+
- WebbPSF Extensions 1.0.4+
- JWST Pipeline 1.3+

Recommended Python packages:

- jwst_backgrounds 1.1.2+
- psutil Library to retrieve information on system utilization and profiling
- tqdm Progress bar for for loops

1.2.2 Installing with pip

You can install the `pynrc` package through pip:

```
$ pip install pynrc
```

If you want to make sure that none of your existing dependencies get upgraded, instead you can do (assuming all dependencies are met!):

```
$ pip install pynrc --no-deps
```

1.2.3 Installing with conda

Todo: Not yet implemented

pyNRC can be installed with conda if you have installed Anaconda or Miniconda. To install pyNRC using the conda-forge Anaconda channel, simply add `-c conda-forge` to the install command:

```
$ conda install -c conda-forge pynrc
```

1.2.4 Installing from source

To get the most up to date version of `pynrc`, install directly from source, though stability is not guaranteed. 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

The above commands only installs the program code. You still must download and install the data files.

Files containing information such as the instrument throughputs, stellar models, and exoplanet models are already distributed through `webbpsf_ext`. In addition, `pynrc` requires a number of files to simulate realistic detector data with DMS-like formatting and headers. In general, these are not necessary to run `pynrc` and use its ETC capabilities and simple simulations. But, in order to create DMS and pipeline-compliant data, you must download these files and define the `PYNRC_PATH` environment variable.

1. Download the following file: pynrc_data_all_v1.0.0.tar [approx. 17.0 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 should now be able to successfully `import pynrc` in a Python session.

1.2.6 Testing

Todo: Not yet implemented

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 with new Conda Environment

This installation tutorial assumes a clean installation with Anaconda via:

```
$ conda create -n py39 python=3.9 anaconda
```

and has been verified on Python 3.9 using the following modules:

- Numpy 1.20
- Matplotlib 3.5
- Scipy 1.7
- Astropy 5.0
- Astroquery 0.4.3

1.3.1 Configure Conda Channels

We will first install a few packages that live in the AstroConda and Conda-Forge channels. If you're already working in an AstroConda environment, then you should be all set and can probably skip most of these steps and jump to *Installing WebbPSF Extensions*.

If you have some other Conda installation, such as indicated above, then you can simply add the AstroConda and Conda-Forge channels to your `.condarc` file, which appends the appropriate URL to Conda's channel search path:

```
# Writes changes to ~/.condarc
$ conda config --append channels https://ssb.stsci.edu/astroconda
$ conda config --append channels conda-forge
```

Now your `.condarc` file should look something like the following:

```
channels:
  - defaults
  - https://ssb.stsci.edu/astroconda
  - conda-forge
```

1.3.2 Installing Pysynphot

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

```
$ conda install pysynphot
```

or from PyPi:

```
$ pip install pysynphot
```

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 easiest way to install WebbPSF without inducing package conflicts is to install some of its main dependencies, then WebbPSF using the `--no-deps` flag. In this particular example, we use a combination of `conda` and `pip`, because of minor issues installing `photutils` dependencies.

```
$ conda install photutils
$ pip install pysiaf poppy
$ pip install webbpsf --no-deps
```

Note: The `synphot` package has been ignored in this case, because `pynrc` currently uses the slightly older `pysynphot` package. For details on installing `synphot` as well as other installation methods, see the WebbPSF documentation. Configuring `pynrc` to use `synphot` is under development.

WebbPSF Data Files

You will also need to download and install WebbPSF data files: webbpsf-data-1.0.0.tar.gz [approx. 280 MB]. Follow the same procedure as with the Pysynphot data files, setting the `WEBBPSF_PATH` environment variable to point towards your `webbpsf-data` directory.

Matplotlib 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. The latter option is probably preferred for most cases.

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 STScI, requiring an internet connection to access. However, `pynrc` comes with a simpler background estimator in the event `jwst_background` is not installed or no functioning internet. In this sense, `jwst_backgrounds` is not a strict requirement for running `pynrc`.

This module requires `healpy` to run:

```
$ conda install healpy
```

Then install JWST Backgrounds with pip:

```
$ pip install jwst_backgrounds
```

1.3.5 Installing Astroquery

Astroquery is a set of tools for querying astronomical web forms and databases. It is used within `pynrc` to query Simbad and Gaia databases to search for sources and obtain basic astrometry, fluxes, and spectral types.

Install via `conda`:

```
$ conda install astroquery
```

1.3.6 Installing JWST Pipeline

In order to create DMS-like datasets, pyNRC uses data models from the JWST pipeline (https://github.com/spacetelescope/jwst). Again, easiest to install via `pip`:

```
$ pip install jwst
```

The JWST pipeline is under significant development, so it's a good idea to keep this up-to-date with new releases by regularly running:

```
$ pip install jwst --upgrade
```

CRDS Data Files

Configure the calibration reference database (CRDS) by defining the CRDS directory that will store downloaded cal files. For example, in `.bashrc` shell file:

```
export CRDS_PATH='$HOME/data/crds_cache/'
export CRDS_SERVER_URL='https://jwst-crds.stsci.edu'
```

1.3.7 Installing WebbPSF Extensions

The `webbpsf_ext` package calculates and stores polynomial relationships between PSFs with respect to wavelength, focal plane position, and WFE drift in order to quickly generate arbitrary NIRCam PSFs without having to simulate a new PSF on the fly.

```
pip install webbpsf_ext
```

Set the environment variable `WEBBPSF_EXT_PATH` to point to some data directory. All PSF coefficients will be saved here as they are generated to be reused later. For example, in `.bashrc` shell file, add:

```
export WEBBPSF_EXT_PATH='$HOME/data/webbpsf_ext_data/'
```

1.3.8 Installing pyNRC

Finally, we are ready to install `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 guaranteed. 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 .
```

This creates an editable installation, which is great for helping to develop the code, create bug reports, pull requests to GitHub, etc. Make sure to switch to the `develop` branch after installation in order to get access to the latest code base.

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`.

Files containing information such as the instrument throughputs, stellar models, and exoplanet models are already distributed through `webbpsf_ext`. In addition, `pynrc` requires a number of files to simulate realistic detector data with DMS-like formatting and headers. In general, these are not necessary to run `pynrc`'s ETC capabilities and simple simulations. But, in order to create DMS and pipeline-compliant data, you must download these files and define the `PYNRC_PATH` environment variable.

1. Download the following file: pynrc_data_all_v1.0.0.tar [approx. 17.0 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 should now be able to successfully `import pynrc` in a Python session.

1.3.9 Environment Variables

In the end, you should have a number of environment variables in your `.bashrc` (or equivalnet):

```
export CRDS_PATH='$HOME/data/crds_cache/'
export CRDS_SERVER_URL='https://jwst-crds.stsci.edu'
export PYSYN_CDBS='$HOME/data/cdbs/'
export WEBBPSF_PATH='$HOME/data/webbpsf-data/'
export WEBBPSF_EXT_PATH='$HOME/data/webbpsf_ext_data/'
export PYNRC_DATA='$HOME/data/pynrc_data/'
```

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

# Enable inline plotting at lower left
%matplotlib inline
```

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
```

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', fov_pix=33, oversample=4)
print('\nFilter: {};  Pupil Mask: {};  Image Mask: {};  Module: {}'\
      .format(nrc.filter, nrc.pupil_mask, nrc.image_mask, nrc.module))
```

```
[   webbpsf:INFO] NIRCam aperture name updated to NRCA1_FULL
[   webbpsf:INFO] NIRCam pixel scale switched to 0.063000 arcsec/pixel for the long wave␣
↪channel.
[   webbpsf:INFO] NIRCam aperture name updated to NRCA5_FULL
[     pynrc:INFO] RAPID readout mode selected.
[     pynrc:INFO] Setting ngroup=1, nf=1, nd1=0, nd2=0, nd3=0.
[     pynrc:INFO] Initializing SCA 485/A5
[     pynrc:INFO] Suggested SIAF aperture name: NRCA5_FULL
[     pynrc:INFO] RAPID readout mode selected.
[     pynrc:INFO] Setting ngroup=1, nf=1, nd1=0, nd2=0, nd3=0.
[     pynrc:INFO] Initializing SCA 485/A5
[webbpsf_ext:INFO] Loading /Users/jarron/NIRCam/webbpsf_ext_data/psf_coeffs/NIRCam/LWA_
↪F430M_CLEAR_NONE_pix33_os4_jsig5_r0.00_th+0.0_RevAAslice0_siwfe_distort_legendre.fits
[     pynrc:INFO] Suggested SIAF aperture name: NRCA5_FULL

Filter: F430M;  Pupil Mask: None;  Image Mask: None;  Module: A
```

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

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

```
Detector Info Keywords:
{'wind_mode': 'FULL', 'nout': 4, 'xpix': 2048, 'ypix': 2048, 'x0': 0, 'y0': 0, 'read_mode
↪': 'RAPID', 'nint': 1, 'ngroup': 1, 'nf': 1, 'nd1': 0, 'nd2': 0, 'nd3': 0}
```

PSF settings are stored and modified in the same manner as WebbPSF (https://webbpsf.readthedocs.io/en/stable/usage.html) with a couple minor modifications: 1. The calculated field of view is specified as `nrc.fov_pix` along with the `nrc.oversample` attributes. 2. In addition, you can turn on/off distortions via the `nrc.include_distortions` attribute. 3. The primary method for calculating PSFs has changed to `nrc.calc_psf_from_coeff`.

You can quickly obtain a brief overview of important settings through the `psf_info` dictionary property. These properties can also be updated via the `nrc.update_psf_coeff()` function, which immediately generates a new set of PSF coefficients.

[6]:
```
print('PSF Info:')
print(nrc.psf_info)
```

```
PSF Info:
{'fov_pix': 33, 'oversample': 4, 'npsf': 7, 'ndeg': 6, 'include_si_wfe': True, 'include_
↪distortions': True, 'jitter': 'gaussian', 'jitter_sigma': 0.005, 'offset_r': 0,
↪'offset_theta': 0, 'bar_offset': None, 'pupil': '/Users/jarron/NIRCam/webbpsf-data/
↪jwst_pupil_RevW_npix1024.fits.gz', 'pupilopd': ('JWST_OTE_OPD_RevAA_prelaunch_
↪predicted.fits.gz', 0)}
```

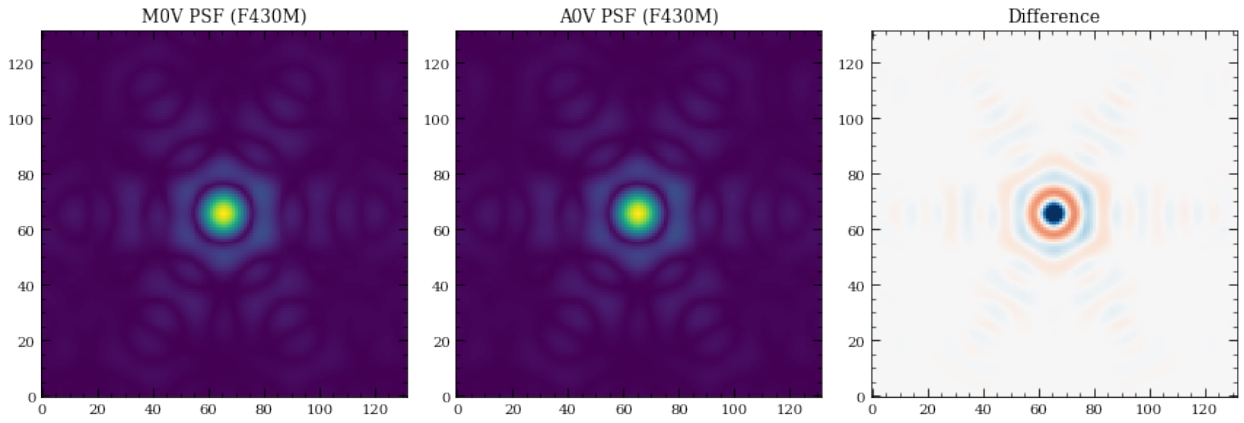
PSF coefficient information is stored in the `psf_coeff` attribute. An associated header file exists in the `psf_coeff_header`, showing all of the parameters used to generate that data. This data is accessed by many of the NIRCam class functions to generate PSFs with arbitrary wavelength weights, such as the `calc_psf_from_coeff()` function.

[7]:
```
# 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)
hdul_M0V = nrc.calc_psf_from_coeff(sp=sp_M0V, return_oversample=True)
hdul_A0V = nrc.calc_psf_from_coeff(sp=sp_A0V, return_oversample=True)

psf_M0V = hdul_M0V[0].data
psf_A0V = hdul_A0V[0].data

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')
```

(continues on next page)

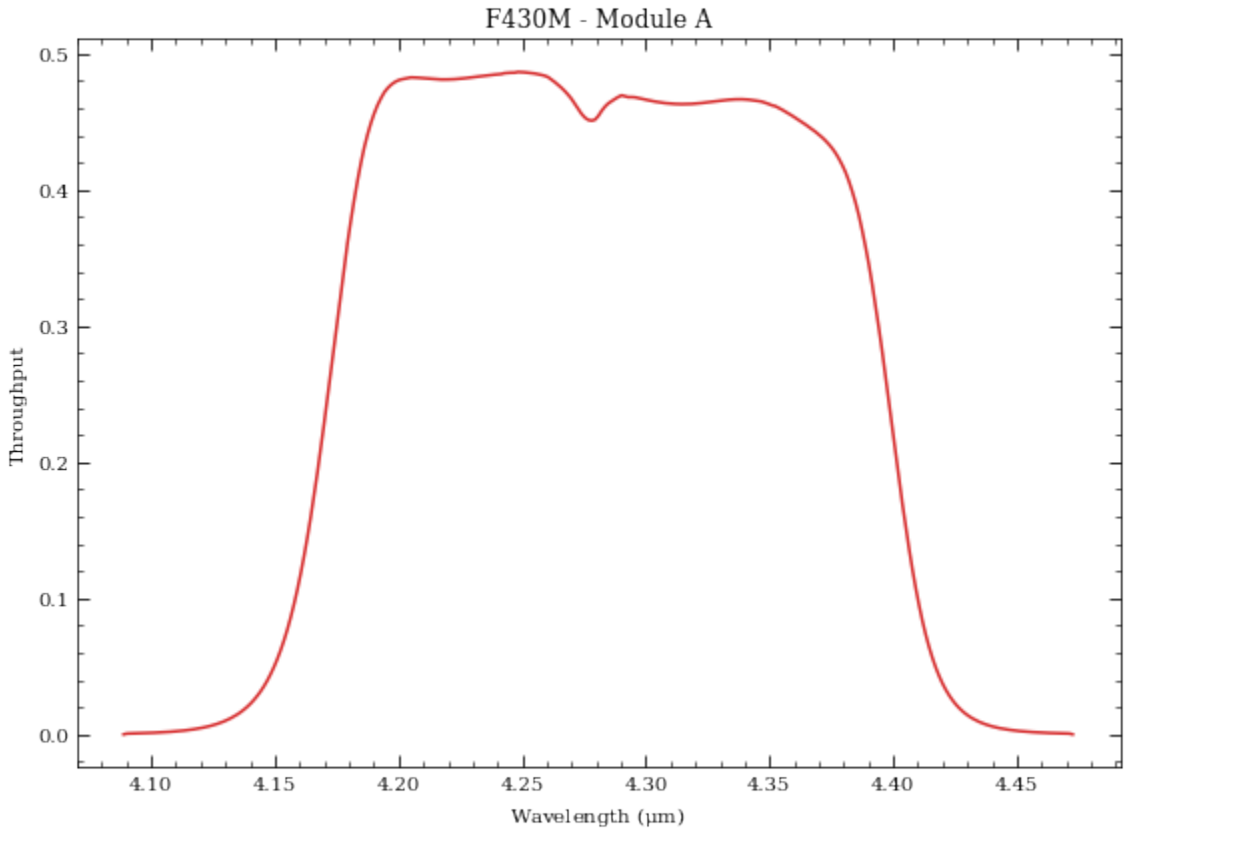
(continued from previous page)

```
fig.tight_layout()
```



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

[8]:
```
fig, ax = plt.subplots(1,1)
nrc.plot_bandpass(ax=ax, color='C3')

fig.tight_layout()
```



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, but can go as high as 95%.

[9]:
```
# 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_tot1 :   21.474
  t_int_tot2 :    0.000
  t_exp      :   21.474
  t_acq      :   21.479
```

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

[10]:
```
# 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 (point source): 12.18 vegamag
F430M Saturation Limit assuming G2V source (extended): 8.48 vegamag/arcsec^2

Dictionary Info: ({'satlim': 12.179917081677289, 'units': 'vegamag', 'bp_lim': 'F430M',
↪'Spectrum': 'G2V'}, {'satlim': 8.478633456035855, 'units': 'vegamag/arcsec^2', 'bp_lim
↪': 'F430M', 'Spectrum': 'G2V'})
```

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.

[11]:
```
# Spectrum of an M0V star (not normalized)
sp_M0V = pynrc.stellar_spectrum('M0V')
```

(continues on next page)

(continued from previous page)

```
# 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 (point source): 12.31 vegamag
Ks-Band Saturation Limit for F430M assuming M0V source (extended): 8.61 vegamag/arcsec^2
```

Now, let's get the same saturation limit assuming a 128x128 detector subarray (faster frame rate).

[12]:
```
nrc.update_detectors(wind_mode='WINDOW', xpix=128, ypix=128)
sat_lims = nrc.sat_limits(sp=sp_M0V, bp_lim=bp_k, verbose=True)
```

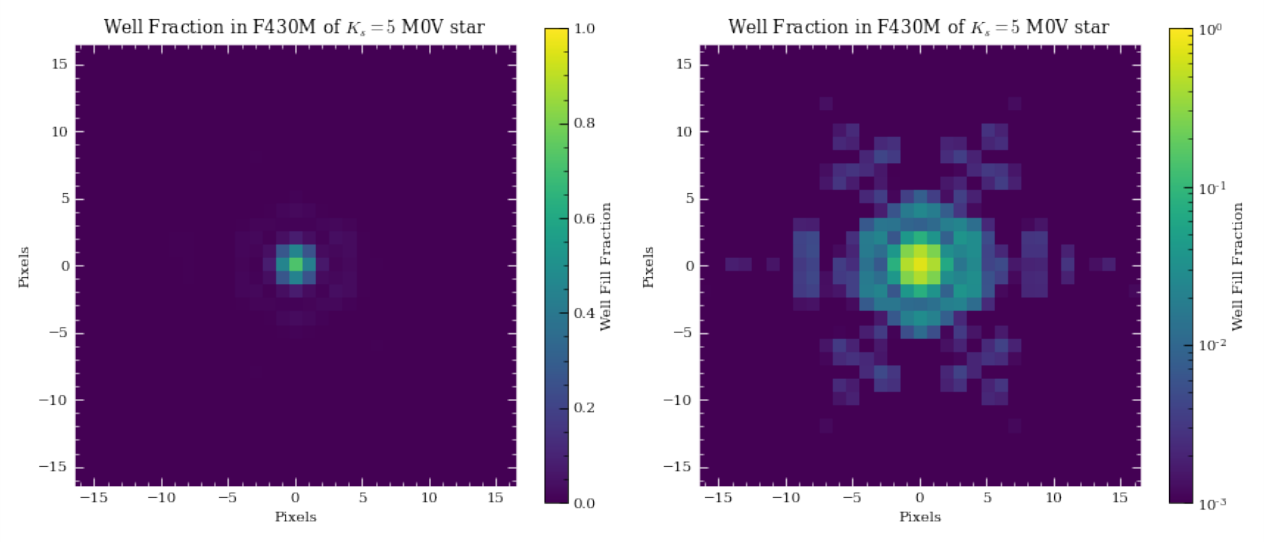
```
Ks-Band Saturation Limit for F430M assuming M0V source (point source): 7.89 vegamag
Ks-Band Saturation Limit for F430M assuming M0V source (extended): 4.18 vegamag/arcsec^2
```

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

[13]:
```
# 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: 0.72
```

[14]:
```
# Plot the well fill levels for each pixel
fig, axes = plt.subplots(1,2, figsize=(12,5))

for i,ax in enumerate(axes):
    extent = 0.5 * nrc.psf_info['fov_pix'] * np.array([-1,1,-1,1])
    if i==0:
        cax = ax.imshow(sat_levels, extent=extent, vmin=0, vmax=1)
    else:
        norm = matplotlib.colors.LogNorm(vmin=0.001, vmax=1)
        cax = ax.imshow(sat_levels, extent=extent, norm=norm)
    ax.set_xlabel('Pixels')
    ax.set_ylabel('Pixels')
    ax.set_title('Well Fraction in {} of $K_s = 5$ M0V star'.format(nrc.filter))

    cbar = fig.colorbar(cax, ax=ax)
    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')

fig.tight_layout()
```



Information for slitless grism observations show wavelength-dependent results.

[15]:
```
nrc = pynrc.NIRCam(filter='F444W', pupil_mask='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               4.44
     4.00               4.44
     4.10               4.33
     4.20               4.21
     4.30               4.06
     4.40               3.82
     4.50               3.64
     4.60               3.47
     4.70               3.30
     4.80               3.13
     4.90               2.92
     5.00               2.35
```

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.

[16]:
```
nrc = pynrc.NIRCam(filter='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': 1052.2087, 't_int_tot1': 1052.20346, 't_int_tot2': 0.0}
```

[17]:
```
sens = nrc.sensitivity(nsig=5, units='vegamag', verbose=True)
```

```
Point Source Sensitivity (5-sigma): 23.35 vegamag
Surface Brightness Sensitivity (5-sigma): 21.23 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.

[18]:
```
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): 77.08 sigma
Surface Brightness SNR (20.00 vegamag/arcsec^2): 14.59 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?

[19]:
```
# Setup observation
nrc = pynrc.NIRCam(filter='F430M', wind_mode='WINDOW', xpix=160, ypix=160)

# Spectrum of an M2V star
bp_k = pynrc.bp_2mass('K')
sp_M0V = pynrc.stellar_spectrum('M0V', 18, 'vegamag', bp_k)
```

[20]:
```
# Run optimizer. Result is a sorted by efficiency.
tbl = nrc.ramp_optimize(sp_M0V, 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
---------- ---- ---- ---------- ---------- ---------- -------- -------- --------
DEEP8         5   10      24.52     245.20     248.04    115.7    0.005    7.344
MEDIUM8       7   11      18.95     208.42     211.54    101.1    0.004    6.949
```

(continues on next page)

(continued from previous page)

```
MEDIUM8       7   12     18.95    227.37    230.78    105.6    0.004    6.949
MEDIUM2       9   10     22.85    228.48    231.32    101.7    0.005    6.687
MEDIUM2       9   11     22.85    251.33    254.46    106.7    0.005    6.687
SHALLOW4     10   18     13.65    245.76    250.87    100.3    0.003    6.334
SHALLOW4     10   19     13.65    259.41    264.81    103.1    0.003    6.334
DEEP2         5   11     22.85    251.33    254.46     99.2    0.005    6.218
...         ...  ...       ...       ...       ...      ...      ...      ...
RAPID        10  634      2.79   1766.58   1946.51    101.5    0.001    2.300
RAPID        10  635      2.79   1769.36   1949.58    101.6    0.001    2.300
RAPID        10  636      2.79   1772.15   1952.65    101.7    0.001    2.300
RAPID        10  637      2.79   1774.94   1955.72    101.7    0.001    2.300
RAPID        10  638      2.79   1777.72   1958.79    101.8    0.001    2.300
RAPID        10  639      2.79   1780.51   1961.86    101.9    0.001    2.300
RAPID        10  640      2.79   1783.30   1964.93    102.0    0.001    2.300
RAPID        10  641      2.79   1786.08   1968.00    102.1    0.001    2.300
Length = 55 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).

[21]:
```
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,
                        well_frac_max=0.9, 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  615      2.79   1713.64   1888.17    100.0    0.778    2.300
RAPID        10  616      2.79   1716.42   1891.24    100.0    0.778    2.300
RAPID        10  617      2.79   1719.21   1894.31    100.1    0.778    2.300
RAPID        10  618      2.79   1722.00   1897.38    100.2    0.778    2.300
RAPID        10  619      2.79   1724.78   1900.45    100.3    0.778    2.300
RAPID        10  620      2.79   1727.57   1903.52    100.4    0.778    2.300
RAPID        10  621      2.79   1730.35   1906.59    100.5    0.778    2.300
RAPID        10  622      2.79   1733.14   1909.66    100.5    0.778    2.300
...         ...  ...       ...       ...       ...      ...      ...      ...
BRIGHT1       6  699      3.07   2142.46   2340.84    101.6    0.856    2.099
BRIGHT1       6  700      3.07   2145.53   2344.19    101.6    0.856    2.099
BRIGHT1       6  701      3.07   2148.59   2347.54    101.7    0.856    2.099
BRIGHT1       6  702      3.07   2151.66   2350.89    101.8    0.856    2.099
BRIGHT1       6  703      3.07   2154.72   2354.23    101.8    0.856    2.099
BRIGHT1       6  704      3.07   2157.79   2357.58    101.9    0.856    2.099
BRIGHT1       6  705      3.07   2160.85   2360.93    102.0    0.856    2.099
BRIGHT1       6  706      3.07   2163.92   2364.28    102.1    0.856    2.099
```

(continues on next page)

(continued from previous page)

```
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 much more efficient exposure settings.

[22]:
```
tbl = nrc.ramp_optimize(sp_M0V, sp_bright=sp_F0V, snr_goal=100, ng_min=5, nint_min=10,
                        well_frac_max=5, 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   14     16.16    226.26    230.23    100.1    4.511    6.597
MEDIUM8       6   15     16.16    242.42    246.67    103.6    4.511    6.597
SHALLOW4     10   18     13.65    245.76    250.87    100.3    3.811    6.334
SHALLOW4     10   19     13.65    259.41    264.81    103.1    3.811    6.334
MEDIUM8       5   21     13.37    280.87    286.83    103.9    3.733    6.135
MEDIUM2       7   16     17.28    276.41    280.95    100.1    4.822    5.971
MEDIUM2       7   17     17.28    293.69    298.51    103.2    4.822    5.971
SHALLOW2     10   22     13.10    288.11    294.36     98.8    3.656    5.759
...         ...  ...       ...       ...       ...      ...      ...      ...
RAPID        10  634      2.79   1766.58   1946.51    101.5    0.778    2.300
RAPID        10  635      2.79   1769.36   1949.58    101.6    0.778    2.300
RAPID        10  636      2.79   1772.15   1952.65    101.7    0.778    2.300
RAPID        10  637      2.79   1774.94   1955.72    101.7    0.778    2.300
RAPID        10  638      2.79   1777.72   1958.79    101.8    0.778    2.300
RAPID        10  639      2.79   1780.51   1961.86    101.9    0.778    2.300
RAPID        10  640      2.79   1783.30   1964.93    102.0    0.778    2.300
RAPID        10  641      2.79   1786.08   1968.00    102.1    0.778    2.300
Length = 54 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]:
```
# Import the usual libraries
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

# Enable inline plotting at lower left
%matplotlib inline
```

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

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

from astropy.table import Table

# Progress bar
from tqdm.auto import tqdm, trange
```

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(filter='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
```

(continues on next page)

(continued from previous page)

```
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     99.3    0.628    8.277
DEEP8         7    5     35.67    178.33    179.75    111.0    0.628    8.277
DEEP2         8    4     39.57    158.27    159.40     98.5    0.696    7.804
DEEP2         8    5     39.57    197.83    199.25    110.2    0.696    7.804
DEEP8         5    7     24.52    171.64    173.63    101.4    0.432    7.697
MEDIUM8       8    8     21.73    173.87    176.14    100.5    0.383    7.575
MEDIUM8       8    9     21.73    195.61    198.16    106.6    0.383    7.575
MEDIUM2      10    7     25.63    179.44    181.43     98.6    0.451    7.322
...         ...  ...       ...       ...       ...      ...      ...      ...
RAPID        10  555      2.79   1546.45   1703.96    101.5    0.049    2.458
RAPID        10  556      2.79   1549.24   1707.03    101.6    0.049    2.458
RAPID        10  557      2.79   1552.02   1710.10    101.7    0.049    2.458
RAPID        10  558      2.79   1554.81   1713.17    101.7    0.049    2.458
RAPID        10  559      2.79   1557.60   1716.24    101.8    0.049    2.458
RAPID        10  560      2.79   1560.38   1719.31    101.9    0.049    2.458
RAPID        10  561      2.79   1563.17   1722.38    102.0    0.049    2.458
RAPID         8 1046      2.23   2331.66   2628.51    102.0    0.039    1.990
Length = 55 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  539      2.79   1501.87   1654.84    100.0    0.049    2.458
RAPID        10  540      2.79   1504.66   1657.91    100.1    0.049    2.458
BRIGHT1      10  145      5.29    767.65    808.80     99.8    0.093    3.509
BRIGHT1      10  146      5.29    772.95    814.38    100.1    0.093    3.509
BRIGHT2      10   93      5.57    518.27    544.66     99.7    0.098    4.270
BRIGHT2      10   94      5.57    523.84    550.52    100.2    0.098    4.270
SHALLOW2     10   20     13.10    261.92    267.60     99.6    0.230    6.089
SHALLOW2     10   21     13.10    275.02    280.98    102.1    0.230    6.089
SHALLOW4     10   16     13.65    218.45    222.99     99.7    0.240    6.675
SHALLOW4     10   17     13.65    232.11    236.93    102.8    0.240    6.675
MEDIUM2      10    7     25.63    179.44    181.43     98.6    0.451    7.322
MEDIUM2      10    8     25.63    205.08    207.35    105.4    0.451    7.322
MEDIUM8       8    8     21.73    173.87    176.14    100.5    0.383    7.575
MEDIUM8       8    9     21.73    195.61    198.16    106.6    0.383    7.575
DEEP2         8    4     39.57    158.27    159.40     98.5    0.696    7.804
DEEP2         8    5     39.57    197.83    199.25    110.2    0.696    7.804
DEEP8         7    4     35.67    142.66    143.80     99.3    0.628    8.277
DEEP8         7    5     35.67    178.33    179.75    111.0    0.628    8.277
```

[7]:
```
# Do the same thing, but for coronagraphic mask instead
nrc = pynrc.NIRCam(filter='F430M', image_mask='MASK430R', pupil_mask='CIRCLYOT',
```

(continues on next page)

(continued from previous page)

```
                    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   94    104.77   9848.00   9950.36     99.6    0.001    0.998
MEDIUM8      10   96    104.77  10057.53  10162.07    100.7    0.001    0.998
DEEP8        20   14    414.79   5807.03   5822.27    104.6    0.003    1.370
DEEP8        19   14    393.41   5507.69   5522.94    101.2    0.003    1.362
```

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 when occulting the primary with the coronagraph masks. **Of course, this assumes the companion exists in the background limit as opposed to the contrast limit.**

1.5.2 Example 2: Exoplanet Coronagraphy

We want to observe GJ 504 for an hour in the F444W filter using the MASK430R coronagraph. - 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(filter='F444W', pupil_mask='CIRCLYOT', image_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
---------- ---- ---- --------- --------- --------- -------- -------- --------
DEEP2        17   10    344.23   3442.31   3453.20    672.8    0.777   11.448
DEEP2        16   10    322.85   3228.50   3239.39    651.5    0.728   11.446
DEEP2        16   12    322.85   3874.20   3887.27    713.7    0.728   11.446
DEEP2        15   12    301.47   3617.63   3630.70    689.4    0.680   11.440
DEEP2        14   12    280.09   3361.06   3374.13    664.0    0.632   11.431
DEEP8        17   10    350.65   3506.45   3517.34    676.2    0.791   11.402
DEEP8        16   10    329.26   3292.64   3303.53    655.3    0.743   11.401
DEEP8        15   12    307.88   3694.60   3707.67    694.1    0.695   11.398
...         ...  ...       ...       ...       ...      ...      ...      ...
BRIGHT2      10  162     21.38   3463.69   3640.10    518.1    0.048    8.587
BRIGHT1      10  166     20.31   3371.75   3552.52    463.1    0.046    7.768
BRIGHT1      10  168     20.31   3412.38   3595.32    465.8    0.046    7.768
BRIGHT1      10  170     20.31   3453.00   3638.12    468.6    0.046    7.768
RAPID        10  304     10.69   3249.88   3580.93    358.9    0.024    5.997
RAPID        10  306     10.69   3271.26   3604.48    360.1    0.024    5.997
RAPID        10  308     10.69   3292.64   3628.04    361.2    0.024    5.997
RAPID         9  334      9.62   3213.53   3577.25    331.8    0.022    5.548
Length = 36 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    358.9    0.024    5.997
RAPID        10  306     10.69   3271.26   3604.48    360.1    0.024    5.997
BRIGHT1      10  166     20.31   3371.75   3552.52    463.1    0.046    7.768
BRIGHT1      10  168     20.31   3412.38   3595.32    465.8    0.046    7.768
BRIGHT2      10  158     21.38   3378.17   3550.22    511.7    0.048    8.587
BRIGHT2      10  160     21.38   3420.93   3595.16    514.9    0.048    8.587
SHALLOW2     10   68     50.24   3416.65   3490.70    606.1    0.113   10.258
SHALLOW2     10   70     50.24   3517.14   3593.37    615.0    0.113   10.258
SHALLOW4     10   66     52.38   3457.28   3529.15    619.9    0.118   10.434
SHALLOW4     10   68     52.38   3562.04   3636.09    629.2    0.118   10.434
MEDIUM2      10   34     98.35   3343.96   3380.98    638.0    0.222   10.971
MEDIUM2      10   36     98.35   3540.66   3579.86    656.5    0.222   10.971
MEDIUM8      10   32    104.77   3352.51   3387.36    638.0    0.236   10.962
MEDIUM8      10   34    104.77   3562.04   3599.07    657.7    0.236   10.962
DEEP2        17   10    344.23   3442.31   3453.20    672.8    0.777   11.448
DEEP2        16   10    322.85   3228.50   3239.39    651.5    0.728   11.446
DEEP8        17   10    350.65   3506.45   3517.34    676.2    0.791   11.402
```

(continues on next page)

(continued from previous page)

```
DEEP8        16   10    329.26   3292.64   3303.53    655.3    0.743   11.401
```

[12]:
```
# The SHALLOWs, DEEPs, and MEDIUMs are very similar for SNR and efficiency.
# Let's go with SHALLOW2 for more GROUPS & INTS
# MEDIUM8 would be fine as well.
nrc.update_detectors(read_mode='SHALLOW2', ngroup=10, nint=70)

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

```
t_acq     :   3593.36880
t_exp     :   3517.14160
t_frame   :      1.06904
t_group   :      5.34520
t_int     :     50.24488
t_int_tot1:     51.33384
t_int_tot2:     51.33384
```

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

```
Point Source Sensitivity (5-sigma): 21.43 vegamag
Surface Brightness Sensitivity (5-sigma): 22.74 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 M0V dwarf. Each grism resolution element should have SNR~100.

[14]:
```
# M0V star normalized to K=12 mags
bp_k = S.ObsBandpass('k')
sp_M0V = pynrc.stellar_spectrum('M0V', 12, 'vegamag', bp_k)
```

[15]:
```
nrc = pynrc.NIRCam(filter='F444W', pupil_mask='GRISMR', wind_mode='STRIPE', ypix=128)
```

[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_M0V, snr_goal=100, nint_min=10, ng_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
```

(continues on next page)

(continued from previous page)

```
---------- ---- ---- --------- --------- --------- -------- -------- --------
DEEP2        10   10    123.03   1230.27   1237.08    332.4    0.071    9.449
DEEP8        10   10    127.08   1270.82   1277.64    336.0    0.074    9.399
MEDIUM2      10   10     62.19    621.89    628.71    231.2    0.036    9.219
MEDIUM8      10   10     66.25    662.45    669.27    236.1    0.038    9.127
SHALLOW4     10   10     33.12    331.23    338.04    161.8    0.019    8.802
SHALLOW2     10   10     31.77    317.71    324.52    157.6    0.018    8.746
BRIGHT2      10   12     13.52    162.23    170.41     98.8    0.008    7.567
BRIGHT2      10   13     13.52    175.75    184.61    102.8    0.008    7.567
BRIGHT1      10   15     12.84    192.65    202.87    100.0    0.007    7.021
BRIGHT1      10   16     12.84    205.49    216.40    103.3    0.007    7.021
RAPID        10   42      6.76    283.91    312.52     99.0    0.004    5.599
RAPID        10   43      6.76    290.67    319.96    100.2    0.004    5.599
RAPID        10   44      6.76    297.43    327.41    101.3    0.004    5.599
RAPID        10   45      6.76    304.19    334.85    102.5    0.004    5.599
```

[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   42      6.76    283.91    312.52     99.0    0.004    5.599
RAPID        10   43      6.76    290.67    319.96    100.2    0.004    5.599
BRIGHT1      10   15     12.84    192.65    202.87    100.0    0.007    7.021
BRIGHT1      10   16     12.84    205.49    216.40    103.3    0.007    7.021
BRIGHT2      10   12     13.52    162.23    170.41     98.8    0.008    7.567
BRIGHT2      10   13     13.52    175.75    184.61    102.8    0.008    7.567
SHALLOW2     10   10     31.77    317.71    324.52    157.6    0.018    8.746
SHALLOW4     10   10     33.12    331.23    338.04    161.8    0.019    8.802
MEDIUM2      10   10     62.19    621.89    628.71    231.2    0.036    9.219
MEDIUM8      10   10     66.25    662.45    669.27    236.1    0.038    9.127
DEEP2        10   10    123.03   1230.27   1237.08    332.4    0.071    9.449
DEEP8        10   10    127.08   1270.82   1277.64    336.0    0.074    9.399
```

[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     :    338.04264
t_exp     :    331.22530
t_frame   :      0.67597
t_group   :      3.37985
t_int     :     33.12253
t_int_tot1:     33.80374
t_int_tot2:     33.80374
```

[19]:
```
# Print final wavelength-dependent SNR
# For spectroscopy, the snr_goal is the median over the bandpass
snr_dict = nrc.sensitivity(sp=sp_M0V, forwardSNR=True, units='mJy', verbose=True)
```

```
F444W SNR for M0V source
   Wave      SNR    Flux (mJy)
--------- --------- ----------
     3.80      9.52       4.54
     3.90    230.66       4.34
     4.00    246.18       4.15
     4.10    235.68       3.97
     4.20    221.71       3.73
     4.30    197.92       3.26
     4.40    184.55       3.19
     4.50    169.03       2.85
     4.60    154.66       2.79
     4.70    139.31       2.63
     4.80    131.21       2.70
     4.90    115.05       2.60
     5.00     57.52       2.38
     5.10      1.30       2.46
```

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.calc_psf_from_coeff(sp=sp_M0V, return_hdul=False, return_
↪oversample=False)

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

# Shrink/expand nx (fill value of 0)
# Then shrink to a size excluding wspec=0
# This assumes simulated spectrum is centered
imspec = nrc_utils.pad_or_cut_to_size(imspec, (ny,nx))
wspec = nrc_utils.pad_or_cut_to_size(wspec, nx)
```

[21]:
```
# Add simple zodiacal background
im_slope = imspec + nrc.bg_zodi()
```

[22]:
```
# Create a series of ramp integrations based on the current NIRCam settings
# Output is a single HDUList with 10 INTs
# Ignore detector non-linearity to return output in e-/sec
kwargs = {
        'apply_nonlinearity' : False,
        'apply_flats'        : False,
}
res = nrc.simulate_level1b('M0V Target', 0, 0, '2023-01-01', '12:00:00',
                           im_slope=im_slope, return_hdul=True, **kwargs)
```

```
PPC Amps:   0%|          | 0/4 [00:00<?, ?it/s]
Frames:   0%|          | 0/1 [00:00<?, ?it/s]
Frames:   0%|          | 0/1 [00:00<?, ?it/s]
Frames:   0%|          | 0/1 [00:00<?, ?it/s]
Frames:   0%|          | 0/1 [00:00<?, ?it/s]
PPC Amps:   0%|          | 0/4 [00:00<?, ?it/s]
Frames:   0%|          | 0/1 [00:00<?, ?it/s]
Frames:   0%|          | 0/1 [00:00<?, ?it/s]
Frames:   0%|          | 0/1 [00:00<?, ?it/s]
Frames:   0%|          | 0/1 [00:00<?, ?it/s]
Frames:   0%|          | 0/1 [00:00<?, ?it/s]
Frames:   0%|          | 0/1 [00:00<?, ?it/s]
Ramps:   0%|          | 0/10 [00:00<?, ?it/s]
  0%|          | 0/14 [00:00<?, ?it/s]
Uncorr 1/f:   0%|          | 0/4 [00:00<?, ?it/s]
ACN:   0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/14 [00:00<?, ?it/s]
Uncorr 1/f:   0%|          | 0/4 [00:00<?, ?it/s]
ACN:   0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/14 [00:00<?, ?it/s]
Uncorr 1/f:   0%|          | 0/4 [00:00<?, ?it/s]
ACN:   0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/14 [00:00<?, ?it/s]
Uncorr 1/f:   0%|          | 0/4 [00:00<?, ?it/s]
ACN:   0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/14 [00:00<?, ?it/s]
Uncorr 1/f:   0%|          | 0/4 [00:00<?, ?it/s]
ACN:   0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/14 [00:00<?, ?it/s]
Uncorr 1/f:   0%|          | 0/4 [00:00<?, ?it/s]
ACN:   0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/14 [00:00<?, ?it/s]
Uncorr 1/f:   0%|          | 0/4 [00:00<?, ?it/s]
ACN:   0%|          | 0/4 [00:00<?, ?it/s]
```

```
  0%|          | 0/14 [00:00<?, ?it/s]
Uncorr 1/f:   0%|          | 0/4 [00:00<?, ?it/s]
ACN:   0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/14 [00:00<?, ?it/s]
Uncorr 1/f:   0%|          | 0/4 [00:00<?, ?it/s]
ACN:   0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/14 [00:00<?, ?it/s]
Uncorr 1/f:   0%|          | 0/4 [00:00<?, ?it/s]
ACN:   0%|          | 0/4 [00:00<?, ?it/s]
[     pynrc:WARNING] Source RA, Dec = (0.0, 0.0) deg is not visible on 2023-01-01!
```

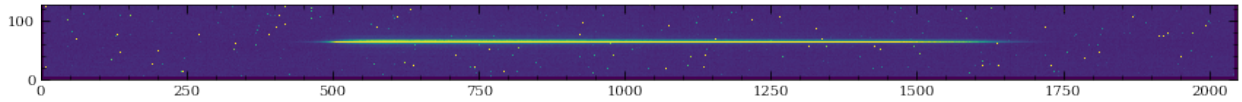
[23]:
```
res.info()
```

```
Filename: (No file associated with this HDUList)
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     117   ()
  1  SCI           1 ImageHDU        44   (2048, 128, 10, 10)   uint16
  2  ZEROFRAME     1 ImageHDU        12   (2048, 128, 10)   uint16
  3  GROUP         1 BinTableHDU     36   100R x 13C   ['I', 'I', 'I', 'J', 'I', '26A',
↪'I', 'I', 'I', 'I', '36A', 'D', 'D']
  4  INT_TIMES     1 BinTableHDU     24   10R x 7C   ['J', 'D', 'D', 'D', 'D', 'D', 'D']
```

[24]:
```
tvals = nrc.Detector.times_group_avg

header = res['PRIMARY'].header
data_all = res['SCI'].data
slope_list = []
for data in tqdm(data_all):
    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 = jl_poly_fit(tvals, ref.data, deg=1)
    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.Detector.gain
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
```

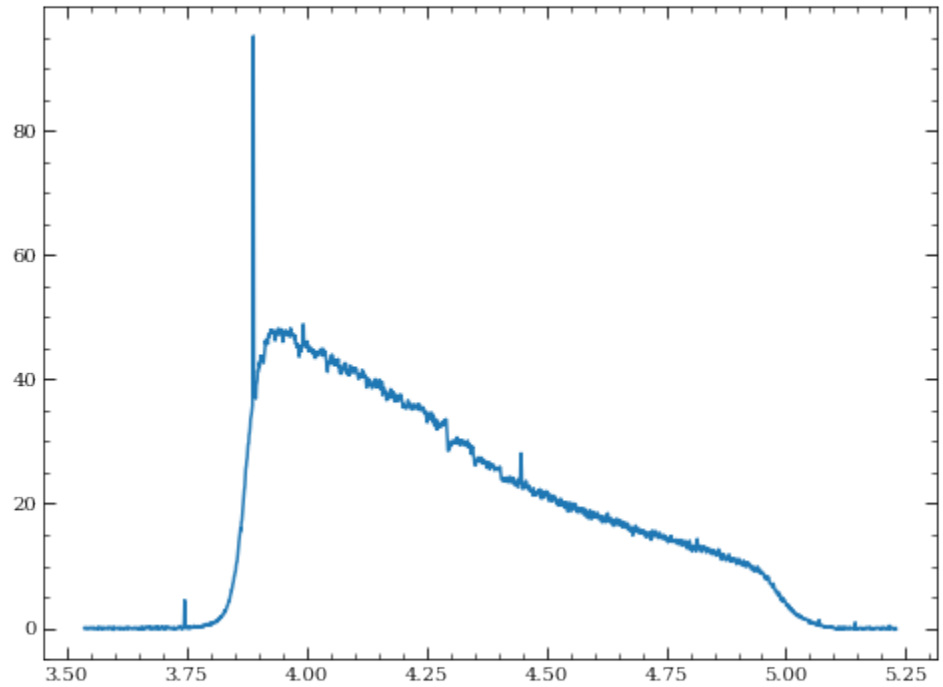
[25]:
```
fig, ax = plt.subplots(1,1, figsize=(12,3))
ax.imshow(slope_sim, vmin=0, vmax=10)

fig.tight_layout()
```



[26]:
```
ind = wspec>0

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

[27]:
```
ind = wspec>0
plt.plot(wspec[ind], slope_sim[63,ind])
```

[27]:
```
[<matplotlib.lines.Line2D at 0x7fc309945150>]
```



[28]:
```
# Extract 2 spectral x 5 spatial pixels

# First, cut out the central 5 pixels
wspec_sub = wspec[ind]
sh_new  = (5, len(wspec_sub))
slope_sub = nrc_utils.pad_or_cut_to_size(slope_sim, sh_new)
slope_sub_ideal = nrc_utils.pad_or_cut_to_size(imspec, sh_new)

# 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
```

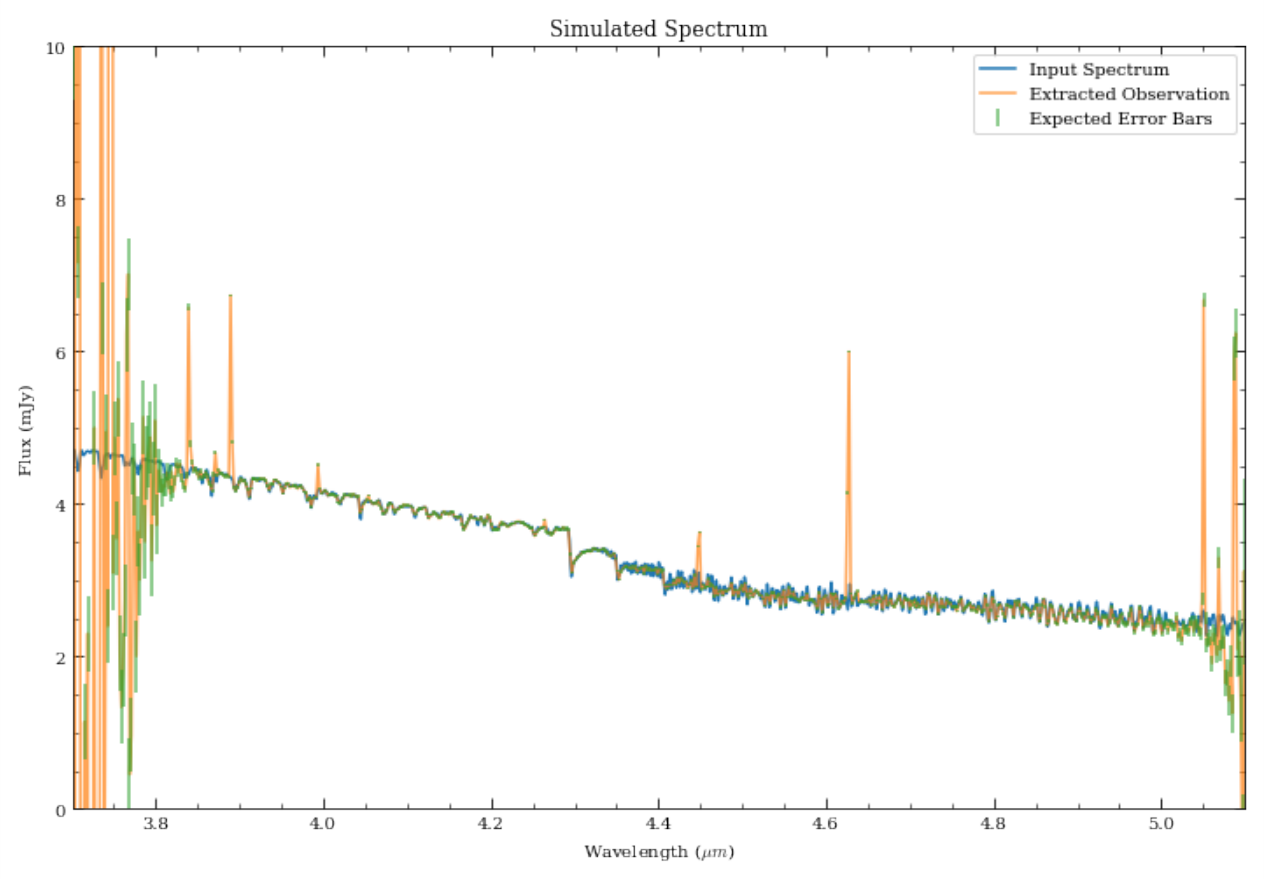
(continues on next page)

(continued from previous page)

```
sp_M0V.convert('mjy')
rsrf = spec_ideal / sp_M0V.sample(wspec_sub*1e4)

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

[29]:
```
# 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)
```

[30]:
```
fig, ax = plt.subplots(1,1, figsize=(12,8))
ax.plot(sp_M0V.wave/1e4, sp_M0V.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, color='C2')

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).

[31]:
```
nrc = pynrc.NIRCam('F322W2', pupil_mask='GRISM0', wind_mode='STRIPE', ypix=64)
```

[32]:
```
# K6V star at K=8.4 mags
bp_k = S.ObsBandpass('k')
sp_K6V = pynrc.stellar_spectrum('K6V', 8.4, 'vegamag', bp_k)
```

[33]:
```
# 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
```

[34]:
```
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      24  460     16.01   7363.99   7523.08  30955.5    0.494  356.894
BRIGHT1      24  461     16.01   7380.00   7539.44  30989.1    0.494  356.894
BRIGHT1      24  462     16.01   7396.01   7555.79  31022.7    0.494  356.894
BRIGHT1      24  463     16.01   7412.01   7572.15  31056.3    0.494  356.894
BRIGHT1      24  464     16.01   7428.02   7588.50  31089.8    0.494  356.894
BRIGHT2      23  470     15.67   7363.99   7526.54  30623.8    0.484  352.989
BRIGHT2      23  471     15.67   7379.66   7542.56  30656.4    0.484  352.989
BRIGHT2      23  472     15.67   7395.32   7558.57  30688.9    0.484  352.989
...         ...  ...       ...       ...       ...      ...      ...      ...
SHALLOW2     10  462     16.01   7396.01   7555.79  30678.0    0.494  352.929
SHALLOW2     10  463     16.01   7412.01   7572.15  30711.2    0.494  352.929
SHALLOW2     10  464     16.01   7428.02   7588.50  30744.4    0.494  352.929
RAPID        30  713     10.22   7285.65   7532.24  30585.0    0.315  352.408
RAPID        30  714     10.22   7295.87   7542.81  30606.5    0.315  352.408
RAPID        30  715     10.22   7306.08   7553.37  30627.9    0.315  352.408
RAPID        30  716     10.22   7316.30   7563.94  30649.3    0.315  352.408
RAPID        30  717     10.22   7326.52   7574.50  30670.7    0.315  352.408
Length = 20 rows
```

[35]:
```
# 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  30585.0    0.315  352.408
RAPID        30  714     10.22   7295.87   7542.81  30606.5    0.315  352.408
BRIGHT1      24  460     16.01   7363.99   7523.08  30955.5    0.494  356.894
BRIGHT1      24  461     16.01   7380.00   7539.44  30989.1    0.494  356.894
BRIGHT2      23  470     15.67   7363.99   7526.54  30623.8    0.484  352.989
BRIGHT2      23  471     15.67   7379.66   7542.56  30656.4    0.484  352.989
SHALLOW2     10  460     16.01   7363.99   7523.08  30611.6    0.494  352.929
SHALLOW2     10  461     16.01   7380.00   7539.44  30644.8    0.494  352.929
```

[36]:
```
# Even though BRIGHT1 has a slight efficiency preference over RAPID
# and BRIGHT2, we decide to choose RAPID, because we are convinced
# that saving all data (and no coadding) is a better option.
# If APT informs you that the data rates or total data shorage is
```

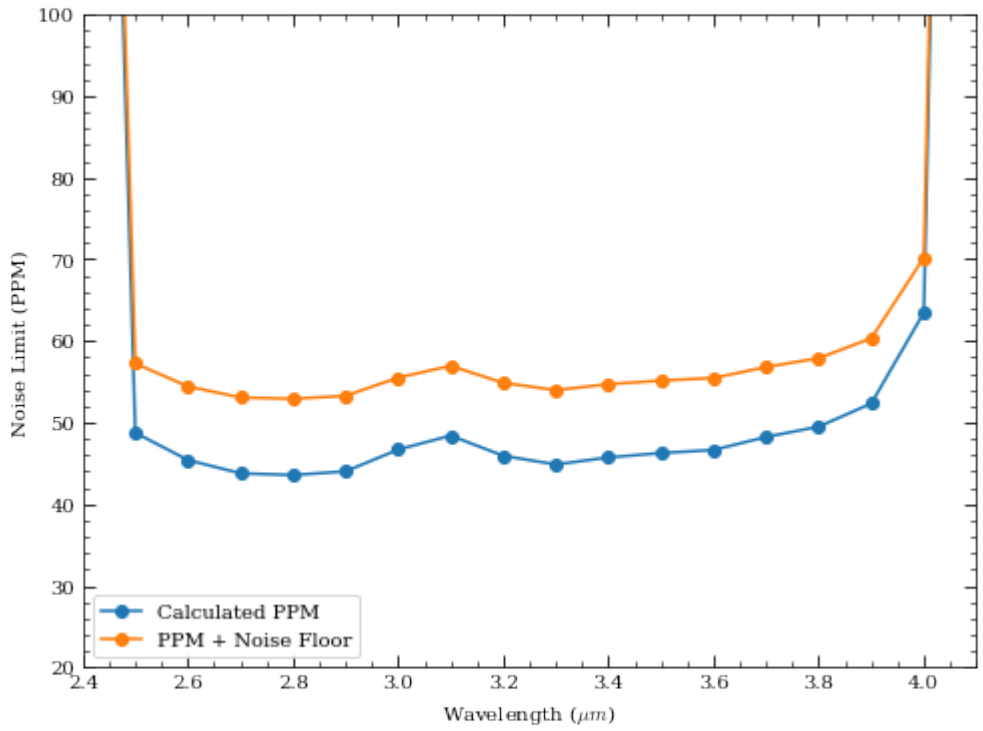
(continues on next page)

(continued from previous page)

```
# an issue, you can select one of the other options.

# Update to RAPID, ngroup=30, nint=700 and plot PPM
nrc.update_detectors(read_mode='RAPID', ngroup=30, nint=700)
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)
```

[37]:
```
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()
```

[37]:
```
<matplotlib.legend.Legend at 0x7fc2f96f0590>
```



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?

[38]:
```
# Detection bandpass is F200W
nrc = pynrc.NIRCam(filter='F200W')

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

[39]:
```
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    9   1052.20   9469.83   9555.73     10.5    0.020    0.107
MEDIUM8      10   10   1052.20  10522.03  10618.67     11.1    0.020    0.107
DEEP8         8    5   1589.04   7945.21   7988.16      9.5    0.031    0.106
DEEP8         7    6   1374.31   8245.84   8299.53      9.7    0.026    0.106
MEDIUM8       8   11    837.47   9212.15   9319.52     10.2    0.016    0.106
DEEP8         6    8   1159.57   9276.57   9351.73     10.3    0.022    0.106
MEDIUM8       9   10    944.84   9448.36   9544.99     10.4    0.018    0.106
DEEP8         8    6   1589.04   9534.25   9587.94     10.4    0.031    0.106
...         ...  ...       ...       ...       ...      ...      ...      ...
BRIGHT1      10   49    204.00   9995.93  10511.30      6.6    0.004    0.064
BRIGHT1      10   50    204.00  10199.93  10726.04      6.7    0.004    0.064
BRIGHT1      10   51    204.00  10403.93  10940.77      6.8    0.004    0.064
RAPID        10   91    107.37   9770.46  10736.78      4.9    0.002    0.047
RAPID        10   92    107.37   9877.83  10854.88      5.0    0.002    0.047
RAPID        10   93    107.37   9985.20  10972.98      5.0    0.002    0.047
RAPID        10   94    107.37  10092.56  11091.09      5.0    0.002    0.047
RAPID        10   95    107.37  10199.93  11209.19      5.0    0.002    0.047
Length = 66 rows
```

[40]:
```
# 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   91    107.37   9770.46  10736.78      4.9    0.002    0.047
RAPID        10   92    107.37   9877.83  10854.88      5.0    0.002    0.047
BRIGHT1      10   47    204.00   9587.94  10081.83      6.5    0.004    0.064
BRIGHT1      10   48    204.00   9791.93  10296.57      6.6    0.004    0.064
```

(continues on next page)

(continued from previous page)

```
BRIGHT2      10   45    214.74   9663.09  10135.52      7.8    0.004    0.077
BRIGHT2      10   46    214.74   9877.83  10360.99      7.9    0.004    0.077
SHALLOW2     10   18    504.63   9083.31   9265.84      9.3    0.010    0.096
SHALLOW2     10   19    504.63   9587.94   9781.20      9.6    0.010    0.096
SHALLOW4     10   18    526.10   9469.83   9652.36     10.1    0.010    0.102
SHALLOW4     10   19    526.10   9995.93  10189.20     10.3    0.010    0.102
MEDIUM2      10    9    987.78   8890.05   8975.94      9.8    0.019    0.103
MEDIUM2      10   10    987.78   9877.83   9974.46     10.4    0.019    0.103
MEDIUM8      10    9   1052.20   9469.83   9555.73     10.5    0.020    0.107
MEDIUM8      10   10   1052.20  10522.03  10618.67     11.1    0.020    0.107
DEEP2         9    5   1739.36   8696.78   8739.74      9.6    0.033    0.103
DEEP2         8    6   1524.62   9147.73   9201.42      9.9    0.029    0.103
DEEP8         8    5   1589.04   7945.21   7988.16      9.5    0.031    0.106
DEEP8         7    6   1374.31   8245.84   8299.53      9.7    0.026    0.106
```

[41]:
```
# 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_tot1  : 1052.203
  t_int_tot2  : 1062.940
  t_exp       : 10522.035
  t_acq       : 10618.671
```

[42]:
```
# 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']])
```

[43]:
```
tbl
```

[43]:
```
<Table length=4>
 Sigma  Point (nJy) Extended (nJy/asec^2) Point (AB Mag) Extended (AB Mag/asec^2)
float64   float64          float64           float64              float64
------- ----------- --------------------- -------------- ------------------------
      1        1.10                 32.05          31.30                    27.64
      3        3.30                 96.57          30.10                    26.44
      5        5.52                161.67          29.55                    25.88
     10       11.14                326.99          28.78                    25.11
```

[ ]:
```

```

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

# Enable inline plotting at lower left
%matplotlib inline
```

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

# Progress bar
from tqdm.auto import tqdm, trange

# 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)]
```

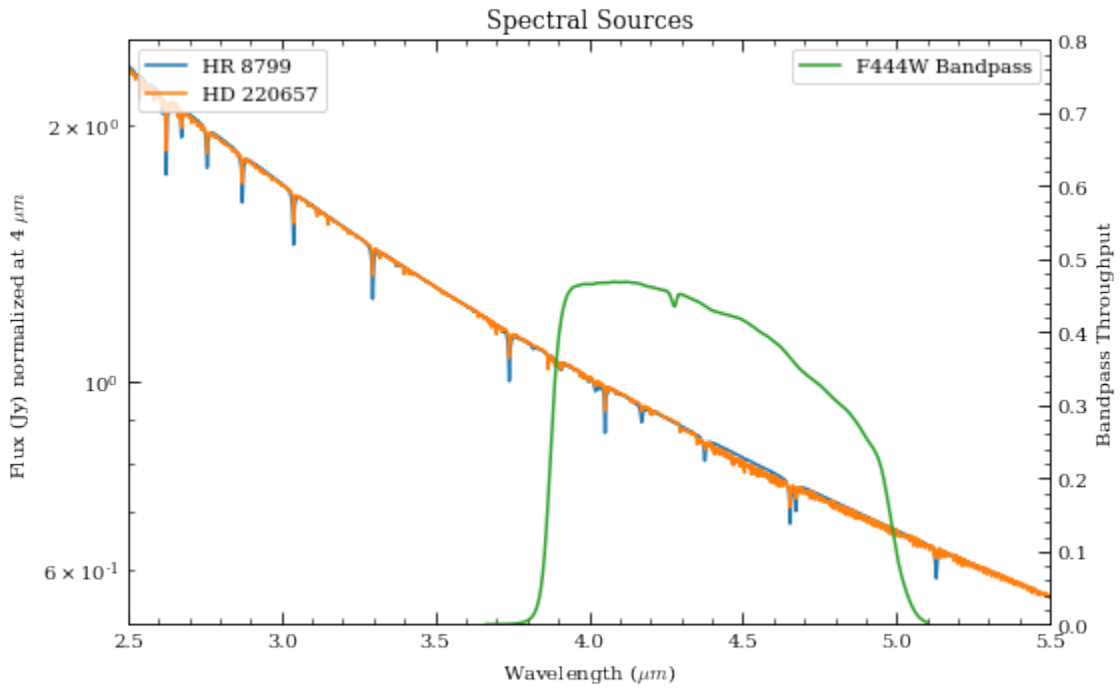
[4]:
```
name_sci, dist_sci, age, spt_sci, Teff_sci, feh_sci, logg_sci, mag_sci, bp_sci = args_
↪sources[0]
name_ref, spt_ref, Teff_ref, feh_ref, logg_ref, mag_ref, bp_ref = ref_sources[0]

# 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])
```

(continues on next page)

(continued from previous page)

```
    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])
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).

The `obs_hci` class also allows one to specify WFE drift values in terms of nm RMS. The `wfe_ref_drift` parameter defaults to 2nm between

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. This process only happens once, then stores the resulting coefficient residuals to disk for future quick retrieval.

Extended disk models can also be specified upon initialization using the `disk_params` keyword (which should be a dictionary).

The `large_grid` keyword controls the quality of PSF variations near and under the corongraphic masks. If False, then a sparse grid is used (faster to generate during initial calculations; less disk space and memory). If True, then a higher density grid is calculated (~2.5 hrs for initial creation; ~3.5x larger sizes), which produces improved PSFs at the SGD positions. For purposes of this demo, we set it to False.

[6]:
```
# The initial call make take some time, as it will need to generate coefficients
# to calculate PSF variations across wavelength, WFE drift, and mask location
filt, mask, pupil = ('F444W',  'MASK430R', 'CIRCLYOT')
wind_mode, subsize = ('WINDOW', 320)
fov_pix, oversample = (321, 2)

obs = pynrc.obs_hci(sp_sci, dist_sci, sp_ref=sp_ref, use_ap_info=False,
                    filter=filt, image_mask=mask, pupil_mask=pupil,
                    wind_mode=wind_mode, xpix=subsize, ypix=subsize,
                    fov_pix=fov_pix, oversample=oversample, large_grid=True)
```

Some information for the reference observation is stored in the attribute `obs.Detector_ref`, which is a separate NIRCam `DetectorOps` class that we use to keep track of the detector a multiaccum configurations, which may differe between science and reference observations. Settings for the reference observation can be updated using the `obs.gen_ref()` function.

[7]:
```
# Set default WFE drift values between Roll1, Roll2, and Ref

# WFE drift amount between rolls
obs.wfe_roll_drift = 2

# Drift amount between Roll 1 and Reference.
obs.wfe_ref_drift = 5
```

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 sequence 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.gen_ref_det(read_mode='DEEP8', ngroup=16, nint=5)
```

```
New Ramp Settings
  read_mode  :    DEEP8
  nf         :        8
  nd2        :       12
  ngroup     :       16
  nint       :        5
New Detector Settings
  wind_mode  :   WINDOW
  xpix       :      320
  ypix       :      320
  x0         :      914
  y0         :     1512
New Ramp Times
  t_group    :   21.381
  t_frame    :    1.069
  t_int      :  329.264
  t_int_tot1 :  330.353
  t_int_tot2 :  330.353
  t_exp      : 1646.322
  t_acq      : 1651.766
```

1.6.4 Add Planets

There are four known giant planets orbiting HR 8799. 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, 2022. 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.

Their are a few exoplanet models available to `pynrc` (SB12, BEX, COND), but let's choose those from Spiegel & Burrows (2012).

[9]:
```
# Projected locations for date 11/01/2022
# These are prelimary positions, but within constrained orbital parameters
loc_list = [(-1.625, 0.564), (0.319, 0.886), (0.588, -0.384), (0.249, 0.294)]

# 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.delete_planets().
obs.delete_planets()
for i, loc in enumerate(loc_list):
    obs.add_planet(model='SB12', 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 verify orientation
PA1 = 85            # Telescope V3 PA
PA_offset = -1*PA1  # Image field is rotated opposite direction
im_planets = obs.gen_planets_image(PA_offset=PA_offset, return_oversample=False)
```

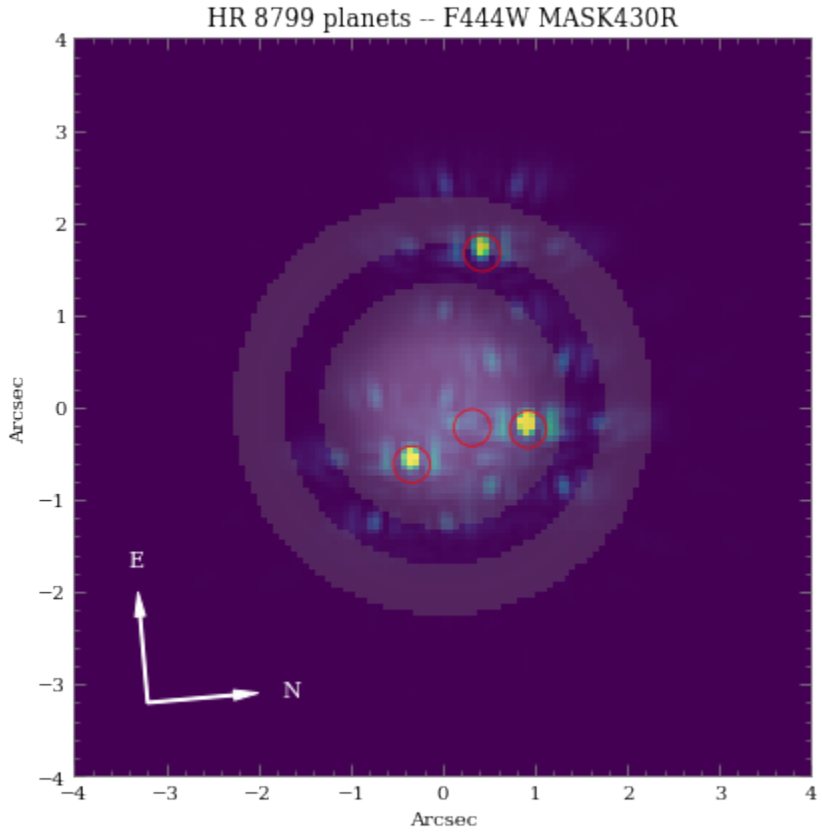
```
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
```

[12]:
```
from matplotlib.patches import Circle
from pynrc.nrc_utils import plotAxes
from pynrc.obs_nircam import get_cen_offsets

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

xasec = obs.det_info['xpix'] * obs.pixelscale
yasec = obs.det_info['ypix'] * obs.pixelscale
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.Detector.detid
im_mask = obs.mask_images['DETSAMP']
# Do some masked transparency overlays
masked = np.ma.masked_where(im_mask>0.98*im_mask.max(), im_mask)
ax.imshow(1-masked, extent=extent, alpha=0.3, cmap='Greys_r', vmin=-0.5)

for loc in loc_list:
    xc, yc = get_cen_offsets(obs, idl_offset=loc, PA_offset=PA_offset)
    circle = Circle((xc,yc), radius=xylim/20., alpha=0.7, lw=1, edgecolor='red',␣
↪facecolor='none')
    ax.add_artist(circle)

xlim = ylim = np.array([-1,1])*xylim
xlim = xlim + obs.bar_offset
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.image_mask))
```

(continues on next page)

(continued from previous page)

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

plotAxes(ax, width=1, headwidth=5, alength=0.15, angle=PA_offset,
                    position=(0.1,0.1), 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* despite providing a similiar magnitude as *d*. This is primarily due to its location relative to the occulting mask reducing throughput along with confusion of bright diffraction spots from the other nearby sources.

Note: the circled regions of the expected planet positions don't perfectly align with the PSFs, because the LW wavelengths have a slight dispersion through the Lyot mask material.

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 11/1/2022, 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`.

Images generated with the `gen_roll_image` method will also include random pointing offsets described in the `pointing_info` dictionary. These can be generated by calling `obs.gen_pointing_offsets()`.

[20]:
```
# Create pointing offset with a random seed for reproducibility
obs.gen_pointing_offsets(rand_seed=1234, verbose=True)
```

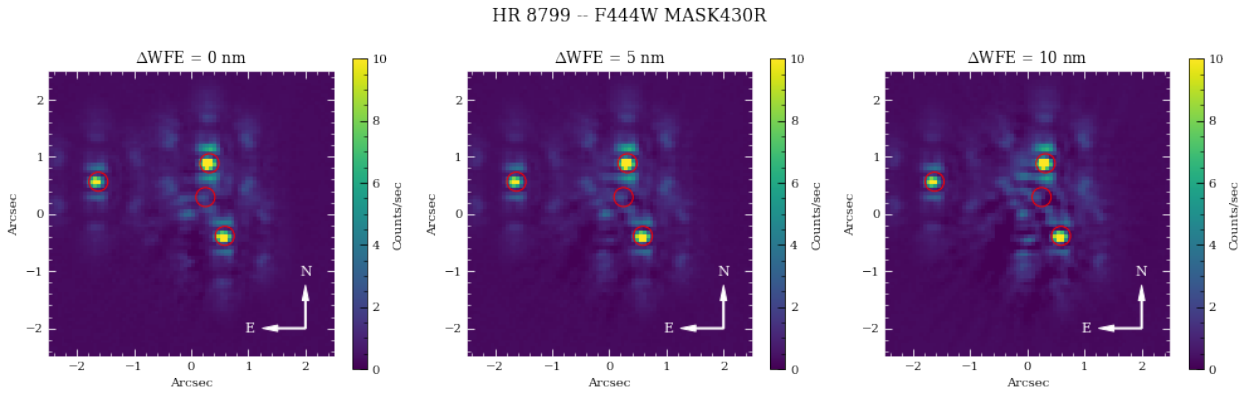
```
Pointing Info
  sgd_type   : None
  slew_std   : 5
  fsm_std    : None
  roll1      : [0.00370446 0.0007631 ]
  roll2      : [0.00431872 0.0145655 ]
  ref        : [-0.00801918  0.0003205 ]
```

[21]:
```
# Cycle through a few WFE drift values
wfe_list = [0,5,10]

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

# A dictionary of HDULists
hdul_dict = {}
for wfe_drift in tqdm(wfe_list):
    # Assume drift between Roll1 and Roll2 is 2 nm WFE
    wfe_roll_drift = 0 if wfe_drift<2 else 2
    hdulist = obs.gen_roll_image(PA1=PA1, PA2=PA2,
                                 wfe_ref_drift=wfe_drift, wfe_roll_drift=wfe_roll_drift)
    hdul_dict[wfe_drift] = hdulist
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
```

```
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
```

[22]:
```
from pynrc.nb_funcs 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=10)

    # 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=(0.9,0.1),␣
↪label1='E', label2='N')

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



The majority of the speckle noise here originates from small pointing offsets between the roll positions and reference observation. These PSF centering mismatches dominate the subtraction residuals compared to the WFE drift variations. Small-grid dithers acquired during the reference observations should produce improved subtraction performance through PCA/KLIP algorithms. To get a better idea of the post-processing performance, we re-run these observations assuming perfect target acquisition.

[23]:
```
hdul_dict = {}
for wfe_drift in tqdm(wfe_list):
    # Assume drift between Roll1 and Roll2 is 2 nm WFE
    wfe_roll_drift = 0 if wfe_drift<2 else 2
```

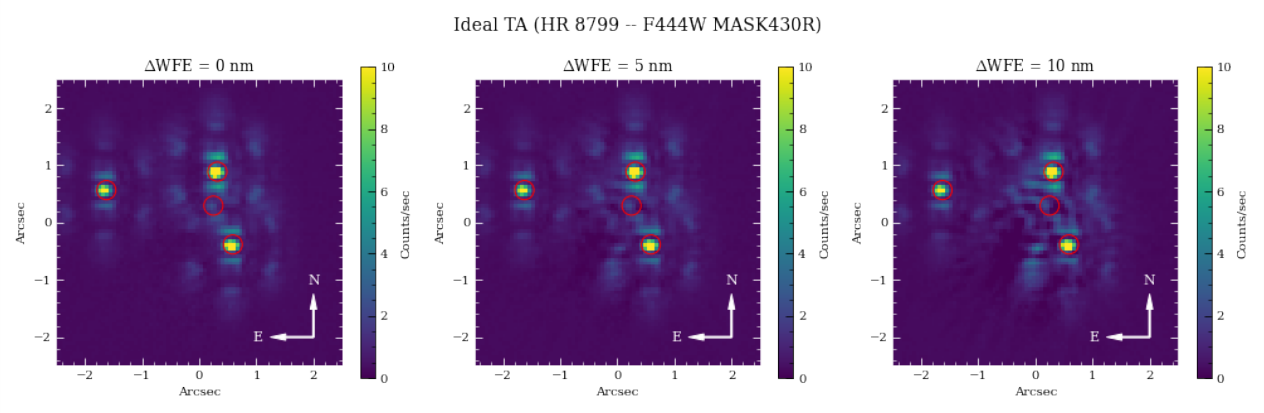
(continues on next page)

(continued from previous page)

```
    # Assume perfect centering by setting xyoff_***=(0,0)
    hdulist = obs.gen_roll_image(PA1=PA1, PA2=PA2,
                                 wfe_ref_drift=wfe_drift, wfe_roll_drift=wfe_roll_drift,
                                 xyoff_roll1=(0,0), xyoff_roll2=(0,0), xyoff_ref=(0,0))
    hdul_dict[wfe_drift] = hdulist
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
```

[24]:
```
from pynrc.nb_funcs 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=10)

    # 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=(0.9,0.1),␣
↪label1='E', label2='N')

fig.suptitle('Ideal TA ({} -- {} {})'.format(name_sci, obs.filter, obs.image_mask),␣
↪fontsize=14)
fig.tight_layout()
fig.subplots_adjust(top=0.85)
```



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).

In order to better understand the relative contributes of WFE drift to contrast loss, we're going to ignore telescope pointing offsets by explicitly passing `xoff_* = (0,0)` keywords for Roll 1, Roll 2, and Ref observations.

[25]:
```
nsig = 5
roll_angle = np.abs(PA2 - PA1)

curves = []
for wfe_drift in tqdm(wfe_list):
    # Assume drift between Roll1 and Roll2 is 2 nm WFE
    wfe_roll_drift = 0 if wfe_drift<2 else 2

    # Generate contrast curves
    result = obs.calc_contrast(roll_angle=roll_angle, nsig=nsig,
                               wfe_ref_drift=wfe_drift, wfe_roll_drift=wfe_roll_drift,
                               xyoff_roll1=(0,0), xyoff_roll2=(0,0), xyoff_ref=(0,0))
    curves.append(result)
```

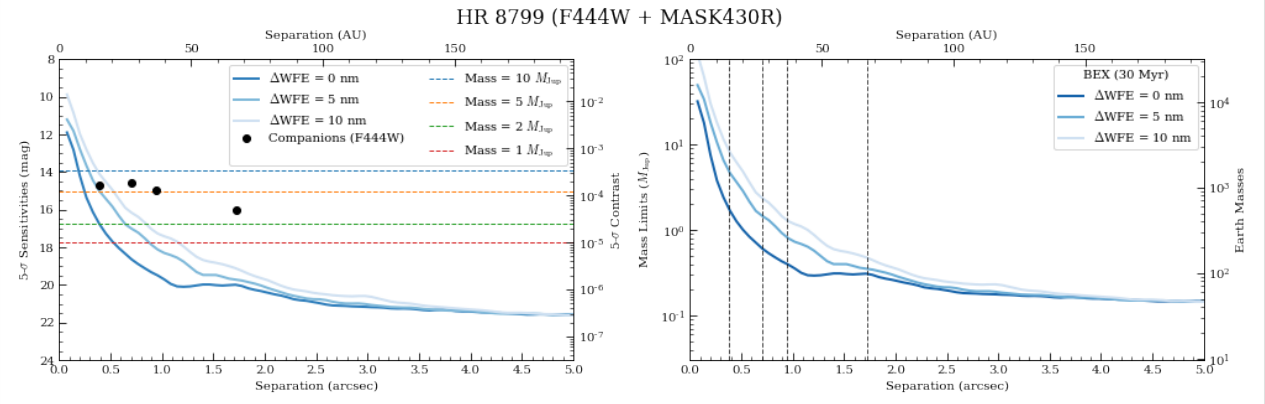
```
  0%|          | 0/3 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
```

[26]:
```
from pynrc.nb_funcs import plot_contrasts, plot_planet_patches, plot_contrasts_mjup,␣
↪update_yscale
import matplotlib.patches as mpatches
```

(continues on next page)

(continued from previous page)

```
# 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]
ax1, ax2, ax3 = plot_contrasts_mjup(curves, nsig, wfe_list, obs=obs, age=age,
                                    ax=ax, twin_ax=True, xr=xr, yr=None, return_
↪axes=True)
yr = [0.03,100]
for xval in planet_dist:
    ax.plot([xval,xval],yr, lw=1, ls='--', color='k', alpha=0.7)
update_yscale(ax1, 'log', ylim=yr)
yr_temp = np.array(ax1.get_ylim()) * 318.0
update_yscale(ax2, 'log', ylim=yr_temp)
# ax.set_yscale('log')
# ax.set_ylim([0.08,100])
ax.legend(loc='upper right', title='BEX ({:.0f} Myr)'.format(age))

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

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



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_frac` keyword.

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

[27]:
```
# 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(f'NGROUP={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(f'{obs.sp_sci.name} flux at {obs.filter}: {mag_sci:0.2f} mags')
print(f'{obs.sp_ref.name} flux at {obs.filter}: {mag_ref:0.2f} mags')
```

```
NGROUP=2
F444W Saturation Limit assuming Flat spectrum in photlam source (point source): 1.29␣
↪vegamag
F444W Saturation Limit assuming Flat spectrum in photlam source (extended): 10.33␣
↪vegamag/arcsec^2

NGROUP=16
F444W Saturation Limit assuming Flat spectrum in photlam source (point source): 3.89␣
↪vegamag
F444W Saturation Limit assuming Flat spectrum in photlam source (extended): 12.93␣
↪vegamag/arcsec^2

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 saturate. However, the reference source should have some saturated pixels before the end of an integration.

[28]:
```
# Well level of each pixel for science source
sci_levels1 = obs.saturation_levels(ngroup=2, exclude_planets=True)
sci_levels2 = obs.saturation_levels(ngroup=ng_max, exclude_planets=True)

# Which pixels are saturated? Assume sat level at 90% full well.
sci_mask1 = sci_levels1 > 0.9
sci_mask2 = sci_levels2 > 0.9
```

[29]:
```
# Well level of each pixel for reference source
ref_levels1 = obs.saturation_levels(ngroup=2, do_ref=True)
```

(continues on next page)

(continued from previous page)

```
ref_levels2 = obs.saturation_levels(ngroup=ng_max, do_ref=True)

# Which pixels are saturated? Assume sat level at 90% full well.
ref_mask1 = ref_levels1 > 0.9
ref_mask2 = ref_levels2 > 0.9
```

[30]:
```
# 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
44 saturated pixel at NGROUP=16
```

[31]:
```
# Saturation Mask for science target

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

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

    xasec = obs.det_info['xpix'] * obs.pixelscale
    yasec = obs.det_info['ypix'] * obs.pixelscale
    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:
```

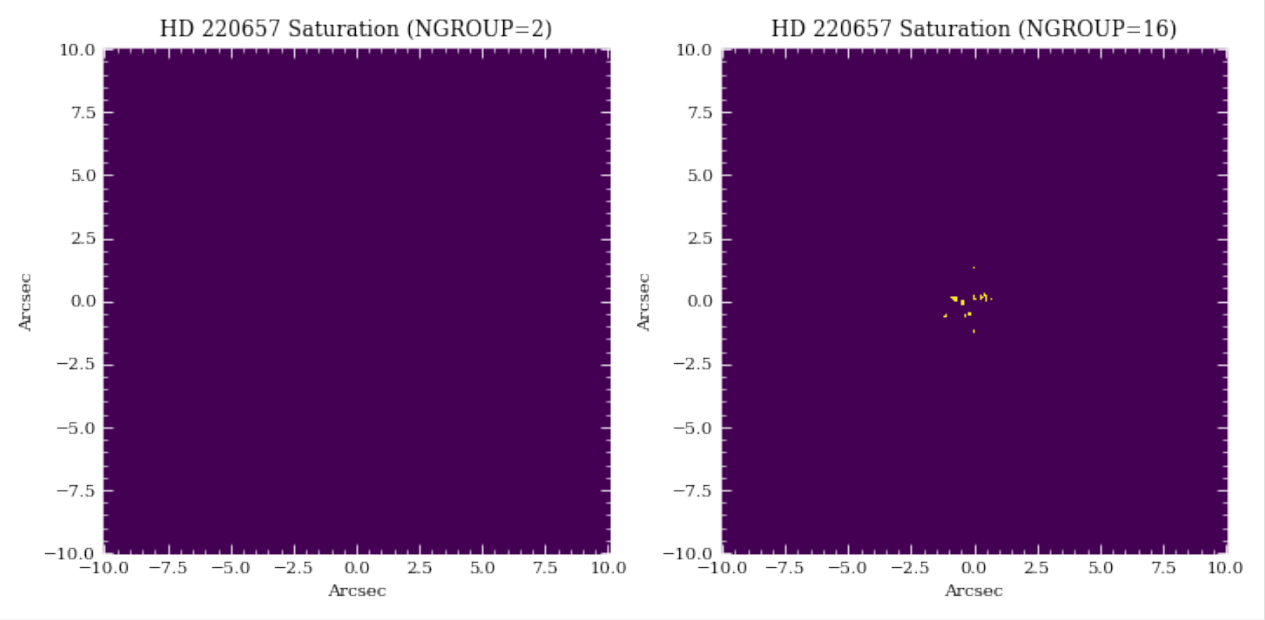
(continues on next page)

(continued from previous page)

```
        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.
```

[32]:
```
# Saturation Mask for reference

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

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

    xasec = obs.det_info['xpix'] * obs.pixelscale
    yasec = obs.det_info['ypix'] * obs.pixelscale
    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(f'{sp.name} Saturation (NGROUP=2)')
    axes[1].set_title(f'{sp.name} Saturation (NGROUP={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.')
```



[ ]:

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

# Enable inline plotting at lower left
%matplotlib inline
```

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

# Progress bar
from tqdm.auto import tqdm, trange

# 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.

[4]:
```
# 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 = 'votables/'
```

[5]:
```
# 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=False, robust=False, wlim=[1,5])

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

```
[0.98590364]
```

[6]:
```
# 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=[0.5,10])

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

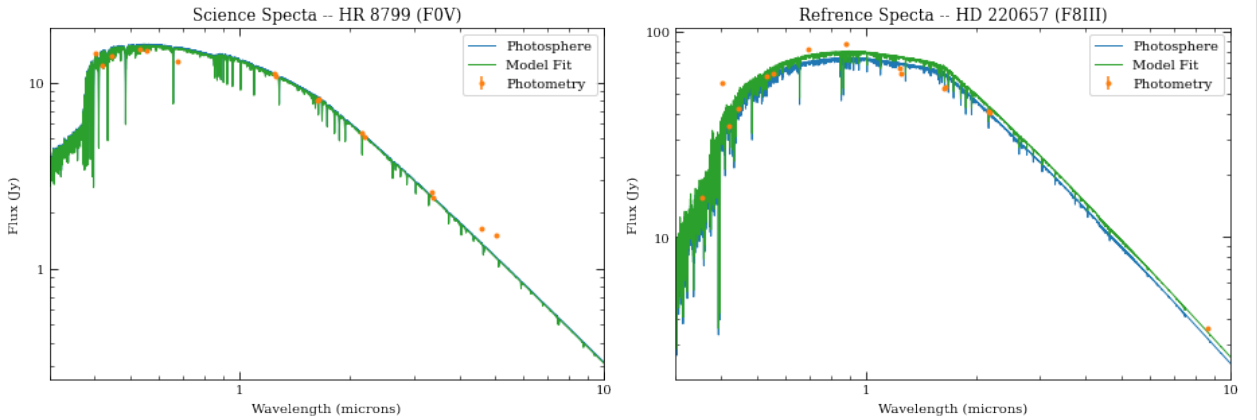
```
[1.07580856]
```

[7]:
```
# Plot spectra
fig, axes = plt.subplots(1,2, figsize=(13,4.5))
```

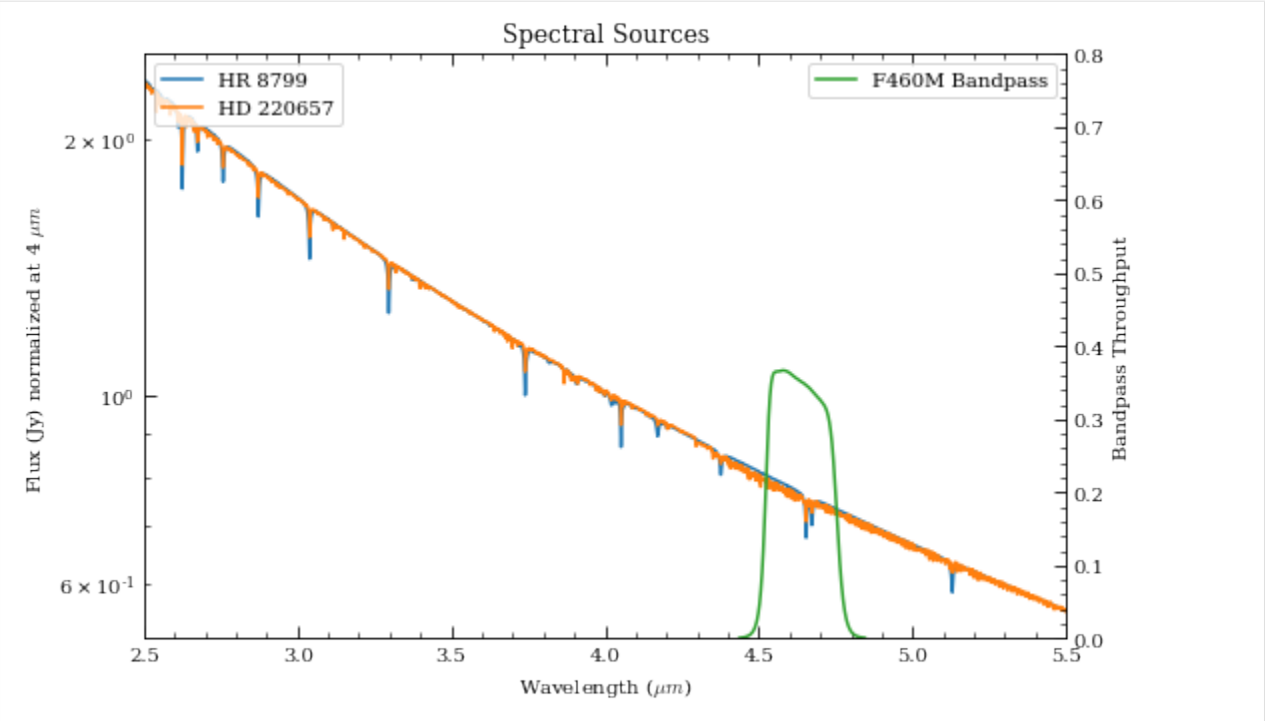
(continues on next page)

(continued from previous page)

```
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()
```



[8]:
```
# 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')

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 'sci' coordinate convention. Specifying this 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 at `bar_offset=8` arcsec from the bar center.

[9]:
```
filt, mask, pupil = ('F460M',  'MASKLWB', 'WEDGELYOT')
wind_mode, subsize = ('WINDOW', 320)
fov_pix, oversample = (321, 2)

obs = pynrc.obs_hci(sp_sci, dist_sci, sp_ref=sp_ref, bar_offset=8, use_ap_info=False,
                    filter=filt, image_mask=mask, pupil_mask=pupil,
                    wind_mode=wind_mode, xpix=subsize, ypix=subsize,
                    fov_pix=fov_pix, oversample=oversample, large_grid=True)
```

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

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.gen_ref_det(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         :      275
  y0         :     1522
New Ramp Times
  t_group    :    2.138
  t_frame    :    1.069
  t_int      :   21.381
  t_int_tot1 :   22.470
  t_int_tot2 :   22.470
  t_exp      :  855.232
  t_acq      :  898.790
```

1.7.4 Add Planets

There are four known giant planets orbiting HR 8799. 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, 2022. 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.

Their are a few exoplanet models available to `pynrc` (SB12, BEX, COND). Let's choose those from Spiegel & Burrows (2012).

[11]:
```
# Projected locations for date 11/01/2022
# These are prelimary positions, but within constrained orbital parameters
loc_list = [(-1.625, 0.564), (0.319, 0.886), (0.588, -0.384), (0.249, 0.294)]

# Estimated magnitudes within F444W 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.delete_planets().
obs.delete_planets()
for i, loc in enumerate(loc_list):
    obs.add_planet(model='SB12', 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 verify orientation
PA1 = 85            # Telescope V3 PA
PA_offset = -1*PA1  # Image field is rotated opposite direction
im_planets = obs.gen_planets_image(PA_offset=PA_offset, return_oversample=False)
```

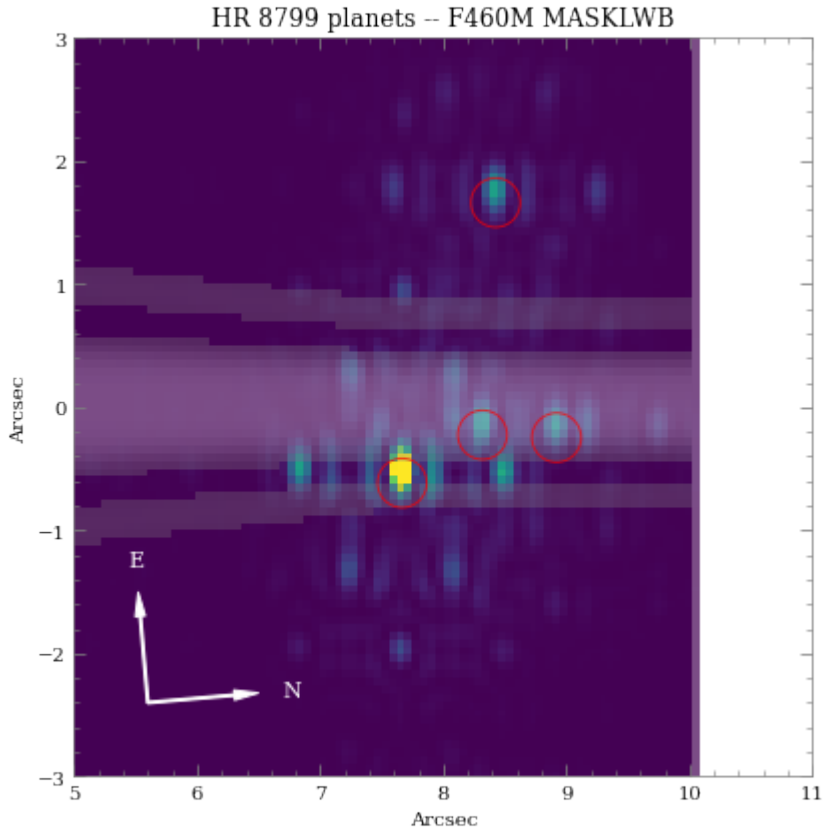
```
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
```

[14]:
```
from matplotlib.patches import Circle
from pynrc.nrc_utils import plotAxes
from pynrc.obs_nircam import get_cen_offsets

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

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

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

# Overlay the coronagraphic mask
detid = obs.Detector.detid
im_mask = obs.mask_images['DETSAMP']
# Do some masked transparency overlays
masked = np.ma.masked_where(im_mask>0.95*im_mask.max(), im_mask)
ax.imshow(1-masked, extent=extent, alpha=0.3, cmap='Greys_r', vmin=-0.5)


for loc in loc_list:
    xc, yc = get_cen_offsets(obs, idl_offset=loc, PA_offset=PA_offset)
    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 + obs.bar_offset
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.image_mask))
```

(continues on next page)

(continued from previous page)

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

plotAxes(ax, width=1, headwidth=5, alength=0.15, angle=PA_offset,
                  position=(0.1,0.1), 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`.

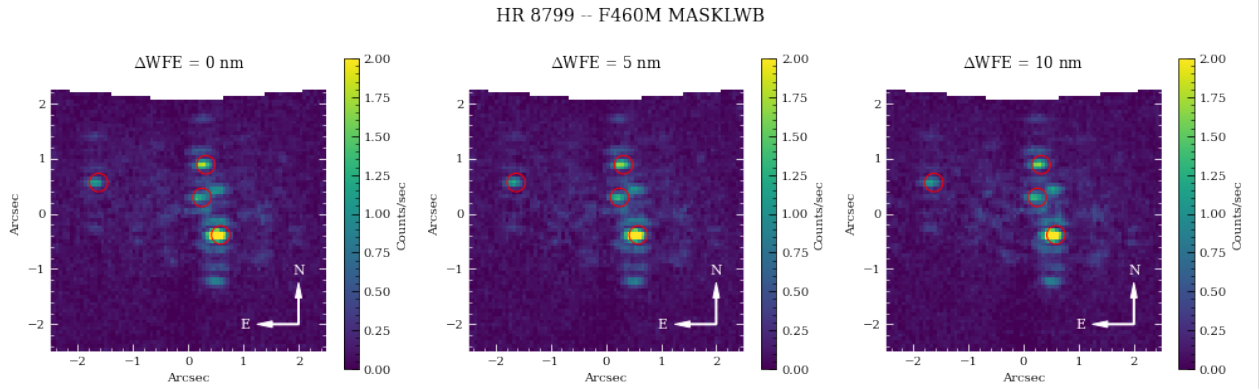
Contrast Curves

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]

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

# A dictionary of HDULists
hdul_dict = {}
for wfe_drift in tqdm(wfe_list):
    # Assume drift between Roll1 and Roll2 is 2 nm WFE
    wfe_roll_drift = 0 if wfe_drift<2 else 2
    # Assume perfect pointing (ie., xyoff_*** = (0,0) )
    # to approximate results of advanced post-processing
    hdulist = obs.gen_roll_image(PA1=PA1, PA2=PA2,
                                 wfe_ref_drift=wfe_drift, wfe_roll_drift=wfe_roll_drift,
                                 xyoff_roll1=(0,0), xyoff_roll2=(0,0), xyoff_ref=(0,0))
    hdul_dict[wfe_drift] = hdulist
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
```

[16]:
```
from pynrc.nb_funcs 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)

    # 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=(0.9,0.1),␣
↪label1='E', label2='N')

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



[17]:
```
nsig = 5
roll_angle = np.abs(PA2 - PA1)

curves = []
for wfe_drift in tqdm(wfe_list):
    # Assume drift between Roll1 and Roll2 is 2 nm WFE
    wfe_roll_drift = 0 if wfe_drift<2 else 2

    # Generate contrast curves
    result = obs.calc_contrast(roll_angle=roll_angle, nsig=nsig,
                               wfe_ref_drift=wfe_drift, wfe_roll_drift=wfe_roll_drift,
                               xyoff_roll1=(0,0), xyoff_roll2=(0,0), xyoff_ref=(0,0))
```

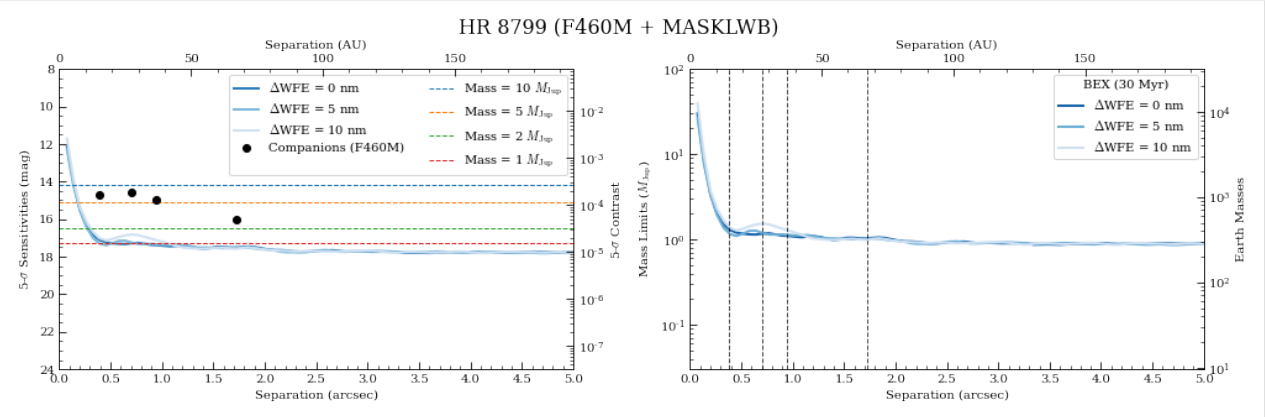
(continues on next page)

(continued from previous page)

```
    curves.append(result)
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
Companions:   0%|          | 0/4 [00:00<?, ?it/s]
```

[18]:
```
from pynrc.nb_funcs import plot_contrasts, plot_planet_patches, plot_contrasts_mjup,␣
↪update_yscale
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]
ax1, ax2, ax3 = plot_contrasts_mjup(curves, nsig, wfe_list, obs=obs, age=age,
                                    ax=ax, twin_ax=True, xr=xr, yr=None, return_
↪axes=True)
yr = [0.03,100]
for xval in planet_dist:
    ax.plot([xval,xval],yr, lw=1, ls='--', color='k', alpha=0.7)
update_yscale(ax1, 'log', ylim=yr)
yr_temp = np.array(ax1.get_ylim()) * 318.0
update_yscale(ax2, 'log', ylim=yr_temp)
ax.legend(loc='upper right', title='BEX ({:.0f} Myr)'.format(age))

fig.suptitle('{} ({} + {})'.format(name_sci, obs.filter, obs.image_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 DMS Level 1b Example

1.8.1 HR 8799 GTO 1194

This program will search for previously unknown planets using NIRCam in the F356W and F444W filters using the MASK430R for both filters. In addition, we will probe the physical characterization of the known planets, HR8789bcde, using multi-filter photometry with the LW Bar mask. The medium-band filters will be observed with a fiducial override to place the primary source on the narrow end of the occulting mask. The NIRCam observations will use two roll angles (± 5 deg) and a reference star to assist with suppression of residuals in the coronagraphic image.

This notebook uses output from the APT file of PID 1194 to simulate the obsevations and save them to DMS-like FITS files (Level 1b data).

[1]:
```
# Import the usual libraries
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
#import matplotlib.patches as mpatches

# Enable inline plotting
%matplotlib inline

# Progress bar
from tqdm.auto import trange, tqdm
```

[2]:
```
import pynrc
from pynrc.simul.ngNRC import create_level1b_FITS

# 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.
```

[3]:
```
from astropy import units as u
from astropy.coordinates import SkyCoord, Distance
from astropy.time import Time
from astropy.table import Table
```

[4]:
```
import os

import pynrc
from pynrc import nrc_utils
from pynrc.simul.ngNRC import make_gaia_source_table, make_simbad_source_table
from pynrc.simul.ngNRC import create_level1b_FITS

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

1.8.2 APT Inputs

From the final APT file, we need to export a number of files, including the `.timing.json`, `.smart_accounting`, `.pointing`, and `.xml` files. These are then parsed by `pynrc` to configure a series of observation visits and associated NIRCam objects.

[5]:
```
import os

# Read in APT
pid = 1194
pid_str = f'pid{pid:05d}'

save_dir = f'/Users/jarron/NIRCam/Data/NRC_Sims/Sim_{pid_str}/'
# save_dir = f'/data/NIRData/NRC_Sims/Sim_{pid_str}/'

# APT input files
apt_file_dir = '../../notebooks/APT_output/'
fprefix = f'pid{pid}'
json_file     = f'{apt_file_dir}{fprefix}.timing.json'
sm_acct_file  = f'{apt_file_dir}{fprefix}.smart_accounting'
pointing_file = f'{apt_file_dir}{fprefix}.pointing'
xml_file      = f'{apt_file_dir}{fprefix}.xml'

# Make sure files exist
for f in [json_file, sm_acct_file, pointing_file, xml_file]:
    print(f, os.path.isfile(f))
```

```
../../notebooks/APT_output/pid1194.timing.json True
../../notebooks/APT_output/pid1194.smart_accounting True
../../notebooks/APT_output/pid1194.pointing True
../../notebooks/APT_output/pid1194.xml True
```

1.8.3 Source Definitions

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.

The output spectra will then be placed into a target dictionary to be ingested into the DMS simulator portion of pyNRC.

[6]:
```
# 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_sci = ('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
args_ref = ('HD 220657', 'F8III', 5888, -0.01, 3.22, 3.04, bp_k)

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

[7]:
```
# Fit spectrum to SED photometry
name_sci, dist_sci, age, spt_sci, Teff_sci, feh_sci, logg_sci, mag_sci, bp_sci = args_sci
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 = pynrc.source_spectrum(*args, **kwargs)

src.fit_SED(use_err=False, robust=False, wlim=[1,5])

# Final source spectrum (pysynphot)
sp_sci = src.sp_model
```

```
[0.98590364]
```

[8]:
```
# Do the same for the reference source
name_ref, spt_ref, Teff_ref, feh_ref, logg_ref, mag_ref, bp_ref = args_ref
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 = pynrc.source_spectrum(*args, **kwargs)

ref.fit_SED(use_err=False, robust=False, wlim=[0.5,10])

# Final reference spectrum (pysynphot)
sp_ref = ref.sp_model
```

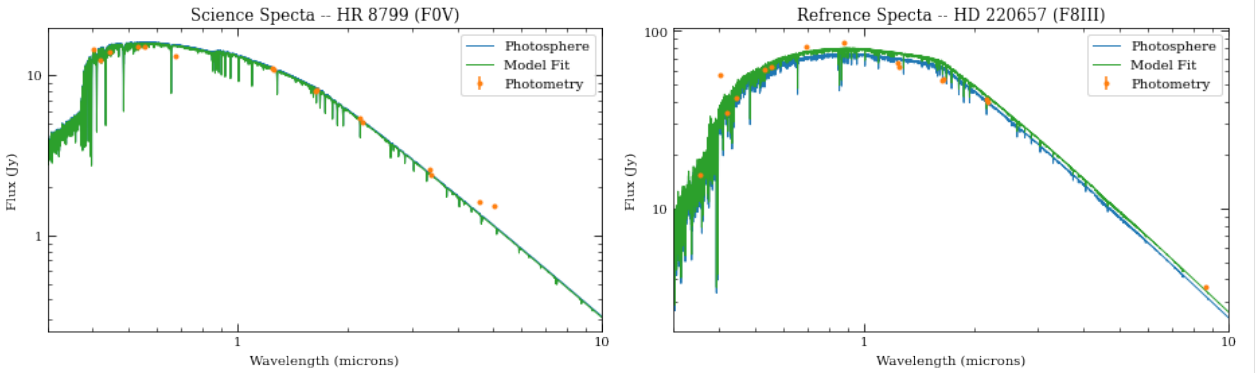
```
[1.07580856]
```

[9]:
```
# Plot spectra
fig, axes = plt.subplots(1,2, figsize=(13,4))
src.plot_SED(xr=[0.3,10], ax=axes[0])
ref.plot_SED(xr=[0.3,10], ax=axes[1])
```

(continues on next page)

(continued from previous page)

```
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()
```



1.8.4 Target Information

For each target specified in the APT file, we want to populate a dictionary with coordinate information and astrophysical properties. The dictionary keys should match the APT "Name in the Proposal" so that

For each designated target, there are four types of objects that can be added: stellar source, point source companions, disk object, and/or a table of point sources specifying magnitudes in NIRCam filters.

1. Stellar source:

   ```
   params_star = {
   'sptype' : 'A0V', 'Teff': 10325, 'metallicity' : 0, 'log_g' : 4.09,
   'v_mag' : 5.690, 'j_mag': 5.768, 'h_mag': 5.753, 'k_mag': 5.751,
   },
   ```

   to generate a spectrum using `pynrc.stellar_spectrum` or add the parameter directly

   ```
   params_star = {'sp' : sp_star}
   ```

2. Companions (where `bp_renorm` is a `pysynphot` bandpass):

   ```
   params_hci_companions = {
       'b' : {'xy':(-1.625,  0.564), 'runits':'arcsec', 'renorm_args':(16.0, 'vegamag',
   ↪ bp_renorm)},
       'c' : {'xy':( 0.319,  0.886), 'runits':'arcsec', 'renorm_args':(15.0, 'vegamag',
   ↪ bp_renorm)},
       'd' : {'xy':( 0.588, -0.384), 'runits':'arcsec', 'renorm_args':(14.6, 'vegamag',
   ↪ bp_renorm)},
       'e' : {'xy':( 0.249,  0.294), 'runits':'arcsec', 'renorm_args':(14.7, 'vegamag',
   ↪ bp_renorm)},
   }
   ```

   See `obs_hci.add_planet()` function for more information and advanced functionality.

3. Companion disk model:

```
params_disk_model = {
    'file': 'HD10647.fits',  # path to file
    'wavelength': 3.0,       # model wavelength (um)
    'pixscale': 0.01575,     # input image arcsec/pixel
    'units': 'Jy/pixel',     # model flux units (e.g., mJy/arcsec, Jy/pixel, etc)
    'dist': 17.34,           # assumed distance to model (pc)
    'cen_star': True,        # does the model include stellar flux?
}
```

4. Table of stellar sources:

```
from astropy.io import ascii

cat_file = 'lmc_catalog.cat'
names = [
    'index', 'ra', 'dec', 'F070W', 'F090W', 'F115W', 'F140M', 'F150W', 'F150W2',
    'F162M', 'F164N', 'F182M', 'F187N', 'F200W', 'F210M', 'F212N', 'F250M', 'F277W',
    'F300M','F322W2', 'F323N', 'F335M', 'F356W', 'F360M', 'F405N', 'F410M', 'F430M',
    'F444W', 'F460M', 'F466N', 'F470N', 'F480M'
]

src_tbl = ascii.read(cat_file, names=names)
```

Use `make_gaia_source_table` and `make_simbad_source_table` to query Gaia DR2 and Simbad to auto-generate tables.

[10]:
```
# Initialize target dictionary
targ_dict = {}

# Bandpass corresponding to companion renormalization flux
bp_pl_norm = pynrc.read_filter('F430M')
# Assumed companion magnitudes in filter bandpass
comp_mags = np.array([16.0, 15.0, 14.6, 14.7])

# Dictionary keywords should match APT target names
targ_dict['HR8799'] = {
    'type' : 'FixedTargetType',
    'TargetName' : 'HR8799', 'TargetArchiveName' : 'HR8799',
    'EquatorialCoordinates' : "23 07 28.7155 +21 08 3.30",
    'RAProperMotion'  : 108.551*u.mas/u.yr,
    'DecProperMotion' : -49.639*u.mas/u.yr,
    'parallax' : 24.76*u.mas, 'age_Myr' : 30,
    'params_star' : {'sp' : sp_sci},
    'params_companions' : {
        'b' : {'xy':(-1.625,  0.564), 'runits':'arcsec', 'mass':10,
               'renorm_args':(comp_mags[0], 'vegamag', bp_pl_norm)
              },
        'c' : {'xy':( 0.319,  0.886), 'runits':'arcsec', 'mass':10,
               'renorm_args':(comp_mags[1], 'vegamag', bp_pl_norm)
              },
        'd' : {'xy':( 0.588, -0.384), 'runits':'arcsec', 'mass':10,
               'renorm_args':(comp_mags[2], 'vegamag', bp_pl_norm)
              },
```

(continues on next page)

(continued from previous page)

```
        'e' : {'xy':( 0.249,  0.294), 'runits':'arcsec', 'mass':10,
               'renorm_args':(comp_mags[3], 'vegamag', bp_pl_norm)
              },
    },
    'params_disk_model' : None,
    'src_tbl' : None,
}
```

[11]:
```
# Reference source
targ_dict['HD220657'] = {
    'type' : 'FixedTargetType',
    'TargetName' : 'HD220657', 'TargetArchiveName' : 'ups Peg',
    'EquatorialCoordinates' : "23 25 22.7835 +23 24 14.76",
    'RAProperMotion'  : 192.19*u.mas/u.yr,
    'DecProperMotion' : 36.12*u.mas/u.yr,
    'parallax' : 19.14*u.mas, 'age_Myr' : None,
    'params_star' : {'sp' : sp_ref},
    'params_companions' : None,
    'params_disk_model' : None,
    'src_tbl' : None,
}
```

[12]:
```
# Populate coordinates and calculate distance from parallax info
for k in targ_dict.keys():
    d = targ_dict[k]
    dist = Distance(parallax=d['parallax']) if d['parallax'] is not None else None
    c = SkyCoord(d['EquatorialCoordinates'], frame='icrs', unit=(u.hourangle, u.deg),
                 pm_ra_cosdec=d['RAProperMotion'], pm_dec=d['DecProperMotion'],
                 distance=dist, obstime='J2000')
    d['sky_coords'] = c
    d['ra_J2000'], d['dec_J2000'] = (c.ra.deg, c.dec.deg)
    d['dist_pc'] = c.distance.value if dist is not None else None

    # Auto-generate source tables
    src_tbl = make_gaia_source_table(c)
    d['src_tbl'] = src_tbl if len(src_tbl)>0 else None
```

```
INFO: Query finished. [astroquery.utils.tap.core]
[astroquery:INFO] Query finished.
```

```
Filters:   0%|          | 0/29 [00:00<?, ?it/s]
```

```
INFO: Query finished. [astroquery.utils.tap.core]
[astroquery:INFO] Query finished.
```

```
Filters:   0%|          | 0/29 [00:00<?, ?it/s]
```

1.8.5 Create Observation Parameters

These will used for input into the data ramp simulator and directly correspond to parameters for DMS FITS creation to ingest into the JWST pipeline

[13]:
```
sim_config = {
    # APT input files
    'json_file'     : json_file,
    'sm_acct_file'  : sm_acct_file,
    'pointing_file' : pointing_file,
    'xml_file'      : xml_file,
    # Output directory
    'save_dir'      : save_dir,

    # Initialize random seeds if repeatability is required
    # Create separate random number generators for dithers and noise
    'rand_seed_init'  : 1234,

    # Date and time of observations
    'obs_date' : '2022-11-04',
    'obs_time' : '12:00:00',
    # Position angle of observatory
    # User should check acceptable range in APT's Roll Analysis
    'pa_v3'    : 90.0,

    # Source information
    'params_targets' : targ_dict,

    # PSF size information for WebbPSF_ext
    'params_webbpsf' : {'fov_pix': None, 'oversample': 2},
    # Position-dependent PSFs for convolution
    'params_psfconv' : {'npsf_per_full_fov': 9, 'osamp': 1, 'sptype': 'G0V'},
    # Wavefront error drift settings
    'params_wfedrift' : {'case': 'BOL', 'slew_init': 10, 'plot': False, 'figname': None},
    # For coronagraphic masks, sample large grid of points?
    'large_grid' : True,

    # Slew and dither pointing uncertainties
    'large_slew' : 100.0,   # Slew to target (mas)
    'ta_sam'     : 5.0,     # SAM movements from TA position (mas)
    'std_sam'    : 5.0,     # Standard dither values (mas)
    'sgd_sam'    : 2.5,     # Small grid dithers (mas)

    # Type of image files to save; can be supplied directly
    'save_slope' : False,  # Save ideal noiseless slope images to FITS
    'save_dms'   : False,  # Save DMS-like ramps to FITS
    'dry_run'    : False,  # Perform a dry-run, not generating any data, just printing␣
↪visit info

    # Noise components to include in full DMS output
    'params_noise' : {
        'include_poisson'   : True,     # Photon Noise
        'include_dark'      : True,     # Dark current
```

(continues on next page)

(continued from previous page)

```
        'include_bias'      : True,     # Bias image offset
        'include_ktc'       : True,     # kTC Noise
        'include_rn'        : True,     # Read Noise
        'include_cpink'     : True,     # Correlated 1/f noise between channel
        'include_upink'     : True,     # Channel-dependent 1/f noise
        'include_acn'       : True,     # Alternating column noise
        'apply_ipc'         : True,     # Interpixel capacitance
        'apply_ppc'         : True,     # Post-pixel coupling
        'amp_crosstalk'     : True,     # Amplifier crosstalk
        'include_refoffsets': True,     # Reference offsets
        'include_refinst'   : True,     # Reference pixel instabilities
        'include_colnoise'  : True,     # Transient detector column noise
        'add_crs'           : True,     # Include cosmic ray
        'cr_model'          : 'SUNMAX', # Cosmic ray model ('SUNMAX', 'SUNMIN', or 'FLARES')
        'cr_scale'          : 1,        # Cosmic ray probabilities scaling
        'apply_nonlinearity': True,     # Apply non-linearity
        'random_nonlin'     : True,     # Add randomness to non-linearity
        'apply_flats'       : True,     # pixel-to-pixel QE variations and field-dep␣
↪illum
    },
}
```

1.8.6 Perform simulations of observations

The next few cells demonstrate how to simulate observations for a single visit in the program (`001:001`). In order to generate a subset of observations, the `create_level1b_FITS` function has a few keyword settings, including `visit_id`, `apname`, `filter`, and `detname`.

In addition, there are some diagnostic-level options to ensure simulations are being generated as expected before going through the long process of making everything. makes sense during

- `dry_run`: Won't generate any image data, but instead runs through each observation, printing detector info, SIAF aperture name, filter, visit IDs, exposure numbers, and dither information. If set to `None`, then grabs keyword from `sim_config` argument, otherwise defaults to `False` if not specified. If paired with `save_dms`, then will generate an empty set of DMS FITS files with headers populated, but data set to all zeros.
- `save_slope`: Saves noiseless slope images to a separate DMS-like FITS file that is names `slope_{DMSfilename}`. If set to `None`, then grabs keyword from `sim_config`, otherwise defaults to False if not found. **Is effect if ``dry_run=True``.**
- `save_dms`: Option to disable simulation of ramp data and creation of DMS FITS. If `dry_run=True`, then setting `save_dms=True` will save DMS FITS files populated with all zeros. If set to `None`, then grabs keyword from `sim_config`; if no keyword is found, then defaults to `True` if `dry_run=False`, otherwise False.

[14]:
```
# Perform dry-run test first
# prints (detname, aperture, filter, target, visit, grp/seq/act id, expid, offset, dWFE,␣
↪dtime start)
create_level1b_FITS(sim_config, dry_run=True, save_slope=False, save_dms=False,
                    visit_id='001:001')
```

```
Obs Params:   0%|          | 0/70 [00:00<?, ?it/s]
```

```
Exposures:   0%|          | 0/5 [00:00<?, ?it/s]
```

```
NRCA5 NRCA5_FULL_MASK430R F335M HR8799 001:001 03104 1 (+0.003, -0.005) 1.84 2915
Exposures:   0%|          | 0/5 [00:00<?, ?it/s]
NRCA5 NRCA5_FULL_TAMASK430R F335M HR8799 001:001 03102 1 (+0.020, -0.017) 1.86 2720
Exposures:   0%|          | 0/5 [00:00<?, ?it/s]
NRCA5 NRCA5_MASK430R F356W HR8799 001:001 03106 1 (+0.003, -0.005) 1.92 3107
Exposures:   0%|          | 0/5 [00:00<?, ?it/s]
NRCA5 NRCA5_MASK430R F444W HR8799 001:001 03107 1 (+0.003, -0.005) 1.95 3706
Exposures:   0%|          | 0/5 [00:00<?, ?it/s]
NRCA5 NRCA5_TAMASK430R F335M HR8799 001:001 02102 1 (+0.020, -0.017) 1.86 2536
```

[ ]:
```
# Create ideal slope images, saved to Level-1b formation
# Will print messages:
#   Saving: slope_jw01194001001_03104_00001_nrca5_uncal.fits
create_level1b_FITS(sim_config, dry_run=False, save_slope=True, save_dms=False,
                    visit_id='001:001')
```

[ ]:
```
# Generate simulated Level-1b FITS files
# Will print messages:
#   Saving: pynrc_jw01194001001_03106_00001_nrca5_uncal.fits
create_level1b_FITS(sim_config, dry_run=False, save_slope=False, save_dms=True,
                    visit_id='001:001', apname='NRCA5_MASK430R')
```

[ ]:
```

```

1.9 Detailed API

1.9.1 Detector Classes

DetectorOps([detector, wind_mode, xpix, ...])	Class to hold detector operations information.
detops	

pynrc.DetectorOps

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 read_mode NIRCam Ramp Readout mode such as 'RAPID', 'BRIGHT1', etc.

kwtype read_mode str

keyword nint Number of integrations (ramps).

kwtype nint int

keyword ngroup Number of groups in a integration.

kwtype ngroup int

keyword nf Number of frames per group.

kwtype nf int

keyword nd1 Number of drop frame after reset (before first group read).

kwtype nd1 int

keyword nd2 Number of drop frames within a group (ie., groupgap).

kwtype nd2 int

keyword nd3 Number of drop frames after final read frame in ramp.

kwtype nd3 int

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)
```

Module Attributes

channel	Detector channel 'SW' or 'LW' (inferred from detector ID)
chsize	Size of Amplifier Channel
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.
fastaxis	Fast readout direction in sci coords
mask_act	Active pixel mask for det coordinates

continues on next page

Table 2 – continued from previous page

mask_channels	Channel masks for det coordinates
mask_ref	Reference pixel mask for det coordinates
module	NIRCam modules A or B (inferred from detector ID)
nff	Number of fast row resets that occur before Reset Frame
nout	Number of simultaneous detector output channels stripes
ref_info	Array of reference pixel borders [lower, upper, left, right].
scaid	Selected SCA ID from detectors in the *scaid_list* attribute.
scaid_list	Allowed SCA IDs
slowaxis	Slow readout direction in sci coords
time_exp	Total photon collection time for all ramps.
time_frame	Determine frame time (sec) 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_int_eff	Same as time_ramp_eff, except that 'int' follows the JWST nomenclature
time_ramp	Photon collection time for a single ramp.
time_ramp_eff	Effective ramp time for slope fit tf*(ng-1)
time_row_reset	NFF Row Resets time per integration
time_total	Total exposure acquisition time
time_total_int1	Total time for all frames in first ramp of exposure.
time_total_int2	Total time for all frames in a subsequent ramp.
times_group_avg	Times at each averaged group since reset
wind_mode	Window mode attribute
x0	
xpix	
y0	
ypix	

pynrc.DetectorOps.channel

property DetectorOps.**channel**

Detector channel 'SW' or 'LW' (inferred from detector ID)

pynrc.DetectorOps.chsize

property DetectorOps.**chsize**
Size of Amplifier Channel

pynrc.DetectorOps.detid

property DetectorOps.**detid**
Selected Detector ID from detectors in the *detid_list* attribute. A1, A2, etc.

pynrc.DetectorOps.detid_list

property DetectorOps.**detid_list**
Allowed Detector IDs

pynrc.DetectorOps.detname

property DetectorOps.**detname**
Selected Detector ID from detectors in the *scaid_list* attribute. NRCA1, NRCA2, etc.

pynrc.DetectorOps.fastaxis

property DetectorOps.**fastaxis**
Fast readout direction in sci coords

pynrc.DetectorOps.mask_act

property DetectorOps.**mask_act**
Active pixel mask for det coordinates

pynrc.DetectorOps.mask_channels

property DetectorOps.**mask_channels**
Channel masks for det coordinates

pynrc.DetectorOps.mask_ref

property DetectorOps.**mask_ref**
Reference pixel mask for det coordinates

pynrc.DetectorOps.module

property DetectorOps.**module**
NIRCam modules A or B (inferred from detector ID)

pynrc.DetectorOps.nff

property DetectorOps.**nff**
Number of fast row resets that occur before Reset Frame

pynrc.DetectorOps.nout

property DetectorOps.**nout**
Number of simultaneous detector output channels stripes

pynrc.DetectorOps.ref_info

property DetectorOps.**ref_info**
Array of reference pixel borders [lower, upper, left, right].

pynrc.DetectorOps.scaid

property DetectorOps.**scaid**
Selected SCA ID from detectors in the *scaid_list* attribute. 481, 482, etc.

pynrc.DetectorOps.scaid_list

property DetectorOps.**scaid_list**
Allowed SCA IDs

pynrc.DetectorOps.slowaxis

property DetectorOps.**slowaxis**
Slow readout direction in sci coords

pynrc.DetectorOps.time_exp

property DetectorOps.**time_exp**
Total photon collection time for all ramps.

pynrc.DetectorOps.time_frame

property DetectorOps.**time_frame**
Determine frame time (sec) based on xpix, ypix, and wind_mode.

pynrc.DetectorOps.time_group

property DetectorOps.**time_group**
Time per group based on time_frame, nf, and nd2.

pynrc.DetectorOps.time_int

property DetectorOps.**time_int**
Same as time_ramp, except that 'int' follows the JWST nomenclature

pynrc.DetectorOps.time_int_eff

property DetectorOps.**time_int_eff**
Same as time_ramp_eff, except that 'int' follows the JWST nomenclature

pynrc.DetectorOps.time_ramp

property DetectorOps.**time_ramp**
Photon collection time for a single ramp.

pynrc.DetectorOps.time_ramp_eff

property DetectorOps.**time_ramp_eff**
Effective ramp time for slope fit tf*(ng-1)

pynrc.DetectorOps.time_row_reset

property DetectorOps.**time_row_reset**
NFF Row Resets time per integration

pynrc.DetectorOps.time_total

property DetectorOps.**time_total**
Total exposure acquisition time

pynrc.DetectorOps.time_total_int1

property DetectorOps.**time_total_int1**
Total time for all frames in first ramp of exposure.

Includes resets and excess drops, as well as NFF Rows Reset.

pynrc.DetectorOps.time_total_int2

property DetectorOps.**time_total_int2**
Total time for all frames in a subsequent ramp.

Includes resets and excess drops, as well as NFF Rows Reset. Only differs from time_total_int1 in case nr1 != nr2

pynrc.DetectorOps.times_group_avg

property DetectorOps.**times_group_avg**
Times at each averaged group since reset

pynrc.DetectorOps.wind_mode

property DetectorOps.**wind_mode**
Window mode attribute

pynrc.DetectorOps.x0

property DetectorOps.**x0**

pynrc.DetectorOps.xpix

property DetectorOps.**xpix**

pynrc.DetectorOps.y0

property DetectorOps.**y0**

pynrc.DetectorOps.ypix

property DetectorOps.**ypix**

pynrc.detops

Functions

config2(input[, intype])	NIRCam CONFIG2 (0x4011) Register
create_detops(header[, DMS, read_mode, ...])	NIRCam Detector class from header
nrc_header(det_class[, filter, pupil, ...])	Simulated header
tuples_to_dict(pairs[, verbose])	Take a list of paired tuples and convert to a dictionary where the first element of each tuple is the key and the second element is the value.

pynrc.detops.config2

pynrc.detops.**config2**(*input*, *intype='int'*)

NIRCam CONFIG2 (0x4011) Register

Return a dictionary of configuration parameters depending on the value of CONFIG2 register (4011).

Parameters

- **input** (*int, str*) – Value of CONFIG2, nominally as an int. Binary and Hex values can also be passed as strings.
- **intype** (*str*) – Input type (int, hex, or bin) for integer, hex, string, or binary string.

pynrc.detops.create_detops

pynrc.detops.**create_detops**(*header*, *DMS=False*, *read_mode=None*, *nint=None*, *ngroup=None*, *detector=None*, *wind_mode=None*, *xpix=None*, *ypix=None*, *x0=None*, *y0=None*, *nff=None*)

NIRCam Detector class from header

Create a NIRCam detector class based on header settings. Can override settings with a variety of keyword arguments.

Parameters

- **header** (*obj*) – Header from NIRCam FITS file
- **DMS** (*bool*) – Is header format from Data Management Systems? Otherwises, ISIM-like.

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.
- **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.

pynrc.detops.nrc_header

pynrc.detops.**nrc_header**(*det_class*, *filter=None*, *pupil=None*, *obs_time=None*, *header=None*, *DMS=True*, *targ_name=None*)

Simulated header

Create a generic NIRCam FITS header from a detector_ops class.

Parameters

- **filter** (*str*) – Name of filter element.
- **pupil** (*str*) – Name of pupil element.
- **DMS** (*bool*) – Make the header in a format used by Data Management Systems.
- **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)
```

- **header** (*obj*) – Can pass an existing header that will be updated. This has not been fully tested.
- **targ_name** (*str*) – Standard astronomical catalog name for a target. Otherwise, it will be UNKNOWN.

pynrc.detops.tuples_to_dict

pynrc.detops.**tuples_to_dict**(*pairs*, *verbose=False*)

Take a list of paired tuples and convert to a dictionary where the first element of each tuple is the key and the second element is the value.

Classes

det_timing([wind_mode, xpix, ypix, x0, y0, ...])	Class to hold detector operations information.
multiaccum([read_mode, nint, ngroup, nf, ...])	A class for defining MULTIACCUM ramp settings.

pynrc.detops.det_timing

class pynrc.detops.**det_timing**(*wind_mode='FULL'*, *xpix=2048*, *ypix=2048*, *x0=0*, *y0=0*, *mode='SLOW'*, *nff=None*, ***kwargs*)

Bases: `object`

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

Parameters

- **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.

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 = det_timing(**kwargs)
```

Set the keywords directly:

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

__init__(*wind_mode='FULL'*, *xpix=2048*, *ypix=2048*, *x0=0*, *y0=0*, *mode='SLOW'*, *nff=None*, ***kwargs*)

Methods

__init__([wind_mode, xpix, ypix, x0, y0, ...])	

int_times_table(date_start, time_start[, ...])	Create and populate the INT_TIMES table, which is saved as a separate extension in the output data file.
pix_timing_map([same_scan_direction, ...])	Create array of pixel times for a single ramp.
pixel_noise([ng, nf, verbose])	Noise values per pixel.
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.

Attributes

chsize	Size of Amplifier Channel
mask_act	Active pixel mask for det coordinates
mask_channels	Channel masks for det coordinates
mask_ref	Reference pixel mask for det coordinates
nff	Number of fast row resets that occur before Reset Frame
nout	Number of simultaneous detector output channels stripes
ref_info	Array of reference pixel borders [lower, upper, left, right].
time_exp	Total photon collection time for all ramps.
time_frame	Determine frame time (sec) 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_int_eff	Same as time_ramp_eff, except that 'int' follows the JWST nomenclature
time_ramp	Photon collection time for a single ramp.
time_ramp_eff	Effective ramp time for slope fit tf*(ng-1)
time_row_reset	NFF Row Resets time per integration
time_total	Total exposure acquisition time
time_total_int1	Total time for all frames in first ramp of exposure.
time_total_int2	Total time for all frames in a subsequent ramp.
times_group_avg	Times at each averaged group since reset
wind_mode	Window mode attribute
x0	
xpix	
y0	
ypix	

property chsize
Size of Amplifier Channel

int_times_table(*date_start*, *time_start*, *offset_seconds=None*)
Create and populate the INT_TIMES table, which is saved as a separate extension in the output data file.

Parameters

- **date_start** (*str*) – Date string of observation ('2020-02-28')
- **time_start** (*str*) – Time string of observation ('12:24:56')
- **offset_seconds** (*None or float*) – Time from beginning of observation until start of integration.

Returns **int_times_tab** (*astropy.table.Table*) – Table of starting, mid, and end times for each integration

property mask_act
Active pixel mask for det coordinates

property mask_channels
Channel masks for det coordinates

property mask_ref
Reference pixel mask for det coordinates

property nff
Number of fast row resets that occur before Reset Frame

property nout
Number of simultaneous detector output channels stripes

pix_timing_map(*same_scan_direction=None*, *reverse_scan_direction=None*, *avg_groups=False*, *reset_zero=False*, *return_flat=False*)
Create array of pixel times for a single ramp.

Each pixel value corresponds to the precise time at which that pixel was read out during the ramp acquisition. The first pixel(s) have t=0.

Parameters

- **same_scan_direction** (*bool*) – Are all the output channels read in the same direction? By default fast-scan readout direction is `[-->,<--,-->,<--]` If `same_scan_direction`, then all `-->`
- **reverse_scan_direction** (*bool*) – If `reverse_scan_direction`, then `[<--,-->,<--,-->]` or all `<--`
- **avg_groups** (*bool*) – For groups where nf>1, the telescope data gets averaged via a bit-shifter. Setting `avg_groups=True` also averages the pixel times in a similar manner. Default is True.
- **return_flat** (*bool*)

Keyword Arguments **reset_zero** (*bool*) – Return timing relative to when reset to get photon-collection time of each pixel. Otherwise, t=0 corresponds to very first pixel(s) read in the ramp.

Returns *ndarray* – If `return_flat=True` then the data is a flattened array for a single channel output. Otherwise, the output is a data cube of the same size and shape of the raw data with these detector settings.

Example

Assume you have a cube of raw full frame data (RAPID, ngroup=5). Create a det_timing instance and get channel:

```
>>> d = det_timing(ngroup=5)
>>> tarr = d.pixel_timing_map(return_flat=True, avg_groups=True)
```

```
>>> nx, ny = (d.xpix, d.ypix)
>>> nout   = d.nout      # Number of amplifier output channels
>>> chsize = d.chsize    # Channel size (x-direction)
>>> # Reshape into (nz, ny, nout, chsize)
>>> data = data.reshape([-1,ny,nout,chsize])
>>> # Reverse odd channels in x-direction to match even chans
```

(continues on next page)

(continued from previous page)

```
>>> for ch in range(nout):
>>>     if np.mod(ch,2)==1:
>>>         data[:,:,ch,:] = data[:,:,ch,::-1]
>>> # Final data reshaped into 4 flattened output channels
>>> data = data.transpose([0,1,3,2]).reshape([-1,nout])
>>> # Can plot this like plt.plot(tarr, data) to make nout line plots
```

pixel_noise(*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

- **ng** (*None or int or image*) – Option to explicitly state 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.multiaccum.ngroup*.
- **nf** (*int*) – Option to explicitly states number of frames in each group. By default will use *self.multiaccum.nf*.
- **verbose** (*bool*) – Print out results at the end.

Keyword Arguments

- **rn** (*float*) – Read Noise per pixel (e-).
- **ktc** (*float*) – kTC noise (in e-). Only valid for single frame (n=1)
- **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.
- **idark** (*float*) – Dark current in e-/sec/pix.
- **fsrc** (*float*) – Flux of source 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.

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.

property ref_info

Array of reference pixel borders [lower, upper, left, right].

property time_exp

Total photon collection time for all ramps.

property time_frame
Determine frame time (sec) based on xpix, ypix, and wind_mode.

property time_group
Time per group based on time_frame, nf, and nd2.

property time_int
Same as time_ramp, except that 'int' follows the JWST nomenclature

property time_int_eff
Same as time_ramp_eff, except that 'int' follows the JWST nomenclature

property time_ramp
Photon collection time for a single ramp.

property time_ramp_eff
Effective ramp time for slope fit tf*(ng-1)

property time_row_reset
NFF Row Resets time per integration

property time_total
Total exposure acquisition time

property time_total_int1
Total time for all frames in first ramp of exposure.

Includes resets and excess drops, as well as NFF Rows Reset.

property time_total_int2
Total time for all frames in a subsequent ramp.

Includes resets and excess drops, as well as NFF Rows Reset. Only differs from time_total_int1 in case nr1 != nr2

property times_group_avg
Times at each averaged group since reset

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.

property wind_mode
Window mode attribute

pynrc.detops.multiaccum

class pynrc.detops.**multiaccum**(*read_mode='RAPID'*, *nint=1*, *ngroup=1*, *nf=1*, *nd1=0*, *nd2=0*, *nd3=0*, *nr1=1*, *nr2=1*, *wind_mode='FULL'*, ***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.
- **wind_mode** (*str*) – Set to determine maximum number of allowed groups.

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

__init__(*read_mode='RAPID'*, *nint=1*, *ngroup=1*, *nf=1*, *nd1=0*, *nd2=0*, *nd3=0*, *nr1=1*, *nr2=1*, *wind_mode='FULL'*, ***kwargs*)

Methods

__init__([read_mode, nint, ngroup, nf, nd1, ...])	
to_dict([verbose])	Export ramp settings to a dictionary.

Attributes

nd1	Number of drop frame after reset (before first group read).
nd2	Number of drop frames within a group (aka, groupgap).
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.
nr1	Number of reset frames before first integration.

continues on next page

Table 8 – continued from previous page

nr2	Number of reset frames for subsequent integrations.
nread_tot	Total number of read frames in a ramp, including drops
patterns_list	Allowed NIRCam MULTIACCUM patterns
read_mode	Selected Read Mode in the *patterns_list* attribute.

property nd1
Number of drop frame after reset (before first group read).

property nd2
Number of drop frames within a group (aka, groupgap).

property nd3
Number of drop frames after final read frame in ramp.

property nf
Number of frames per group.

property ngroup
Number of groups in a ramp (integration).

property nint
Number of ramps (integrations) in an exposure.

property nr1
Number of reset frames before first integration.

property nr2
Number of reset frames for subsequent integrations.

property nread_tot
Total number of read frames in a ramp, including drops

property patterns_list
Allowed NIRCam MULTIACCUM patterns

property read_mode
Selected Read Mode in the *patterns_list* attribute.

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

1.9.2 Observation Classes

NIRCam([filter, pupil_mask, image_mask, ...])	NIRCam base instrument class
nrc_hci([wind_mode, xpix, ypix, large_grid, ...])	NIRCam coronagraphy (and direct imaging)
obs_hci(sp_sci, distance[, sp_ref, ...])	NIRCam coronagraphic observations

pynrc.NIRCam

NIRCam base instrument class

Creates a NIRCam instrument class that holds all the information pertinent to an observation using a given observation. This class extends the NIRCam subclass `webbpsf_ext.NIRCam_ext`, to generate PSF coefficients to calculate an arbitrary PSF based on wavelength, field position, and WFE drift.

In addition to PSF generation, includes ability to estimate detector saturation limits, sensitivities, and perform ramp optimizations.

Parameters

- **filter** (*str*) – Name of input filter.
- **pupil_mask** (*str, None*) – Pupil elements such as grisms or lyot stops (default: None).
- **image_mask** (*str, None*) – Specify which coronagraphic occulter (default: None).
- **ND_acq** (*bool*) – Add in neutral density attenuation in throughput and PSF creation? Used primarily for sensitivity and saturation calculations. Not recommended for simulations (TBI).
- **detector** (*int or str*) – NRC[A-B][1-5] or 481-490
- **apname** (*str*) – Pass specific SIAF aperture name, which will update pupil mask, image mask, and detector subarray information.
- **autogen_coeffs** (*bool*) – Automatically generate base PSF coefficients. Equivalent to performing `self.gen_psf_coeff()`. Default: True WFE drift and field-dependent coefficients should be run manually via `gen_wfedrift_coeff`, `gen_wfefield_coeff`, and `gen_wfemask_coeff`.

keyword wind_mode Window mode type ‘FULL’, ‘STRIPE’, ‘WINDOW’.

kwtype wind_mode str

keyword xpix Size of window in x-pixels for frame time calculation.

kwtype xpix int

keyword ypix Size of window in y-pixels for frame time calculation.

kwtype ypix int

keyword x0 Lower-left x-coord position of detector window.

kwtype x0 int

keyword y0 Lower-left y-coord position of detector window.

kwtype y0 int

keyword read_mode NIRCam Ramp Readout mode such as ‘RAPID’, ‘BRIGHT1’, etc.

kwtype read_mode str

keyword nint Number of integrations (ramps).

kwtype nint int

keyword ngroup Number of groups in a integration.

kwtype ngroup int

keyword nf Number of frames per group.

kwtype nf int

keyword nd1 Number of drop frame after reset (before first group read).

kwtype nd1 int

keyword nd2 Number of drop frames within a group (ie., groupgap).

kwtype nd2 int

keyword nd3 Number of drop frames after final read frame in ramp.

kwtype nd3 int

keyword nr1 Number of reset frames within first ramp.

kwtype nr1 int

keyword nr2 Number of reset frames for subsequent ramps.

kwtype nr2 int

keyword PSF Keywords

keyword ============

keyword fov_pix 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."

kwtype fov_pix int

keyword oversample Factor to oversample during WebbPSF calculations. Default 2 for coronagraphy and 4 otherwise.

kwtype oversample int

keyword include_si_wfe Include SI WFE measurements? Default=True.

kwtype include_si_wfe bool

keyword include_distortions If True, will include a distorted version of the PSF.

kwtype include_distortions bool

keyword pupil File name or HDUList specifying telescope entrance pupil. Can also be an OTE_Linear_Model.

kwtype pupil str

keyword pupilopd Tuple (file, index) or filename or HDUList specifying OPD. Can also be an OTE_Linear_Model.

kwtype pupilopd tuple or HDUList

keyword wfe_drift Wavefront error drift amplitude in nm.

kwtype wfe_drift float

keyword offset_r Radial offset from the center in arcsec.

kwtype offset_r float

keyword offset_theta Position angle for radial offset, in degrees CCW.

kwtype offset_theta float

keyword bar_offset For wedge masks, option to set the PSF position across the bar.

kwtype bar_offset float

keyword jitter Currently either 'gaussian' or None.

kwtype jitter str or None

keyword jitter_sigma If `jitter = 'gaussian'`, then this is the size of the blurring effect.

kwtype jitter_sigma float

keyword npsf Number of wavelengths/PSFs to fit.

kwtype npsf int

keyword ndeg Degree of polynomial fit.

kwtype ndeg int

keyword nproc Manual setting of number of processor cores to break up PSF calculation. If set to None, this is determined based on the requested PSF size, number of available memory, and hardware processor cores. The automatic calculation endeavors to leave a number of resources available to the user so as to not crash the user's machine.

kwtype nproc int

keyword save Save the resulting PSF coefficients to a file? (default: True)

kwtype save bool

keyword force Forces a recalculation of PSF even if saved PSF exists. (default: False)

kwtype force bool

keyword quick Only perform a fit over the filter bandpass with a lower default polynomial degree fit. (default: True)

kwtype quick bool

keyword use_legendre Fit with Legendre polynomials, an orthonormal basis set. (default: True)

kwtype use_legendre bool

Module Attributes

LONG_WAVELENGTH_MAX	
LONG_WAVELENGTH_MIN	
ND_acq	Use Coronagraphic ND acquisition square?
SHORT_WAVELENGTH_MAX	
SHORT_WAVELENGTH_MIN	
aperturename	SIAF aperture name for detector pixel to sky coords transformations
bandpass	Return bandpass throughput
channel	
coron_substrate	Include coronagraphic substrate material?
det_info	Dictionary housing detector info parameters and keywords.
detector	Detector selected for simulated PSF
detector_list	Detectors on which the simulated PSF could lie
detector_position	The pixel position in (X, Y) on the detector, relative to the currently-selected SIAF aperture subarray.

continues on next page

Table 10 – continued from previous page

fastaxis	Fast readout direction in sci coords
filter	Currently selected filter name (e.g.
filter_list	List of available filter names for this instrument
fov_pix	
image_mask	Currently selected image plane mask, or None for direct imaging
is_coron	Observation with coronagraphic mask (incl Lyot stop)?
is_dark	
is_grism	
is_lyot	Is a Lyot mask in the pupil wheel?
module	
multiaccum	*multiaccum* object
multiaccum_times	Exposure timings in dictionary
name	
ndeg	
npsf	Number of wavelengths/PSFs to fit
options	A dictionary capable of storing other arbitrary options, for extensibility.
oversample	
pixelscale	Detector pixel scale, in arcsec/pixel
psf_info	PSF parameters
pupil	Filename *or* fits.HDUList for JWST pupil mask.
pupil_mask	Currently selected Lyot pupil mask, or None for direct imaging
pupilopd	Filename *or* fits.HDUList for JWST pupil OPD.
quick	Perform quicker coeff calculation over limited bandwidth?
save_dir	Coefficient save directory
save_name	Coefficient file name
scaid	SCA ID (481, 482, .
siaf_ap	SIAF Aperture object
siaf_ap_names	Give all possible SIAF aperture names
slowaxis	Slow readout direction in sci coords
telescope	
wave_fit	Wavelength range to fit
well_level	Detector well level in units of electrons

pynrc.NIRCam.LONG_WAVELENGTH_MAX

NIRCam.**LONG_WAVELENGTH_MAX = 5.299999999999999e-06**

pynrc.NIRCam.LONG_WAVELENGTH_MIN

NIRCam.**LONG_WAVELENGTH_MIN = 2.35e-06**

pynrc.NIRCam.ND_acq

property NIRCam.**ND_acq**
Use Coronagraphic ND acquisition square?

pynrc.NIRCam.SHORT_WAVELENGTH_MAX

NIRCam.**SHORT_WAVELENGTH_MAX = 2.35e-06**

pynrc.NIRCam.SHORT_WAVELENGTH_MIN

NIRCam.**SHORT_WAVELENGTH_MIN = 6e-07**

pynrc.NIRCam.aperturename

property NIRCam.**aperturename**
SIAF aperture name for detector pixel to sky coords transformations

pynrc.NIRCam.bandpass

property NIRCam.**bandpass**
Return bandpass throughput

pynrc.NIRCam.channel

property NIRCam.**channel**

pynrc.NIRCam.coron_substrate

property NIRCam.**coron_substrate**
Include coronagraphic substrate material?

pynrc.NIRCam.det_info

property NIRCam.**det_info**
Dictionary housing detector info parameters and keywords.

pynrc.NIRCam.detector

property NIRCam.**detector**
Detector selected for simulated PSF

Used in calculation of field-dependent aberrations. Must be selected from detectors in the *detector_list* attribute.

pynrc.NIRCam.detector_list

property NIRCam.**detector_list**
Detectors on which the simulated PSF could lie

pynrc.NIRCam.detector_position

property NIRCam.**detector_position**
The pixel position in (X, Y) on the detector, relative to the currently-selected SIAF aperture subarray. By default the SIAF aperture will correspond to the full-frame detector, so (X,Y) will in that case be absolute (X,Y) pixels on the detector. But if you select a subarray aperture name from the SIAF, then the (X,Y) are interpreted as (X,Y) within that subarray.

Please note, this is X,Y order - **not** a Pythonic y,x axes ordering.

pynrc.NIRCam.fastaxis

property NIRCam.**fastaxis**
Fast readout direction in sci coords

pynrc.NIRCam.filter

property NIRCam.**filter**
Currently selected filter name (e.g. F200W)

pynrc.NIRCam.filter_list

NIRCam.**filter_list = None**
List of available filter names for this instrument

pynrc.NIRCam.fov_pix

property NIRCam.**fov_pix**

pynrc.NIRCam.image_mask

property NIRCam.**image_mask**
Currently selected image plane mask, or None for direct imaging

pynrc.NIRCam.is_coron

property NIRCam.**is_coron**
Observation with coronagraphic mask (incl Lyot stop)?

pynrc.NIRCam.is_dark

property NIRCam.**is_dark**

pynrc.NIRCam.is_grism

property NIRCam.**is_grism**

pynrc.NIRCam.is_lyot

property NIRCam.**is_lyot**
Is a Lyot mask in the pupil wheel?

pynrc.NIRCam.module

property NIRCam.**module**

pynrc.NIRCam.multiaccum

property NIRCam.**multiaccum**
multiaccum object

pynrc.NIRCam.multiaccum_times

property NIRCam.**multiaccum_times**
Exposure timings in dictionary

t_frame : Time of a single frame. t_group : Time of a single group (read frames + drop frames). t_int : Photon collection time for a single ramp/integration. t_int_tot1: Total time for all frames (reset+read+drop) in a first ramp. t_int_tot2: Total time for all frames (reset+read+drop) in a subsequent ramp. t_exp : Total photon collection time for all ramps. t_acq : Total acquisition time to complete exposure with all overheads.

pynrc.NIRCam.name

NIRCam.**name** = 'Instrument'

pynrc.NIRCam.ndeg

property NIRCam.**ndeg**

pynrc.NIRCam.npsf

property NIRCam.**npsf**
Number of wavelengths/PSFs to fit

pynrc.NIRCam.options

NIRCam.**options** = {}
A dictionary capable of storing other arbitrary options, for extensibility. The following are all optional, and may or may not be meaningful depending on which instrument is selected.

This is a superset of the options provided in `poppy.Instrument.options`.

Parameters

- **source_offset_r** (*float*) – Radial offset of the target from the center, in arcseconds
- **source_offset_theta** (*float*) – Position angle for that offset, in degrees CCW.
- **pupil_shift_x, pupil_shift_y** (*float*) – Relative shift of the intermediate (coronagraphic) pupil in X and Y relative to the telescope entrance pupil, expressed as a decimal between -1.0-1.0 Note that shifting an array too much will wrap around to the other side unphysically, but for reasonable values of shift this is a non-issue. This option only has an effect for optical models that have something at an intermediate pupil plane between the telescope aperture and the detector.
- **pupil_rotation** (*float*) – Relative rotation of the intermediate (coronagraphic) pupil relative to the telescope entrace pupil, expressed in degrees counterclockwise. This option only has an effect for optical models that have something at an intermediate pupil plane between the telescope aperture and the detector.
- **rebin** (*bool*) – For output files, write an additional FITS extension including a version of the output array rebinned down to the actual detector pixel scale?
- **jitter** (*string "gaussian" or None*) – Type of jitter model to apply. Currently only convolution with a Gaussian kernel of specified width *jitter_sigma* is implemented. (default: None)
- **jitter_sigma** (*float*) – Width of the jitter kernel in arcseconds (default: 0.006 arcsec, 1 sigma per axis)
- **parity** (*string "even" or "odd"*) – You may wish to ensure that the output PSF grid has either an odd or even number of pixels. Setting this option will force that to be the case by increasing npix by one if necessary. Note that this applies to the number detector pixels, rather than the subsampled pixels if oversample > 1.

- **force_coron** (*bool*) – Set this to force full coronagraphic optical propagation when it might not otherwise take place (e.g. calculate the non-coronagraphic images via explicit propagation to all optical surfaces, FFTing to intermediate pupil and image planes whether or not they contain any actual optics, rather than taking the straight-to-MFT shortcut)
- **no_sam** (*bool*) – Set this to prevent the SemiAnalyticMethod coronagraph mode from being used when possible, and instead do the brute-force FFT calculations. This is usually not what you want to do, but is available for comparison tests. The SAM code will in general be much faster than the FFT method, particularly for high oversampling.

pynrc.NIRCam.oversample

property NIRCam.**oversample**

pynrc.NIRCam.pixelscale

NIRCam.**pixelscale = 0.025**
Detector pixel scale, in arcsec/pixel

pynrc.NIRCam.psf_info

property NIRCam.**psf_info**
PSF parameters

pynrc.NIRCam.pupil

NIRCam.**pupil = None**
Filename *or* fits.HDUList for JWST pupil mask. Usually there is no need to change this.

pynrc.NIRCam.pupil_mask

property NIRCam.**pupil_mask**
Currently selected Lyot pupil mask, or None for direct imaging

pynrc.NIRCam.pupilopd

NIRCam.**pupilopd = None**
Filename *or* fits.HDUList for JWST pupil OPD.

This can be either a full absolute filename, or a relative name in which case it is assumed to be within the instrument's *data/OPDs/* directory, or an actual fits.HDUList object corresponding to such a file. If the file contains a datacube, you may set this to a tuple (filename, slice) to select a given slice, or else the first slice will be used.

pynrc.NIRCam.quick

property NIRCam.**quick**
Perform quicker coeff calculation over limited bandwidth?

pynrc.NIRCam.save_dir

property NIRCam.**save_dir**
Coefficient save directory

pynrc.NIRCam.save_name

property NIRCam.**save_name**
Coefficient file name

pynrc.NIRCam.scaid

property NIRCam.**scaid**
SCA ID (481, 482, … 489, 490)

pynrc.NIRCam.siaf_ap

property NIRCam.**siaf_ap**
SIAF Aperture object

pynrc.NIRCam.siaf_ap_names

property NIRCam.**siaf_ap_names**
Give all possible SIAF aperture names

pynrc.NIRCam.slowaxis

property NIRCam.**slowaxis**
Slow readout direction in sci coords

pynrc.NIRCam.telescope

NIRCam.**telescope = 'JWST'**

pynrc.NIRCam.wave_fit

property NIRCam.**wave_fit**
Wavelength range to fit

pynrc.NIRCam.well_level

property NIRCam.**well_level**
Detector well level in units of electrons

pynrc.nrc_hci

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 effectively 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'.
- **large_grid** (*bool*) – Use a large number (high-density) of grid points to create coefficients. If True, then produces a higher fidelity PSF variations across the FoV, but will take much longer to genrate on the first pass and requires more disk space and memory while running.
- **bar_offset** (*float*) – Custom offset position along bar mask (-10 to +10 arcsec).
- **use_ap_info** (*bool*) – For subarray observations, the mask reference points are not actually in the center of the array. Set this to true to shift the sources to actual aperture reference location. Default is to place in center of array.
- **autogen_coeffs** (*bool*) – Automatically generate base PSF coefficients. Equivalent to performing *self.gen_psf_coeff()*. *gen_wfedrift_coeff*, and *gen_wfemask_coeff*. Default: True.
- **sgd_type** (*str or None*) – Small grid dither pattern. Valid types are '9circle', '5box', '5diamond', '3bar', or '5bar'. If 'auto', then defaults are '5diamond' for round masks, '5bar' for bar masks, and '5diamond' for direct imaging. If None, then no FSM pointings, but there will be a single slew.
- **fsm_std** (*float*) – One-sigma accuracy per axis of fine steering mirror positions. This provides randomness to each position relative to the nominal central position. Ignored for central position. Values should be in units of mas.
- **slew_std** (*float*) – One-sigma accuracy per axis of the initial slew. This is applied to all positions and gives a baseline offset relative to the desired mask center. ***Values should be in units of mas***

Module Attributes

LONG_WAVELENGTH_MAX	
LONG_WAVELENGTH_MIN	
ND_acq	Use Coronagraphic ND acquisition square?
SHORT_WAVELENGTH_MAX	
SHORT_WAVELENGTH_MIN	
aperturename	SIAF aperture name for detector pixel to sky coords transformations
bandpass	Return bandpass throughput
bar_offset	Offset position along bar mask (arcsec).
channel	
coron_substrate	Include coronagraphic substrate material?
det_info	Dictionary housing detector info parameters and key-words.
detector	Detector selected for simulated PSF
detector_list	Detectors on which the simulated PSF could lie
detector_position	The pixel position in (X, Y) on the detector, relative to the currently-selected SIAF aperture subarray.
fastaxis	Fast readout direction in sci coords
filter	Currently selected filter name (e.g.
filter_list	List of available filter names for this instrument
fov_pix	
image_mask	Currently selected image plane mask, or None for direct imaging
is_coron	Observation with coronagraphic mask (incl Lyot stop)?
is_dark	
is_grism	
is_lyot	Is a Lyot mask in the pupil wheel?
module	
multiaccum	*multiaccum* object
multiaccum_times	Exposure timings in dictionary
name	
ndeg	
npsf	Number of wavelengths/PSFs to fit
options	A dictionary capable of storing other arbitrary options, for extensibility.
oversample	
pixelscale	Detector pixel scale, in arcsec/pixel

continues on next page

Table 11 – continued from previous page

psf_info	PSF parameters
pupil	Filename *or* fits.HDUList for JWST pupil mask.
pupil_mask	Currently selected Lyot pupil mask, or None for direct imaging
pupilopd	Filename *or* fits.HDUList for JWST pupil OPD.
quick	Perform quicker coeff calculation over limited bandwidth?
save_dir	Coefficient save directory
save_name	Coefficient file name
scaid	SCA ID (481, 482, .
siaf_ap	SIAF Aperture object
siaf_ap_names	Give all possible SIAF aperture names
slowaxis	Slow readout direction in sci coords
telescope	
wave_fit	Wavelength range to fit
well_level	Detector well level in units of electrons

pynrc.nrc_hci.LONG_WAVELENGTH_MAX

nrc_hci.**LONG_WAVELENGTH_MAX = 5.299999999999999e-06**

pynrc.nrc_hci.LONG_WAVELENGTH_MIN

nrc_hci.**LONG_WAVELENGTH_MIN = 2.35e-06**

pynrc.nrc_hci.ND_acq

property nrc_hci.**ND_acq**
Use Coronagraphic ND acquisition square?

pynrc.nrc_hci.SHORT_WAVELENGTH_MAX

nrc_hci.**SHORT_WAVELENGTH_MAX = 2.35e-06**

pynrc.nrc_hci.SHORT_WAVELENGTH_MIN

nrc_hci.**SHORT_WAVELENGTH_MIN = 6e-07**

pynrc.nrc_hci.aperturename

property nrc_hci.**aperturename**
SIAF aperture name for detector pixel to sky coords transformations

pynrc.nrc_hci.bandpass

property nrc_hci.**bandpass**
Return bandpass throughput

pynrc.nrc_hci.bar_offset

property nrc_hci.**bar_offset**
Offset position along bar mask (arcsec).

pynrc.nrc_hci.channel

property nrc_hci.**channel**

pynrc.nrc_hci.coron_substrate

property nrc_hci.**coron_substrate**
Include coronagraphic substrate material?

pynrc.nrc_hci.det_info

property nrc_hci.**det_info**
Dictionary housing detector info parameters and keywords.

pynrc.nrc_hci.detector

property nrc_hci.**detector**
Detector selected for simulated PSF

Used in calculation of field-dependent aberrations. Must be selected from detectors in the *detector_list* attribute.

pynrc.nrc_hci.detector_list

property nrc_hci.**detector_list**
Detectors on which the simulated PSF could lie

pynrc.nrc_hci.detector_position

property nrc_hci.**detector_position**
The pixel position in (X, Y) on the detector, relative to the currently-selected SIAF aperture subarray. By default the SIAF aperture will correspond to the full-frame detector, so (X,Y) will in that case be absolute (X,Y) pixels on the detector. But if you select a subarray aperture name from the SIAF, then the (X,Y) are interpreted as (X,Y) within that subarray.

Please note, this is X,Y order - **not** a Pythonic y,x axes ordering.

pynrc.nrc_hci.fastaxis

property nrc_hci.**fastaxis**
Fast readout direction in sci coords

pynrc.nrc_hci.filter

property nrc_hci.**filter**
Currently selected filter name (e.g. F200W)

pynrc.nrc_hci.filter_list

nrc_hci.**filter_list = None**
List of available filter names for this instrument

pynrc.nrc_hci.fov_pix

property nrc_hci.**fov_pix**

pynrc.nrc_hci.image_mask

property nrc_hci.**image_mask**
Currently selected image plane mask, or None for direct imaging

pynrc.nrc_hci.is_coron

property nrc_hci.**is_coron**
Observation with coronagraphic mask (incl Lyot stop)?

pynrc.nrc_hci.is_dark

property nrc_hci.**is_dark**

pynrc.nrc_hci.is_grism

property nrc_hci.**is_grism**

pynrc.nrc_hci.is_lyot

property nrc_hci.**is_lyot**
Is a Lyot mask in the pupil wheel?

pynrc.nrc_hci.module

property nrc_hci.**module**

pynrc.nrc_hci.multiaccum

property nrc_hci.**multiaccum**
multiaccum object

pynrc.nrc_hci.multiaccum_times

property nrc_hci.**multiaccum_times**
Exposure timings in dictionary

t_frame : Time of a single frame. t_group : Time of a single group (read frames + drop frames). t_int : Photon collection time for a single ramp/integration. t_int_tot1: Total time for all frames (reset+read+drop) in a first ramp. t_int_tot2: Total time for all frames (reset+read+drop) in a subsequent ramp. t_exp : Total photon collection time for all ramps. t_acq : Total acquisition time to complete exposure with all overheads.

pynrc.nrc_hci.name

nrc_hci.**name** = 'Instrument'

pynrc.nrc_hci.ndeg

property nrc_hci.**ndeg**

pynrc.nrc_hci.npsf

property nrc_hci.**npsf**
Number of wavelengths/PSFs to fit

pynrc.nrc_hci.options

nrc_hci.**options = {}**
A dictionary capable of storing other arbitrary options, for extensibility. The following are all optional, and may or may not be meaningful depending on which instrument is selected.

This is a superset of the options provided in `poppy.Instrument.options`.

Parameters

- **source_offset_r** (*float*) – Radial offset of the target from the center, in arcseconds
- **source_offset_theta** (*float*) – Position angle for that offset, in degrees CCW.
- **pupil_shift_x, pupil_shift_y** (*float*) – Relative shift of the intermediate (coronagraphic) pupil in X and Y relative to the telescope entrance pupil, expressed as a decimal between -1.0-1.0 Note that shifting an array too much will wrap around to the other side unphysically, but for reasonable values of shift this is a non-issue. This option only has an effect for optical models that have something at an intermediate pupil plane between the telescope aperture and the detector.
- **pupil_rotation** (*float*) – Relative rotation of the intermediate (coronagraphic) pupil relative to the telescope entrace pupil, expressed in degrees counterclockwise. This option only has an effect for optical models that have something at an intermediate pupil plane between the telescope aperture and the detector.
- **rebin** (*bool*) – For output files, write an additional FITS extension including a version of the output array rebinned down to the actual detector pixel scale?
- **jitter** (*string "gaussian" or None*) – Type of jitter model to apply. Currently only convolution with a Gaussian kernel of specified width *jitter_sigma* is implemented. (default: None)
- **jitter_sigma** (*float*) – Width of the jitter kernel in arcseconds (default: 0.006 arcsec, 1 sigma per axis)
- **parity** (*string "even" or "odd"*) – You may wish to ensure that the output PSF grid has either an odd or even number of pixels. Setting this option will force that to be the case by increasing npix by one if necessary. Note that this applies to the number detector pixels, rather than the subsampled pixels if oversample > 1.
- **force_coron** (*bool*) – Set this to force full coronagraphic optical propagation when it might not otherwise take place (e.g. calculate the non-coronagraphic images via explicit propagation to all optical surfaces, FFTing to intermediate pupil and image planes whether or not they contain any actual optics, rather than taking the straight-to-MFT shortcut)
- **no_sam** (*bool*) – Set this to prevent the SemiAnalyticMethod coronagraph mode from being used when possible, and instead do the brute-force FFT calculations. This is usually not what you want to do, but is available for comparison tests. The SAM code will in general be much faster than the FFT method, particularly for high oversampling.

pynrc.nrc_hci.oversample

property nrc_hci.**oversample**

pynrc.nrc_hci.pixelscale

nrc_hci.**pixelscale = 0.025**
Detector pixel scale, in arcsec/pixel

pynrc.nrc_hci.psf_info

property nrc_hci.**psf_info**
PSF parameters

pynrc.nrc_hci.pupil

nrc_hci.**pupil = None**
Filename *or* fits.HDUList for JWST pupil mask. Usually there is no need to change this.

pynrc.nrc_hci.pupil_mask

property nrc_hci.**pupil_mask**
Currently selected Lyot pupil mask, or None for direct imaging

pynrc.nrc_hci.pupilopd

nrc_hci.**pupilopd = None**
Filename *or* fits.HDUList for JWST pupil OPD.

This can be either a full absolute filename, or a relative name in which case it is assumed to be within the instrument's *data/OPDs/* directory, or an actual fits.HDUList object corresponding to such a file. If the file contains a datacube, you may set this to a tuple (filename, slice) to select a given slice, or else the first slice will be used.

pynrc.nrc_hci.quick

property nrc_hci.**quick**
Perform quicker coeff calculation over limited bandwidth?

pynrc.nrc_hci.save_dir

property nrc_hci.**save_dir**
Coefficient save directory

pynrc.nrc_hci.save_name

property nrc_hci.**save_name**
Coefficient file name

pynrc.nrc_hci.scaid

property nrc_hci.**scaid**
SCA ID (481, 482, … 489, 490)

pynrc.nrc_hci.siaf_ap

property nrc_hci.**siaf_ap**
SIAF Aperture object

pynrc.nrc_hci.siaf_ap_names

property nrc_hci.**siaf_ap_names**
Give all possible SIAF aperture names

pynrc.nrc_hci.slowaxis

property nrc_hci.**slowaxis**
Slow readout direction in sci coords

pynrc.nrc_hci.telescope

nrc_hci.**telescope = 'JWST'**

pynrc.nrc_hci.wave_fit

property nrc_hci.**wave_fit**
Wavelength range to fit

pynrc.nrc_hci.well_level

property nrc_hci.**well_level**
Detector well level in units of electrons

pynrc.obs_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 RMS between the science and reference targets. Expected values are between ~3-10 nm.
- **wfe_roll_drift** (*float*) – WFE drift in nm RMS 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_params** (*dict*) –

  Arguments describing disk model information for a given FITS file:

  - 'file' : Path to model file or an HDUList.
  - 'pixscale' : Pixel scale for model image (arcsec/pixel).
  - 'dist' : Assumed model distance in parsecs.
  - 'wavelength' : Wavelength of observation in microns.
  - 'units' : String of assumed flux units (ie., MJy/arcsec^2 or muJy/pixel)
  - 'cen_star' : True/False. Is a star already placed in the central pixel?
- **autogen_coeffs** (*bool*) – Automatically generate base PSF coefficients. Equivalent to performing *self.gen_psf_coeff()*. *gen_wfedrift_coeff*, and *gen_wfemask_coeff*. Default: True.
- **use_ap_info** (*bool*) – For subarray observations, the mask reference points are not actually in the center of the array. Set this to True to shift the sources to actual aperture reference location. Otherwise, default will place in center of array.

- **sgd_type** (*str or None*) – Small grid dither pattern. Valid types are '9circle', '5box', '5diamond', '3bar', or '5bar'. If 'auto', then defaults are '5diamond' for round masks, '5bar' for bar masks, and '5diamond' for direct imaging. If None, then no FSM pointings, but there will be a single slew.
- **fsm_std** (*float*) – One-sigma accuracy per axis of fine steering mirror positions. This provides randomness to each position relative to the nominal central position. Ignored for central position. Values should be in units of mas.
- **slew_std** (*float*) – One-sigma accuracy per axis of the initial slew. This is applied to all positions and gives a baseline offset relative to the desired mask center. ***Values should be in units of mas***

Module Attributes

LONG_WAVELENGTH_MAX	
LONG_WAVELENGTH_MIN	
ND_acq	Use Coronagraphic ND acquisition square?
SHORT_WAVELENGTH_MAX	
SHORT_WAVELENGTH_MIN	
aperturename	SIAF aperture name for detector pixel to sky coords transformations
bandpass	Return bandpass throughput
bar_offset	Offset position along bar mask (arcsec).
channel	
coron_substrate	Include coronagraphic substrate material?
det_info	Dictionary housing detector info parameters and keywords.
detector	Detector selected for simulated PSF
detector_list	Detectors on which the simulated PSF could lie
detector_position	The pixel position in (X, Y) on the detector, relative to the currently-selected SIAF aperture subarray.
disk_params	
fastaxis	Fast readout direction in sci coords
filter	Currently selected filter name (e.g.
filter_list	List of available filter names for this instrument
fov_pix	
image_mask	Currently selected image plane mask, or None for direct imaging
is_coron	Observation with coronagraphic mask (incl Lyot stop)?
is_dark	
is_grism	
is_lyot	Is a Lyot mask in the pupil wheel?

continues on next page

Table 12 – continued from previous page

module	
multiaccum	*multiaccum* object
multiaccum_times	Exposure timings in dictionary
name	
ndeg	
npsf	Number of wavelengths/PSFs to fit
options	A dictionary capable of storing other arbitrary options, for extensibility.
oversample	
pixelscale	Detector pixel scale, in arcsec/pixel
planets	Planet info (if any exists)
psf_info	PSF parameters
pupil	Filename *or* fits.HDUList for JWST pupil mask.
pupil_mask	Currently selected Lyot pupil mask, or None for direct imaging
pupilopd	Filename *or* fits.HDUList for JWST pupil OPD.
quick	Perform quicker coeff calculation over limited bandwidth?
save_dir	Coefficient save directory
save_name	Coefficient file name
scaid	SCA ID (481, 482, .
siaf_ap	SIAF Aperture object
siaf_ap_names	Give all possible SIAF aperture names
slowaxis	Slow readout direction in sci coords
telescope	
wave_fit	Wavelength range to fit
well_level	Detector well level in units of electrons
wfe_ref_drift	WFE drift (nm) of ref obs relative to sci obs
wfe_roll_drift	WFE drift (nm) of Roll2 obs relative to Roll1 obs

pynrc.obs_hci.LONG_WAVELENGTH_MAX

obs_hci.**LONG_WAVELENGTH_MAX = 5.299999999999999e-06**

pynrc.obs_hci.LONG_WAVELENGTH_MIN

obs_hci.**LONG_WAVELENGTH_MIN = 2.35e-06**

pynrc.obs_hci.ND_acq

property obs_hci.**ND_acq**
Use Coronagraphic ND acquisition square?

pynrc.obs_hci.SHORT_WAVELENGTH_MAX

obs_hci.**SHORT_WAVELENGTH_MAX = 2.35e-06**

pynrc.obs_hci.SHORT_WAVELENGTH_MIN

obs_hci.**SHORT_WAVELENGTH_MIN = 6e-07**

pynrc.obs_hci.aperturename

property obs_hci.**aperturename**
SIAF aperture name for detector pixel to sky coords transformations

pynrc.obs_hci.bandpass

property obs_hci.**bandpass**
Return bandpass throughput

pynrc.obs_hci.bar_offset

property obs_hci.**bar_offset**
Offset position along bar mask (arcsec).

pynrc.obs_hci.channel

property obs_hci.**channel**

pynrc.obs_hci.coron_substrate

property obs_hci.**coron_substrate**
Include coronagraphic substrate material?

pynrc.obs_hci.det_info

property obs_hci.**det_info**
Dictionary housing detector info parameters and keywords.

pynrc.obs_hci.detector

property obs_hci.**detector**
Detector selected for simulated PSF

Used in calculation of field-dependent aberrations. Must be selected from detectors in the *detector_list* attribute.

pynrc.obs_hci.detector_list

property obs_hci.**detector_list**
Detectors on which the simulated PSF could lie

pynrc.obs_hci.detector_position

property obs_hci.**detector_position**
The pixel position in (X, Y) on the detector, relative to the currently-selected SIAF aperture subarray. By default the SIAF aperture will correspond to the full-frame detector, so (X,Y) will in that case be absolute (X,Y) pixels on the detector. But if you select a subarray aperture name from the SIAF, then the (X,Y) are interpreted as (X,Y) within that subarray.

Please note, this is X,Y order - **not** a Pythonic y,x axes ordering.

pynrc.obs_hci.disk_params

property obs_hci.**disk_params**

pynrc.obs_hci.fastaxis

property obs_hci.**fastaxis**
Fast readout direction in sci coords

pynrc.obs_hci.filter

property obs_hci.**filter**
Currently selected filter name (e.g. F200W)

pynrc.obs_hci.filter_list

obs_hci.**filter_list = None**
List of available filter names for this instrument

pynrc.obs_hci.fov_pix

property obs_hci.**fov_pix**

pynrc.obs_hci.image_mask

property obs_hci.**image_mask**
Currently selected image plane mask, or None for direct imaging

pynrc.obs_hci.is_coron

property obs_hci.**is_coron**
Observation with coronagraphic mask (incl Lyot stop)?

pynrc.obs_hci.is_dark

property obs_hci.**is_dark**

pynrc.obs_hci.is_grism

property obs_hci.**is_grism**

pynrc.obs_hci.is_lyot

property obs_hci.**is_lyot**
Is a Lyot mask in the pupil wheel?

pynrc.obs_hci.module

property obs_hci.**module**

pynrc.obs_hci.multiaccum

property obs_hci.**multiaccum**
multiaccum object

pynrc.obs_hci.multiaccum_times

property obs_hci.**multiaccum_times**
Exposure timings in dictionary

t_frame : Time of a single frame. t_group : Time of a single group (read frames + drop frames). t_int : Photon collection time for a single ramp/integration. t_int_tot1: Total time for all frames (reset+read+drop) in a first ramp. t_int_tot2: Total time for all frames (reset+read+drop) in a subsequent ramp. t_exp : Total photon collection time for all ramps. t_acq : Total acquisition time to complete exposure with all overheads.

pynrc.obs_hci.name

obs_hci.**name** = **'Instrument'**

pynrc.obs_hci.ndeg

property obs_hci.**ndeg**

pynrc.obs_hci.npsf

property obs_hci.**npsf**
Number of wavelengths/PSFs to fit

pynrc.obs_hci.options

obs_hci.**options** = **{}**
A dictionary capable of storing other arbitrary options, for extensibility. The following are all optional, and may or may not be meaningful depending on which instrument is selected.

This is a superset of the options provided in `poppy.Instrument.options`.

Parameters

- **source_offset_r** (*float*) – Radial offset of the target from the center, in arcseconds
- **source_offset_theta** (*float*) – Position angle for that offset, in degrees CCW.
- **pupil_shift_x, pupil_shift_y** (*float*) – Relative shift of the intermediate (coronagraphic) pupil in X and Y relative to the telescope entrance pupil, expressed as a decimal between -1.0-1.0 Note that shifting an array too much will wrap around to the other side unphysically, but for reasonable values of shift this is a non-issue. This option only has an effect for optical models that have something at an intermediate pupil plane between the telescope aperture and the detector.
- **pupil_rotation** (*float*) – Relative rotation of the intermediate (coronagraphic) pupil relative to the telescope entrace pupil, expressed in degrees counterclockwise. This option only has an effect for optical models that have something at an intermediate pupil plane between the telescope aperture and the detector.
- **rebin** (*bool*) – For output files, write an additional FITS extension including a version of the output array rebinned down to the actual detector pixel scale?
- **jitter** (*string "gaussian" or None*) – Type of jitter model to apply. Currently only convolution with a Gaussian kernel of specified width *jitter_sigma* is implemented. (default: None)

- **jitter_sigma** (*float*) – Width of the jitter kernel in arcseconds (default: 0.006 arcsec, 1 sigma per axis)
- **parity** (*string "even" or "odd"*) – You may wish to ensure that the output PSF grid has either an odd or even number of pixels. Setting this option will force that to be the case by increasing npix by one if necessary. Note that this applies to the number detector pixels, rather than the subsampled pixels if oversample > 1.
- **force_coron** (*bool*) – Set this to force full coronagraphic optical propagation when it might not otherwise take place (e.g. calculate the non-coronagraphic images via explicit propagation to all optical surfaces, FFTing to intermediate pupil and image planes whether or not they contain any actual optics, rather than taking the straight-to-MFT shortcut)
- **no_sam** (*bool*) – Set this to prevent the SemiAnalyticMethod coronagraph mode from being used when possible, and instead do the brute-force FFT calculations. This is usually not what you want to do, but is available for comparison tests. The SAM code will in general be much faster than the FFT method, particularly for high oversampling.

pynrc.obs_hci.oversample

property obs_hci.**oversample**

pynrc.obs_hci.pixelscale

obs_hci.**pixelscale = 0.025**
Detector pixel scale, in arcsec/pixel

pynrc.obs_hci.planets

property obs_hci.**planets**
Planet info (if any exists)

pynrc.obs_hci.psf_info

property obs_hci.**psf_info**
PSF parameters

pynrc.obs_hci.pupil

obs_hci.**pupil = None**
Filename *or* fits.HDUList for JWST pupil mask. Usually there is no need to change this.

pynrc.obs_hci.pupil_mask

property obs_hci.**pupil_mask**
Currently selected Lyot pupil mask, or None for direct imaging

pynrc.obs_hci.pupilopd

obs_hci.**pupilopd = None**
Filename *or* fits.HDUList for JWST pupil OPD.

This can be either a full absolute filename, or a relative name in which case it is assumed to be within the instrument's *data/OPDs/* directory, or an actual fits.HDUList object corresponding to such a file. If the file contains a datacube, you may set this to a tuple (filename, slice) to select a given slice, or else the first slice will be used.

pynrc.obs_hci.quick

property obs_hci.**quick**
Perform quicker coeff calculation over limited bandwidth?

pynrc.obs_hci.save_dir

property obs_hci.**save_dir**
Coefficient save directory

pynrc.obs_hci.save_name

property obs_hci.**save_name**
Coefficient file name

pynrc.obs_hci.scaid

property obs_hci.**scaid**
SCA ID (481, 482, … 489, 490)

pynrc.obs_hci.siaf_ap

property obs_hci.**siaf_ap**
SIAF Aperture object

pynrc.obs_hci.siaf_ap_names

property obs_hci.**siaf_ap_names**
Give all possible SIAF aperture names

pynrc.obs_hci.slowaxis

property obs_hci.**slowaxis**
Slow readout direction in sci coords

pynrc.obs_hci.telescope

obs_hci.**telescope = 'JWST'**

pynrc.obs_hci.wave_fit

property obs_hci.**wave_fit**
Wavelength range to fit

pynrc.obs_hci.well_level

property obs_hci.**well_level**
Detector well level in units of electrons

pynrc.obs_hci.wfe_ref_drift

property obs_hci.**wfe_ref_drift**
WFE drift (nm) of ref obs relative to sci obs

pynrc.obs_hci.wfe_roll_drift

property obs_hci.**wfe_roll_drift**
WFE drift (nm) of Roll2 obs relative to Roll1 obs

1.9.3 Simulations

simul.ngNRC	ngNRC - NIRCam Detector Noise Simulator
simul.apt	APT conversion procedures
simul.dms	DMS data format routines
simul.skyvec2ins	skyvec2ins JWST Coronagraph Visibility Calculator

pynrc.simul.ngNRC

ngNRC - NIRCam Detector Noise Simulator

Modification History:

Nov 2021

- Auto-generate source tables from Gaia DR2 or Simbad queries
- Add WFE drifts.

Oct 2021

- Use numpy random number generator objects to produce repeatable results.

Sept 2021

- Major refactor, splitting out slope_to_level1b and slope_to_fitswriter
- Added linearity, flat fields, and cosmic rays

Apr 2021

- Deprecate nghxrg, SCANoise, and slope_to_ramp
- Instead use slope_to_ramps

Oct 2020

- Restructure det noise and ramp creation
- DMS simulations using JWST pipeline data models

Feb 2017

- Add ngNRC to pyNRC code base

Aug 2016, J.M. Leisenring, UA/Steward

- Modified how the detector and multiaccum info is handled
- Copied detector and multiaccum classes from pyNRC
- In the future, we will want to integrate this directly so that any changes made in the pyNRC classes are accounted.

July 2016, J.M. Leisenring, UA/Steward

- Updated many things and more for nghxrg (v3.0)

Feb 2016, J.M. Leisenring, UA/Steward

- First Release

Functions

add_col_noise(data_in, ...[, rand_seed])	Add RTN Column Noise
add_cosmic_rays(data[, scenario, scale, ...])	Add random cosmic rays to data cube
add_ipc(im[, alpha_min, alpha_max, kernel])	Convolve image with IPC kernel
add_ppc(im[, ppc_frac, nchans, kernel, ...])	Add Post-Pixel Coupling (PPC)
add_xtalk(data, det[, coeffs])	Add amplifier crosstalk to each frame in data cube
apply_flat(cube, det, imflat_full)	Apply flat field
create_level1b_FITS(sim_config[, detname, ...])	Generate Level1b DMS-like FITS files.

continues on next page

Table 14 – continued from previous page

fft_noise(pow_spec[, nstep_out, fmin, f, ...])	Random Noise from Power Spectrum
gen_col_noise(ramp_column_varations, prob_bad)	Generate RTN Column Noise
gen_dark_ramp(dark, out_shape[, tf, gain, ...])	Assumes a constant dark current rate, either in image form or single value.
gen_ramp_biases(ref_dict[, nchan, ...])	Generate a ramp of bias offsets
gen_wfe_drift(obs_input[, case, iec_period, ...])	Create WFE drift information over time
make_gaia_source_table(coords[, ...])	Create source table from GAIA DR2 query
make_ramp_poisson(im_slope, det[, out_ADU, ...])	Create a ramp with only photon noise.
make_simbad_source_table(coords[, ...])	
pink_noise(nstep_out[, pow_spec, f, fmin, alpha])	Generate random pink noise
save_slope_image(im_slope, obs_params[, ...])	Save slope image as Level1b-like FITS
sim_dark_ramp(det, slope_image[, ...])	Simulate a dark current ramp based on input det class and a super dark image.
sim_image_ramp(det, im_slope[, verbose])	Simulate an image ramp based on input det class and slope image.
sim_noise_data(det[, rd_noise, u_pink, ...])	Simulate Noise Ramp
simulate_detector_ramp(det, cal_obj[, ...])	Return a single simulated ramp
slope_to_fitswriter(det, cal_obj[, ...])	Simulate HDUList from slope image
slope_to_level1b(im_slope, obs_params[, ...])	Simulate DMS HDUList from slope image
sources_to_level1b(source_table, nircam_obj, ...)	Simulate DMS HDUList from slope image
sources_to_slope(source_table, nircam_obj, ...)	Create a slope image from a table or sources
xtalk_image(frame, det[, coeffs])	Create image of crosstalk signal

pynrc.simul.ngNRC.add_col_noise

pynrc.simul.ngNRC.**add_col_noise**(*data_in*, *ramp_column_varations*, *prob_bad*, *rand_seed=None*)

Add RTN Column Noise

This function takes the random telegraph noise templates derived from CV3 data and adds it to an idealized dark ramp. These column variations likely come from noise in column-specifc FETs jumping between two discrete states, possibly within the detector column bus.

This function randomly draws a number of template column variation ramps, then randomly assigns them to different columns in *super_dark_ramp*. The nubmer of columns (and whether or not a column is assigned a random variation) is based on the *prob_bad* variable.

Parameters

- **data_in** (*ndarray*) – Idealized ramp of size (nz,ny,nx)
- **ramp_column_variations** (*ndarray*) – The column-average ramp variations of size (nz,nx). These are added to a given columnn.
- **prob_bad** (*float*) – Probability that a given column is subject to these column variations.

pynrc.simul.ngNRC.add_cosmic_rays

pynrc.simul.ngNRC.**add_cosmic_rays**(*data*, *scenario='SUNMAX'*, *scale=1*, *tframe=10.73677*, *ref_info=[4, 4, 4, 4]*, *rand_seed=None*)

Add random cosmic rays to data cube

pynrc.simul.ngNRC.add_ipc

pynrc.simul.ngNRC.**add_ipc**(*im*, *alpha_min=0.0065*, *alpha_max=None*, *kernel=None*)

Convolve image with IPC kernel

Given an image in electrons, apply IPC convolution. NIRCam average IPC values (alpha) reported 0.005 - 0.006.

Parameters

- **im** (*ndarray*) – Input image or array of images.
- **alpha_min** (*float*) – Minimum coupling coefficient between neighboring pixels. If alpha_max is None, then this is taken to be constant with respect to signal levels.
- **alpha_max** (*float or None*) – Maximum value of coupling coefficent. If specificed, then coupling between pixel pairs is assumed to vary depending on signal values. See Donlon et al., 2019, PASP 130.
- **kernel** (*ndarry or None*) – Option to directly specify the convolution kernel. *alpha_min* and *alpha_max* are ignored.

Examples

Constant Kernel

```
>>> im_ipc = add_ipc(im, alpha_min=0.0065)
```

Constant Kernel (manual)

```
>>> alpha = 0.0065
>>> k = np.array([[0,alpha,0], [alpha,1-4*alpha,alpha], [0,alpha,0]])
>>> im_ipc = add_ipc(im, kernel=k)
```

Signal-dependent Kernel

```
>>> im_ipc = add_ipc(im, alpha_min=0.0065, alpha_max=0.0145)
```

pynrc.simul.ngNRC.add_ppc

pynrc.simul.ngNRC.**add_ppc**(*im*, *ppc_frac=0.002*, *nchans=4*, *kernel=None*, *same_scan_direction=False*, *reverse_scan_direction=False*, *in_place=False*)

Add Post-Pixel Coupling (PPC)

This effect is due to the incomplete settling of the analog signal when the ADC sample-and-hold pulse occurs. The measured signals for a given pixel will have a value that has not fully transitioned to the real analog signal. Mathematically, this can be treated in the same way as IPC, but with a different convolution kernel.

Parameters

- **im** (*ndarray*) – Image or array of images

- **ppc_frac** (*float*) – Fraction of signal contaminating next pixel in readout.
- **kernel** (*ndarry or None*) – Option to directly specify the convolution kernel, in which case *ppc_frac* is ignored.
- **nchans** (*int*) – Number of readout output channel amplifiers.
- **same_scan_direction** (*bool*) – Are all the output channels read in the same direction? By default fast-scan readout direction is `[-->,<--,-->,<--]` If `same_scan_direction`, then all `-->`
- **reverse_scan_direction** (*bool*) – If `reverse_scan_direction`, then `[<--,-->,<--,-->]` or all `<--`
- **in_place** (*bool*) – Apply in place to input image.

pynrc.simul.ngNRC.add_xtalk

pynrc.simul.ngNRC.**add_xtalk**(*data*, *det*, *coeffs=None*)
Add amplifier crosstalk to each frame in data cube

Parameters

- **data** (*ndarray*) – 2D or 3D data cube
- **det** (*pynrc.DetectorOps*) – Detector class corresponding to data.
- **coeffs** (*None or Table*) – Table of coefficients corresponding to detector crosstalk behavior.

pynrc.simul.ngNRC.apply_flat

pynrc.simul.ngNRC.**apply_flat**(*cube*, *det*, *imflat_full*)
Apply flat field

Includes pixel-to-pixel QE variations or instrument's optical flatfield (e.g., throughput variations across the field).

Parameters

- **cube** (*ndarray*) – Simulated ramp data in e-. These should be intrinsic flux values with Poisson noise, but prior to read noise, kTC, IPC, etc. Size (nz,ny,nx). In det coords. Can either be full frame or match *det* subarray. Returns *det* subarray shape.
- **det** (*Detector Class*) – Desired detector class output
- **imflat_full** (*ndarray*) – Full field image of flat field in det coords. Will get trimmed if necessary.

pynrc.simul.ngNRC.create_level1b_FITS

pynrc.simul.ngNRC.**create_level1b_FITS**(*sim_config*, *detname=None*, *apname=None*, *filter=None*, *visit_id=None*, *dry_run=None*, *save_slope=None*, *save_dms=None*)
Generate Level1b DMS-like FITS files.

Todo:

- Tracking image persistence

- Static column noise that shifts every integration
- Currently only has WFE drifts of coronagraphic on-mask stars (sci and ref)
- Also drifts non-HCI observations, generating new PSF grids each exposure

---

Keyword Arguments

- **detname** (*None or int or str*) – Option to supply a valid detector name. If set, only the currently specified SCA will be simulated.
- **apname** (*None or str*) – Similar to *detname* keyword, can supply a specific SIAF aperture name that exists within the observation. Only that aperture will be simulated.
- **filter** (*None or str*) – Specify a filter within observation to be simulated. Can be combined with *detname* and *apname* keywords. Should have the form "ABC:XYZ", or "ObsNum:VisitNum" (e.g., "005:001" for observation 5, visit 1).
- **visit_id** (*None or str*) – Specify the visit ID to simulate.
- **dry_run** (*bool or None*) – Won't generate any image data, but instead runs through each observation, printing detector info, SIAF aperture name, filter, visit IDs, exposure numbers, and dither information. If set to None, then grabs keyword from *sim_config*, otherwise defaults to False if not found. If paired with *save_dms*, then will generate an empty set of DMS FITS files with headers populated, but data set to all zeros.
- **save_slope** (*bool or None*) – Saves noiseless slope images to a separate DMS-like FITS file that is names 'slope_{DMSfilename}'. If set to None, then grabs keyword from *sim_config*, otherwise defaults to False if not found. No effect if dry_run=True.
- **save_dms** (*bool or None*) – Option to disable simulation of ramp data and creation of DMS FITS. If dry_run=True, then setting save_dms=True will save DMS FITS files populated with all zeros. If set to None, then grabs keyword from *sim_config*; if no keyword is found, then defaults to True if dry_run=False, otherwise False.

pynrc.simul.ngNRC.fft_noise

pynrc.simul.ngNRC.**fft_noise**(*pow_spec*, *nstep_out=None*, *fmin=None*, *f=None*, *pad_mode='edge'*, *rand_seed=None*, ***kwargs*)

Random Noise from Power Spectrum

Returns a noised array where the instrinsic distribution follows that of the input power spectrum. The output has an instrinsic standard deviation scaled to 1.0.

Parameters

- **pow_spec** (*ndarray*) – Input power spectrum from which to generate noise distribution.
- **nstep_out** (*int*) – Desired size of the output noise array. If smaller than *pow_spec* then it just truncates the results to the appropriate size. If larger, then pow_spec gets padded by the specified *pad_mode*.
- **fmin** (*float or None*) – Low-frequency cutoff. Power spectrum values below this cut-off point get set equal to the power spectrum value at fmin.
- **f** (*ndarray or None*) – An array the same size as pow_spec and is only used when fmin is set. If set to None, then *f = np.fft.rfftfreq(n_ifft)* where *n_ifft* is the size of the result of *rifft(pow_spec)* assuming a delta time of unity.

- **pad_mode** (*str or function*) – One of the following string values or a user supplied function. Default is 'edge'.

  'constant' (default) Pads with a constant value.

  'edge' Pads with the edge values of array.

  'linear_ramp' Pads with the linear ramp between end_value and the array edge value.

  'maximum' Pads with the maximum value of all or part of the vector along each axis.

  'mean' Pads with the mean value of all or part of the vector along each axis.

  'median' Pads with the median value of all or part of the vector along each axis.

  'minimum' Pads with the minimum value of all or part of the vector along each axis.

  'reflect' Pads with the reflection of the vector mirrored on the first and last values of the vector along each axis.

  'symmetric' Pads with the reflection of the vector mirrored along the edge of the array.

  'wrap' Pads with the wrap of the vector along the axis. The first values are used to pad the end and the end values are used to pad the beginning.

pynrc.simul.ngNRC.gen_col_noise

pynrc.simul.ngNRC.**gen_col_noise**(*ramp_column_varations*, *prob_bad*, *nz=108*, *nx=2048*, *rand_seed=None*)
Generate RTN Column Noise

This function takes the random telegraph noise templates derived from CV3 data and generates a random noise set to add to an dark ramp sim. These column variations likely come from RTN in column-specifc FETs jumping between two discrete states, possibly within the detector column bus.

This function randomly draws a number of template column variation ramps, then randomly assigns them to different columns in *super_dark_ramp*. The nubmer of columns (and whether or not a column is assigned a random variation) is based on the *prob_bad* variable.

pynrc.simul.ngNRC.gen_dark_ramp

pynrc.simul.ngNRC.**gen_dark_ramp**(*dark*, *out_shape*, *tf=10.73677*, *gain=1*, *ref_info=None*, *avg_ramp=None*, *include_poisson=True*, *rand_seed=None*, ***kwargs*)
Assumes a constant dark current rate, either in image form or single value. If gain is supplied, then input is assumed to be in DN/sec, otherwise e-/sec. Output will be e-.

Parameters

- **dark** (*ndarray or float*) – Dark slope image or constant value. Assumed to be DN/sec. If gain=1, then also e-/sec. If this value is intended to be e-/sec, then simply set gain=1.
- **out_shape** (*tuple, list, ndarray*) – Desired shape of output ramp (nframes, ny, nx). If *dark* is an array, then dark.shape == out_shape[1:] == (ny,nx).
- **tf** (*float*) – Frame time in seconds
- **gain** (*float*) – Gain of detector in e-/sec. If specified to be other than 1, then we assume *dark* to be in units of DN/sec.
- **avg_ramp** (*ndarray*) – Time-dependent flux of average dark ramp.

pynrc.simul.ngNRC.gen_ramp_biases

pynrc.simul.ngNRC.**gen_ramp_biases**(*ref_dict*, *nchan=None*, *data_shape=(2, 2048, 2048)*, *include_refinst=True*, *ref_border=[4, 4, 4, 4]*, *rand_seed=None*)

Generate a ramp of bias offsets

Parameters

- **ref_dict** (*dict*) – Dictionary of reference behaviors.
- **nchan** (*int*) – Specify number of output channels. If not set, then will automatically determine from *ref_dict*. This allows us to set nchan=1 for Window Mode while using the first channel info provided in *ref_dict*.
- **data_shape** (*array like*) – Shape of output (nz,ny,nx)
- **include_refinst** (*bool*) – Include instabilities in the offsets?
- **ref_border** (*list*) – Number of references pixels [lower, upper, left, right]

pynrc.simul.ngNRC.gen_wfe_drift

pynrc.simul.ngNRC.**gen_wfe_drift**(*obs_input*, *case='BOL'*, *iec_period=300*, *slew_init=10*, *rand_seed=None*, *t0_offset=False*, *plot=False*, *figname=None*)

Create WFE drift information over time

Parameters

- **obs_input** (*DMS_input class*) – Class to generate a series of observation dictionaries in order to build DMS-like files. Loads APT files to generate the necessary observation information.
- **case** (*string*) – Either "BOL" for current best estimate at beginning of life, or "EOL" for more conservative prediction at end of life.
- **iec_period** (*float*) – IEC heater switching period in seconds.
- **slew_init** (*float*) – Assumed slew difference relative to previous program
- **rand_seed** (*None or int*) – Seed value to initialize random number generator to obtain repeatable values.
- **t0_offset** (*bool*) – Shift delta WFE drift values to 0 at time t=0 (beginning of program)? If set to False, then relative drift values correspond to beginning of the previous randomly-generated program.
- **plot** (*bool*) – Create a plot of the slew angles and associated RMS drift components.
- **figname** (*string*) – Output name (path) to save plot figure.

pynrc.simul.ngNRC.make_gaia_source_table

pynrc.simul.ngNRC.**make_gaia_source_table**(*coords*, *remove_cen_star=True*, *radius=<Quantity 6. arcmin>*, *teff_default=5800*)

Create source table from GAIA DR2 query

Generates a table of objects by performing a cone search around a set of input coordinates. The output table includes both coordinates and extrapolated photometry. All returned coordinates are for epoch=2015.5 for GAIA DR2.

The process involves performing a search query around a set of input coordinates, then using the GAIA magnitudes and Teff to extrapolate magnitudes in the NIRCam filters. If Teff isn't supplied, then it defaults to solar temperature. If an object's parallax isn't available, then the object is assumed to have a flat spectrum in F_lambda.

Parameters

- **coords** (*SkyCoords*) – Astropy SkyCoords object.
- **remove_cen_star** (*bool*) – Output will exclude the star associated with the input coordinates (anything with 0.1" is removed from table).
- **radius** (*Units*) – Radius to perfrom search. Default is 6', which should encompass NIRCam's full FoV, including both Modules A and B.
- **teff_default** (*string*) – Default stellar effective temperature to assume for sources without GAIA information to extrapolate to longer wavelengths.

pynrc.simul.ngNRC.make_ramp_poisson

pynrc.simul.ngNRC.**make_ramp_poisson**(*im_slope*, *det*, *out_ADU=True*, *zero_data=False*, *include_poisson=True*, *rand_seed=None*)

Create a ramp with only photon noise. Useful for quick simulations.

im_slope : Slope image (detector coordinates) in e-/sec det : Detector information class out_ADU : Convert to 16-bit UINT? zero_data: Return the so-called "zero frame"?

pynrc.simul.ngNRC.make_simbad_source_table

pynrc.simul.ngNRC.**make_simbad_source_table**(*coords*, *remove_cen_star=True*, *radius=<Quantity 6. arcmin>*, *spt_default='G2V'*)

pynrc.simul.ngNRC.pink_noise

pynrc.simul.ngNRC.**pink_noise**(*nstep_out*, *pow_spec=None*, *f=None*, *fmin=None*, *alpha=- 1*, ***kwargs*)

Generate random pink noise

Parameters

- **nstep_out** (*int*) – Desired size of the output noise array. If smaller than *pow_spec* then it just truncates the results to the appropriate size. If larger, then pow_spec gets padded by the specified *pad_mode*.
- **pow_spec** (*ndarray*) – Option to input the power spectrum instead of regenerating it every time. Make sure this was generated with powers of 2 for faster processing.
- **f** (*ndarray or None*) – An array the same size as pow_spec. If set to None, then will create an array of appropriate size assuming a delta time of unity.
- **fmin** (*float or None*) – Low-frequency cutoff. Power spectrum values below this cut-off point get set equal to the power spectrum value at fmin.
- **alpha** (*float*) – Power spectrum index to generate if *pow_spec* is not specified directly.

Keyword Arguments

- **pad_mode** (*str or function*) – One of the following string values or a user supplied function. Default is 'edge'.

'constant' (default) Pads with a constant value.

'edge' Pads with the edge values of array.

'linear_ramp' Pads with the linear ramp between end_value and the array edge value.

'maximum' Pads with the maximum value of all or part of the vector along each axis.

'mean' Pads with the mean value of all or part of the vector along each axis.

'median' Pads with the median value of all or part of the vector along each axis.

'minimum' Pads with the minimum value of all or part of the vector along each axis.

'reflect' Pads with the reflection of the vector mirrored on the first and last values of the vector along each axis.

'symmetric' Pads with the reflection of the vector mirrored along the edge of the array.

'wrap' Pads with the wrap of the vector along the axis. The first values are used to pad the end and the end values are used to pad the beginning.

- **rand_seed** (*None or int*) – Seed value to initialize random number generator to obtain repeatable values.

pynrc.simul.ngNRC.save_slope_image

pynrc.simul.ngNRC.**save_slope_image**(*im_slope*, *obs_params*, *save_dir=None*, ***kwargs*)
Save slope image as Level1b-like FITS

pynrc.simul.ngNRC.sim_dark_ramp

pynrc.simul.ngNRC.**sim_dark_ramp**(*det*, *slope_image*, *ramp_avg_ch=None*, *ramp_avg_tf=10.73677*, *out_ADU=False*, *verbose=False*, ***kwargs*)
Simulate a dark current ramp based on input det class and a super dark image.

By default, returns ramp in terms of e- using gain information provide in *det* input. To return in terms of ADU, set *out_ADU=True* (divides by gain).

Parameters

- **det** (*Detector Class*) – Desired detector class output
- **slope_image** (*ndarray*) – Input slope image (DN/sec). Can either be full frame or match *det* subarray. Returns *det* subarray shape.

Keyword Arguments

- **ramp_avg_ch** (*ndarray or None*) – Time-dependent flux of average dark ramp for each amplifier channel for dark current simulations.
- **ramp_avg_tf** (*float*) – Delta time between between *ramp_avg_ch* points.
- **out_ADU** (*bool*) – Divide by gain to get value in ADU (float).
- **include_poisson** (*bool*) – Include Poisson noise from photons?
- **verbose** (*bool*) – Print some messages.

pynrc.simul.ngNRC.sim_image_ramp

pynrc.simul.ngNRC.**sim_image_ramp**(*det*, *im_slope*, *verbose=False*, ***kwargs*)

Simulate an image ramp based on input det class and slope image. Uses the *sim_dark_ramp* function. By default, returns ramp in terms of e- using gain information provide in *det* input. To return in terms of ADU, set *out_ADU=True* (divides by gain).

Parameters

- **det** (*Detector Class*) – Desired detector class output
- **im_slope** (*ndarray*) – Input slope image (e-/sec). *NOTE* - This is different than sim_dark_ramp, which assumed DN/sec. Can either be full frame or match *det* subarray. Returns *det* subarray shape.
- **include_poisson** (*bool*) – Include Poisson noise from photons? Default: True.

Keyword Arguments

- **out_ADU** (*bool*) – Divides by gain to get output value in ADU (float).
- **verbose** (*bool*) – Print some messages.

pynrc.simul.ngNRC.sim_noise_data

pynrc.simul.ngNRC.**sim_noise_data**(*det*, *rd_noise=[5, 5, 5, 5]*, *u_pink=[1, 1, 1, 1]*, *c_pink=3*, *acn=0*, *pow_spec_corr=None*, *corr_scales=None*, *fcorr_lim=[1, 10]*, *ref_ratio=0.8*, *rand_seed=None*, *verbose=True*, ***kwargs*)

Simulate Noise Ramp

Simulate the noise components of a ramp, including white noise as well as 1/f (pink) noise components that are uncorrelated and correlated between amplifier channels.

Parameters

- **det** (*det_timing* class) – Class holding detector operations information. See *detops.det_timing* for generic class, or *pynrc_core.DetectorOps* for NIRCam specific timing.
- **rd_noise** (*array like or float or None*) – Array of white noise values (std dev per frame) for each output channel, or a single value. If an array, must match the number amplifier values specified in *det.nchan*.
- **u_pink** (*array like or float or None*) – Array of uncorrelated pink noise (std dev per frame) for each output channel, or a single value. If an array, must match the number amplifier values specified in *det.nchan*.
- **c_pink** (*float or None*) – Standard deviation of the pink noise correlated between channels.
- **pow_spec_corr** (*ndarray*) – Option to input a custom power spectrum for the correlated noise.
- **corr_scales** (*array like*) – Instead of *pow_spec_corr*, input the scale factors of the two 1/f components [low freq, highfreq]).
- **fcorr_lim** (*array like*) – Low- and high- frequency cut-off points for *corr_scales* factors. The first element of *corr_scales* is applied to those frequencies below *fcorr_lim[0]*, while the second element corresponds to frequencies above *fcorr_lim[1]*.

pynrc.simul.ngNRC.simulate_detector_ramp

pynrc.simul.ngNRC.**simulate_detector_ramp**(*det*, *cal_obj*, *im_slope=None*, *cframe='sci'*, *out_ADU=False*, *include_poisson=True*, *include_dark=True*, *include_bias=True*, *include_ktc=True*, *include_rn=True*, *apply_ipc=True*, *apply_ppc=True*, *include_cpink=True*, *include_upink=True*, *include_acn=True*, *include_refoffsets=True*, *include_refinst=True*, *include_colnoise=True*, *col_noise=None*, *amp_crosstalk=True*, *add_crs=True*, *cr_model='SUNMAX'*, *cr_scale=1*, *apply_flats=None*, *apply_nonlinearity=True*, *random_nonlin=False*, *latents=None*, *rand_seed=None*, *return_zero_frame=None*, *return_full_ramp=False*, *prog_bar=True*, ***kwargs*)

Return a single simulated ramp

Parameters

- **det** (*Detector Class*) – Desired detector class output
- **cal_obj** (*nircam_cal class*) – NIRCam calibration class that holds the necessary calibration info to simulate a ramp.
- **im_slope** (*ndarray*) – Input slope image of observed scene. Can either be full frame or match *det* subarray. Returns *det* subarray shape.
- **cframe** (*str*) – Coordinate frame of input image, 'sci' or 'det'. Output will be in same coordinates.

Keyword Arguments

- **return_zero_frame** (*bool or None*) – For DMS data, particularly readout patterns with averaged frames, this returns the very first raw read in the ramp.
- **return_full_ramp** (*bool*) – By default, we average groups and drop frames as specified in the *det* input. If this keyword is set to True, then return all raw frames within the ramp. The last set of *nd2* drop frames are omitted.
- **out_ADU** (*bool*) – If true, divide by gain and convert to 16-bit UINT.
- **include_poisson** (*bool*) – Include photon noise?
- **include_dark** (*bool*) – Add dark current?
- **include_bias** (*bool*) – Add detector bias?
- **include_ktc** (*bool*) – Add kTC noise?
- **include_rn** (*bool*) – Add readout noise per frame?
- **include_cpink** (*bool*) – Add correlated 1/f noise to all amplifiers?
- **include_upink** (*bool*) – Add uncorrelated 1/f noise to each amplifier?
- **include_acn** (*bool*) – Add alternating column noise?
- **apply_ipc** (*bool*) – Include interpixel capacitance?
- **apply_ppc** (*bool*) – Apply post-pixel coupling to linear analog signal?
- **include_refoffsets** (*bool*) – Include reference offsts between amplifiers and odd/even columns?
- **include_refinst** (*bool*) – Include reference/active pixel instabilities?

- **include_colnoise** (*bool*) – Add in column noise per integration?
- **col_noise** (*ndarray or None*) – Option to explicitly specifiy column noise distribution in order to shift by one for subsequent integrations
- **amp_crosstalk** (*bool*) – Crosstalk between amplifiers?
- **add_crs** (*bool*) – Add cosmic ray events? See Robberto et al 2010 (JWST-STScI-001928).
- **cr_model** (*str*) – Cosmic ray model to use: 'SUNMAX', 'SUNMIN', or 'FLARES'.
- **cr_scale** (*float*) – Scale factor for probabilities.
- **apply_nonlinearity** (*bool*) – Apply non-linearity? If False, then warning if out_ADU=True
- **random_nonlin** (*bool*) – Add randomness to the linearity coefficients?
- **apply_flats** (*bool*) – Apply sub-pixel QE variations (crosshatching and illumination)?
- **latents** (*None or ndarray*) – (TODO) Apply persistence from previous integration.
- **prog_bar** (*bool*) – Show a progress bar for this ramp generation?

pynrc.simul.ngNRC.slope_to_fitswriter

pynrc.simul.ngNRC.**slope_to_fitswriter**(*det*, *cal_obj*, *im_slope=None*, *cframe='det'*, *filter=None*, *pupil=None*, *targ_name=None*, *obs_time=None*, *file_out=None*, *out_ADU=True*, *return_results=True*, *rand_seed=None*, ***kwargs*)

Simulate HDUList from slope image

FITSWriter-like output. DMS output has been depreceated and moved to *slope_to_level1b* and *sources_to_level1b*.

Parameters

- **det** (*Detector Class*) – Desired detector class output
- **dark_cal_obj** (*nircam_cal class*) – NIRCam calibration class that holds the necessary calibration info to simulate a ramp.
- **im_slope** (*ndarray*) – Input slope image of observed scene. Assumed to be in detector coordinates. If an image cube, then number of images must match the number of integration (*nint*) in *det* class.
- **cframe** (*str*) – Orientation of im_slope. Either 'det' or 'sci' coordinate frame.
- **filter** (*str*) – Name of filter element for header
- **pupil** (*str*) – Name of pupil element for header
- **targ_name** (*str*) – Target name (optional)
- **obs_time** (*datetime*) – Specifies when the observation was considered to be executed. If not specified, then it will choose the current time. This information is added to the header. Must be a datetime object:

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

- **file_out** (*str or None*) – Name (including directory) to save FITS file. If None, then won't save; make sure to set return_results=True.
- **out_ADU** (*bool*) – If true, divide by gain and convert to 16-bit UINT.

- **return_results** (*bool*) – Return HDUList result? Otherwise,

Keyword Arguments

- **return_full_ramp** (*bool*) – By default, we average groups and drop frames as specified in the *det* input. If this keyword is set to True, then return all raw frames within the ramp. The last set of *nd2* frames will be omitted.
- **include_dark** (*bool*) – Add dark current?
- **include_bias** (*bool*) – Add detector bias?
- **include_ktc** (*bool*) – Add kTC noise?
- **include_rn** (*bool*) – Add readout noise per frame?
- **include_cpink** (*bool*) – Add correlated 1/f noise to all amplifiers?
- **include_upink** (*bool*) – Add uncorrelated 1/f noise to each amplifier?
- **include_acn** (*bool*) – Add alternating column noise?
- **apply_ipc** (*bool*) – Include interpixel capacitance?
- **apply_ppc** (*bool*) – Apply post-pixel coupling to linear analog signal?
- **include_refinst** (*bool*) – Include reference/active pixel instabilities?
- **include_colnoise** (*bool*) – Add in column noise per integration?
- **col_noise** (*ndarray or None*) – Option to explicitly specify column noise distribution in order to shift by one for subsequent integrations
- **amp_crosstalk** (*bool*) – Crosstalk between amplifiers?
- **add_crs** (*bool*) – Add cosmic ray events?
- **latents** (*None*) – Apply persistence.
- **linearity_map** (*ndarray*) – Add non-linearity.

pynrc.simul.ngNRC.slope_to_level1b

pynrc.simul.ngNRC.**slope_to_level1b**(*im_slope*, *obs_params*, *cal_obj=None*, *save_dir=None*, *cframe='sci'*, *out_ADU=True*, ***kwargs*)

Simulate DMS HDUList from slope image

Requires input of obs_params input dictionary as generated from APT input files (see *DMS_input* class in apt.py).

Also, make sure the *calib* directory exists in PYNRC_PATH and is populated with detector calibration information.

Look at keyword args to exclude specific detector effects.

Output is saved to disk and will not be returned by the function.

Parameters

- **im_slope** (*ndarray*) – Slope in e-/sec of image from all sky sources, including Zodiacal background. Should exclude dark current background, which is handled separately from calib directory.
- **obs_params** (*dict*) – Dictionary of parameters to populate DMS header. See *create_DMS_HDUList* in dms.py.

- **cal_obj** (`pynrc.nircam_cal`) – DMS object built from exported APT files. See *DMS_input* in apt.py.
- **save_dir** (*None or str*) – Option to override output directory as specified in *obs_params* dictionary. If not specified as either a function keyword or in *obs_params*, then files are saved in current working directory.
- **cframe** (*str*) – Coordinate frame of input slope, 'sci' or 'det'.

Keyword Arguments

- **include_dark** (*bool*) – Add dark current?
- **include_bias** (*bool*) – Add detector bias?
- **include_ktc** (*bool*) – Add kTC noise?
- **include_rn** (*bool*) – Add readout noise per frame?
- **include_cpink** (*bool*) – Add correlated 1/f noise to all amplifiers?
- **include_upink** (*bool*) – Add uncorrelated 1/f noise to each amplifier?
- **include_acn** (*bool*) – Add alternating column noise?
- **apply_ipc** (*bool*) – Include interpixel capacitance?
- **apply_ppc** (*bool*) – Apply post-pixel coupling to linear analog signal?
- **include_refoffsets** (*bool*) – Include reference offests between amplifiers and odd/even columns?
- **include_refinst** (*bool*) – Include reference/active pixel instabilities?
- **include_colnoise** (*bool*) – Add in column noise per integration?
- **col_noise** (*ndarray or None*) – Option to explicitly specify column noise distribution in order to shift by one for subsequent integrations
- **amp_crosstalk** (*bool*) – Crosstalk between amplifiers?
- **add_crs** (*bool*) – Add cosmic ray events? See Robberto et al 2010 (JWST-STScI-001928).
- **cr_model** (*str*) – Cosmic ray model to use: 'SUNMAX', 'SUNMIN', or 'FLARES'.
- **cr_scale** (*float*) – Scale factor for probabilities.
- **latents** (*None*) – Apply persistence.
- **apply_nonlinearity** (*bool*) – Apply non-linearity?
- **random_nonlin** (*bool*) – Add randomness to the linearity coefficients?
- **prog_bar** (*bool*) – Show a progress bar for this ramp generation?

pynrc.simul.ngNRC.sources_to_level1b

pynrc.simul.ngNRC.**sources_to_level1b**(*source_table*, *nircam_obj*, *obs_params*, *tel_pointing*, *hdul_psfs=None*, *cal_obj=None*, *im_bg=None*, *out_ADU=True*, *save_dir=None*, ***kwargs*)

Simulate DMS HDUList from slope image

Requires input of obs_params input dictionary as generated from APT input files (see *DMS_input* class in apt.py).

Also, make sure the *calib* directory exists in PYNRC_PATH and is populated with detector calibration information.

Look at keyword args to exclude specific detector effects.

Output is saved to disk and will not be returned by the function.

Parameters

- **source_table** (*astropy Table*) – Table of objects in across the region, including headers 'ra', 'dec', and object fluxes in NIRCam filter in vega mags where headers are labeled the filter name (e.g, 'F444W').
- **nircam_obj** (*pynrc.NIRCam*) – NIRCam instrument class for PSF generation.
- **obs_params** (*dict*) – Dictionary of parameters to populate DMS header. See *create_obs_params* in apt.py and *level1b_data_model* in dms.py.
- **tel_pointing** (`webbpsf_ext.jwst_point`) – JWST telescope pointing information. Holds pointing coordinates and dither information for a given telescope visit.
- **cal_obj** (`pynrc.nircam_cal`) – NIRCam calibration class that holds the necessary calibration info to simulate a ramp.
- **im_bg** (*None or ndarray*) – Option to specify a pre-generated image (or single value) of the Zodiacal background emission. If not specified, then gets automatically generating.
- **save_dir** (*None or str*) – Option to override output directory as specified in *obs_params* dictionary. If not specified as either a function keyword or in *obs_params*, then files are saved in current working directory.

Keyword Arguments

- **npsf_per_full_fov** (*int*) – Number of PSFs across one dimension of the instrument's field of view. If a coronagraphic observation, then this is for the nominal coronagrahic field of view.
- **sptype** (*str*) – Spectral type, such as 'A0V' or 'K2III'.
- **wfe_drift** (*float*) – Desired WFE drift value relative to default OPD.
- **osamp** (*int*) – Sampling of output PSF relative to detector sampling. If *hdul_psfs* is specified, then the 'OSAMP' header keyword takes precedence.
- **use_coeff** (*bool*) – If True, uses *calc_psf_from_coeff*, other WebbPSF's built-in *calc_psf*. Coefficients are much faster
- **Keywords** (*Ramp Gen*) –
- **==========** –
- **include_dark** (*bool*) – Add dark current?
- **include_bias** (*bool*) – Add detector bias?
- **include_ktc** (*bool*) – Add kTC noise?
- **include_rn** (*bool*) – Add readout noise per frame?
- **include_cpink** (*bool*) – Add correlated 1/f noise to all amplifiers?
- **include_upink** (*bool*) – Add uncorrelated 1/f noise to each amplifier?
- **include_acn** (*bool*) – Add alternating column noise?
- **apply_ipc** (*bool*) – Include interpixel capacitance?
- **apply_ppc** (*bool*) – Apply post-pixel coupling to linear analog signal?

- **include_refoffsets** (*bool*) – Include reference offests between amplifiers and odd/even columns?
- **include_refinst** (*bool*) – Include reference/active pixel instabilities?
- **include_colnoise** (*bool*) – Add in column noise per integration?
- **col_noise** (*ndarray or None*) – Option to explicitly specify column noise distribution in order to shift by one for subsequent integrations
- **amp_crosstalk** (*bool*) – Crosstalk between amplifiers?
- **add_crs** (*bool*) – Add cosmic ray events? See Robberto et al 2010 (JWST-STScI-001928).
- **cr_model** (*str*) – Cosmic ray model to use: 'SUNMAX', 'SUNMIN', or 'FLARES'.
- **cr_scale** (*float*) – Scale factor for probabilities.
- **latents** (*None*) – Apply persistence.
- **apply_nonlinearity** (*bool*) – Apply non-linearity?
- **random_nonlin** (*bool*) – Add randomness to the linearity coefficients?
- **prog_bar** (*bool*) – Show a progress bar for this ramp generation?

pynrc.simul.ngNRC.sources_to_slope

pynrc.simul.ngNRC.**sources_to_slope**(*source_table*, *nircam_obj*, *obs_params*, *tel_pointing*, *hdul_psfs=None*, *im_bg=None*, *cframe_out='sci'*, ***kwargs*)

Create a slope image from a table or sources

Parameters

- **source_table** (*astropy Table*) – Table of objects in across the region, including headers 'ra', 'dec', and object fluxes in NIRCam filter in vega mags where headers are labeled the filter name (e.g, 'F444W').
- **nircam_obj** (*pynrc.NIRCam*) – NIRCam instrument class for PSF generation.
- **obs_params** (*dict*) – Dictionary of parameters to populate DMS header. See *create_obs_params* in apt.py and *level1b_data_model* in dms.py.
- **tel_pointing** (webbpsf_ext.jwst_point) – JWST telescope pointing information. Holds pointing coordinates and dither information for a given telescope visit.
- **hdul_psfs** (*HDUList*) – Option to pass a pre-generated HDUList of PSFs across the field of view. If set to None, then generated automatically.
- **im_bg** (*None or ndarray*) – Option to specify a pre-generated image (or single value) of the Zodiacal background emission. If not specified, then gets automatically generating.
- **cframe_out** (*str*) – Desired output coordinate frame, either 'sci' or 'det'

Keyword Arguments

- **npsf_per_full_fov** (*int*) – Number of PSFs across one dimension of the instrument's field of view. If a coronagraphic observation, then this is for the nominal coronagrahic field of view.
- **sptype** (*str*) – Spectral type, such as 'A0V' or 'K2III'.
- **wfe_drift** (*float*) – Desired WFE drift value relative to default OPD.

- **osamp** (*int*) – Sampling of output PSF relative to detector sampling. If *hdul_psfs* is specified, then the 'OSAMP' header keyword takes precedence.
- **use_coeff** (*bool*) – If True, uses *calc_psf_from_coeff*, other WebbPSF's built-in *calc_psf*. Coefficients are much faster

pynrc.simul.ngNRC.xtalk_image

pynrc.simul.ngNRC.**xtalk_image**(*frame*, *det*, *coeffs=None*)

Create image of crosstalk signal

Add amplifier crosstalk to each frame in data cube

Parameters

- **frame** (*ndarray*) – An image to calculate and add crosstalk to.
- **det** (*pynrc.DetectorOps*) – Detector class corresponding to data.
- **coeffs** (*None or Table*) – Table of coefficients corresponding to detector crosstalk behavior.

pynrc.simul.apt

APT conversion procedures

Functions

build_dict_from_xml(xml_file, keys[, verbose])	Read in the .xml file from APT, and output dictionary of parameters.
create_det_class(visit_dict, exp_id, detname)	
create_obs_params(filt, pupil, mask, det, ...)	Generate obs_params dictionary
file_segmenting(det[, max_size_MB])	
gen_all_apt_visits(xml_file, pointing_file, ...)	Read in APT output files and return a dictionary that holds all necessary visit information to create an observation in DMS format.
gen_jwst_pointing(visit_dict, obs_params[, ...])	Create telescope pointing sequence for a given visit / exposure.
gen_pointing_info(*args, **kwargs)	**Deprecated.
get_ditherinfo(xml_file[, verbose])	Dither information
get_exp_type(visit_type, visit_mode, pupil)	Possible Exposure Types:
get_filter_info(xml_file[, verbose])	Filter information for each exposure
get_orient_specreq(xml_file)	Grab V3 PA Range special requirements from XML file
get_pointing_info(pointing_files[, propid, ...])	Read in information from APT's pointing file.
get_proposal_info(xml_file[, verbose])	Get basic program information
get_readmodes(xml_file[, verbose])	Readout information for each exposure
get_roll_info(xml_file)	
get_siaf_detectors(apname)	List detectors associated with SIAF aperture name
get_target_info(xml_file[, verbose])	Target for each each exposure

continues on next page

Table 15 – continued from previous page

get_tel_angles(ra, dec[, obs_date, obs_time])	For a given RA, Dec and date, return the telescope pitch and V3 PA angles.
get_timing_info(timing_json_file, ...)	
pitch_vs_time(xml_file, pointing_file, ...)	
populate_obs_params(visit_dict, exp_id, ...)	Create obs_params from visit dictionary
read_subarray_definition_file(filename)	Read in the file that contains a list of subarray names and related information
update_eng_detectors(visit_dict)	Update detectors used for engineering templates

pynrc.simul.apt.build_dict_from_xml

pynrc.simul.apt.**build_dict_from_xml**(*xml_file*, *keys*, *verbose=False*)

Read in the .xml file from APT, and output dictionary of parameters.

Parameters **infile (str)** – Path to input .xml file

Returns *dict* – Dictionary with extracted observation parameters

Raises **ValueError:** – If an .xml file is provided that includes an APT template that is not supported. If the .xml file includes a fiducial pointing override with an unknown subarray specification

pynrc.simul.apt.create_det_class

pynrc.simul.apt.**create_det_class**(*visit_dict*, *exp_id*, *detname*, *grp_id=1*, *seq_id=1*, *act_id='01'*)

pynrc.simul.apt.create_obs_params

pynrc.simul.apt.**create_obs_params**(*filt*, *pupil*, *mask*, *det*, *siaf_ap*, *ra_dec*, *date_obs*, *time_obs='12:00:00.000'*, *pa_v3=None*, *siaf_ap_obs=None*, *xyoff_idl=(0, 0)*, *visit_level='TARGET'*, *visit_type='SCIENCE'*, *time_series=False*, *time_visit_offset=0*, *time_exp_offset=0*, *segNum=None*, *segTot=None*, *int_range=None*, *filename=None*, ***kwargs*)

Generate obs_params dictionary

An obs_params dictionary is used to create a jwst data model (e.g., Level1bModel). Additional **kwargs will add/update elements to the final output dictionary.

Parameters

- **filt** (*str*) – Observed filter
- **pupil** (*str*) – Observed pupil mask (e.g., GRISMR, GRISMC, CIRCLYOT, etc)
- **det** (*DetectorOps*) – NIRCam detector operations class
- **siaf_ap** (*pysiaf Aperture*) – SIAF aperture class used for telescope pointing
- **ra_dec** (*tuple, list*) – RA and Dec in degrees associated with observation pointing
- **data_obs** (*str*) – YYYY-MM-DD
- **Keyword Arg**

- **==========**
- **time_obs** (*str*) – HH:MM:SS
- **pa_v3** (*float or None*) – Telescope V3 position angle. If set to None, then will automatically determine from date and ra/dec.
- **siaf_ap_obs** (*pysiaf Aperture*) – SIAF aperture class used for to observe (if different from *siaf_ap*)
- **xyoff_idl** (*tuple, list*) – (x,y) offset in arcsec ('idl' coords) to dither observation
- **visit_type** (*str*) – 'T_ACQ', 'CONFIRM', or 'SCIENCE'
- **time_series** (*bool*) – Is this a time series observation?
- **time_exp_offset** (*float*) – Exposure start time (in seconds) relative to beginning of observation execution.
- **segNum** (*int*) – The segment number of the current product. Only for TSO.
- **segTot** (*int*) – The total number of segments. Only for TSO.
- **int_range** (*list*) – Integration indices to use
- **filename** (*str or None*) – Name of output filename.

pynrc.simul.apt.file_segmenting

pynrc.simul.apt.**file_segmenting**(*det*, *max_size_MB=320*)

pynrc.simul.apt.gen_all_apt_visits

pynrc.simul.apt.**gen_all_apt_visits**(*xml_file*, *pointing_file*, *sm_acct_file*, *json_file*, *rand_seed=None*)

Read in APT output files and return a dictionary that holds all necessary visit information to create an observation in DMS format. Each visit is placed in an ordered dictionary according to that within the Smart Accounting file.

pynrc.simul.apt.gen_jwst_pointing

pynrc.simul.apt.**gen_jwst_pointing**(*visit_dict*, *obs_params*, *base_std=None*, *dith_std=None*, *rand_seed=None*, *rand_seed_base=None*)

Create telescope pointing sequence for a given visit / exposure.

Keyword Arguments

- **base_std** (*float or array-like or None*) – The 1-sigma pointing uncertainty per axis for telescope slew. If None, then standard deviation is chosen to be either 5 mas or 100 mas, depending on *use_ta* attribute (default: True).
- **dith_std** (*float or array-like or None*) – The 1-sigma pointing uncertainty per axis for dithers. If None, then standard deviation is chosen to be either 2.5 or 5 mas, depending on *use_sgd* attribute (default: True).
- **rand_seed** (*None or int*) – Random seed to use for generating repeatable random offsets.
- **rand_seed_base** (*None or int*) – Use a separate random seed for telescope slew offset. Then, *rand_seed* corresponds only to relative dithers. Useful for multiple exposures with same initial slew, but independent dither pattern realizations.

pynrc.simul.apt.gen_pointing_info

pynrc.simul.apt.**gen_pointing_info**(**args*, ***kwargs*)
Deprecated. Use ``gen_jwst_pointing`` instead. Create telescope pointing sequence for a given visit / exposure.

pynrc.simul.apt.get_ditherinfo

pynrc.simul.apt.**get_ditherinfo**(*xml_file*, *verbose=False*)
Dither information

pynrc.simul.apt.get_exp_type

pynrc.simul.apt.**get_exp_type**(*visit_type*, *visit_mode*, *pupil*)

Possible Exposure Types: NRC_DARK, NRC_FLAT, NRC_LED, NRC_FOCUS, NRC_TACQ, NRC_TACONFIRM, NRC_IMAGE, NRC_GRISM, NRC_CORON, NRC_WFSS, NRC_TSIMAGE, NRC_TSGRISM

pynrc.simul.apt.get_filter_info

pynrc.simul.apt.**get_filter_info**(*xml_file*, *verbose=False*)
Filter information for each exposure

pynrc.simul.apt.get_orient_specreq

pynrc.simul.apt.**get_orient_specreq**(*xml_file*)
Grab V3 PA Range special requirements from XML file

TODO: Value is specified in the aperture PA, not V3 PA. Need to calculate V3 PA.

pynrc.simul.apt.get_pointing_info

pynrc.simul.apt.**get_pointing_info**(*pointing_files*, *propid=0*, *verbose=False*, *all_inst=False*)
Read in information from APT's pointing file.

Parameters

- **file** (*str*) – Name of APT-exported pointing file to be read
- **propid** (*int*) – Proposal ID number (integer). This is used to create various ID fields
- **all_inst** (*bool*) – If False, only NIRCam, otherwise get all instruments.

Returns **pointing** (*dict*) – Dictionary of pointing-related information

Todo: extract useful information from header? check visit numbers set parallel proposal number correctly

pynrc.simul.apt.get_proposal_info

pynrc.simul.apt.**get_proposal_info**(*xml_file*, *verbose=False*)
Get basic program information

pynrc.simul.apt.get_readmodes

pynrc.simul.apt.**get_readmodes**(*xml_file*, *verbose=False*)
Readout information for each exposure

pynrc.simul.apt.get_roll_info

pynrc.simul.apt.**get_roll_info**(*xml_file*)

pynrc.simul.apt.get_siaf_detectors

pynrc.simul.apt.**get_siaf_detectors**(*apname*)
List detectors associated with SIAF aperture name

pynrc.simul.apt.get_target_info

pynrc.simul.apt.**get_target_info**(*xml_file*, *verbose=False*)
Target for each each exposure

pynrc.simul.apt.get_tel_angles

pynrc.simul.apt.**get_tel_angles**(*ra*, *dec*, *obs_date='2022-03-01'*, *obs_time='12:00:00'*)
For a given RA, Dec and date, return the telescope pitch and V3 PA angles.

pynrc.simul.apt.get_timing_info

pynrc.simul.apt.**get_timing_info**(*timing_json_file*, *smart_accounting_file*)

pynrc.simul.apt.pitch_vs_time

pynrc.simul.apt.**pitch_vs_time**(*xml_file*, *pointing_file*, *timing_json_file*, *smart_accounting_file*, *obs_date='2022-03-01'*, *obs_time='12:00:00'*, *pitch_init=None*, *nvals=1000*)

pynrc.simul.apt.populate_obs_params

pynrc.simul.apt.**populate_obs_params**(*visit_dict*, *exp_id*, *detname*, *date_obs*, *time_obs='12:00:00.000'*, *pa_v3=None*, *segNum=None*, *segTot=None*, *int_range=None*, *det=None*, *obs_params=None*, *grp_id=1*, *seq_id=1*, *act_id='01'*, ***kwargs*)

Create obs_params from visit dictionary

An obs_params dictionary is used to create a jwst data model (e.g., Level1bModel). If passing `obs_params` parameter, this gets updated based on the input arguments. Additional `**kwargs` will add/update elements to the final output dictionary

Parameters

- **visit_dict** (*dict*) – Uses gen_all_apt_visits() to create a dictionary of visit information. Each visit has a series of exposure IDs.
- **exp_id** (*int*) – Unique exposure ID generate observations
- **detname** (*str*) – Options NRC[A/B][1-5]
- **data_obs** (*str*) – YYYY-MM-DD
- **time_obs** (*str*) – HH:MM:SS
- **Keyword Arg**
- **==========**
- **pa_v3** (*float or None*) – Option to specify telescope V3 position angle. If not set, then automatically calculated from RA/Dec and observation date/time.
- **segNum** (*int*) – The segment number of the current product. Only for TSO.
- **segTot** (*int*) – The total number of segments. Only for TSO.
- **int_range** (*list*) – Integration indices to use
- **obs_params** (*dict*) – An initial obs_params dictionary. Any duplicate keywords will be updated.

pynrc.simul.apt.read_subarray_definition_file

pynrc.simul.apt.**read_subarray_definition_file**(*filename*)

Read in the file that contains a list of subarray names and related information

Parameters **filename** (*str*) – Name of the ascii file containing the table of subarray information

Returns **data** (*astropy.table.Table*) – Table containing subarray information

pynrc.simul.apt.update_eng_detectors

pynrc.simul.apt.**update_eng_detectors**(*visit_dict*)

Update detectors used for engineering templates

Engineering templates always use all detectors operated with the same subarray settings, but there are not always SIAF apertures for every detector for certain subarray settings (e.g., SUBGRISM), so some of the auto-generated detector from *get_siaf_detectors* will usually be incorrect.

Classes

AptInput([input_xml, pointing_file, ...])	Summary
DMS_input(xml_file, pointing_file, ...[, ...])	Class to generate a series of observation dictionaries in order to build DMS-like files.
ReadAPTXML()	Class to open and parse XML files from APT.

pynrc.simul.apt.AptInput

class pynrc.simul.apt.**AptInput**(*input_xml=None*, *pointing_file=None*, *output_dir=None*, *output_csv=None*, *observation_list_file=None*)

Bases: object

Summary

exposure_tab
Description

Type TYPE

input_xml
Description

Type str

observation_list_file
Description

Type str

obstab
Description

Type TYPE

output_csv
Description

Type TYPE

pointing_file
Description

Type str

__init__(*input_xml=None*, *pointing_file=None*, *output_dir=None*, *output_csv=None*, *observation_list_file=None*)

Methods

__init__([input_xml, pointing_file, ...])	
add_epochs(intab)	NOT CURRENTLY USED
add_observation_info(intab)	Add information about each observation.
add_options([parser, usage])	
base36encode(integer)	Translate a base 10 integer to base 36
check_aperture_override()	
combine_dicts(dict1, dict2)	Combine two dictionaries into a single dictionary.
create_input_table([verbose])	Main function for creating a table of parameters for each exposure
expand_for_detectors(input_dictionary)	Expand dictionary to have one entry per detector, rather than the one line per module that is in the input
extract_grism_aperture(apertures, filter_name)	In the case of a Grism observation (WFSS or GRISM TSO), where a given crossing filter is used, find the appropriate aperture to use.
extract_value(line)	Extract text from xml line
full_path(in_path)	If the input path is not None, expand any environment variables and make an absolute path.
get_pointing_info(file[, propid, verbose])	Read in information from APT's pointing file.
global_alignment_pointing(obs_dict)	Adjust the pointing dictionary information for global alignment observations.
tight_dithers(input_dict)	In NIRCam, when the 'FULL' dither pattern is used, it is possible to set the number of primary dithers to '3TIGHT' rather than just a number (e.g.

add_epochs(*intab*)
NOT CURRENTLY USED

add_observation_info(*intab*)
Add information about each observation.

Catalog names, dates, PAV3 values, etc., which are retrieved from the observation list yaml file.

Parameters **intab** (*obj*) – astropy.table.Table containing exposure information

Returns **intab** (*obj*) – Updated table with information from the observation list yaml file added.

base36encode(*integer*)
Translate a base 10 integer to base 36

Parameters **integer** (*int*) – a base 10 integer

Returns **integer** (*int*) – The integer translated to base 36

combine_dicts(*dict1*, *dict2*)
Combine two dictionaries into a single dictionary.

Parameters

- **dict1** (*dict*) – dictionary
- **dict2** (*dict*) – dictionary

Returns **combined** (*dict*) – Combined dictionary

`create_input_table`(*verbose=False*)
Main function for creating a table of parameters for each exposure

Parameters **verbose** (*bool*) – If True, extra information is printed to the log

`expand_for_detectors`(*input_dictionary*)
Expand dictionary to have one entry per detector, rather than the one line per module that is in the input

Parameters **input_dictionary** (*dict*) – dictionary containing one entry per module

Returns **observation_dictionary** (*dict*) – dictionary expanded to have one entry per detector

`extract_grism_aperture`(*apertures*, *filter_name*)
In the case of a Grism observation (WFSS or GRISM TSO), where a given crossing filter is used, find the appropriate aperture to use.

Parameters

- **apertures** (*list*) – List of possible grism apertures
- **filter_name** (*str*) – Name of crossing filter

Returns **apertures** (*list*) – Modified list containig the correct aperture

`extract_value`(*line*)
Extract text from xml line

Parameters **line** (*str*) – Line from xml file

Returns **line** (*str*) – Text between > and < in the input line

`full_path`(*in_path*)
If the input path is not None, expand any environment variables and make an absolute path. Return the updated path.

Parameters **in_path** (*str*) – Path to be expanded

Returns **in_path** (*str*) – Expanded, absolute path

`get_pointing_info`(*file*, *propid=0*, *verbose=False*)
Read in information from APT's pointing file.

Parameters

- **file** (*str*) – Name of APT-exported pointing file to be read
- **propid** (*int*) – Proposal ID number (integer). This is used to create various ID fields

Returns **pointing** (*dict*) – Dictionary of pointing-related information

---

Todo: extract useful information from header? check visit numbers set parallel proposal number correctly

---

`global_alignment_pointing`(*obs_dict*)
Adjust the pointing dictionary information for global alignment observations. Some of the entries need to be changed from NIRCam to FGS. Remember that not all observations in the dictionary will necessarily be WfscGlobalAlignment template. Be sure the leave all other templates unchanged.

Parameters **obs_dict** (*dict*) – Dictionary of observation parameters, as returned from add_observation_info()

Returns **obs_dict** (*dict*) – Dictionary with modified values for FGS pointing in Global Alignment templates

tight_dithers(*input_dict*)

In NIRCam, when the 'FULL' dither pattern is used, it is possible to set the number of primary dithers to '3TIGHT' rather than just a number (e.g. '3'). If the number of dithers is set to '3TIGHT' remove 'TIGHT' from the entries and leave only the number behind.

Parameters **input_dict** (*dict*) – Dictionary where each key points to a list containing observation details. For example, input_dict['PrimarDither'] is a list of the number of primary dithers for all observations

Returns **input_dict** (*dict*) – Updated dictionary where 'TIGHT' has been removed from PrimaryDither list

pynrc.simul.apt.DMS_input

class pynrc.simul.apt.**DMS_input**(*xml_file*, *pointing_file*, *json_file*, *sm_acct_file*, *save_dir=None*, *obs_date='2022-03-01'*, *obs_time='12:00:00'*, *pa_v3=None*, *rand_seed_init=None*)

Bases: object

Class to generate a series of observation dictionaries in order to build DMS-like files. Loads APT files to generate the necessary observation information.

Usage:

json_file = 'NRC-21.timing.json' sm_acct_file = 'NRC-21.smart_accounting' pointing_file = 'NRC-21.pointing' xml_file = 'NRC-21.xml'

obs_input = DMS_input(xml_file, pointing_file, json_file, sm_acct_file) # Update observation time and telescope PA obs_input.obs_date = '2022-03-01' obs_input.obs_time = '12:00:00.000' obs_input.pa_v3 = 45

Create a list of observation parameters obs_params_all = obs_input.gen_all_obs_params()

Select one of the parameter dictionaries obs_params = obs_params_all[0]

Create a series of science data based on observation params # that results in sci_data and zero_data 16-bit numpy arrays out_model = level1b_data_model(obs_params, sci_data, zero_data)

Save to FITS file # Performs minor updates to the saved FITS file save_level1b_fits(out_model, obs_params, save_dir)

__init__(*xml_file*, *pointing_file*, *json_file*, *sm_acct_file*, *save_dir=None*, *obs_date='2022-03-01'*, *obs_time='12:00:00'*, *pa_v3=None*, *rand_seed_init=None*)

Methods

__init__(xml_file, pointing_file, json_file, ...)	
gen_all_obs_params()	Generate a full set of parameters for all exposures
gen_obs_params(visit_id, exp_id, det_id[, ...])	Generate a single set of observation parameters for a given exposure
gen_pitch_array([nvals, pitch_init])	
make_jwst_point(visit_id, exp_id, detname[, ...])	Create jwst_point object

Attributes

obs_date	Date of observations
obs_time	Start time of observations
pa_v3	

gen_all_obs_params()
Generate a full set of parameters for all exposures

gen_obs_params(*visit_id*, *exp_id*, *det_id*, *det=None*, *seg_num=None*, *seg_tot=None*, *int_range=None*, *grp_id=1*, *seq_id=1*, *act_id='01'*)
Generate a single set of observation parameters for a given exposure

make_jwst_point(*visit_id*, *exp_id*, *detname*, *obs_params=None*, *base_std=0*, *dith_std=0*, *rand_seed=None*, *grp_id=1*, *seq_id=1*, *act_id='01'*)
Create jwst_point object

Parameters

- **visit_id** (*str*) – obsnum:visitnum, for example: '007:001'
- **exp_id** (*int*) – Exposure number
- **detname** (*str*) – Name of detector, such as 'NRCA5'.

Keyword Arguments

- **obs_params** (*dict*) – Option to pass any already generated obs_params dictionary rather than generating it automatically. Should be part of the associated visit.
- **base_std** (*float or None*) – The 1-sigma pointing uncertainty per axis for telescope slew. If None, then standard deviation is chosen to be either 5 mas or 100 mas, depending on *use_ta* attribute (default: True).
- **dith_std** (*float or None*) – The 1-sigma pointing uncertainty per axis for dithers. If None, then standard deviation is chosen to be either 2.5 or 5 mas, depending on *use_sgd* attribute (default: True).
- **rand_seed** (*None or int*) – Random seed to use for generating repeatable random offsets.

property obs_date
Date of observations

property obs_time
Start time of observations

pynrc.simul.apt.ReadAPTXML

class pynrc.simul.apt.**ReadAPTXML**
Bases: object

Class to open and parse XML files from APT. Can read templates for NircamImaging, NircamEngineeringImaging, WfscCommissioning, WfscGlobaLAlignment, WfscCoarsePhasing, WfscFinePhasing (incomplete), and NircamWfss modes.

apt
APT namespace for XML files

Type str

APTObservationParams
Dictionary of APT parameters that accumulates all parameters in all tiles and all observation. Passed out to be further parsed in the apt_inputs script

Type dict

obs_tuple_list
Compiled list of all the parameters for all tiles in a single observation

Type list

__init__()

Methods

__init__()	
add_exposure(dictionary, tup)	Add an exposure to the exposure dictionary
append_to_exposures_dictionary(...)	Append exposure(s) information from a dictionary to an existing exposures dictionary
get_tracking_type(observation)	Determine whether the observation uses sidereal or non-sidereal tracking
read_coarsephasing_template(template, ...)	
read_commissioning_template(template, ...)	
read_finephasing_template(template, ...)	
read_generic_imaging_template(template, ...)	Read imaging template content regardless of instrument.
read_globalalignment_template(template, ...)	
read_miri_coronagraphy_template(template, ...)	Parse a MIRI coronagraphy observation template from an APT xml file.
read_nircam_coronagraphy_template(template, ...)	Parse a NIRCam coronagraphy observation template from an APT xml file.
read_nircam_grism_time_series(template, ...)	Parse a NIRCam Grism Time Series observation template from an APT xml file.
read_nircam_imaging_time_series(template, ...)	Parse a NIRCam Imaging Time Series observation template from an APT xml file.
read_nircam_wfss_template(template, ...)	Parse a NIRCam WFSS observation template from an APT xml file.
read_niriss_ami_template(template, ...[, ...])	Parse a NIRISS AMI observation template from an APT xml file.
read_niriss_wfss_template(template, ...[, ...])	Parse a NIRISS WFSS observation template from an APT xml file.
read_parallel_exposures(obs, ...[, verbose])	Read the exposures of the parallel instrument.
read_xml(infile[, verbose])	Main function.

continues on next page

Table 20 – continued from previous page

separate_pupil_and_filter(filter_string)	Filters listed for NIRCam observations can take the form 'F164N+F444W' in cases where filters in the filter wheel and the pupil wheel are used in combination.

add_exposure(*dictionary*, *tup*)
Add an exposure to the exposure dictionary

Parameters

- **dictionary** (*dict*) – Information on individual exposures
- **tup** (*tuple*) – A tuple contianing information to add to dictionary as the next exposure

Returns **dictionary** (*dict*) – With new exposure added

append_to_exposures_dictionary(*exp_dictionary*, *exposure_seq_dict*, *prop_param_dict*)
Append exposure(s) information from a dictionary to an existing exposures dictionary

Parameters

- **exp_dictionary** (*dict*) – Dictionary containing information on multiple exposures
- **exposure_seq_dict** (*dict*) – Dictionary containing information on a single exposure. This dictionary should have the same keys as exp_dictionary. The contents of this dictionary will be added to exp_dictionary
- **prop_param_dict** (*dict*) – A dictionary containing proposal-wide information, such as title and PI name
- **Reutrns**
- **———-**
- **exp_dictionary** (*dict*) – With the new exposure(s) added

get_tracking_type(*observation*)
Determine whether the observation uses sidereal or non-sidereal tracking

Parameters **observation** (*etree xml element*) – xml content of observation

Returns **tracking_type** (*str*) – "sidereal" or "non-sidereal" based on the target used in the observation

read_generic_imaging_template(*template*, *template_name*, *obs*, *proposal_parameter_dictionary*, *verbose=False*, *parallel=False*)
Read imaging template content regardless of instrument.

Save content to object attributes. Support for coordinated parallels is included.

Parameters

- **template** (*etree xml element*) – xml content of template
- **template_name** (*str*) – name of the template
- **obs** (*etree xml element*) – xml content of observation
- **proposal_parameter_dictionary** (*dict*) – Dictionary of proposal parameters to extract from template

Returns **exposures_dictionary** (*OrderedDict*) – Dictionary containing relevant exposure parameters

`read_miri_coronagraphy_template`(*template*, *template_name*, *obs*, *proposal_param_dict*, *parallel=False*, *verbose=False*)

Parse a MIRI coronagraphy observation template from an APT xml file. Produce an exposure dictionary that lists all exposures (excluding dithers) from the template.

Parameters

- **template** (*lxml.etree._Element*) – Template section from APT xml
- **template_name** (*str*) – The type of template (e.g. 'NirissAmi')
- **obs** (*lxml.etree._Element*) – Observation section from APT xml
- **proposal_param_dict** (*dict*) – Dictionary of proposal level information from the xml file (e.g. PI, Science Category, etc)
- **parallel** (*bool*) – If True, template should be for parallel observations. If False, NIRISS WFSS observation is assumed to be prime

Returns

- **exposures_dictionary** (*dict*) – Dictionary containing details on all exposures contained within the template. These details include things like filter, pupil, readout pattern, subarray, etc
- **exp_len** (*int*) – Dictionary length to use when comparing to that from a parallel observation. This is not necessarily the same as the true length of the dictionary due to the way in which APT groups overvations

`read_nircam_coronagraphy_template`(*template*, *template_name*, *obs*, *proposal_param_dict*, *parallel=False*, *verbose=False*)

Parse a NIRCam coronagraphy observation template from an APT xml file. Produce an exposure dictionary that lists all exposures (excluding dithers) from the template.

Parameters

- **template** (*lxml.etree._Element*) – Template section from APT xml
- **template_name** (*str*) – The type of template (e.g. 'NirissAmi')
- **obs** (*lxml.etree._Element*) – Observation section from APT xml
- **proposal_param_dict** (*dict*) – Dictionary of proposal level information from the xml file (e.g. PI, Science Category, etc)
- **parallel** (*bool*) – If True, template should be for parallel observations. If False, NIRISS WFSS observation is assumed to be prime

Returns

- **exposures_dictionary** (*dict*) – Dictionary containing details on all exposures contained within the template. These details include things like filter, pupil, readout pattern, subarray, etc
- **exp_len** (*int*) – Dictionary length to use when comparing to that from a parallel observation. This is not necessarily the same as the true length of the dictionary due to the way in which APT groups overvations

`read_nircam_grism_time_series`(*template*, *template_name*, *obs*, *proposal_parameter_dictionary*)

Parse a NIRCam Grism Time Series observation template from an APT xml file. Produce an exposure dictionary that lists all exposures (excluding dithers) from the template

Parameters

- **template** (*lxml.etree._Element*) – Template section from APT xml

- **template_name** (*str*) – The type of template (e.g. 'NirissWfss')
- **obs** (*lxml.etree._Element*) – Observation section from APT xml
- **proposal_param_dict** (*dict*) – Dictionary of proposal level information from the xml file (e.g. PI, Science Category, etc)

Returns **exposures_dictionary** (*dict*) – Dictionary containing details on all exposures contained within the template. These details include things like filter, pupil, readout pattern, subarray, etc. Specifically for Grism Time Series, there will be entries for the TA exposure and the Time Series exposure.

`read_nircam_imaging_time_series`(*template*, *template_name*, *obs*, *proposal_parameter_dictionary*)
Parse a NIRCam Imaging Time Series observation template from an APT xml file. Produce an exposure dictionary that lists all exposures (excluding dithers) from the template

Parameters

- **template** (*lxml.etree._Element*) – Template section from APT xml
- **template_name** (*str*) – The type of template (e.g. 'NirissWfss')
- **obs** (*lxml.etree._Element*) – Observation section from APT xml
- **proposal_param_dict** (*dict*) – Dictionary of proposal level information from the xml file (e.g. PI, Science Category, etc)

Returns **exposures_dictionary** (*dict*) – Dictionary containing details on all exposures contained within the template. These details include things like filter, pupil, readout pattern, subarray, etc. Specifically for Grism Time Series, there will be entries for the TA exposure and the Time Series exposure.

`read_nircam_wfss_template`(*template*, *template_name*, *obs*, *proposal_param_dict*)
Parse a NIRCam WFSS observation template from an APT xml file. Produce an exposure dictionary that lists all exposures (excluding dithers) from the template.

Parameters

- **template** (*lxml.etree._Element*) – Template section from APT xml
- **template_name** (*str*) – The type of template (e.g. 'NirissWfss')
- **obs** (*lxml.etree._Element*) – Observation section from APT xml
- **proposal_param_dict** (*dict*) – Dictionary of proposal level information from the xml file (e.g. PI, Science Category, etc)

Returns **exposures_dictionary** (*dict*) – Dictionary containing details on all exposures contained within the template. These details include things like filter, pupil, readout pattern, subarray, etc

`read_niriss_ami_template`(*template*, *template_name*, *obs*, *proposal_param_dict*, *parallel=False*, *verbose=False*)
Parse a NIRISS AMI observation template from an APT xml file. Produce an exposure dictionary that lists all exposures (excluding dithers) from the template.

Parameters

- **template** (*lxml.etree._Element*) – Template section from APT xml
- **template_name** (*str*) – The type of template (e.g. 'NirissAmi')
- **obs** (*lxml.etree._Element*) – Observation section from APT xml

- **proposal_param_dict** (*dict*) – Dictionary of proposal level information from the xml file (e.g. PI, Science Category, etc)
- **parallel** (*bool*) – If True, template should be for parallel observations. If False, NIRISS WFSS observation is assumed to be prime

Returns

- **exposures_dictionary** (*dict*) – Dictionary containing details on all exposures contained within the template. These details include things like filter, pupil, readout pattern, subarray, etc
- **exp_len** (*int*) – Dictionary length to use when comparing to that from a parallel observation. This is not necessarily the same as the true length of the dictionary due to the way in which APT groups overvations

`read_niriss_wfss_template`(*template*, *template_name*, *obs*, *proposal_param_dict*, *parallel=False*, *verbose=False*)

Parse a NIRISS WFSS observation template from an APT xml file. Produce an exposure dictionary that lists all exposures (excluding dithers) from the template.

Parameters

- **template** (*lxml.etree._Element*) – Template section from APT xml
- **template_name** (*str*) – The type of template (e.g. 'NirissWfss')
- **obs** (*lxml.etree._Element*) – Observation section from APT xml
- **proposal_param_dict** (*dict*) – Dictionary of proposal level information from the xml file (e.g. PI, Science Category, etc)
- **parallel** (*bool*) – If True, template should be for parallel observations. If False, NIRISS WFSS observation is assumed to be prime

Returns

- **exposures_dictionary** (*dict*) – Dictionary containing details on all exposures contained within the template. These details include things like filter, pupil, readout pattern, subarray, etc
- **exp_len** (*int*) – Dictionary length to use when comparing to that from a parallel observation. This is not necessarily the same as the true length of the dictionary due to the way in which APT groups overvations

`read_parallel_exposures`(*obs*, *exposures_dictionary*, *proposal_parameter_dictionary*, *verbose=False*)

Read the exposures of the parallel instrument.

Parameters

- **obs** (*APT xml element*) – Observation section of xml file
- **exposures_dictionary** (*dict*) – Exposures of the prime instrument
- **proposal_parameter_dictionary** (*dict*) – Parameters to extract
- **verbose** (*bool*) – Verbosity

Returns **parallel_exposures_dictionary** (*dict*) – Parallel exposures.

`read_xml`(*infile*, *verbose=False*)

Main function. Read in the .xml file from APT, and output dictionary of parameters.

Parameters **infile (str)** – Path to input .xml file

Returns *dict* – Dictionary with extracted observation parameters

Raises `ValueError:` – If an .xml file is provided that includes an APT template that is not supported. If the .xml file includes a fiducial pointing override with an unknown subarray specification

`separate_pupil_and_filter`(*filter_string*)

Filters listed for NIRCam observations can take the form 'F164N+F444W' in cases where filters in the filter wheel and the pupil wheel are used in combination. This function separates the two values.

Parameters **filter_string** (*str*) – Filter name as given in xml file from APT

Returns

- **filter_name** (*str*) – Name of the filter in the filter wheel
- **pupil_name** (*str*) – Name of the filter in the pupil wheel

pynrc.simul.dms

DMS data format routines

Functions

DMS_filename(obs_id_info, detname[, segNum, ...])	jw<ppppp><ooo><vvv>_<gg><s><aa>_<eeeee>(-<"seg"NNN>)_<detector>_<prodType>.fits
compute_local_roll(pa_v3, ra_ref, dec_ref, ...)	Computes the position angle of V3 (measured N to E) at the reference point of an aperture.
create_DMS_HDUList(sci_data, zero_data, ...)	Save Level 1b to FITS file
create_group_entry(integration, groupnum, ...)	Add the GROUP extension to the output file
dec_to_base36(val)	Convert decimal integer to base 36 (0-Z)
jw_obs_id(pid, obs_num, visit_num, visit_gp, ...)	JWST Observation info and file naming convention
level1b_data_model(obs_params[, sci_data, ...])	obs_params : dict
populate_group_table(starttime, grouptime, ...)	Create some reasonable values to fill the GROUP extension table.
save_level1b_fits(outModel, obs_params[, ...])	Save Level1bModel to FITS and update headers
update_dms_headers(filename, obs_params)	Given the name of a valid partially populated level 1b JWST file, add a couple simple WCS parameters from the SIAF keywords, which contain information about the telescope pointing.
update_headers_pynrc_info(filename, ...)	Add pynrc info to headers

pynrc.simul.dms.DMS_filename

pynrc.simul.dms.`DMS_filename`(*obs_id_info*, *detname*, *segNum=None*, *prodType='uncal'*)

jw<ppppp><ooo><vvv>_<gg><s><aa>_<eeeee>(-<"seg"NNN>)_<detector>_<prodType>.fits

pynrc.simul.dms.compute_local_roll

pynrc.simul.dms.**compute_local_roll**(*pa_v3*, *ra_ref*, *dec_ref*, *v2_ref*, *v3_ref*)
Computes the position angle of V3 (measured N to E) at the reference point of an aperture.

Parameters

- **pa_v3** (*float*) – Position angle of V3 at (V2, V3) = (0, 0) [in deg]
- **v2_ref, v3_ref** (*float*) – Reference point in the V2, V3 frame [in arcsec]
- **ra_ref, dec_ref** (*float*) – RA and DEC corresponding to V2_REF and V3_REF, [in deg]

Returns **new_roll** (*float*) – The value of ROLL_REF (in deg)

pynrc.simul.dms.create_DMS_HDUList

pynrc.simul.dms.**create_DMS_HDUList**(*sci_data*, *zero_data*, *obs_params*, *save_dir=None*, ***kwargs*)
Save Level 1b to FITS file

pynrc.simul.dms.create_group_entry

pynrc.simul.dms.**create_group_entry**(*integration*, *groupnum*, *endday*, *endmilli*, *endsubmilli*, *endgroup*, *xd*, *yd*, *gap*, *comp_code*, *comp_text*, *barycentric*, *heliocentric*)
Add the GROUP extension to the output file

Parameters

- **integration** (*int*) – Integration number
- **group_number** (*int*) – Group number
- **endday** (*int*) – Days since Jan 1 2000
- **endmilli** (*integer*) – Milliseconds of the day for given time
- **endsubmilli** (*int*) – Time since last millisecond?
- **endgroup** (*str*) – End group time, e.g. '2016-01-18T02:43:26.061'
- **xd** (*int*) – Number_of_columns e.g. 2048
- **yd** (*int*) – Number_of_rows e.g. 2048
- **gap** (*int*) – Number of gaps in telemetry
- **comp_code** (*int*) – Completion code number e.g. 0 (nominal?)
- **comp_text** (*str*) –

  Completion code text e.g. 'COMPLETE'-from howard 'Normal Completion' - from mark
- **barycentric** (*float*) – Barycentric end time (mjd) 57405.11165225
- **heliocentric** (*float*) – Heliocentric end time (mjd) 57405.1163058

Returns **group** (*nump.ndarray*) – Input values organized into format needed for group entry in JWST formatted file

pynrc.simul.dms.dec_to_base36

pynrc.simul.dms.**dec_to_base36**(*val*)
Convert decimal integer to base 36 (0-Z)

pynrc.simul.dms.jw_obs_id

pynrc.simul.dms.**jw_obs_id**(*pid*, *obs_num*, *visit_num*, *visit_gp*, *seq_id*, *act_id*, *exp_num*)
JWST Observation info and file naming convention

jw<ppppp><ooo><vvv>_<gg><s><aa>_<eeeee>(-<"seg"NNN>)_<detector>_<prodType>.fits

ppppp: program ID number ooo: observation number vvv: visit number

gg: visit group s: parallel sequence ID (1=prime, 2-5=parallel) aa: activity number (base 36) (only for WFSC, coarse and fine phasing)

eeeee: exposure number segNNN: the text "seg" followed by a three-digit segment number (optional)

detector: detector name (e.g. 'nrca1', 'nrcblong', 'mirimage')

prodType: product type identifier (e.g. 'uncal', 'rate', 'cal')

An example Stage 1 product FITS file name is: jw93065002001_02101_00001_nrca1_rate.fits

pynrc.simul.dms.level1b_data_model

pynrc.simul.dms.**level1b_data_model**(*obs_params*, *sci_data=None*, *zero_data=None*)

obs_params [dict] Dictionary of parameters to populate DMS header. See *create_DMS_HDUList* in dms.py.

pynrc.simul.dms.populate_group_table

pynrc.simul.dms.**populate_group_table**(*starttime*, *grouptime*, *ramptime*, *numint*, *numgroup*, *ny*, *nx*)
Create some reasonable values to fill the GROUP extension table. These will not be completely correct because access to other ssb scripts and more importantly, databases, is necessary. But they should be close.

Parameters

- **starttime** (*astropy.time.Time*) – Starting time of exposure
- **grouptime** (*float*) – Exposure time of a single group (seconds)
- **ramptime** (*float*) – Exposure time of the entire exposure (seconds)
- **numint** (*int*) – Number of integrations in data
- **numgroup** (*int*) – Number of groups per integration
- **ny** (*int*) – Number of pixels in the y dimension
- **nx** (*int*) – Number of pixels in the x dimension

Returns **grouptable** (*numpy.ndarray*) – Group extension data for all groups in the exposure

pynrc.simul.dms.save_level1b_fits

pynrc.simul.dms.**save_level1b_fits**(*outModel*, *obs_params*, *save_dir=None*, ***kwargs*)
Save Level1bModel to FITS and update headers

pynrc.simul.dms.update_dms_headers

pynrc.simul.dms.**update_dms_headers**(*filename*, *obs_params*)
Given the name of a valid partially populated level 1b JWST file, add a couple simple WCS parameters from the SIAF keywords, which contain information about the telescope pointing.

It presumes all the accessed keywords are present from the JWST pipeline data model.

Parameters

- **filename** (*str*) – file name
- **obs_params** (*dict*) – Dictionary of parameters to populate DMS header. See *create_DMS_HDUList* in dms.py.

pynrc.simul.dms.update_headers_pynrc_info

pynrc.simul.dms.**update_headers_pynrc_info**(*filename*, *obs_params*, ***kwargs*)
Add pynrc info to headers

pynrc.simul.skyvec2ins

skyvec2ins JWST Coronagraph Visibility Calculator

Developed by Chris Stark (cstark@stsci.edu), translated to Python from IDL by Joseph Long (jlong@stsci.edu), Brendan Hagan, Bryony Nickson (bnickson@stsci.edu) and Mees Fix (mfix@stsci.edu)

The allowed pointing of JWST leads to target visibility that depends on ecliptic latitude, and the range of roll angles allowed depends on solar elongation. The allowed PAs for a target can thus be a complicated function of time. As a result, it can be difficult to 1) understand the possible orientations of a given target on the detector, 2) determine the ideal roll angle offsets for multi-roll observations, and 3) determine a group of targets that are simultaneously visible. The JWST Coronagraph Visibility Calculator (CVC) was created to address these issues and assist with creating APT programs and diagnosing scheduling errors.

We stress that the CVC is designed to provide quick illustrations of the possible observable orientations for a given target. As such, the CVC rapidly approximates JWST's pointing restrictions and does not query the official JWST Proposal Constraint Generator (PCG). The CVC does not include detailed pointing restrictions like Earth and Moon avoidance, etc. Additionally, results may differ from official constraints by a degree or so. Users should treat the results as close approximations.

Functions

ad2lb(alpha_rad, delta_rad)	Convert equatorial coordinates (RA, Dec, i.e. alpha, delta) to ecliptic coordinates (lambda, beta) according to Eq 3 in Leinert et al. 1998.
ei2lb(elong, inc)	Convert alternative ecliptic coordinates (epsilon, i) to ecliptic coordinates (lambda-lambda_sun, beta) according to Eq 12 in Leinert et al. 1998.
lb2ad(lambda_rad, beta_rad)	Convert ecliptic coordinates (lambda, beta) to equatorial coordinates (RA, Dec, i.e. alpha, delta) according to Eq 4 in Leinert et al. 1998.
lb2ei(lmlsun, beta)	Convert ecliptic coordinates (lambda-lambda_sun, beta) to alternative ecliptic coordinates (epsilon, i) according to Eq 11 in Leinert et al. 1998.
skyvec2ins(ra, dec, start_date[, npoints, ...])	JWST coronagraphic target visibility calculator.
sun_ecliptic_longitude(start_date)	Compute ecliptic longitude of sun on a given start date using equations from http://aa.usno.navy.mil/faq/docs/SunApprox.php [broken].

pynrc.simul.skyvec2ins.ad2lb

pynrc.simul.skyvec2ins.**ad2lb**(*alpha_rad*, *delta_rad*)

Convert equatorial coordinates (RA, Dec, i.e. alpha, delta) to ecliptic coordinates (lambda, beta) according to Eq 3 in Leinert et al. 1998.

Parameters

- **alpha_rad** (*ndarray*) – Right ascension in radians.
- **delta_rad** (*ndarray*) – Declination in radians.
- **lambda_rad** (*ndarray*) – Ecliptic longitude in radians.
- **beta_rad** – Ecliptic latitude in radians.

pynrc.simul.skyvec2ins.ei2lb

pynrc.simul.skyvec2ins.**ei2lb**(*elong*, *inc*)

Convert alternative ecliptic coordinates (epsilon, i) to ecliptic coordinates (lambda-lambda_sun, beta) according to Eq 12 in Leinert et al. 1998.

Parameters

- **elong** (*numpy.ndarray*) – Elongation (angular distance from the sun) in radians.
- **inc** (*ndarray*) – Inclination (position angle counted from the ecliptic counterclockwise) in radians.

Returns

- **lmlsun** (*numpy.ndarray*) – Differential helioecliptic longitude in radians.
- **beta** (*numpy.ndarray*) – Ecliptic latitude in radians.

pynrc.simul.skyvec2ins.lb2ad

pynrc.simul.skyvec2ins.**lb2ad**(*lambda_rad*, *beta_rad*)

Convert ecliptic coordinates (lambda, beta) to equatorial coordinates (RA, Dec, i.e. alpha, delta) according to Eq 4 in Leinert et al. 1998.

Parameters

- **lambda_rad** (*ndarray*) – Ecliptic longitude in radians.
- **beta_rad** (*numpy.ndarray*) – Ecliptic latitude in radians.

Returns

- **alpha** (*ndarray*) – Equatorial Right Ascension in radians.
- **delta** (*ndarray*) – Equatorial Declination in radians.

pynrc.simul.skyvec2ins.lb2ei

pynrc.simul.skyvec2ins.**lb2ei**(*lmlsun*, *beta*)

Convert ecliptic coordinates (lambda-lambda_sun, beta) to alternative ecliptic coordinates (epsilon, i) according to Eq 11 in Leinert et al. 1998.

Parameters

- **beta** (*numpy.ndarray*) – Ecplitic longitude in radians.
- **lmlsun** (*numpy.ndarray*) – Ecliptic lattitude in radians.

Returns

- **elong** (*numpy.ndarray*) – Elongation (angular distance from the sun to the field-of-view) in radians.
- **inc** (*numpy.ndarray*) – Inclination (position angle counted from the ecliptic counterclockwise) in radians.

pynrc.simul.skyvec2ins.skyvec2ins

pynrc.simul.skyvec2ins.**skyvec2ins**(*ra*, *dec*, *start_date*, *npoints=360*, *nrolls=15*, *maxvroll=7.0*)

JWST coronagraphic target visibility calculator.

Parameters

- **ra** (*float*) – Right ascension of science target in decimal degrees (0-360)
- **dec** (*float*) – Declination of science target in decimal degrees (-90, 90)
- **start_date** (*datetime.datetime*) – Start date of the year-long interval evaluated by skyvec2ins
- **npoints** (*int*) – Number of points to sample in the year-long interval to find observable dates (default: 360)
- **nrolls** (*int*) – Number of roll angles in the allowed roll angle range to sample at each date (default: 15)
- **maxvroll** (*float*) – Maximum number of degrees positive or negative roll around the boresight to allow (as designed: 7.0)
- **.. note::** – *lambda_rad0* is the longitude of quadrature at day 0 of the code, so it should be 90 deg W of the solar longitude.

Returns

- **x** (*numpy.ndarray*) – Float array of length *npoints* containing days from starting date
- **observable** (*numpy.ndarray*) – uint8 array of shape (*nrolls*, *npoints*) that is 1 where the target is observable and 0 otherwise
- **elongation_rad** (*numpy.ndarray*) – Float array of length *npoints* containing elongation of the observatory in radians
- **roll_rad** (*numpy.ndarray*) – Float array of shape (*nrolls*, *npoints*) containing V3 PA in radians
- **c1_x, c1_y, c2_x, c2_y, c3_x, c3_y** (*numpy.ndarray*) – Float array of shape (*nrolls*, *npoints*) containing the location of the companions in "Idl" (ideal) frame coordinates
- **n_x, n_y, e_x, e_y** (*numpy.ndarray*) – Float array of shape (*nrolls*, *npoints*) containing the location of a reference "north" vector and "east" vector from the center in "Idl" (ideal) frame coordinates

pynrc.simul.skyvec2ins.sun_ecliptic_longitude

pynrc.simul.skyvec2ins.**sun_ecliptic_longitude**(*start_date*)

Compute ecliptic longitude of sun on a given start date using equations from http://aa.usno.navy.mil/faq/docs/SunApprox.php [broken].

Parameters **start_date** (*datetime*) – Start date of the year-long interval evaluated by skyvec2ins.

Returns **lambda_sun** (*float*) – The longitude of quadrature at day 0.

1.9.4 Data Reduction

reduce.calib
reduce.ref_pixels

pynrc.reduce.calib

Functions

apply_linearity(cube, det, coeff_dict)	Apply pixel linearity corrections to ramp
apply_nonlin(cube, det, coeff_dict[, ...])	Apply pixel non-linearity to ideal ramp
broken_pink_powspec(freq, scales[, fcut1, ...])	
calc_cdsnoise(data[, temporal, spatial, ...])	Calculate CDS noise from input image cube
calc_eff_noise(allfiles[, superbias, ...])	Determine Effective Noise
calc_ktc(bias_sigma_arr[, binsize, return_std])	Calculate kTC (Reset) Noise
calc_linearity_coeff(data, sat_vals, well_depth)	counts_cut : None or float
calc_nonlin_coeff(data, sat_vals, well_depth)	counts_cut : None or float
chisqr_red(yvals[, yfit, err, dof, err_func])	Calculate reduced chi square metric

continues on next page

Table 24 – continued from previous page

cube_fit(tarr, data[, bias, sat_vals, ...])	
deconv_single_image(im, kfft)	Image deconvolution for a kernel
find_group_sat(file[, DMS, bias, sat_vals, ...])	Group at which 98% of pixels are saturated
find_sat(data[, bias, ref_info, bit_depth])	Given a data cube, find the values in ADU in which data reaches hard saturation.
fit_corr_powspec(freq, ps[, flim1, flim2, alpha])	Fit Correlated Noise Power Spectrum
fit_func_var_ex(params, det, patterns, ...)	Function for lsq fit to get excess variance
gen_cds_dict(allfiles[, DMS, superbias, ...])	Generate dictionary of CDS noise info
gen_col_variations(allfiles[, super_bias, ...])	Create a series of column offset models
gen_ref_dict(allfiles, super_bias[, ...])	Generate Reference Pixel Behavior Dictionary
gen_super_bias(allfiles[, DMS, mn_func, ...])	Generate a Super Bias Image
gen_super_dark(allfiles[, super_bias, DMS])	Average together all dark ramps to create a super dark ramp.
gen_super_ramp(allfiles[, super_bias, DMS, ...])	Average together all linearity ramps to create a super ramp.
get_bias_offsets(data[, nchan, ref_bot, ...])	Get Reference Bias Characteristics
get_fits_data(fits_file[, return_header, ...])	Read in FITS file data
get_flat_fields(im_slope[, split_low_high, ...])	Calculate QE variations in flat field
get_freq_array(pow_spec[, dt, nozero, npix_odd])	Return frequencies associated with power spectrum
get_ipc_kernel(imdark[, tint, boxsize, ...])	Derive IPC/PPC Convolution Kernels
get_linear_coeffs(allfiles[, super_bias, ...])	
get_oddeven_offsets(data[, nchan, ref_bot, ...])	Even/Odd Column Offsets
get_power_spec(data[, nchan, calc_cds, ...])	Calculate the power spectrum of an input data ramp in a variety of ways.
get_power_spec_all(allfiles[, super_bias, ...])	Return the average power spectra (white, 1/f noise correlated and uncorrelated) of all FITS files.
get_ref_instability(data[, nchan, ref_bot, ...])	Reference Pixel Instability
ipc_deconvolve(imarr, kernel[, kfft])	Simple IPC image deconvolution
pixel_linearity_gains(frame, coeff_arr[, ...])	Given some image data and coefficient
plot_dark_histogram(im, ax[, binsize, ...])	
plot_kernel(kern[, ax, return_figax])	Plot image of IPC or PPC kernel
pow_spec_ramp(data, nchan[, nroh, nfoh, ...])	Get power spectrum within frames of input ramp
pow_spec_ramp_pix(data, nchan[, ...])	Get power spectrum of pixels within ramp
ppc_deconvolve(im, kernel[, kfft, nchans, ...])	PPC image deconvolution
ramp_derivative(y[, dx, fit0, deg, ifit])	Get the frame-by-frame derivative of a ramp.
ramp_resample(data, det_new[, return_zero_frame])	Resample a RAPID dataset into new detector format
time_to_sat(data, sat_vals[, dt, sat_calc, ...])	Determine time of saturation

pynrc.reduce.calib.apply_linearity

pynrc.reduce.calib.**apply_linearity**(*cube*, *det*, *coeff_dict*)

Apply pixel linearity corrections to ramp

Linearize a bias-subtracted, ref-pixel-corrected ramp and convert from units of DN to electrons.

Parameters

- **cube** (*ndarray*) – Ramp data in DN of size (nz,ny,nx). Should be bias-subtracted and ref-pixel-corrected. Should match det subarray shape.

- **det** (*Detector Class*) – NIRCam detector class.
- **coeff_dict** (*ndarray*) –

  Dictionary holding coefficient information:

  – 'cf_nonlin' : Set of polynomial coefficients of size (ncf,ny,nx).

  – 'use_legendre' : Coefficients use Legendre polynomials?

  – 'lxmap' : Legendre polynomial normalization range, usually [0,1e5]

  Possible to separately fit lower flux values:

  – 'counts_cut' : Flux cut-off value in electrons

  – 'cf_nonlin_low' : Coefficients for flux values below counts_cut

pynrc.reduce.calib.apply_nonlin

pynrc.reduce.calib.**apply_nonlin**(*cube*, *det*, *coeff_dict*, *randomize=True*, *rand_seed=None*)

Apply pixel non-linearity to ideal ramp

Given a simulated cube of data in electrons, apply non-linearity coefficients to obtain values in DN (ADU). This

Parameters

- **cube** (*ndarray*) – Simulated ramp data in e-. These should be intrinsic flux values with Poisson noise, but prior to read noise, kTC, IPC, etc. Size (nz,ny,nx). Should match det subarray shape.
- **det** (*Detector Class*) – Desired detector class output
- **coeff_dict** (*ndarray*) –

  Dictionary holding coefficient information:

  – 'cf_nonlin' : Set of polynomial coefficients of size (ncf,ny,nx).

  – 'use_legendre' : Coefficients use Legendre polynomials?

  – 'lxmap' : Legendre polynomial normalization range, usually [0,1e5]

  – 'sat_vals' : An image indicating what saturation levels in DN for each pixel

  Possible to separately fit lower flux values:

  – 'counts_cut' : Flux cut-off value in electrons

  – 'cf_nonlin_low' : Coefficients for flux values below counts_cut

  To include randomization in line with observed variation:

  – 'cflin0_mean' : Average 0th-order coefficient

  – 'cflin0_std' : Measured standard deviation of 0th-order coefficent

  – 'corr_slope' : Slope of linear correlation between 0th-order and higher orders

  – 'corr_intercept' : Intercept of linear Correaltion between 0th-order and higher orders

Keyword Arguments **randomize** (*bool*) – Add variation to the non-linearity coefficients

pynrc.reduce.calib.broken_pink_powspec

pynrc.reduce.calib.**broken_pink_powspec**(*freq*, *scales*, *fcut1=1*, *fcut2=10*, *alpha=- 1*, ***kwargs*)

pynrc.reduce.calib.calc_cdsnoise

pynrc.reduce.calib.**calc_cdsnoise**(*data*, *temporal=True*, *spatial=True*, *std_func=<function std>*)
Calculate CDS noise from input image cube

pynrc.reduce.calib.calc_eff_noise

pynrc.reduce.calib.**calc_eff_noise**(*allfiles*, *superbias=None*, *temporal=True*, *spatial=True*, *ng_all=None*, *DMS=False*, *kw_ref=None*, *std_func=<function medabsdev>*, *kernel_ipc=None*, *kernel_ppc=None*, *read_pattern='RAPID'*)

Determine Effective Noise

Calculates the slope noise (in DN/sec) assuming a linear fits to a variety number of groups. The idea is to visualize the reduction in noise as you increase the number of groups in the fit and compare it to theoretical predictions (ie., slope noise formula).

Parameters

- **allfiles** (*list*) – List of input file names.
- **DMS** (*bool*) – Are files DMS formatted?
- **superbias** (*ndarray*) – Super bias to subtract from each dataset.
- **temporal** (*bool*) – Calculate slope noise using pixels' temporal distribution?
- **spatial** (*bool*) – Calcualte slope noise using pixel spatial distribution?
- **ng_all** (*array-like*) – Array of group to perform linear fits for slope calculations.
- **kw_ref** (*dict*) – Dictionary of keywords to pass to reference correction routine.
- **std_func** (*func*) – Function for calculating spatial distribution.
- **kernel_ipc** (*ndarray*) – IPC kernel to perform deconvolution on slope images.
- **kernel_ppc** (*ndarray*) – Similar to *kernel_ipc* except for PPC.
- **read_pattern** (*string*) – Reformulate data as if it were acquired using a read pattern other than RAPID.

pynrc.reduce.calib.calc_ktc

pynrc.reduce.calib.**calc_ktc**(*bias_sigma_arr*, *binsize=0.25*, *return_std=False*)
Calculate kTC (Reset) Noise

Use the uncertainty image from super bias to calculate the kTC noise. This function generates a histogram of the pixel uncertainties and takes the peak of the distribution as the pixel reset noise.

Parameters

- **bias_sigma_arr** (*ndarray*) – Image of the pixel uncertainties.
- **binsize** (*float*) – Size of the histogram bins.
- **return_std** (*bool*) – Also return the standard deviation of the distribution?

pynrc.reduce.calib.calc_linearity_coeff

pynrc.reduce.calib.**calc_linearity_coeff**(*data*, *sat_vals*, *well_depth*, *sat_calc=0.98*, *ref_info=[4, 4, 4, 4]*, *counts_cut=None*, *deg=8*, *use_legendre=True*, *lxmap=[0, 100000.0]*, *nonlin=False*, ***kwargs*)

counts_cut [None or float] Option to fit two sets of polynomial coefficients to lower and uppper values. 'counts_cut' specifies the division in values of electrons. Useful for pixels with different non-linear behavior at low flux levels. Recommended values of 15000 e-.

pynrc.reduce.calib.calc_nonlin_coeff

pynrc.reduce.calib.**calc_nonlin_coeff**(*data*, *sat_vals*, *well_depth*, *sat_calc=0.98*, *ref_info=[4, 4, 4, 4]*, *counts_cut=None*, *deg=8*, *use_legendre=True*, *lxmap=[0, 100000.0]*, ***kwargs*)

counts_cut [None or float] Option to fit two sets of polynomial coefficients to lower and uppper values. 'counts_cut' specifies the division in values of electrons. Useful for pixels with different non-linear behavior at low flux levels. Recommended values of 15000 e-.

pynrc.reduce.calib.chisqr_red

pynrc.reduce.calib.**chisqr_red**(*yvals*, *yfit=None*, *err=None*, *dof=None*, *err_func=<function std>*)
Calculate reduced chi square metric

If yfit is None, then yvals assumed to be residuals. In this case, *err* should be specified.

Parameters

- **yvals** (*ndarray*) – Sampled values.
- **yfit** (*ndarray*) – Model fit corresponding to *yvals*.
- **dof** (*int*) – Number of degrees of freedom (nvals - nparams - 1).
- **err** (*ndarray or float*) – Uncertainties associated with *yvals*. If not specified, then use yvals point-to-point differences to estimate a single value for the uncertainty.
- **err_func** (*func*) – Error function uses to estimate *err*.

pynrc.reduce.calib.cube_fit

pynrc.reduce.calib.**cube_fit**(*tarr*, *data*, *bias=None*, *sat_vals=None*, *sat_frac=0.95*, *deg=1*, *fit_zero=False*, *verbose=False*, *ref_info=[4, 4, 4, 4]*, *use_legendre=False*, *lxmap=None*, *return_lxmap=False*, *return_chired=False*)

pynrc.reduce.calib.deconv_single_image

pynrc.reduce.calib.**deconv_single_image**(*im*, *kfft*)
Image deconvolution for a kernel

pynrc.reduce.calib.find_group_sat

pynrc.reduce.calib.**find_group_sat**(*file*, *DMS=False*, *bias=None*, *sat_vals=None*, *sat_calc=0.998*)
Group at which 98% of pixels are saturated

pynrc.reduce.calib.find_sat

pynrc.reduce.calib.**find_sat**(*data*, *bias=None*, *ref_info=[4, 4, 4, 4]*, *bit_depth=16*)
Given a data cube, find the values in ADU in which data reaches hard saturation.

pynrc.reduce.calib.fit_corr_powspec

pynrc.reduce.calib.**fit_corr_powspec**(*freq*, *ps*, *flim1=[0, 1]*, *flim2=[10, 100]*, *alpha=- 1*, ***kwargs*)
Fit Correlated Noise Power Spectrum

Fit the scaling factors of the 1/f power law components observed in the correlated noise power spectra. This function separately calculates the high-freq and low- freq scale factor components defined by the fcut params. The mid-frequency ranges are interpolated in log space.

Parameters

- **freq** (*ndarray*) – Input frequencies corresponding to power spectrum.
- **ps** (*ndarray*) – Input power spectrum to fit.
- **flim1** (*float*) – Fit frequencies within this range to get scaling for low frequency 1/f noise.
- **flim2** (*float*) – Fit frequencies within this range to get scaling for high frequency 1/f noise.
- **alpha** (*float*) – Noise power spectrum scaling

pynrc.reduce.calib.fit_func_var_ex

pynrc.reduce.calib.**fit_func_var_ex**(*params*, *det*, *patterns*, *ng_all_list*, *en_dn_list*, *read_noise=None*, *idark=None*, *ideal_Poisson=False*)
Function for lsq fit to get excess variance

pynrc.reduce.calib.gen_cds_dict

pynrc.reduce.calib.**gen_cds_dict**(*allfiles*, *DMS=False*, *superbias=None*, *mask_good_arr=None*, *same_scan_direction=False*)
Generate dictionary of CDS noise info

Calculate read noise for:

1. Total noise (no column correcton)
2. 1/f noise (no column correcton)

3. Intrinsic read noise (w/ column correcton)
4. Both temporal and spatial

pynrc.reduce.calib.gen_col_variations

pynrc.reduce.calib.**gen_col_variations**(*allfiles*, *super_bias=None*, *super_dark_ramp=None*, *DMS=False*, ***kwargs*)

Create a series of column offset models

Returns a series of ramp variations to add to entire columns as well as the probability a given column will be affected.

Likely due to FETS in the ASIC preamp or ADC or detector column buffer jumping around and causing entire columns within a ramp to transition between two states.

pynrc.reduce.calib.gen_ref_dict

pynrc.reduce.calib.**gen_ref_dict**(*allfiles*, *super_bias*, *super_dark_ramp=None*, *DMS=False*, ***kwargs*)

Generate Reference Pixel Behavior Dictionary

pynrc.reduce.calib.gen_super_bias

pynrc.reduce.calib.**gen_super_bias**(*allfiles*, *DMS=False*, *mn_func=<function median>*, *std_func=<function std>*, *return_std=False*, *deg=1*, *nsplit=3*, ***kwargs*)

Generate a Super Bias Image

Read in a number of dark ramps, perform a polynomial fit to the data, and return the average of all bias offsets. This a very simple procedure that is useful for estimating an initial bias image. Will not work well for weird pixels.

pynrc.reduce.calib.gen_super_dark

pynrc.reduce.calib.**gen_super_dark**(*allfiles*, *super_bias=None*, *DMS=False*, ***kwargs*)

Average together all dark ramps to create a super dark ramp. First subtracts a bias frame. Tries to decipher t=0 intercept for odd behaving pixels.

pynrc.reduce.calib.gen_super_ramp

pynrc.reduce.calib.**gen_super_ramp**(*allfiles*, *super_bias=None*, *DMS=False*, *grp_max=None*, *sat_vals=None*, ***kwargs*)

Average together all linearity ramps to create a super ramp. Subtracts a bias frame to determine more appropriate pixel by pixel average. Tries to decipher t=0 intercept for odd behaving pixels. Also returns bias offsets.

pynrc.reduce.calib.get_bias_offsets

pynrc.reduce.calib.**get_bias_offsets**(*data*, *nchan=4*, *ref_bot=True*, *ref_top=True*, *npix_ref=4*)
Get Reference Bias Characteristics

Given some ramp data, determine the average master bias offset as well as the relative individual amplifier offsets. Also return the frame-to-frame variations caused by the preamp resets.

pynrc.reduce.calib.get_fits_data

pynrc.reduce.calib.**get_fits_data**(*fits_file*, *return_header=False*, *bias=None*, *reffix=False*, *DMS=False*, *int_ind=0*, *grp_ind=None*, ***kwargs*)
Read in FITS file data

Parameters

- **fname** (*str*) – FITS file (including path) to open.
- **return_header** (*bool*) – Return header as well as data?
- **bias** (*ndarray*) – If specified, will subtract bias image from ramp.
- **reffix** (*bool*) – Perform reference correction?
- **DMS** (*bool*) – Is the FITS file DMS format?
- **int_ind** (*int*) – If DMS format, select integration index to extract. DMS FITS files usually have all integrations within a given exposure in a single FITS extension, which can be quite large.
- **grp_ind** (*2-element array*) – Option to index specific groups from the data. For instance *grp_ind=[0:10]* will select only the first 10 groups from the FITS cube.

Keyword Arguments

- **altcol** (*bool*) – Calculate separate reference values for even/odd columns. (default: True)
- **supermean** (*bool*) – Add back the overall mean of the reference pixels. (default: False)
- **top_ref** (*bool*) – Include top reference rows when correcting channel offsets. (default: True)
- **bot_ref** (*bool*) – Include bottom reference rows when correcting channel offsets. (default: True)
- **ntop** (*int*) – Specify the number of top reference rows. (default: 4)
- **nbot** (*int*) – Specify the number of bottom reference rows. (default: 4)
- **mean_func** (*func*) – Function used to calculate averages. (default: *robust.mean*)
- **left_ref** (*bool*) – Include left reference cols when correcting 1/f noise. (default: True)
- **right_ref** (*bool*) – Include right reference cols when correcting 1/f noise. (default: True)
- **nleft** (*int*) – Specify the number of left reference columns. (default: 4)
- **nright** (*int*) – Specify the number of right reference columns. (default: 4)
- **perint** (*bool*) – Smooth side reference pixel per integration, otherwise do frame-by-frame. (default: False)
- **avg_type** (*str*) – Type of side column averaging to perform to determine ref pixel drift. Allowed values are 'pixel', 'frame', or 'int' (default: 'frame'):

  - '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*) – Use Savitsky-Golay filter method rather than FFT. (default: True)
- **winsize** (*int*) – Size of the window filter. (default: 31)
- **order** (*int*) – Order of the polynomial used to fit the samples. (default: 3)

pynrc.reduce.calib.get_flat_fields

pynrc.reduce.calib.**get_flat_fields**(*im_slope*, *split_low_high=True*, *smth_sig=10*, *ref_info=[4, 4, 4, 4]*)
Calculate QE variations in flat field

pynrc.reduce.calib.get_freq_array

pynrc.reduce.calib.**get_freq_array**(*pow_spec*, *dt=1*, *nozero=False*, *npix_odd=False*)
Return frequencies associated with power spectrum

Parameters

- **pow_spec** (*ndarray*) – Power spectrum to obtain associated frequency array.
- **dt** (*float*) – Delta time between corresponding elements in time domain.
- **nozero** (*bool*) – Set freq[0] = freq[1] to remove zeros? This is mainly so we don't obtain NaN's later when calculating 1/f noise.
- **npix_odd** (*bool*) – We normally assume that the original time-domain data was comprised of an even number of pixels. However, if it were actually odd, the frequency array will be slightly shifted. Set this to True if the intrinsic data that was used to generate the pow_spec had an odd number of elements.

pynrc.reduce.calib.get_ipc_kernel

pynrc.reduce.calib.**get_ipc_kernel**(*imdark*, *tint=None*, *boxsize=5*, *nchans=4*, *bg_remove=True*, *hotcut=[5000, 50000]*, *calc_ppc=False*, *same_scan_direction=False*, *reverse_scan_direction=False*, *ref_info=[4, 4, 4, 4]*, *suppress_error_msg=False*)
Derive IPC/PPC Convolution Kernels

Find the IPC and PPC kernels used to convolve detector pixel data. Finds all hot pixels within hotcut parameters and measures the average relative flux within adjacent pixels.

Parameters

- **imdark** (*ndarray*) – Image to search for hot pixels in units of DN or DN/sec. If in terms of DN/sec, make sure to set *tint* to convert to raw DN.
- **Keyword Parameters**
- **==========**
- **tint** (*float or None*) – Integration time to convert dark current rate into raw pixel values (DN). If None, then input image is assumed to be in units of DN.

- **boxsize** (*int*) – Size of the box. Should be odd. If even, will increment by 1.
- **nchans** (*int*) – Number of amplifier channels; necessary for PPC measurements.
- **bg_remove** (*bool*) – Remove the average dark current values for each hot pixel cut-out. Only works if boxsize>3.
- **hotcut** (*array-like*) – Min and max values of hot pixels (above bg and bias) to consider.
- **calc_ppc** (*bool*) – Calculate and return post-pixel coupling?
- **same_scan_direction** (*bool*) – Are all the output channels read in the same direction? By default fast-scan readout direction is `[-->,<--,-->,<--]` If `same_scan_direction`, then all `-->`
- **reverse_scan_direction** (*bool*) – If `reverse_scan_direction`, then `[<--,-->,<--,-->]` or all `<--`

pynrc.reduce.calib.get_linear_coeffs

pynrc.reduce.calib.**get_linear_coeffs**(*allfiles*, *super_bias=None*, *DMS=False*, *kppc=None*, *kipc=None*, *counts_cut=None*, *deg=8*, *use_legendre=True*, *lxmap=[0, 100000.0]*, *return_satvals=False*, *nonlin=False*, *sat_calc=0.98*, ***kwargs*)

pynrc.reduce.calib.get_oddeven_offsets

pynrc.reduce.calib.**get_oddeven_offsets**(*data*, *nchan=4*, *ref_bot=True*, *ref_top=True*, *bias_off=None*, *amp_off=None*)

Even/Odd Column Offsets

Return the per-amplifier offsets of the even and odd columns relative after subtraction of the matster and amplifier bias offsets.

pynrc.reduce.calib.get_power_spec

pynrc.reduce.calib.**get_power_spec**(*data*, *nchan=4*, *calc_cds=True*, *kw_powspec=None*, *per_pixel=False*, *return_corr=False*, *return_ucorr=False*, *mn_func=<function mean>*)

Calculate the power spectrum of an input data ramp in a variety of ways.

If return_corr and return_ucorr are both False, then will return (ps_all, None, None).

Parameters

- **calc_cds** (*bool*) – Power spectrum of CDS pairs or individual frames?
- **per_pixel** (*bool*) – Calculate average power spectrum of each pixel along ramp (frame timescales)? If False, samples pixels within a frame (pixel read timescales)
- **return_corr** (*bool*) – Return power spectrum of channel correlated 1/f noise?
- **return_ucorr** (*bool*) – Return power spectra of channel-dependent (uncorrelated) 1/f noise?
- **kw_powspec** (*dict*) – Keyword arguments to pass to *pow_spec_ramp* function.
- **mn_func** (*func*) – Function to use to perform averaging of individual power spectra.

pynrc.reduce.calib.get_power_spec_all

pynrc.reduce.calib.**get_power_spec_all**(*allfiles*, *super_bias=None*, *det=None*, *DMS=False*, *include_oh=False*, *same_scan_direction=False*, *reverse_scan_direction=False*, *calc_cds=True*, *return_corr=False*, *return_ucorr=False*, *per_pixel=False*, *mn_func=<function mean>*, *kw_reffix=None*)

Return the average power spectra (white, 1/f noise correlated and uncorrelated) of all FITS files.

Parameters

- **allfiles** (*array-like*) – List of FITS files to operate on.
- **super_bias** (*ndarray*) – Option to subtract a super bias image from all frames in a ramp. Provides slightly better statistical averaging for reference pixel correction routines.
- **det** (*Detector class*) – Option to pass known NIRCam detector class. This will get generated from a FITS header if not specified.
- **DMS** (*bool*) – Are the files DMS formatted or FITSWriter?
- **include_oh** (*bool*) – Zero-pad the data to insert line and frame overhead pixels?
- **same_scan_direction** (*bool*) – Are all the output channels read in the same direction? By default fast-scan readout direction is `[-->,<--,-->,<--]` If `same_scan_direction`, then all `-->`
- **reverse_scan_direction** (*bool*) – If `reverse_scan_direction`, then `[<--,-->,<--,-->]` or all `<--`
- **calc_cds** (*bool*) – Power spectrum of CDS pairs or individual frames?
- **per_pixel** (*bool*) – Calculate average power spectrum of each pixel along ramp (frame timescales)? If False, samples pixels within a frame (pixel read timescales).
- **return_corr** (*bool*) – Return power spectrum of channel correlated 1/f noise?
- **return_ucorr** (*bool*) – Return power spectra of channel-dependent (uncorrelated) 1/f noise?
- **kw_powspec** (*dict*) – Keyword arguments to pass to *pow_spec_ramp* function.
- **mn_func** (*func*) – Function to use to perform averaging of individual power spectra.

pynrc.reduce.calib.get_ref_instability

pynrc.reduce.calib.**get_ref_instability**(*data*, *nchan=4*, *ref_bot=True*, *ref_top=True*, *mn_func=<function median>*)

Reference Pixel Instability

Determine the instability of the average reference pixel values relative to the active pixels on a frame-to-frame basis. The procedure is to compute a series of CDS frames, then look at the peak distributions of the active pixels relative to the reference pixels.

pynrc.reduce.calib.ipc_deconvolve

pynrc.reduce.calib.**ipc_deconvolve**(*imarr*, *kernel*, *kfft=None*, ***kwargs*)

Simple IPC image deconvolution

Given an image (or image cube), apply an IPC deconvolution kernel to obtain the intrinsic flux distribution. Should also work for PPC kernels. This simply calculates the FFT of the image(s) and kernel, divides them, then applies an iFFT to determine the deconvolved image.

If performing PPC deconvolution, make sure to perform channel-by-channel with the kernel in the appropriate scan direction. IPC is usually symmetric, so this restriction may not apply. See *ppc_deconvolve* function. Calls *ppc_deconvolve* for asymmetric (left-right) IPC kernels.

Parameters

- **im** (*ndarray*) – Image or array of images.
- **kernel** (*ndarry*) – Deconvolution kernel. Ignored if *kfft* is specified.
- **kfft** (*Complex ndarray*) – Option to directy supply the kernel's FFT rather than calculating it within the function. The supplied ndarray should have shape (ny,nx) equal to the input *im*. Useful if calling `ipc_deconvolve` multiple times.
- **symmetric** (*bool*) – Is the input IPC kernel symmetric?

Keyword Arguments

- **in_place** (*bool*) – Perform calculate in place (overwrites original image).
- **nchans** (*int*) – Number of amplifier channels.
- **same_scan_direction** (*bool*) – Are all the output channels read in the same direction? By default fast-scan readout direction is `[-->,<--,-->,<--]` If `same_scan_direction`, then all `-->`
- **reverse_scan_direction** (*bool*) – If `reverse_scan_direction`, then `[<--,-->, <--,-->]` or all `<--`

pynrc.reduce.calib.pixel_linearity_gains

pynrc.reduce.calib.**pixel_linearity_gains**(*frame*, *coeff_arr*, *use_legendre=True*, *lxmap=[0, 100000.0]*)

Given some image data and coefficient

pynrc.reduce.calib.plot_dark_histogram

pynrc.reduce.calib.**plot_dark_histogram**(*im*, *ax*, *binsize=0.0001*, *return_ax=False*, *label='Active Pixels'*, *plot_fit=True*, *plot_cumsum=True*, *color='C1'*, *xlim=None*, *xlim_std=7*)

pynrc.reduce.calib.plot_kernel

pynrc.reduce.calib.**plot_kernel**(*kern*, *ax=None*, *return_figax=False*)
Plot image of IPC or PPC kernel

Parameters

- **kern** (*ndarray*) – Kernel image (3x3 or 5x5, etc) to plot.
- **ax** (*axes*) – Axes to plot kernel on. If None, will create new figure and axes subplot.
- **return_figax** (*bool*) – Return the (figure, axes) for user manipulations?

pynrc.reduce.calib.pow_spec_ramp

pynrc.reduce.calib.**pow_spec_ramp**(*data*, *nchan*, *nroh=0*, *nfoh=0*, *nframes=1*, *expand_npix=False*, *same_scan_direction=False*, *reverse_scan_direction=False*, *mn_func=<function mean>*, *return_freq=False*, *dt=1*, ***kwargs*)
Get power spectrum within frames of input ramp

Takes an input cube, splits it into output channels, and finds the power spectrum of each frame. Then, calculate the average power spectrum for each channel.

Use *nroh* and *nfoh* to expand the frame size to encapsulate the row and frame overheads not included in the science data. These just zero-pad the array.

Parameters

- **data** (*ndarray*) – Input Image cube.
- **nchan** (*int*) – Number of amplifier channels.
- **nroh** (*int*) – Number of pixel overheads per row.
- **nfoh** (*int*) – Number of row overheads per frame.
- **nframes** (*int*) – Number of frames to use to calculate an power spectrum. Normally we just use 1 frame time
- **expand_npix** (*bool*) – Should we zero-pad the array to a power of two factor for incresed speed?
- **same_scan_direction** (*bool*) – Are all the output channels read in the same direction? By default fast-scan readout direction is `[-->,<--,-->,<--]` If `same_scan_direction`, then all `-->`
- **reverse_scan_direction** (*bool*) – If `reverse_scan_direction`, then `[<--,-->,<--,-->]` or all `<--`

pynrc.reduce.calib.pow_spec_ramp_pix

pynrc.reduce.calib.**pow_spec_ramp_pix**(*data*, *nchan*, *expand_nstep=False*, *mn_func=<function mean>*, *return_freq=False*, *dt=1*, ***kwargs*)
Get power spectrum of pixels within ramp

Takes an input cube, splits it into output channels, and finds the power spectrum of each pixel. Return the average power spectrum for each channel.

Parameters

- **data** (*ndarray*) – Input Image cube.

- **nchan** (*int*) – Number of amplifier channels.
- **expand_nstep** (*bool*) – Should we zero-pad the array to a power of two factor for incresed speed?

pynrc.reduce.calib.ppc_deconvolve

pynrc.reduce.calib.**ppc_deconvolve**(*im*, *kernel*, *kfft=None*, *nchans=4*, *in_place=False*, *same_scan_direction=False*, *reverse_scan_direction=False*, ***kwargs*)

PPC image deconvolution

Given an image (or image cube), apply PPC deconvolution kernel to obtain the intrinsic flux distribution. This performs channel-by-channel deconvolution, taking into account the specific readout directly. This function can also be used for asymmetric IPC kernels.

Parameters

- **im** (*ndarray*) – Image or array of images. Assumes detector coordinates where (0,0) is in bottom left.
- **kernel** (*ndarry*) – Deconvolution kernel. Ignored if *kfft* is specified.
- **kfft** (*Complex ndarray*) – Option to directy supply the kernel's FFT rather than calculating it within the function. The supplied ndarray should have shape (ny,nx) equal to the input *im*. Useful if calling `ppc_deconvolve` multiple times.
- **in_place** (*bool*) – Perform calculate in place (overwrites original image).
- **nchans** (*int*) – Number of amplifier channels.
- **same_scan_direction** (*bool*) – Are all the output channels read in the same direction? By default fast-scan readout direction is `[-->,<--,-->,<--]` If `same_scan_direction`, then all `-->`
- **reverse_scan_direction** (*bool*) – If `reverse_scan_direction`, then `[<--,-->,<--,-->]` or all `<--`

pynrc.reduce.calib.ramp_derivative

pynrc.reduce.calib.**ramp_derivative**(*y*, *dx=None*, *fit0=True*, *deg=2*, *ifit=[0, 10]*)

Get the frame-by-frame derivative of a ramp.

Parameters

- **y** (*ndarray*) – Array of values (1D, 2D or 3D)
- **dx** (*float*) – If dx is supplied, divide by value to get dy/dx.
- **fit0** (*bool*) – In order to find slope of element 0, we have the option to fit some number of values to extrapolate this value. If not set, then dy0 = 2*dy[0] - dy[1].
- **ifit** (*2-element array*) – Indices to fit in order to extrapolate dy0. Don't necessarily want to fit the entire dataset.
- **deg** (*int*) – Polynomial degree to use for extrapolation fit.

pynrc.reduce.calib.ramp_resample

pynrc.reduce.calib.**ramp_resample**(*data*, *det_new*, *return_zero_frame=False*)
Resample a RAPID dataset into new detector format

pynrc.reduce.calib.time_to_sat

pynrc.reduce.calib.**time_to_sat**(*data*, *sat_vals*, *dt=1*, *sat_calc=0.998*, *ref_info=[4, 4, 4, 4]*)
Determine time of saturation

Classes

nircam_cal(scaid[, same_scan_direction, ...])	NIRCam Calibration class
nircam_dark(scaid, datadir, outdir[, ...])	

pynrc.reduce.calib.nircam_cal

class pynrc.reduce.calib.**nircam_cal**(*scaid*, *same_scan_direction=False*, *reverse_scan_direction=False*, *DMS=False*, *verbose=True*)
Bases: *pynrc.reduce.calib.nircam_dark*

NIRCam Calibration class

Assumes that all cal files exist in the calibration directory in PYNRC_PATH.

__init__(*scaid*, *same_scan_direction=False*, *reverse_scan_direction=False*, *DMS=False*, *verbose=True*)

Methods

__init__(scaid[, same_scan_direction, ...])	

calc_cds_noise([cds_type, temperature, temp_key])	Return CDS Noise components for each channel
deconvolve_supers()	Deconvolve the super dark and super bias images
get_cds_dict([force])	Calculate CDS noise for all files
get_column_variations([force])	Get column offset variations
get_dark_slope_image([deg, force])	Calculate dark slope image
get_effective_noise([ideal_Poisson, force])	Calculate effective noise curves for each readout pattern
get_ipc([calc_ppc])	Calculate IPC (and PPC) kernels
get_ktc_noise(**kwargs)	Calculate and store kTC (Reset) Noise
get_linear_coeffs([deg, use_legendre, ...])	Determine linearity coefficents
get_nonlinear_coeffs([deg, use_legendre, ...])	Determine non-linear coefficents
get_pixel_slope_averages([force])	Get average pixel ramp

continues on next page

Table 26 – continued from previous page

get_power_spectrum([include_oh, calc_cds, ...])	**keyword include_oh** Zero-pad the data to insert line and frame overhead pixels?
get_ref_pixel_noise([force])	Generate Dictionary of Reference Pixel behavior info
get_super_bias_init([deg, nsplit, force])	
get_super_bias_update([force])	
get_super_dark_ramp([force])	Create or read super dark ramp and update super bias
get_super_flats([split_low_high, smth_sig, ...])	Get flat field information
plot_bias_darks([save, return_figax, deconvolve])	
plot_cds_noise([tkey, save, return_figax, xlim])	
plot_dark_distribution([save, xlim, ...])	Plot histogram of dark slope
plot_dark_overview([save, xlim_hist, ...])	Plot Overview of Dark Current Characteristics
plot_dark_ramps([save, time_cut, return_figax])	Plot average dark current ramps
plot_dark_ramps_ch([save, time_cut, ...])	Plot fits to each channel dark current ramp
plot_eff_noise([ideal_Poisson, save, ...])	Plot effective noise of slope fits
plot_eff_noise_patterns([ideal_Poisson, ...])	Plot effective noise of slope fits for variety of read patterns
plot_ipc_ppc([k_ipc, k_ppc, save, return_figax])	
plot_power_spectrum([save, cds, return_figax])	
plot_reset_overview([save, binsize, ...])	Overview Plots of Bias and kTC Noise

Attributes

allfiles	
cds_act_dict	
cds_ref_dict	
chsize	Width of output channel
column_prob_bad	
column_variations	
dark_ramp_dict	
dark_shape	Shape of dark ramps
datadir	
eff_noise_dict	

continues on next page

Table 27 – continued from previous page

ipc_alpha_frac	Fractional IPC value (alpha)
kernel_ipc	
kernel_ppc	
ktc_noise	
lindir	
linfiles	
mask_act	
mask_channels	
mask_ref	
nchan	Number of output channels
nchans	Number of output channels
outdir	
pow_spec_dict	
ppc_frac	Fractional PPC value
ref_pixel_dict	
super_bias	
super_bias_deconv	
super_dark	
super_dark_deconv	
super_dark_ramp	
temperature_dict	
time_arr	

calc_cds_noise(*cds_type='spatial'*, *temperature=None*, *temp_key='T_FPA1'*)
Return CDS Noise components for each channel

Parameters

- **cds_type** (*str*) – Return 'spatial', 'temporal', or 'average' noise values?
- **temperature** (*float or None*) – Option to supply temperature at which to interpolate. If None is provided, then returns the median of all noise values.
- **temp_key** (*str*) – Temperature key from *self.temperature_dict* to interpolate over. Generally, either 'T_FPA1' or 'T_FPA2' as those most closely represent the detector operating temperatures.

property chsize
Width of output channel

property dark_shape
Shape of dark ramps

deconvolve_supers()
Deconvolve the super dark and super bias images

get_cds_dict(*force=False*)
Calculate CDS noise for all files

Creates a dictionary of CDS noise components, including total noise, amplifier 1/f noise, correlated 1/f noise, white noise, and reference pixel ratios. Two different methods are used to calculate CDS per pixels: temporal and spatial.

Creates dictionary attributes *self.cds_act_dict* and *self.cds_ref_dict*.

get_column_variations(*force=False*, ***kwargs*)
Get column offset variations

Create a series of column offset models. These are likely FETS in the ASIC preamp or ADC causing entire columns within a ramp to jump around.

get_dark_slope_image(*deg=1*, *force=False*)
Calculate dark slope image

get_effective_noise(*ideal_Poisson=False*, *force=False*)
Calculate effective noise curves for each readout pattern

get_ipc(*calc_ppc=False*)
Calculate IPC (and PPC) kernels

get_ktc_noise(***kwargs*)
Calculate and store kTC (Reset) Noise

Keyword Arguments

- **bias_sigma_arr** (*ndarray*) – Image of the pixel uncertainties.
- **binsize** (*float*) – Size of the histogram bins.
- **return_std** (*bool*) – Also return the standard deviation of the distribution?

get_linear_coeffs(*deg=8*, *use_legendre=True*, *lxmap=[0, 100000.0]*, *counts_cut=None*, *sat_calc=0.98*, *force=False*, *DMS=None*, *super_bias=None*, ***kwargs*)
Determine linearity coefficents

These coefficients allow us to convert from an observed ramp (DN) to an idealized linear ramp (in e-). Values are stored in the dictionary self.linear_dict.

Parameters

- **force** (*bool*) – Force calculation of coefficients.
- **DMS** (*None or bool*) – Option to specifiy if linearity files are DMS format. If set to None, then uses self.DMS.
- **super_bias** (*None or ndarray*) – Option to specify an input super bias image. If not specified, then defaults to self.super_bias.
- **counts_cut** (*None or float*) – Option to fit two sets of polynomial coefficients to lower and uppper values. 'counts_cut' specifies the division in values of electrons. Useful for pixels with different non-linear behavior at low flux levels. Recommended values of 15000 e-.

- **deg** (*int*) – Degree of polynomial to fit. Default=8.
- **use_legendre** (*bool*) – Fit with Legendre polynomial, an orthonormal basis set. Default=True.
- **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()].

get_nonlinear_coeffs(*deg=8*, *use_legendre=True*, *lxmap=[0, 100000.0]*, *counts_cut=15000*, *sat_calc=0.998*, *force=False*, *DMS=None*, *super_bias=None*, ***kwargs*)

Determine non-linear coefficents

These coefficients allow us to go from an ideal linear ramp to some observed (simulated) non-linear ramp. Value are store in the self.nonlinear_dict dictionary.

Parameters

- **force** (*bool*) – Force calculation of coefficients.
- **DMS** (*None or bool*) – Option to specifiy if linearity files are DMS format. If set to None, then uses self.DMS.
- **super_bias** (*None or ndarray*) – Option to specify an input super bias image. If not specified, then defaults to self.super_bias.
- **counts_cut** (*None or float*) – Option to fit two sets of polynomial coefficients to lower and uppper values. 'counts_cut' specifies the division in values of electrons. Useful for pixels with different non-linear behavior at low flux levels. Recommended values of 15000 e-.
- **deg** (*int*) – Degree of polynomial to fit. Default=8.
- **use_legendre** (*bool*) – Fit with Legendre polynomial, an orthonormal basis set. Default=True.
- **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()].

get_pixel_slope_averages(*force=False*)

Get average pixel ramp

get_power_spectrum(*include_oh=False*, *calc_cds=True*, *per_pixel=False*, *return_corr=False*, *return_ucorr=False*, *mn_func=<function mean>*, *force=False*, *save=True*)

Keyword Arguments

- **include_oh** (*bool*) – Zero-pad the data to insert line and frame overhead pixels?
- **calc_cds** (*bool*) – Power spectrum of CDS pairs or individual frames?
- **return_corr** (*bool*) – Return power spectrum of channel correlated 1/f noise?
- **return_ucorr** (*bool*) – Return power spectra of channel-dependent (uncorrelated) 1/f noise?
- **per_pixel** (*bool*) – Calculate average power spectrum of each pixel along ramp (frame timescales)? If False, samples pixels within a frame (pixel read timescales).

get_ref_pixel_noise(*force=False*, ***kwargs*)

Generate Dictionary of Reference Pixel behavior info

get_super_dark_ramp(*force=False*, ***kwargs*)

Create or read super dark ramp and update super bias

get_super_flats(*split_low_high=True*, *smth_sig=10*, *force=False*, ***kwargs*)
Get flat field information

Splits flat field into to lflats and pflats (low and high frequency).

property ipc_alpha_frac
Fractional IPC value (alpha)

property nchan
Number of output channels

property nchans
Number of output channels

plot_bias_darks(*save=False*, *return_figax=False*, *deconvolve=False*)

plot_dark_distribution(*save=False*, *xlim=None*, *return_figax=False*)
Plot histogram of dark slope

plot_dark_overview(*save=False*, *xlim_hist=None*, *return_figax=False*)
Plot Overview of Dark Current Characteristics

plot_dark_ramps(*save=True*, *time_cut=None*, *return_figax=False*)
Plot average dark current ramps

time_cut [float] Some darks show distinct slopes before and after a characteristic time. Setting this keyword will fit separate slopes before and after the specified time. A time of 200 sec is used for SCA 485.

plot_dark_ramps_ch(*save=True*, *time_cut=None*, *return_figax=False*)
Plot fits to each channel dark current ramp

time_cut [float] Some darks show distinct slopes before and after a characteristic time. Setting this keyword will fit separate slopes before and after the specified time. A time of 200 sec is used for SCA 485.

plot_eff_noise(*ideal_Poisson=False*, *save=False*, *return_figax=False*)
Plot effective noise of slope fits

plot_eff_noise_patterns(*ideal_Poisson=False*, *save=False*, *ylim=None*, *return_figax=False*)
Plot effective noise of slope fits for variety of read patterns

plot_reset_overview(*save=False*, *binsize=0.25*, *xlim_hist=None*, *return_figax=False*)
Overview Plots of Bias and kTC Noise

property ppc_frac
Fractional PPC value

pynrc.reduce.calib.nircam_dark

class pynrc.reduce.calib.**nircam_dark**(*scaid*, *datadir*, *outdir*, *lindir=None*, *DMS=False*, *same_scan_direction=False*, *reverse_scan_direction=False*)
Bases: object

__init__(*scaid*, *datadir*, *outdir*, *lindir=None*, *DMS=False*, *same_scan_direction=False*, *reverse_scan_direction=False*)

Methods

__init__(scaid, datadir, outdir[, lindir, ...])	
calc_cds_noise([cds_type, temperature, temp_key])	Return CDS Noise components for each channel
deconvolve_supers()	Deconvolve the super dark and super bias images
get_cds_dict([force])	Calculate CDS noise for all files
get_column_variations([force])	Get column offset variations
get_dark_slope_image([deg, force])	Calculate dark slope image
get_effective_noise([ideal_Poisson, force])	Calculate effective noise curves for each readout pattern
get_ipc([calc_ppc])	Calculate IPC (and PPC) kernels
get_ktc_noise(**kwargs)	Calculate and store kTC (Reset) Noise
get_linear_coeffs([deg, use_legendre, ...])	Determine linearity coefficents
get_nonlinear_coeffs([deg, use_legendre, ...])	Determine non-linear coefficents
get_pixel_slope_averages([force])	Get average pixel ramp
get_power_spectrum([include_oh, calc_cds, ...])	**keyword include_oh** Zero-pad the data to insert line and frame overhead pixels?
get_ref_pixel_noise([force])	Generate Dictionary of Reference Pixel behavior info
get_super_bias_init([deg, nsplit, force])	
get_super_bias_update([force])	
get_super_dark_ramp([force])	Create or read super dark ramp and update super bias
get_super_flats([split_low_high, smth_sig, ...])	Get flat field information
plot_bias_darks([save, return_figax, deconvolve])	
plot_cds_noise([tkey, save, return_figax, xlim])	
plot_dark_distribution([save, xlim, ...])	Plot histogram of dark slope
plot_dark_overview([save, xlim_hist, ...])	Plot Overview of Dark Current Characteristics
plot_dark_ramps([save, time_cut, return_figax])	Plot average dark current ramps
plot_dark_ramps_ch([save, time_cut, ...])	Plot fits to each channel dark current ramp
plot_eff_noise([ideal_Poisson, save, ...])	Plot effective noise of slope fits
plot_eff_noise_patterns([ideal_Poisson, ...])	Plot effective noise of slope fits for variety of read patterns
plot_ipc_ppc([k_ipc, k_ppc, save, return_figax])	
plot_power_spectrum([save, cds, return_figax])	
plot_reset_overview([save, binsize, ...])	Overview Plots of Bias and kTC Noise

Attributes

allfiles	
cds_act_dict	
cds_ref_dict	
chsize	Width of output channel
column_prob_bad	
column_variations	
dark_ramp_dict	
dark_shape	Shape of dark ramps
datadir	
eff_noise_dict	
ipc_alpha_frac	Fractional IPC value (alpha)
kernel_ipc	
kernel_ppc	
ktc_noise	
lindir	
linfiles	
mask_act	
mask_channels	
mask_ref	
nchan	Number of output channels
nchans	Number of output channels
outdir	
pow_spec_dict	
ppc_frac	Fractional PPC value
ref_pixel_dict	
super_bias	
super_bias_deconv	

continues on next page

Table 29 – continued from previous page

super_dark
super_dark_deconv
super_dark_ramp
temperature_dict
time_arr

calc_cds_noise(*cds_type='spatial'*, *temperature=None*, *temp_key='T_FPA1'*)
Return CDS Noise components for each channel

Parameters

- **cds_type** (*str*) – Return 'spatial', 'temporal', or 'average' noise values?
- **temperature** (*float or None*) – Option to supply temperature at which to interpolate. If None is provided, then returns the median of all noise values.
- **temp_key** (*str*) – Temperature key from *self.temperature_dict* to interpolate over. Generally, either 'T_FPA1' or 'T_FPA2' as those most closely represent the detector operating temperatures.

property chsize
Width of output channel

property dark_shape
Shape of dark ramps

deconvolve_supers()
Deconvolve the super dark and super bias images

get_cds_dict(*force=False*)
Calculate CDS noise for all files

Creates a dictionary of CDS noise components, including total noise, amplifier 1/f noise, correlated 1/f noise, white noise, and reference pixel ratios. Two different methods are used to calculate CDS per pixels: temporal and spatial.

Creates dictionary attributes *self.cds_act_dict* and *self.cds_ref_dict*.

get_column_variations(*force=False*, ***kwargs*)
Get column offset variations

Create a series of column offset models. These are likely FETS in the ASIC preamp or ADC causing entire columns within a ramp to jump around.

get_dark_slope_image(*deg=1*, *force=False*)
Calculate dark slope image

get_effective_noise(*ideal_Poisson=False*, *force=False*)
Calculate effective noise curves for each readout pattern

get_ipc(*calc_ppc=False*)
Calculate IPC (and PPC) kernels

get_ktc_noise(***kwargs*)
Calculate and store kTC (Reset) Noise

Keyword Arguments

- **bias_sigma_arr** (*ndarray*) – Image of the pixel uncertainties.
- **binsize** (*float*) – Size of the histogram bins.
- **return_std** (*bool*) – Also return the standard deviation of the distribution?

get_linear_coeffs(*deg=8*, *use_legendre=True*, *lxmap=[0, 100000.0]*, *counts_cut=None*, *sat_calc=0.98*, *force=False*, *DMS=None*, *super_bias=None*, ***kwargs*)

Determine linearity coefficents

These coefficients allow us to convert from an observed ramp (DN) to an idealized linear ramp (in e-). Values are stored in the dictionary self.linear_dict.

Parameters

- **force** (*bool*) – Force calculation of coefficients.
- **DMS** (*None or bool*) – Option to specifiy if linearity files are DMS format. If set to None, then uses self.DMS.
- **super_bias** (*None or ndarray*) – Option to specify an input super bias image. If not specified, then defaults to self.super_bias.
- **counts_cut** (*None or float*) – Option to fit two sets of polynomial coefficients to lower and uppper values. 'counts_cut' specifies the division in values of electrons. Useful for pixels with different non-linear behavior at low flux levels. Recommended values of 15000 e-.
- **deg** (*int*) – Degree of polynomial to fit. Default=8.
- **use_legendre** (*bool*) – Fit with Legendre polynomial, an orthonormal basis set. Default=True.
- **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()].

get_nonlinear_coeffs(*deg=8*, *use_legendre=True*, *lxmap=[0, 100000.0]*, *counts_cut=15000*, *sat_calc=0.998*, *force=False*, *DMS=None*, *super_bias=None*, ***kwargs*)

Determine non-linear coefficents

These coefficients allow us to go from an ideal linear ramp to some observed (simulated) non-linear ramp. Value are store in the self.nonlinear_dict dictionary.

Parameters

- **force** (*bool*) – Force calculation of coefficients.
- **DMS** (*None or bool*) – Option to specifiy if linearity files are DMS format. If set to None, then uses self.DMS.
- **super_bias** (*None or ndarray*) – Option to specify an input super bias image. If not specified, then defaults to self.super_bias.
- **counts_cut** (*None or float*) – Option to fit two sets of polynomial coefficients to lower and uppper values. 'counts_cut' specifies the division in values of electrons. Useful for pixels with different non-linear behavior at low flux levels. Recommended values of 15000 e-.
- **deg** (*int*) – Degree of polynomial to fit. Default=8.
- **use_legendre** (*bool*) – Fit with Legendre polynomial, an orthonormal basis set. Default=True.

- **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()].

get_pixel_slope_averages(*force=False*)
Get average pixel ramp

get_power_spectrum(*include_oh=False*, *calc_cds=True*, *per_pixel=False*, *return_corr=False*, *return_ucorr=False*, *mn_func=<function mean>*, *force=False*, *save=True*)

Keyword Arguments

- **include_oh** (*bool*) – Zero-pad the data to insert line and frame overhead pixels?
- **calc_cds** (*bool*) – Power spectrum of CDS pairs or individual frames?
- **return_corr** (*bool*) – Return power spectrum of channel correlated 1/f noise?
- **return_ucorr** (*bool*) – Return power spectra of channel-dependent (uncorrelated) 1/f noise?
- **per_pixel** (*bool*) – Calculate average power spectrum of each pixel along ramp (frame timescales)? If False, samples pixels within a frame (pixel read timescales).

get_ref_pixel_noise(*force=False*, ***kwargs*)
Generate Dictionary of Reference Pixel behavior info

get_super_dark_ramp(*force=False*, ***kwargs*)
Create or read super dark ramp and update super bias

get_super_flats(*split_low_high=True*, *smth_sig=10*, *force=False*, ***kwargs*)
Get flat field information

Splits flat field into to lflats and pflats (low and high frequency).

property ipc_alpha_frac
Fractional IPC value (alpha)

property nchan
Number of output channels

property nchans
Number of output channels

plot_bias_darks(*save=False*, *return_figax=False*, *deconvolve=False*)

plot_dark_distribution(*save=False*, *xlim=None*, *return_figax=False*)
Plot histogram of dark slope

plot_dark_overview(*save=False*, *xlim_hist=None*, *return_figax=False*)
Plot Overview of Dark Current Characteristics

plot_dark_ramps(*save=True*, *time_cut=None*, *return_figax=False*)
Plot average dark current ramps

time_cut [float] Some darks show distinct slopes before and after a characteristic time. Setting this keyword will fit separate slopes before and after the specified time. A time of 200 sec is used for SCA 485.

plot_dark_ramps_ch(*save=True*, *time_cut=None*, *return_figax=False*)
Plot fits to each channel dark current ramp

time_cut [float] Some darks show distinct slopes before and after a characteristic time. Setting this keyword will fit separate slopes before and after the specified time. A time of 200 sec is used for SCA 485.

plot_eff_noise(*ideal_Poisson=False*, *save=False*, *return_figax=False*)
Plot effective noise of slope fits

plot_eff_noise_patterns(*ideal_Poisson=False*, *save=False*, *ylim=None*, *return_figax=False*)
Plot effective noise of slope fits for variety of read patterns

plot_reset_overview(*save=False*, *binsize=0.25*, *xlim_hist=None*, *return_figax=False*)
Overview Plots of Bias and kTC Noise

property ppc_frac
Fractional PPC value

pynrc.reduce.ref_pixels

Functions

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
channel_averaging(im[, nchans, ...])	Estimate common 1/f noise in image
channel_smooth_butter(im_arr[, order, freq, ...])	Channel smoothing using Butterworth filter
channel_smooth_fft(im_arr[, winsize])	Channel smoothing using smooth_fft
channel_smooth_savgol(im_arr[, winsize, ...])	Channel smoothing using savgol filter
chrem_med(imarr[, nchans, yind, bpmask, ...])	Subtract Amplifier Channel Offsets
mask_helper()	Helper to handle indices and logical indices of a mask
ref_filter(cube[, nchans, in_place, ...])	Optimal Smoothing
reffix_amps(cube[, nchans, in_place, ...])	Correct amplifier offsets
reffix_hxrg(cube[, nchans, in_place, fixcol])	Reference pixel correction function
smooth_fft(data, delt[, first_deriv, ...])	Optimal smoothing algorithm

pynrc.reduce.ref_pixels.calc_avg_amps

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

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

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

Generates 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.channel_averaging

pynrc.reduce.ref_pixels.**channel_averaging**(*im*, *nchans=4*, *same_scan_direction=False*, *off_chans=True*, *mn_func=<function nanmedian>*, ***kwargs*)

Estimate common 1/f noise in image

For a given image, average the channels together to find the common pattern noise present within the channels. Returns an array the same size as the input image.

Parameters **im** (*ndarray*) – Input image

Keyword Arguments

- **nchans** (*int*) – Number of output channels
- **same_scan_direction** (*bool*) – Are all the output channels read in the same direction? By default fast-scan readout direction is `[-->,<--,-->,<--]` If `same_scan_direction`, then all `-->`
- **off_chans** (*bool*) – Calculate indepenent values for each channel using the off channels.
- **mn_func** (*function*) – What function should we use to calculate the average. Default *np.nanmedian*

pynrc.reduce.ref_pixels.channel_smooth_butter

pynrc.reduce.ref_pixels.**channel_smooth_butter**(*im_arr*, *order=3*, *freq=0.1*, *per_line=False*, *mask=None*)

Channel smoothing using Butterworth filter

Parameters **im_arr** (*ndarray*) – Input array of images (intended to be a cube of output channels). Each image is operated on separately. If only two dimensions, then only a single input image is assumed.

Keyword Arguments

- **order** (*int*) – Order of the filter (high order have sharper frequency cut-off)
- **freq** (*float*) – Normalized frequency cut-off (between 0 and 1). 1 is Nyquist.
- **per_line** (*bool*) – Smooth each channel line separately with the hopes of avoiding edge discontinuities.
- **mask** (*bool image or None*) – An image mask of pixels to ignore. Should be same size as im_arr. This can be used to mask pixels that the filter should ignore, such as stellar sources or pixel outliers.

pynrc.reduce.ref_pixels.channel_smooth_fft

pynrc.reduce.ref_pixels.**channel_smooth_fft**(*im_arr*, *winsize=64*)

Channel smoothing using smooth_fft

Function for generating a map of the 1/f noise within a series of input images. The input images should show some clear noise structure for this to be useful. Uses M. Robberto smoothing algorithm.

One might prefer the *channel_smooth_savgol* or *channel_smooth_butter* functions due to their quickness.

Parameters

- **im_arr** (*ndarray*) – Input array of images

- **winsize** (*int*) – Window size chunks to break up

pynrc.reduce.ref_pixels.channel_smooth_savgol

pynrc.reduce.ref_pixels.**channel_smooth_savgol**(*im_arr*, *winsize=31*, *order=3*, *per_line=False*, *mask=None*, ***kwargs*)

Channel smoothing using savgol filter

Parameters **im_arr** (*ndarray*) – Input array of images (intended to be a cube of output channels). Shape should either be (ny, chsize) to smooth a single channel or (nchan, ny, chsize) for multiple channels. Each image is operated on separately. If only two dimensions, then only a single input image is assumed. NaN's will be interpolated over.

Keyword Arguments

- **winsize** (*int*) – Size of the window filter. Should be an odd number.
- **order** (*int*) – Order of the polynomial used to fit the samples.
- **per_line** (*bool*) – Smooth each channel line separately with the hopes of avoiding edge discontinuities.
- **mask** (*bool image or None*) – An image mask of pixels to ignore. Should be same size as im_arr. This can be used to mask pixels that the filter should ignore, such as stellar sources or pixel outliers. A value of True indicates that pixel should be ignored.
- **mode** (*str*) – Must be 'mirror', 'constant', 'nearest', 'wrap' or 'interp'. This determines the type of extension to use for the padded signal to which the filter is applied. When *mode* is 'constant', the padding value is given by *cval*. When the 'interp' mode is selected (the default), no extension is used. Instead, a degree *polyorder* polynomial is fit to the last *window_length* values of the edges, and this polynomial is used to evaluate the last *window_length // 2* output values.
- **cval** (*float*) – Value to fill past the edges of the input if *mode* is 'constant'. Default is 0.0.

pynrc.reduce.ref_pixels.chrem_med

pynrc.reduce.ref_pixels.**chrem_med**(*imarr*, *nchans=4*, *yind=None*, *bpmask=None*, *in_place=True*, *mean_func=<function median>*)

Subtract Amplifier Channel Offsets

Sometimes amplifiers have offsets relative to each other due to imperfect tracking of reference pixels. This function determines the average offset from zero of each channel and subtracts the mean/median from the entire channel.

Parameters

- **imarr** (*ndarray*) – Array of image (or single image).
- **nchans** (*int*) – Number of amplifier readout channels.
- **yind** (*array-like*) – Two element array to select a y-range for calculating the channel offset.
- **bpmask** (*bool array*) – Bad pixel mask (1 for bad, 0 for good). Can either be a single image or image cube of same size as *imarr*.
- **in_place** (*bool*) – Correct in-place? If False, returns a copy of the array with channels offset.
- **mean_func** (*func*) – Function to use for performing the mean calculation.

pynrc.reduce.ref_pixels.mask_helper

pynrc.reduce.ref_pixels.**mask_helper**()

Helper to handle indices and logical indices of a mask

Output: index, a function, with signature indices = index(logical_indices), to convert logical indices of a mask to 'equivalent' indices

Example

```
>>> # linear interpolation of NaNs
>>> mask = np.isnan(y)
>>> x = mask_helper(y)
>>> y[mask]= np.interp(x(mask), x(~mask), y[~mask])
```

pynrc.reduce.ref_pixels.ref_filter

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.reffix_amps

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.reffix_hxrg

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.smooth_fft

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).

Classes

NRC_refs(data, header[, DMS, altcol, do_all])	Reference pixel correction object

pynrc.reduce.ref_pixels.NRC_refs

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.

__init__(*data*, *header*, *DMS=False*, *altcol=True*, *do_all=False*, ***kwargs*)

Methods

__init__(data, header[, DMS, altcol, do_all])	

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

Attributes

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

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`.

property multiaccum
A `multiaccum` object

property multiaccum_times
Exposure timings in dictionary

property refs_bot
Return raw bottom reference values

property refs_left
Return raw left reference values

property refs_right
Return raw right reference values

property refs_top
Return raw top reference values

1.9.5 Utilities

maths.coords	Telescope coordinate information
maths.image_manip	
nrc_utils	pyNRC utility functions
opds	
nb_funcs	
logging_utils	

pynrc.maths.coords

Telescope coordinate information

Functions

Tel2Sci_info(channel, coords[, pupil, ...])	Telescope coords converted to Science coords
det_to_sci(image, detid)	Detector to science orientation
sci_to_det(image, detid)	Science to detector orientation
siafap_sci_coords(inst[, coord_vals, ...])	Return the detector, sci position, and full frame aperture name for a set of coordinate values.

pynrc.maths.coords.Tel2Sci_info

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_sci

pynrc.maths.coords.**det_to_sci**(*image*, *detid*)
Detector to science orientation

Reorient image from detector coordinates to 'sci' coordinate system. This places +V3 up and +V2 to the LEFT. Detector pixel (0,0) is assumed to be in the bottom left. Simply performs axes flips.

Parameters

- **image** (*ndarray*) – Input image to tranform.
- **detid** (*int or str*) – NIRCam detector/SCA ID, either 481-490 or A1-B5.

pynrc.maths.coords.sci_to_det

pynrc.maths.coords.**sci_to_det**(*image*, *detid*)
Science to detector orientation

Reorient image from 'sci' coordinates to detector coordinate system. Assumes +V3 up and +V2 to the LEFT. The result places the detector pixel (0,0) in the bottom left. Simply performs axes flips.

Parameters

- **image** (*ndarray*) – Input image to tranform.
- **detid** (*int or str*) – NIRCam detector/SCA ID, either 481-490 or A1-B5.

pynrc.maths.coords.siafap_sci_coords

pynrc.maths.coords.**siafap_sci_coords**(*inst*, *coord_vals=None*, *coord_frame='tel'*)
Return the detector, sci position, and full frame aperture name for a set of coordinate values.

pynrc.maths.image_manip

Functions

align_LSQ(reference, target[, mask, pad, ...])	Find best shift value
fix_nans_with_med(im[, niter_max, verbose])	Iteratively fix NaNs with surrounding Real data
optimal_difference(im_sci, im_ref, scale[, ...])	Optimize subtraction of ref PSF
scale_ref_image(im1, im2[, mask, ...])	Reference image scaling
shift_subtract(params, reference, target[, ...])	Shift and subtract image

pynrc.maths.image_manip.align_LSQ

pynrc.maths.image_manip.**align_LSQ**(*reference*, *target*, *mask=None*, *pad=False*, *interp='cubic'*, *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.
- **interp** (*str*) – Interpolation for fshift function. Default is 'cubic'. Options are 'linear', 'cubic', or 'quintic'.
- **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.fix_nans_with_med

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.optimal_difference

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.scale_ref_image

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.shift_subtract

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

Shift and subtract image

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.
- **interp** (*str*) – Interpolation for fshift function. Default is 'cubic'. Options are 'linear', 'cubic', or 'quintic'.
- **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.nrc_utils

pyNRC utility functions

Functions

bin_spectrum(sp, wave[, waveunits])	Rebin spectrum
build_mask([module, pixscale, filter, ...])	Create coronagraphic mask image
build_mask_detid(detid[, oversample, ...])	Create mask image for a given detector
channel_select(bp)	Select wavelength channel
coron_ap_locs(module, channel, mask[, ...])	Coronagraph mask aperture locations and sizes
coron_detector(mask, module[, channel])	Return detector name for a given coronagraphic mask, module, and channel.
coron_trans(name[, module, pixelscale, ...])	Build a transmission image of a coronagraphic mask spanning the 20" coronagraphic FoV.
gen_unconvolved_point_source_image(nrc, ...)	Create an unconvolved image with sub-pixel shifts
get_detname(det_id)	Return NRC[A-B][1-5] for valid detector/SCA IDs
grism_background(filter[, pupil, module, ...])	Returns a 1D array of grism Zodiacal/thermal background emission model, including roll-off from pick-off mirror (POM) edges.
grism_background_com(filter[, pupil, ...])	
grism_background_image(filter[, pupil, ...])	Create full grism background image
make_grism_slope(nrc, src_tbl, tel_pointing, ...)	Create slope image
nproc_use_convolve(fov_pix, oversample[, npsf])	Attempt to estimate a reasonable number of processes to use for multiple simultaneous convolve_fft calculations.
offset_bar(filt, mask)	Bar mask offset locations
pickoff_image(ap_obs, v2_obj, v3_obj, flux_obj)	Create an unconvolved image of filled pixel values that have been shifted via bilinear interpolation.
pickoff_xy(ap_obs_name)	Return pickoff mirror FoV x/y limits in terms of science pixel coordinates
pix_noise([ngroup, nf, nd2, tf, rn, ktc, ...])	Noise per pixel
place_grism_spec(nrc, sp, xpix, ypix[, ...])	Create spectral image and place ref wavelenght at (x,y) location
place_grismr_tso(waves, imarr, siaf_ap[, ...])	Shift image such that undeviated wavelength sits at the SIAF aperture reference location.
var_ex_model(ng, nf, params)	Variance Excess Model
zodi_euclid(locstr, year, day[, ...])	IPAC Euclid Background Model
zodi_spec([zfact, ra, dec, thisday])	Zodiacal light spectrum.

pynrc.nrc_utils.bin_spectrum

pynrc.nrc_utils.**bin_spectrum**(*sp*, *wave*, *waveunits='um'*)

Rebin spectrum

Rebin a pysynphot.spectrum to a different 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.

pynrc.nrc_utils.build_mask

pynrc.nrc_utils.**build_mask**(*module='A'*, *pixscale=None*, *filter=None*, *nd_squares=True*)

Create coronagraphic mask image

Return a truncated image of the full coronagraphic mask layout for a given module. Assumes each mask is exactly 20" across.

+V3 is up, and +V2 is to the left.

pynrc.nrc_utils.build_mask_detid

pynrc.nrc_utils.**build_mask_detid**(*detid*, *oversample=1*, *ref_mask=None*, *pupil=None*, *filter=None*, *nd_squares=True*, *mask_holder=True*)

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? This affects holder placement. If None, then defaults based on ref_mask.

pynrc.nrc_utils.channel_select

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.coron_ap_locs

pynrc.nrc_utils.**coron_ap_locs**(*module*, *channel*, *mask*, *pupil=None*, *full=False*)
Coronagraph mask aperture locations and sizes

Returns a dictionary of the detector aperture sizes and locations. Attributes 'cen' and 'loc' are in terms of (x,y) detector pixels. 'cen_sci' is sci coords location.

pynrc.nrc_utils.coron_detector

pynrc.nrc_utils.**coron_detector**(*mask*, *module*, *channel=None*)
Return detector name for a given coronagraphic mask, module, and channel.

pynrc.nrc_utils.coron_trans

pynrc.nrc_utils.**coron_trans**(*name*, *module='A'*, *pixelscale=None*, *npix=None*, *oversample=1*, *nd_squares=True*, *shift_x=None*, *shift_y=None*, *filter=None*)
Build a transmission image of a coronagraphic mask spanning the 20" coronagraphic FoV.

oversample is used only if pixelscale is set to None.

Returns the intensity transmission (square of the amplitude transmission).

pynrc.nrc_utils.gen_unconvolved_point_source_image

pynrc.nrc_utils.**gen_unconvolved_point_source_image**(*nrc*, *tel_pointing*, *ra_deg*, *dec_deg*, *mags*, *expnum=1*, *osamp=1*, *siaf_ap_obs=None*, ***kwargs*)
Create an unconvolved image with sub-pixel shifts

pynrc.nrc_utils.get_detname

pynrc.nrc_utils.**get_detname**(*det_id*)
Return NRC[A-B][1-5] for valid detector/SCA IDs

pynrc.nrc_utils.grism_background

pynrc.nrc_utils.**grism_background**(*filter*, *pupil='GRISM0'*, *module='A'*, *sp_bg=None*, *orders=[1, 2]*, *wref=None*, *upper=9.6*, ***kwargs*)
Returns a 1D array of grism Zodiacal/thermal background emission model, including roll-off from pick-off mirror (POM) edges. By default, this includes light dispersed by the 1st and 2nd grism orders (m=1 and m=2).

For column dipsersion, we ignore the upper region occupied by the coronagraphic mask region by default. The preferred way to include this region is to add the dispersed COM image from the *grism_background_com* function to create the full 2048x2048 image. Or, more simply (but less accurate) is to set an *upper* value of 31.2, which is the approximately distance (in arcsec) from the top of the detector to the top of the coronagraphic field of view.

Parameters

- **filter** (*str*) – Name of filter (Long Wave only).
- **pupil** (*str*) – Either 'GRISM0' ('GRISMR') or 'GRISM90' ('GRISMC').

- **module** (*str*) – NIRCam 'A' or 'B' module.
- **sp_bg** (`pysynphot.spectrum`) – Spectrum of Zodiacal background emission, which gets multiplied by bandpass throughput to determine final wavelength-dependent flux that is then dispersed.
- **orders** (*array-like*) – What spectral orders to include? Valid orders are 1 and 2.
- **wref** (*float or None*) – Option to set the undeviated wavelength, otherwise this will search a lookup table depending on the grism.
- **upper** (*float*) – Set the maximum bounds for out-of-field flux to be dispersed onto the detector. By default, this value is 9.6", corresponding to the bottom of the coronagraphic mask. Use *grism_background_com* to then include image of dispersed COM mask. If you want something simpler, increase this value to 31.2" to assume the coronagraphic FoV is free of any holder blockages or substrate and occulting masks.

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.

pynrc.nrc_utils.grism_background_com

pynrc.nrc_utils.**grism_background_com**(*filter*, *pupil='GRISM90'*, *module='A'*, *sp_bg=None*, *wref=None*, ***kwargs*)

pynrc.nrc_utils.grism_background_image

pynrc.nrc_utils.**grism_background_image**(*filter*, *pupil='GRISM0'*, *module='A'*, *sp_bg=None*, *include_com=True*, ***kwargs*)

Create full grism background image

pynrc.nrc_utils.make_grism_slope

pynrc.nrc_utils.**make_grism_slope**(*nrc*, *src_tbl*, *tel_pointing*, *expnum*, *add_offset=None*, ***kwargs*)

Create slope image

pynrc.nrc_utils.nproc_use_convolve

pynrc.nrc_utils.**nproc_use_convolve**(*fov_pix*, *oversample*, *npsf=None*)

Attempt to estimate a reasonable number of processes to use for multiple simultaneous convolve_fft calculations.

Here we attempt to estimate how many such calculations can happen in parallel without swapping to disk, with a mixture of empiricism and conservatism. One really does not want to end up swapping to disk with huge arrays.

NOTE: Requires psutil package. Otherwise defaults to mp.cpu_count() / 2

Parameters

- **fov_pix** (*int*) – Square size in detector-sampled pixels of final PSF image.
- **oversample** (*int*) – The optical system that we will be calculating for.
- **npsf** (*int*) – Number of PSFs. Sets maximum # of processes.

pynrc.nrc_utils.offset_bar

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)
```

pynrc.nrc_utils.pickoff_image

pynrc.nrc_utils.**pickoff_image**(*ap_obs*, *v2_obj*, *v3_obj*, *flux_obj*, *oversample=1*)
Create an unconvolved image of filled pixel values that have been shifted via bilinear interpolation. The image will then be convolved with a PSF to create the a focal plane image that is the size of the NIRCam pick-off mirror. This image should then be cropped to generate the final detector image.

Returns the tuple (xsci, ysci, image), where xsci and ysci are the science coordinates associated with the image.

Parameters

- **ap_obs** (*str*) – Name of aperture in which the observation is taking place. Necessary to determine pixel locations for stars.
- **v2_obj** (*ndarray*) – List of V2 coordiantes of stellar sources
- **v3_obj** (*ndarray*) – List of V3 coordinates of stellar sources
- **flux_obj** (*ndarray*) – List of fluxes (e-/sec) for each source

Keyword Arguments **oversample** (*int*) – If set, the returns an oversampled version of the image to convolve with PSFs. If set to one, then detector pixels.

pynrc.nrc_utils.pickoff_xy

pynrc.nrc_utils.**pickoff_xy**(*ap_obs_name*)
Return pickoff mirror FoV x/y limits in terms of science pixel coordinates

ap_obs : Aperture to create observation (e.g., 'NRCA5_FULL')

pynrc.nrc_utils.pix_noise

pynrc.nrc_utils.**pix_noise**(*ngroup=2*, *nf=1*, *nd2=0*, *tf=10.73677*, *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 time
- **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.place_grism_spec

pynrc.nrc_utils.**place_grism_spec**(*nrc*, *sp*, *xpix*, *ypix*, *wref=None*, *return_oversample=False*)

Create spectral image and place ref wavelenght at (x,y) location

Given a NIRCam instrument object and input spectrum, create a dispersed PSF and place the undeviated reference wavelength at the specified (xpix,ypix) coordinates (assuming 'sci' coords).

Returned values will be a tuple of (wspec, imspec)

pynrc.nrc_utils.place_grismr_tso

pynrc.nrc_utils.**place_grismr_tso**(*waves*, *imarr*, *siaf_ap*, *wref=None*, *im_coords='sci'*)

Shift image such that undeviated wavelength sits at the SIAF aperture reference location.

Return image in sience coords.

Parameters

- **waves** (*ndarray*) – Wavelength solution of input image
- **imarr** (*ndarray*) – Input dispersed grism PSF (can be multiple PSFs)
- **siaf_ap** (*pysiaf aperture*) – Grism-specific SIAF aperture class to determine final subarray size and reference point

pynrc.nrc_utils.var_ex_model

pynrc.nrc_utils.**var_ex_model**(*ng*, *nf*, *params*)
Variance Excess Model

Measured pixel variance shows a slight excess above the measured values. The input *params* describes this excess variance. This funciton can be used to fit the excess variance for a variety of different readout patterns.

pynrc.nrc_utils.zodi_euclid

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 `urllib3` 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 the 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.

Parameters

- **locstr** (*str*) – This input field must contain either coordinates (as string), or an object name resolveable via NED or SIMBAD.
- **year** (*string*) – Year. Limited to 2018 to 2029 for L2 position.
- **day** (*string*) – Day of year (1-366). Limited to 2018 Day 274 to 2029 Day 120 for L2 position and ido_viewin=0.
- **wavelength** (*array-like*) – Wavelength in microns (0.5-1000).
- **ido_viewin** (*0 or 1*) – If set to 0, returns zodiacal emission at specific location for input time. If set to 1, then gives the median value for times of the year that the object is in a typical spacecraft viewing zone. Currently this is set to solar elongations between 85 and 120 degrees.

References

See the Euclid Help Website for more details.

pynrc.nrc_utils.zodi_spec

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.opds

pynrc.nb_funcs

Functions

average_slopes(hdulist)	For a series of ramps, calculate the slope images then average together.

continues on next page

Table 38 – continued from previous page

disk_rim_model(a_asec, b_asec[, pa, ...])	Simple geometric model of an inner disk rim that simply creates an ellipsoidal ring with a brightness gradient along the major axis.
do_contrast(obs_dict, wfe_list, filt_keys[, ...])	kwargs to pass to calc_contrast() and their defaults:
do_gen_hdus(obs_dict, filt_keys, ...[, ...])	kwargs to pass to gen_roll_image() and their defaults:
do_opt(obs_dict[, tacq_max])	
do_plot_contrasts(curves_ref, curves_roll, ...)	Plot series of contrast curves.
do_plot_contrasts2(key1, key2, curves_all, ...)	
do_sat_levels(obs[, satval, ng_min, ng_max, ...])	Only for obs.hci classes
make_key(filter[, pupil, mask])	Create identification key (string) based on filter, pupil, and mask
model_info(source, filt, dist[, model_dir])	
obs_optimize(obs_dict[, sp_opt, ...])	Perform ramp optimization on each science and reference observation in a list of filter observations.
obs_wfe(wfe_ref_drift, filt_list, sp_sci, dist)	For a given WFE drift and series of filters, create a list of NIRCam observations.
planet_mags(obs[, age, entropy, mass_list, ...])	Exoplanet Magnitudes
plot_contrasts(curves, nsig, wfe_list[, ...])	Plot contrast curves
plot_contrasts_mjup(curves, nsig, wfe_list)	Plot mass contrast curves
plot_hdulist(hdulist[, ext, xr, yr, ax, ...])	
plot_images(obs_dict, hdu_dict, filt_keys, ...)	
plot_images_swlw(obs_dict, hdu_dict, ...[, ...])	
plot_planet_patches(ax, obs[, age, entropy, ...])	Plot exoplanet magnitudes in region corresponding to extinction values.
update_yscale(ax, scale_type[, ylim])	

pynrc.nb_funcs.average_slopes

pynrc.nb_funcs.**average_slopes**(*hdulist*)

For a series of ramps, calculate the slope images then average together.

pynrc.nb_funcs.disk_rim_model

pynrc.nb_funcs.**disk_rim_model**(*a_asec*, *b_asec*, *pa=0*, *sig_asec=0.1*, *flux_frac=0.5*, *flux_tot=1.0*, *flux_units='mJy'*, *wave_um=None*, *dist_pc=None*, *pixsize=0.007*, *fov_pix=401*)

Simple geometric model of an inner disk rim that simply creates an ellipsoidal ring with a brightness gradient along the major axis.

Parameters

- **a_asec** (*float*) – Semi-major axis of ellipse
- **ba_asec** (*float*) – Semi-minor axis of ellipse

Keyword Arguments

- **pa** (*float*) – Position angle of major axis
- **sig_asec** (*float*) – Sigma width of ring model
- **flux_frac** (*float*) – A brightness gradient can be applied along the semi-major axis. This parameter dictates the relative brightness of the minimum flux (at the center of the axis) compared to the flux at the out edge of the geometric ring.
- **flux_tot** (*float*) – The total integrated flux of disk model.
- **flux_units** (*str*) – Units corresponding to *flux_tot*.
- **wave_um** (*float or None*) – Wavelength (in um) corresponding to *flux_tot*. Saved in output FITS header unless the value is None.
- **dist_pc** (*float or None*) – Assumed distance of model (in pc). Saved in output FITS header unless the value is None.
- **pixsize** (*float*) – Desired model pixel size in arcsec.
- **fov_pix** (*int*) – Number of pixels for x/y dimensions of output model data.

pynrc.nb_funcs.do_contrast

pynrc.nb_funcs.**do_contrast**(*obs_dict*, *wfe_list*, *filt_keys*, *nsig=5*, *roll_angle=10*, *verbose=False*, ***kwargs*)

kwargs to pass to calc_contrast() and their defaults:

no_ref = False func_std = robust.medabsdev exclude_disk = True exclude_planets = True exclude_noise = False opt_diff = True fix_sat = False ref_scale_all = False

pynrc.nb_funcs.do_gen_hdus

pynrc.nb_funcs.**do_gen_hdus**(*obs_dict*, *filt_keys*, *wfe_ref_drift*, *wfe_roll_drift*, *return_oversample=True*, ***kwargs*)

kwargs to pass to gen_roll_image() and their defaults:

PA1 = 0 PA2 = 10 zfact = None return_oversample = True exclude_disk = False exclude_noise = False no_ref = False opt_diff = True use_cmask = False ref_scale_all = False xyoff_roll1 = None xyoff_roll2 = None xyoff_ref = None

pynrc.nb_funcs.do_opt

pynrc.nb_funcs.**do_opt**(*obs_dict*, *tacq_max=1800*, ***kwargs*)

pynrc.nb_funcs.do_plot_contrasts

pynrc.nb_funcs.**do_plot_contrasts**(*curves_ref*, *curves_roll*, *nsig*, *wfe_list*, *obs*, *age*, *age2=None*, *sat_rad=0*, *jup_mag=True*, *xr=[0, 10]*, *yr=[22, 8]*, *xr2=[0, 10]*, *yscale2='log'*, *yr2=None*, *save_fig=False*, *outdir=''*, *return_fig_axes=False*, ***kwargs*)

Plot series of contrast curves.

pynrc.nb_funcs.do_plot_contrasts2

pynrc.nb_funcs.**do_plot_contrasts2**(*key1*, *key2*, *curves_all*, *nsig*, *obs_dict*, *wfe_list*, *age*, *sat_dict=None*, *label1='Curves1'*, *label2='Curves2'*, *xr=[0, 10]*, *yr=[24, 8]*, *yscale2='log'*, *yr2=None*, *av_vals=[0, 10]*, *curves_all2=None*, *c1=None*, *c2=None*, *linder_models=True*, *planet_patches=True*, ***kwargs*)

pynrc.nb_funcs.do_sat_levels

pynrc.nb_funcs.**do_sat_levels**(*obs*, *satval=0.95*, *ng_min=2*, *ng_max=None*, *verbose=True*, *plot=True*, *xylim=2.5*, *return_fig_axes=False*)

Only for obs.hci classes

pynrc.nb_funcs.make_key

pynrc.nb_funcs.**make_key**(*filter*, *pupil=None*, *mask=None*)

Create identification key (string) based on filter, pupil, and mask

pynrc.nb_funcs.model_info

pynrc.nb_funcs.**model_info**(*source*, *filt*, *dist*, *model_dir=''*)

pynrc.nb_funcs.obs_optimize

pynrc.nb_funcs.**obs_optimize**(*obs_dict*, *sp_opt=None*, *well_levels=None*, *tacq_max=1800*, ***kwargs*)

Perform ramp optimization on each science and reference observation in a list of filter observations. Updates the detector MULTIACCUM settings for each observation in the dictionary.

snr_goal = 5 snr_frac = 0.02 tacq_max = 1400 tacq_frac = 0.01 nint_min = 15 ng_max = 10

pynrc.nb_funcs.obs_wfe

pynrc.nb_funcs.**obs_wfe**(*wfe_ref_drift*, *filt_list*, *sp_sci*, *dist*, *sp_ref=None*, *args_disk=None*, *wind_mode='WINDOW'*, *subsize=None*, *fov_pix=None*, *verbose=False*, *narrow=False*, *model_dir=None*, *large_grid=False*, ***kwargs*)

For a given WFE drift and series of filters, create a list of NIRCam observations.

pynrc.nb_funcs.planet_mags

pynrc.nb_funcs.**planet_mags**(*obs*, *age=10*, *entropy=13*, *mass_list=[10, 5, 2, 1]*, *av_vals=[0, 25]*, *atmo='hy3s'*, *cond=False*, *linder=False*, ***kwargs*)

Exoplanet Magnitudes

Determine a series of exoplanet magnitudes for given observation. By default, use Spiegel & Burrows 2012 models, but has the option to use the COND models from https://phoenix.ens-lyon.fr/Grids. These are useful because SB12 model grids only ranges from 1-1000 Myr with masses 1-15 MJup.

cond [bool] Instead of plotting sensitivities, use COND models to plot the limiting planet masses.

linder [bool] Instead of plotting sensitivities, use Linder models to plot the limiting planet masses.

file [string] Location and name of COND or Linder file.

pynrc.nb_funcs.plot_contrasts

pynrc.nb_funcs.**plot_contrasts**(*curves*, *nsig*, *wfe_list*, *obs=None*, *sat_rad=None*, *ax=None*, *colors=None*, *xr=[0, 10]*, *yr=[25, 5]*, *return_axes=False*)

Plot contrast curves

Plot a series of contrast curves for corresponding WFE drifts.

Parameters

- **curves** (*list*) – A list with length corresponding to *wfe_list*. Each list element has three arrays in a tuple: the radius in arcsec, n-sigma contrast, and n-sigma sensitivity limit (vega mag).
- **nsig** (*float*) – N-sigma limit corresponding to sensitivities/contrasts.
- **wfe_list** (*array-like*) – List of WFE drift values corresponding to each set of sensitivities in *curves* argument.

Keyword Arguments

- **obs** (`obs_hci`) – Corresponding observation class that created the contrast curves. Uses distances and stellar magnitude to plot contrast and AU distances on opposing axes.
- **sat_rad** (*float*) – Saturation radius in arcsec. If >0, then that part of the contrast curve is excluded from the plot
- **ax** (*matplotlib.axes*) – Axes on which to plot curves.
- **colors** (*None, array-like*) – List of colors for contrast curves. Default is gradient of blues.
- **return_axes** (*bool*) – Return the matplotlib axes to continue plotting. If *obs* is set, then this returns three sets of axes.

pynrc.nb_funcs.plot_contrasts_mjup

pynrc.nb_funcs.**plot_contrasts_mjup**(*curves*, *nsig*, *wfe_list*, *obs=None*, *sat_rad=None*, *age=100*, *ax=None*, *colors=None*, *xr=[0, 10]*, *yr=None*, *file=None*, *linder_models=True*, *twin_ax=False*, *return_axes=False*, ***kwargs*)

Plot mass contrast curves

Plot a series of mass contrast curves for corresponding WFE drifts.

Parameters

- **curves** (*list*) – A list with length corresponding to *wfe_list*. Each list element has three arrays in a tuple: the radius in arcsec, n-sigma contrast, and n-sigma sensitivity limit (vega mag).
- **nsig** (*float*) – N-sigma limit corresponding to sensitivities/contrasts.
- **wfe_list** (*array-like*) – List of WFE drift values corresponding to each set of sensitivities in *curves* argument.

Keyword Arguments

- **obs** (`obs_hci`) – Corresponding observation class that created the contrast curves. Uses distances and stellar magnitude to plot contrast and AU distances on opposing axes. Also necessary for mjup=True.
- **sat_rad** (*float*) – Saturation radius in arcsec. If >0, then that part of the contrast curve is excluded from the plot
- **age** (*float*) – Required for plotting limiting planet masses.
- **file** (*string*) – Location and name of COND or Linder isochrone file.
- **ax** (*matplotlib.axes*) – Axes on which to plot curves.
- **colors** (*None, array-like*) – List of colors for contrast curves. Default is gradient of blues.
- **twin_ax** (*bool*) – Plot opposing axes in alternate units.
- **return_axes** (*bool*) – Return the matplotlib axes to continue plotting. If *obs* is set, then this returns three sets of axes.

pynrc.nb_funcs.plot_hdulist

pynrc.nb_funcs.**plot_hdulist**(*hdulist*, *ext=0*, *xr=None*, *yr=None*, *ax=None*, *return_ax=False*, *cmap=None*, *scale='linear'*, *vmin=None*, *vmax=None*, *axes_color='white'*, *half_pix_shift=False*, *cb_label='Counts/sec'*, ***kwargs*)

pynrc.nb_funcs.plot_images

pynrc.nb_funcs.**plot_images**(*obs_dict*, *hdu_dict*, *filt_keys*, *wfe_drift*, *fov=10*, *save_fig=False*, *outdir=''*, *return_fig_axes=False*)

pynrc.nb_funcs.plot_images_swlw

pynrc.nb_funcs.**plot_images_swlw**(*obs_dict*, *hdu_dict*, *filt_keys*, *wfe_drift*, *fov=10*, *save_fig=False*, *outdir=''*, *return_fig_axes=False*)

pynrc.nb_funcs.plot_planet_patches

pynrc.nb_funcs.**plot_planet_patches**(*ax*, *obs*, *age=10*, *entropy=13*, *mass_list=[10, 5, 2, 1]*, *av_vals=[0, 25]*, *cols=None*, *update_title=False*, *linder=False*, ***kwargs*)

Plot exoplanet magnitudes in region corresponding to extinction values.

pynrc.nb_funcs.update_yscale

pynrc.nb_funcs.**update_yscale**(*ax*, *scale_type*, *ylim=None*)

pynrc.logging_utils

Functions

restart_logging([verbose])	Restart Logging
setup_logging([level, filename, verbose])	Setup Logging

pynrc.logging_utils.restart_logging

pynrc.logging_utils.**restart_logging**(*verbose=True*)

Restart Logging

Restart logging using the same settings as those currently stored in conf.logging_level. This function was shamelessly stolen from WebbPSF utils.py.

Parameters **verbose** (*boolean*) – Should this function print the new logging targets to standard output?

pynrc.logging_utils.setup_logging

pynrc.logging_utils.**setup_logging**(*level='INFO'*, *filename=None*, *verbose=True*)

Setup Logging

Allows selection of logging detail and output locations (screen and/or file). Shamelessly stolen from WebbPSF utils.py.

This is a convenience wrapper to Python's built-in logging package. By default, this sets up log messages to be written to the screen, but the user can also request logging to a file.

Editing the WebbPSF config file to set *autoconfigure_logging = True* (and any of the logging settings you wish to persist) instructs WebbPSF to apply your settings on import. (This is not done by default in case you have configured *logging* yourself and don't wish to overwrite your configuration.)

For more advanced log handling, see the Python logging module's own documentation.

Parameters

- **level** (*str*) – Name of log output to show. Defaults to 'INFO', set to 'DEBUG' for more extensive messages, or to 'WARN' or 'ERROR' for fewer.
- **filename** (*str, optional*) – Filename to write the log output to. If not set, output will just be displayed on screen. (Default: None)

Examples

```
>>> pynrc.setup_logging(filename='pynrc_log.txt')
```

This will save all log messages to 'pynrc_log.txt' in the current directory.

```
>>> pynrc.setup_logging(level='WARN')
```

This will show only WARNING or ERROR messages on screen, and not save any logs to files at all (since the filename argument is None)

Classes

FilterLevelRange(min_level, max_level)

pynrc.logging_utils.FilterLevelRange

class pynrc.logging_utils.**FilterLevelRange**(*min_level*, *max_level*)

Bases: `object`

__init__(*min_level*, *max_level*)

Methods

__init__(min_level, max_level)
`filter`(record)

1.9.6 WebbPSF Extensions

bandpasses
coords
image_manip
logging_utils

continues on next page

Table 42 – continued from previous page

maths	
opds	
psfs	
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.
spectra	
utils	
webbpsf_ext_core	

webbpsf_ext.bandpasses

Functions

bp_2mass(filter)	2MASS Bandpass
bp_gaia(filter[, release])	GAIA Bandpass
bp_igood(bp[, min_trans, fext])	Given a bandpass with transmission 0.0-1.0, return the indices that cover only the region of interest and ignore those wavelengths with very low transmission less than and greater than the bandpass width.
bp_wise(filter)	WISE Bandpass
miri_filter(filter, **kwargs)	No need to include pupil for the
nircam_com_nd([wave_out])	NIRCam COM Neutral Density squares
nircam_com_th([wave_out, ND_acq])	
nircam_filter(filter[, pupil, mask, module, ...])	Read filter bandpass.
nircam_grism_res([pupil, module, m])	NIRCam Grism resolution
nircam_grism_wref([pupil, module])	NIRCam Grism undeviated wavelength
niriss_grism_res([m])	Grism resolution
niriss_grism_wref()	NIRISS Grism undeviated wavelength (um)
read_filter(self, *args, **kwargs)	

webbpsf_ext.bandpasses.bp_2mass

webbpsf_ext.bandpasses.**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.

webbpsf_ext.bandpasses.bp_gaia

webbpsf_ext.bandpasses.**bp_gaia**(*filter*, *release='DR2'*)
GAIA Bandpass

Create a bandpass class for GAIA filter to generate synthetic photometry.

Parameters **filter** (*str*) – Filters 'g', 'bp', or 'rp'

webbpsf_ext.bandpasses.bp_igood

webbpsf_ext.bandpasses.**bp_igood**(*bp*, *min_trans=0.001*, *fext=0.05*)
Given a bandpass with transmission 0.0-1.0, return the indices that cover only the region of interest and ignore those wavelengths with very low transmission less than and greater than the bandpass width.

webbpsf_ext.bandpasses.bp_wise

webbpsf_ext.bandpasses.**bp_wise**(*filter*)
WISE Bandpass

Create a WISE W1-W4 filter bandpass used to generate synthetic photometry.

Parameters **filter** (*str*) – Filter 'w1', 'w2', 'w3', or 'w4'.

Returns pysynphot.obsbandpass – A Pysynphot bandpass object.

webbpsf_ext.bandpasses.miri_filter

webbpsf_ext.bandpasses.**miri_filter**(*filter*, ***kwargs*)
No need to include pupil for the

webbpsf_ext.bandpasses.nircam_com_nd

webbpsf_ext.bandpasses.**nircam_com_nd**(*wave_out=None*)
NIRCam COM Neutral Density squares

Return optical density, where final throughput is equal to 10**(-1*OD).

webbpsf_ext.bandpasses.nircam_com_th

webbpsf_ext.bandpasses.**nircam_com_th**(*wave_out=None*, *ND_acq=False*)

webbpsf_ext.bandpasses.nircam_filter

webbpsf_ext.bandpasses.**nircam_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*, *include_blocking=True*, ***kwargs*)

Read filter bandpass.

Read in filter throughput curve from file generated by STScI. Includes: OTE, NRC mirrors, dichroic, filter curve, and detector QE.

TODO: 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 or grism pupils with or without coron substrate.
- **include_blocking** (*bool*) – Include wide-band blocking filter for those filters in pupil wheel. These include: 'F162M', 'F164N', 'F323N', 'F405N', 'F466N', 'F470N'

Returns `pysynphot.obsbandpass` – A Pysynphot bandpass object.

webbpsf_ext.bandpasses.nircam_grism_res

webbpsf_ext.bandpasses.**nircam_grism_res**(*pupil='GRISM'*, *module='A'*, *m=1*)
NIRCam Grism resolution

Based on the pupil input and module, return the spectral dispersion and resolution as a tuple (res, dw).

Parameters

- **pupil** (*str*) – 'GRISMC' or 'GRISMR', otherwise assume res=1000 pix/um. 'GRISM0' is GRISMR; 'GRISM90' is GRISMC
- **module** (*str*) – 'A' or 'B'
- **m** (*int*) – Spectral order (1 or 2).

webbpsf_ext.bandpasses.nircam_grism_wref

webbpsf_ext.bandpasses.**nircam_grism_wref**(*pupil='GRISM'*, *module='A'*)
NIRCam Grism undeviated wavelength

webbpsf_ext.bandpasses.niriss_grism_res

webbpsf_ext.bandpasses.**niriss_grism_res**(*m=1*)
Grism resolution

Based on the pupil input and module, return the spectral dispersion and resolution as a tuple (res, dw).

Parameters **m** (*int*) – Spectral order (1 or 2).

webbpsf_ext.bandpasses.niriss_grism_wref

webbpsf_ext.bandpasses.**niriss_grism_wref**()
NIRISS Grism undeviated wavelength (um)

webbpsf_ext.bandpasses.read_filter

webbpsf_ext.bandpasses.**read_filter**(*self*, **args*, ***kwargs*)

webbpsf_ext.coords

Functions

NIRCam_V2V3_limits(module[, channel, pupil, ...])	NIRCam V2/V3 bounds +10" border encompassing detector.
ap_radec(ap_obs, ap_ref, coord_ref, pa[, ...])	Aperture reference point(s) RA/Dec
dist_image(image[, pixscale, center, ...])	Pixel distances
gen_sgd_offsets(sgd_type[, slew_std, ...])	Create a series of x and y position offsets for a SGD pattern.
get_NRC_v2v3_limits([pupil, border, ...])	V2/V3 Limits for a given module stored within an dictionary

continues on next page

Table 44 – continued from previous page

get_idl_offset([base_offset, dith_offset, ...])	Calculate pointing offsets in 'idl' coordinates with errors.
plotAxes(ax[, position, label1, label2, ...])	Compass arrows
radec_offset(ra, dec, dist, pos_ang)	Return (RA, Dec) of a position offset relative to some input (RA, Dec).
radec_to_coord(coord_objs, siaf_ref_name, ...)	RA/Dec to 'tel' (arcsec), 'sci' (pixels), or 'det' (pixels)
radec_to_v2v3(coord_objs, siaf_ref_name, ...)	RA/Dec to V2/V3
rtheta_to_xy(r, theta)	Convert (r,theta) to (x,y)
v2v3_to_pixel(ap_obs, v2_obj, v3_obj[, frame])	V2/V3 to pixel coordinates
xy_rot(x, y, ang)	Rotate (x,y) positions to new coords
xy_to_rtheta(x, y)	Convert (x,y) to (r,theta)

webbpsf_ext.coords.NIRCam_V2V3_limits

webbpsf_ext.coords.**NIRCam_V2V3_limits**(*module*, *channel='LW'*, *pupil=None*, *rederive=False*, *return_corners=False*, ***kwargs*)

NIRCam V2/V3 bounds +10" border encompassing detector.

webbpsf_ext.coords.ap_radec

webbpsf_ext.coords.**ap_radec**(*ap_obs*, *ap_ref*, *coord_ref*, *pa*, *base_off=(0, 0)*, *dith_off=(0, 0)*, *get_cenpos=True*, *get_vert=False*)

Aperture reference point(s) RA/Dec

Given the (RA, Dec) and position angle of a given reference aperture, return the (RA, Dec) associated with the reference point (usually center) of a different aperture. Can also return the corner vertices of the aperture.

Typically, the reference aperture (ap_ref) is used for the telescope pointing information (e.g., NRCALL), but you may want to determine locations of the individual detector apertures (NRCA1_FULL, NRCB3_FULL, etc).

Parameters

- **ap_obs** (*str*) – Name of observed aperture (e.g., NRCA5_FULL)
- **ap_ref** (*str*) – Name of reference aperture (e.g., NRCALL_FULL)
- **coord_ref** (*tuple or list*) – Center position of reference aperture (RA/Dec deg)
- **pa** (*float*) – Position angle in degrees measured from North to V3 axis in North to East direction.

Keyword Arguments

- **base_off** (*list or tuple*) – X/Y offset of overall aperture offset (see APT pointing file)
- **dither_off** (*list or tuple*) – Additional offset from dithering (see APT pointing file)
- **get_cenpos** (*bool*) – Return aperture reference location coordinates?
- **get_vert** (*bool*) – Return closed polygon vertices (useful for plotting)?

webbpsf_ext.coords.dist_image

webbpsf_ext.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*) – Pixel 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.

webbpsf_ext.coords.gen_sgd_offsets

webbpsf_ext.coords.**gen_sgd_offsets**(*sgd_type*, *slew_std=5*, *fsm_std=2.5*, *rand_seed=None*)
Create a series of x and y position offsets for a SGD pattern. This includes the central position as the first in the series. By default, will also add random movement errors using the *slew_std* and *fsm_std* keywords. Returned values in arcsec.

Parameters

- **sgd_type** (*str*) – Small grid dither pattern. Valid types are '9circle', '5box', '5diamond', '3bar', '5bar', '5miri', and '9miri' where the first four refer to NIRCam coronagraphic dither positions and the last two are for MIRI coronagraphy.
- **fsm_std** (*float*) – One-sigma accuracy per axis of fine steering mirror positions. This provides randomness to each position relative to the nominal central position. Ignored for central position. Values should be in units of mas.
- **slew_std** (*float*) – One-sigma accuracy per axis of the initial slew. This is applied to all positions and gives a baseline offset relative to the desired mask center. Values should be in units of mas.
- **rand_seed** (*int*) – Input a random seed in order to make reproduceable pseudo-random numbers.

webbpsf_ext.coords.get_NRC_v2v3_limits

webbpsf_ext.coords.**get_NRC_v2v3_limits**(*pupil=None*, *border=10*, *return_corners=False*, ***kwargs*)
V2/V3 Limits for a given module stored within an dictionary

border [float] Extend a border by some number of arcsec.

return_corners [bool] Return the actual aperture corners. Otherwise, values are chosen to be a square in V2/V3.

webbpsf_ext.coords.get_idl_offset

webbpsf_ext.coords.**get_idl_offset**(*base_offset=(0, 0)*, *dith_offset=(0, 0)*, *base_std=0*, *use_ta=True*, *dith_std=0*, *use_sgd=True*, *rand_seed=None*, ***kwargs*)

Calculate pointing offsets in 'idl' coordinates with errors. Inputs come from the APT's .pointing file. For a sequence of dithers, make sure to only calculate the base offset once, and all dithers independently. For instance:

```
>>> base_offset = get_idl_offset(base_std=None)
>>> dith0 = get_idl_offset(base_offset, dith_offset=(0,0), dith_std=None)
>>> dith1 = get_idl_offset(base_offset, dith_offset=(-0.01,+0.01), dith_std=None)
>>> dith2 = get_idl_offset(base_offset, dith_offset=(+0.01,+0.01), dith_std=None)
>>> dith3 = get_idl_offset(base_offset, dith_offset=(+0.01,-0.01), dith_std=None)
>>> dith4 = get_idl_offset(base_offset, dith_offset=(-0.01,-0.01), dith_std=None)
```

Parameters

- **base_offset** (*array-like*) – Corresponds to (BaseX, BaseY) columns in .pointing file (arcsec).
- **dith_offset** (*array-like*) – Corresponds to (DithX, DithY) columns in .pointing file (arcsec).
- **base_std** (*float or array-like or None*) – The 1-sigma pointing uncertainty per axis for telescope slew. If None, then standard deviation is chosen to be either 5 mas or 100 mas, depending on *use_ta* setting.
- **use_ta** (*bool*) – If observation uses a target acquisition, then assume only 5 mas of pointing uncertainty, other 100 mas for "blind" pointing.
- **dith_std** (*float or array-like or None*) – The 1-sigma pointing uncertainty per axis for dithers. If None, then standard deviation is chosen to be either 2.5 or 5 mas, depending on *use_sgd* setting.
- **use_sgd** (*bool*) – If True, then we're employing small-grid dithers with the fine steering mirror, which has a ~2.5 mas uncertainty. Otherwise, assume standard small angle maneuver, which has ~5 mas uncertainty.

webbpsf_ext.coords.plotAxes

webbpsf_ext.coords.**plotAxes**(*ax*, *position=(0.9, 0.1)*, *label1='V2'*, *label2='V3'*, *dir1=[- 1, 0]*, *dir2=[0, 1]*, *angle=0*, *alength=0.12*, *width=1.5*, *headwidth=6*, *color='w'*, *alpha=1*, *fontsize=11*)

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.
- **alpha** (*float*) – Transparency.

webbpsf_ext.coords.radec_offset

webbpsf_ext.coords.**radec_offset**(*ra*, *dec*, *dist*, *pos_ang*)

Return (RA, Dec) of a position offset relative to some input (RA, Dec).

Parameters

- **RA** (*float*) – Input RA in deg.
- **Dec** (*float*) – Input Dec in deg.
- **dist** (*float*) – Angular distance in arcsec. Can also be an array of distances.
- **pos_ang** (*float*) – Position angle (positive angles East of North) in degrees. Can also be an array; must match size of *dist*.

Returns

- *Two elements, RA and Dec of calculated offsets in dec degrees.*
- *If multiple offsets specified, then this will be two arrays*
- *of RA and Dec, where eac array has the same size as inputs.*

webbpsf_ext.coords.radec_to_coord

webbpsf_ext.coords.**radec_to_coord**(*coord_objs*, *siaf_ref_name*, *coord_ref*, *pa_ref*, *frame_out='tel'*, *base_off=(0, 0)*, *dith_off=(0, 0)*)

RA/Dec to 'tel' (arcsec), 'sci' (pixels), or 'det' (pixels)

Convert a series of RA/Dec positions to telescope V2/V3 coordinates (in arcsec).

Parameters

- **coord_objs** (*tuple*) – (RA, Dec) positions (deg), where RA and Dec are numpy arrays.
- **siaf_ref_name** (*str*) – Reference SIAF aperture name (e.g., 'NRCALL_FULL')
- **coord_ref** (*list or tuple*) – RA and Dec towards which reference SIAF points
- **pa** (*float*) – Position angle in degrees measured from North to V3 axis in North to East direction.

Keyword Arguments

- **frame_out** (*str*) – One of 'tel' (arcsec), 'sci' (pixels), or 'det' (pixels).
- **base_off** (*list or tuple*) – X/Y offset of overall aperture offset (see APT pointing file)
- **dither_off** (*list or tuple*) – Additional offset from dithering (see APT pointing file)

webbpsf_ext.coords.radec_to_v2v3

webbpsf_ext.coords.**radec_to_v2v3**(*coord_objs*, *siaf_ref_name*, *coord_ref*, *pa_ref*, *frame_out='tel'*, *base_off=(0, 0)*, *dith_off=(0, 0)*)

RA/Dec to V2/V3

Convert a series of RA/Dec positions to telescope V2/V3 coordinates (in arcsec).

Parameters

- **coord_objs** (*tuple*) – (RA, Dec) positions (deg), where RA and Dec are numpy arrays.
- **siaf_ref_name** (*str*) – Reference SIAF aperture name (e.g., 'NRCALL_FULL')
- **coord_ref** (*list or tuple*) – RA and Dec towards which reference SIAF points
- **pa** (*float*) – Position angle in degrees measured from North to V3 axis in North to East direction.

Keyword Arguments

- **frame_out** (*str*) – One of 'tel' (arcsec), 'sci' (pixels), or 'det' (pixels).
- **base_off** (*list or tuple*) – X/Y offset of overall aperture offset (see APT pointing file)
- **dither_off** (*list or tuple*) – Additional offset from dithering (see APT pointing file)

webbpsf_ext.coords.rtheta_to_xy

webbpsf_ext.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).

webbpsf_ext.coords.v2v3_to_pixel

webbpsf_ext.coords.**v2v3_to_pixel**(*ap_obs*, *v2_obj*, *v3_obj*, *frame='sci'*)

V2/V3 to pixel coordinates

Convert object V2/V3 coordinates into pixel positions.

Parameters

- **ap_obs** (*str*) – Name of observed aperture (e.g., NRCA5_FULL)
- **v2_obj** (*ndarray*) – V2 locations of stellar sources.
- **v3_obj** (*ndarray*) – V3 locations of stellar sources.

Keyword Arguments **frame** (*str*) – 'det' or 'sci' coordinate frame. 'det' is always full frame reference. 'sci' is relative to subarray size if not a full frame aperture.

webbpsf_ext.coords.xy_rot

webbpsf_ext.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

webbpsf_ext.coords.xy_to_rtheta

webbpsf_ext.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

Classes

jwst_point(ap_obs_name, ap_ref_name, ra_ref, ...)	JWST Telescope Pointing information

webbpsf_ext.coords.jwst_point

class webbpsf_ext.coords.**jwst_point**(*ap_obs_name*, *ap_ref_name*, *ra_ref*, *dec_ref*, *pos_ang=0*, *base_offset=(0, 0)*, *dith_offsets=[(0, 0)]*, *exp_nums=None*, *base_std=None*, *dith_std=None*, *rand_seed=None*, *rand_seed_base=None*)
Bases: `object`

JWST Telescope Pointing information

Holds pointing coordinates and dither information for a given telescope visit.

Parameters

- **ap_obs_name** (*str*) – Name of the observed instrument aperture.
- **ap_ref_name** (*str*) – Name of the reference instrument aperture. Can be the same as observed, but not always.
- **ra_ref** (*float*) – RA position (in deg) of the reference aperture.
- **dec_ref** (*float*) – Dec position (in deg) of the reference aperture.

Keyword Arguments

- **pos_ang** (*float*) – Position angle (positive angles East of North) in degrees.
- **exp_nums** (*ndarray or None*) – Option to specify exposure numbers associated with each dither position. Useful fro Visit book-keeping. If not set, then will simply be a *np.arange(self.ndith) + 1*
- **base_offset** (*array-like*) – Corresponds to (BaseX, BaseY) columns in .pointing file (arcsec).
- **dith_offset** (*array-like*) – Corresponds to (DithX, DithY) columns in .pointing file (arcsec).
- **base_std** (*float or array-like or None*) – The 1-sigma pointing uncertainty per axis for telescope slew. If None, then standard deviation is chosen to be either 5 mas or 100 mas, depending on *use_ta* attribute (default: True).
- **dith_std** (*float or array-like or None*) – The 1-sigma pointing uncertainty per axis for dithers. If None, then standard deviation is chosen to be either 2.5 or 5 mas, depending on *use_sgd* attribute (default: True).
- **rand_seed** (*None or int*) – Random seed to use for generating repeatable random offsets.
- **rand_seed_base** (*None or int*) – Use a separate random seed for telescope slew offset. Then, *rand_seed* corresponds only to relative dithers. Useful for multiple exposures with same initial slew, but independent dither pattern realizations.

__init__(*ap_obs_name*, *ap_ref_name*, *ra_ref*, *dec_ref*, *pos_ang=0*, *base_offset=(0, 0)*, *dith_offsets=[(0, 0)]*, *exp_nums=None*, *base_std=None*, *dith_std=None*, *rand_seed=None*, *rand_seed_base=None*)

Methods

__init__(ap_obs_name, ap_ref_name, ra_ref, ...)	

ap_radec([idl_off, get_cenpos, get_vert, ...])	Aperture reference point(s) RA/Dec
attitude_matrix([idl_off, ap_siaf_ref, ...])	Return an attitutde correction matrix
gen_random_offsets([rand_seed, ...])	Generate randomized pointing offsets for each dither position
plot_inst_apertures([subarrays, fill])	Plot all apertures in this instrument's SIAF.
plot_main_apertures([fill])	Plot main SIAF telescope apertures.
plot_obs_aperture([fill])	Plot observed aperture
plot_ref_aperture([fill])	Plot reference aperture
radec_to_frame(coord_objs[, frame_out, ...])	RA/Dec to aperture coordinate frame

Attributes

base_std	Target pointing uncertainty (mas)
dith_std	Dither pointing uncertainty (mas)
exp_nums	Exposure Numbers associated with each dither position
ndith	Number of dither positions

ap_radec(*idl_off=(0, 0)*, *get_cenpos=True*, *get_vert=False*, *ap_siaf_obs=None*, ***kwargs*)
Aperture reference point(s) RA/Dec

Given the (RA, Dec) and position angle some reference aperture, return the (RA, Dec) associated with the reference point (usually center) of a different aperture. Can also return the corner vertices of the aperture.

Typically, the reference aperture (self.siaf_ap_ref) is used for the telescope pointing information (e.g., NRCALL), but you may want to determine locations of the individual detector apertures (NRCA1_FULL, NRCB3_FULL, etc).

Parameters

- **idl_off** (*list or tuple*) – X/Y offset of overall aperture offset. Usually a combination of base_off + dith_off (both in idl coordinates)
- **get_cenpos** (*bool*) – Return aperture reference location coordinates?
- **get_vert** (*bool*) – Return closed polygon vertices (useful for plotting)?
- **ap_siaf_obs** (*pysiaf aperture*) – Specify the aperture for which to obtain RA/Dec reference point. The default is self.siaf_ap_obs.

`attitude_matrix`(*idl_off=(0, 0)*, *ap_siaf_ref=None*, *coord_ref=None*, ***kwargs*)
Return an attitutde correction matrix

Parameters

- **idl_off** (*list or tuple*) – X/Y offset of overall aperture offset. Usually a combination of base_off + dith_off (both in idl coordinates).
- **ap_siaf_ref** (*pysiaf aperture*) – Aperture reference being offset (uses it's V2/V3 ref/center coords). By default, uses self.siaf_ap_ref.
- **coord_ref** (*tuple*) – RA/Dec (in deg) reference coordinate nominally placed at aperture's reference location prior to dither offset. Default is (self.ra_ref, self.dec_ref).

`property base_std`
Target pointing uncertainty (mas)

`property dith_std`
Dither pointing uncertainty (mas)

`property exp_nums`
Exposure Numbers associated with each dither position

`gen_random_offsets`(*rand_seed=None*, *rand_seed_base=None*, *first_dith_zero=True*)
Generate randomized pointing offsets for each dither position

`property ndith`
Number of dither positions

`plot_inst_apertures`(*subarrays=False*, *fill=False*, ***kwargs*)
Plot all apertures in this instrument's SIAF.

Other matplotlib standard parameters may be passed in via `**kwargs` to adjust the style of the displayed lines.

Parameters

- **names** (*list of strings*) – A subset of aperture names, if you wish to plot only a subset
- **subarrays** (*bool*) – Plot all the minor subarrays if True, else just plot the "main" apertures
- **label** (*bool*) – Add text labels stating aperture names
- **units** (*str*) – one of 'arcsec', 'arcmin', 'deg'. Only set for 'idl' and 'tel' frames.
- **clear** (*bool*) – Clear plot before plotting (set to false to overplot)

- **show_frame_origin** (*str or list*) – Plot frame origin (goes to plot_frame_origin()): None, 'all', 'det', 'sci', 'raw', 'idl', or a list of these.
- **mark_ref** (*bool*) – Add markers for the reference (V2Ref, V3Ref) point in each apertyre
- **frame** (*str*) – Which coordinate system to plot in: 'tel', 'idl', 'sci', 'det'
- **ax** (*matplotlib.Axes*) – Desired destination axes to plot into (If None, current axes are inferred from pyplot.)
- **fill** (*bool*) – Whether to color fill the aperture
- **fill_color** (*str*) – Fill color
- **fill_alpha** (*float*) – alpha parameter for filled aperture
- **color** (*matplotlib-compatible color*) – Color specification for this aperture's outline, passed through to *matplotlib.Axes.plot*

`plot_main_apertures`(*fill=False*, ***kwargs*)

Plot main SIAF telescope apertures.

Other matplotlib standard parameters may be passed in via `**kwargs` to adjust the style of the displayed lines.

Parameters

- **darkbg** (*bool*) – Plotting onto a dark background? Will make white outlines instead of black.
- **detector_channels** (*bool*) – Overplot the detector amplifier channels for all apertures.
- **label** (*bool*) – Add text labels stating aperture names
- **units** (*str*) – one of 'arcsec', 'arcmin', 'deg'.
- **show_frame_origin** (*str or list*) – Plot frame origin (goes to plot_frame_origin()): None, 'all', 'det', 'sci', 'raw', 'idl', or a list of these.
- **mark_ref** (*bool*) – Add markers for the reference (V2Ref, V3Ref) point in each apertyre
- **ax** (*matplotlib.Axes*) – Desired destination axes to plot into (If None, current axes are inferred from pyplot.)
- **fill** (*bool*) – Whether to color fill the aperture
- **fill_color** (*str*) – Fill color
- **fill_alpha** (*float*) – alpha parameter for filled aperture
- **color** (*matplotlib-compatible color*) – Color specification for this aperture's outline, passed through to *matplotlib.Axes.plot*

`plot_obs_aperture`(*fill=False*, ***kwargs*)

Plot observed aperture

Parameters

- **names** (*list of strings*) – A subset of aperture names, if you wish to plot only a subset
- **subarrays** (*bool*) – Plot all the minor subarrays if True, else just plot the "main" apertures
- **label** (*bool*) – Add text labels stating aperture names
- **units** (*str*) – one of 'arcsec', 'arcmin', 'deg'. Only set for 'idl' and 'tel' frames.
- **clear** (*bool*) – Clear plot before plotting (set to false to overplot)

- **show_frame_origin** (*str or list*) – Plot frame origin (goes to plot_frame_origin()): None, 'all', 'det', 'sci', 'raw', 'idl', or a list of these.
- **mark_ref** (*bool*) – Add markers for the reference (V2Ref, V3Ref) point in each apertyre
- **frame** (*str*) – Which coordinate system to plot in: 'tel', 'idl', 'sci', 'det', 'sky'
- **ax** (*matplotlib.Axes*) – Desired destination axes to plot into (If None, current axes are inferred from pyplot.)
- **fill** (*bool*) – Whether to color fill the aperture
- **fill_color** (*str*) – Fill color
- **fill_alpha** (*float*) – alpha parameter for filled aperture
- **color** (*matplotlib-compatible color*) – Color specification for this aperture's outline, passed through to *matplotlib.Axes.plot*

`plot_ref_aperture`(*fill=False*, ***kwargs*)
Plot reference aperture

Parameters

- **names** (*list of strings*) – A subset of aperture names, if you wish to plot only a subset
- **subarrays** (*bool*) – Plot all the minor subarrays if True, else just plot the "main" apertures
- **label** (*bool*) – Add text labels stating aperture names
- **units** (*str*) – one of 'arcsec', 'arcmin', 'deg'. Only set for 'idl' and 'tel' frames.
- **clear** (*bool*) – Clear plot before plotting (set to false to overplot)
- **show_frame_origin** (*str or list*) – Plot frame origin (goes to plot_frame_origin()): None, 'all', 'det', 'sci', 'raw', 'idl', or a list of these.
- **mark_ref** (*bool*) – Add markers for the reference (V2Ref, V3Ref) point in each apertyre
- **frame** (*str*) – Which coordinate system to plot in: 'tel', 'idl', 'sci', 'det', 'sky'.
- **ax** (*matplotlib.Axes*) – Desired destination axes to plot into (If None, current axes are inferred from pyplot.)
- **fill** (*bool*) – Whether to color fill the aperture
- **fill_color** (*str*) – Fill color
- **fill_alpha** (*float*) – alpha parameter for filled aperture
- **color** (*matplotlib-compatible color*) – Color specification for this aperture's outline, passed through to *matplotlib.Axes.plot*

`radec_to_frame`(*coord_objs*, *frame_out='tel'*, *idl_offsets=None*)
RA/Dec to aperture coordinate frame

Convert a series of RA/Dec positions to desired telescope SIAF coordinate frame within the observed aperture. Will return a list of SIAF coordinates for all objects at each position.

Parameters

- **coord_objs** (*tuple*) – (RA, Dec) positions (deg), where RA and Dec are numpy arrays.
- **frame_out** (*str*) – One of 'tel' (arcsec), 'sci' (pixels), or 'det' (pixels).
- **idl_offsets** (*None or list of 2-element array*) – Option to specify custom offset locations. Normally this is set to None, and we return RA/Dec for all telescope point positions defined

in *self.position_offsets_act*. However, we can specify offsets here (in 'idl') coordinates if you're only interested in a single position or want a custom location.

webbpsf_ext.image_manip

Functions

convolve_image(hdul_sci_image, hdul_psfs[, ...])	Convolve image with various PSFs
crop_zero_rows_cols(image[, symmetric, ...])	Crop off rows and columns that are all zeros.
distort_image(hdulist_or_filename[, ext, ...])	Distort an image
fourier_imshift(image, xshift, yshift[, ...])	Fourier shift image
frebin(image[, dimensions, scale, total])	Fractional rebin
fshift(inarr[, delx, dely, pad, cval, interp])	Fractional image shift
image_rescale(HDUlist_or_filename, pixscale_out)	Rescale image flux
make_disk_image(inst, disk_params[, ...])	Rescale disk model flux to desired pixel scale and distance.
model_to_hdulist(args_model, sp_star, bandpass)	HDUList from model FITS file.
pad_or_cut_to_size(array, new_shape[, ...])	Resize an array to a new shape by either padding with zeros or trimming off rows and/or columns.
rotate_offset(data, angle[, cen, cval, ...])	Rotate and offset an array.
rotate_shift_image(hdul[, index, angle, ...])	Rotate/Shift image

webbpsf_ext.image_manip.convolve_image

webbpsf_ext.image_manip.**convolve_image**(*hdul_sci_image*, *hdul_psfs*, *return_hdul=False*, *output_sampling=None*, *crop_zeros=True*)

Convolve image with various PSFs

Takes an extended image, breaks it up into subsections, then convolves each subsection with the nearest neighbor PSF. The subsection sizes and locations are determined from PSF 'sci' positions.

Parameters

- **hdul_sci_image** (*HDUList*) –

  Image to convolve. Requires header info of:

  - APERNAME : SIAF aperture that images is placed in
  - PIXELSCL : Pixel scale of image (arcsec/pixel)
  - OSAMP : Oversampling relative to detector pixels
  - CFRAME : Coordinate frame of image ('sci', 'tel', 'idl', 'det')
  - XCEN : Image x-position corresponding to aperture reference location
  - YCEN : Image y-position corresponding to aperture reference location
  - XIND_REF, YIND_REF : Alternative for (XCEN, YCEN)

- **hdul_psfs** (*HDUList*) – Multi-extension FITS. Each HDU element is a different PSF for some location within some field of view. Must have same pixel scale as hdul_sci_image.

Keyword Arguments

- **return_hdul** (*bool*) – Return as an HDUList, otherwise return as an image.

- **output_sampling** (*None or int*) – Sampling output relative to detector. If None, then return same sampling as input image.
- **crop_zeros** (*bool*) – For large images that are zero-padded, this option will first crop off the extraneous zeros (but accounting for PSF size to not tuncate resulting convolution at edges), then place the convolved subarray image back into a full frame of zeros. This process can improve speeds by a factor of a few, with no resulting differences. Should always be set to True; only provided as an option for debugging purposes.

webbpsf_ext.image_manip.crop_zero_rows_cols

webbpsf_ext.image_manip.**crop_zero_rows_cols**(*image*, *symmetric=True*, *return_indices=False*)

Crop off rows and columns that are all zeros.

webbpsf_ext.image_manip.distort_image

webbpsf_ext.image_manip.**distort_image**(*hdulist_or_filename*, *ext=0*, *to_frame='sci'*, *fill_value=0*, *xnew_coords=None*, *ynew_coords=None*, *return_coords=False*, *aper=None*, *sci_cen=None*, *pixelscale=None*, *oversamp=None*)

Distort an image

Apply SIAF instrument distortion to an image that is assumed to be in its ideal coordinates. The header information should contain the relevant SIAF point information, such as SI instrument, aperture name, pixel scale, detector oversampling, and detector position ('sci' coords).

This function then transforms the image to the new coordinate system using scipy's RegularGridInterpolator (linear interpolation).

Parameters

- **hdulist_or_filename** (*str or HDUList*) – A PSF from WebbPSF, either as an HDUlist object or as a filename
- **ext** (*int*) – Extension of HDUList to perform distortion on.
- **fill_value** (*float or None*) – Value used to fill in any blank space by the skewed PSF. Default = 0. If set to None, values outside the domain are extrapolated.
- **to_frame** (*str*) –

  Type of input 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.
- **xnew_coords** (*None or ndarray*) – Array of x-values in new coordinate frame to interpolate onto. Can be a 1-dimensional array of unique values, in which case the final image will be of size (ny_new, nx_new). Or a 2d array that corresponds to full regular grid and has same shape as *ynew_coords* (ny_new, nx_new). If set to None, then final image is same size as input image, and coordinate grid spans the min and max values of siaf_ap.convert(xidl,yidl,'idl',to_frame).

- **ynew_coords** (*None or ndarray*) – Array of y-values in new coordinate frame to interpolate onto. Can be a 1-dimensional array of unique values, in which case the final image will be of size (ny_new, nx_new). Or a 2d array that corresponds to full regular grid and has same shape as *xnew_coords* (ny_new, nx_new). If set to None, then final image is same size as input image, and coordinate grid spans the min and max values of siaf_ap.convert(xidl,yidl,'idl',to_frame).
- **return_coords** (*bool*) – In addition to returning the final image, setting this to True will return the full set of new coordinates. Output will then be (psf_new, xnew, ynew), where all three array have the same shape.
- **aper** (None or `pysiaf.Aperture`) – Option to pass the SIAF aperture if it is already known or specified to save time on generating a new one. If set to None, then automatically determines a new *pysiaf* aperture based on information stored in the header.
- **sci_cen** (*tuple or None*) – Science pixel values associated with center of array. If set to None, then will grab values from DET_X and DET_Y header keywords.
- **pixelscale** (*float or None*) – Pixel scale of input image in arcsec/pixel. If set to None, then will search for PIXELSCL and PIXSCALE keywords in header.
- **oversamp** (*int or None*) – Oversampling of input image relative to native detector pixel scale. If set to None, will search for OSAMP and DET_SAMP keywords.

webbpsf_ext.image_manip.fourier_imshift

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

Shift an image by use of Fourier shift theorem

Parameters

- **image** (*ndarray*) – 2D image or 3D image cube [nz,ny,nx].
- **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

webbpsf_ext.image_manip.frebin

webbpsf_ext.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. Can also input an image cube.

Parameters

- **image** (*ndarray*) – Input image ndarray (1D, 2D). Can also be an image cube assumed to have shape [nz,ny,nx].
- **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

webbpsf_ext.image_manip.fshift

webbpsf_ext.image_manip.**fshift**(*inarr*, *delx=0*, *dely=0*, *pad=False*, *cval=0.0*, *interp='linear'*, ***kwargs*)
Fractional image shift

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

Parameters

- **inarr** (*ndarray*) – 1D, or 2D array to be shifted. Can also be an image cube assume with shape [nz,ny,nx].
- **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.
- **interp** (*str*) – Type of interpolation to use during the sub-pixel shift. Valid values are ‘linear’, ‘cubic’, and ‘quintic’.

Returns *ndarray* – Shifted image

webbpsf_ext.image_manip.image_rescale

webbpsf_ext.image_manip.**image_rescale**(*HDUlist_or_filename*, *pixscale_out*, *pixscale_in=None*, *dist_in=None*, *dist_out=None*, *cen_star=True*, *shape_out=None*)

Rescale image flux

Scale the flux and rebin an image to some new pixel scale and distance. The object's physical units (AU) are assumed to be constant, so the total angular size changes if the distance to the object changes.

IT IS RECOMMENDED THAT UNITS BE IN PHOTONS/SEC/PIXEL (not mJy/arcsec)

Parameters

- **HDUlist_or_filename** (*HDUList or str*) – Input either an HDUList or file name.
- **pixscale_out** (*float*) – Desired pixel scale (asec/pix) of returned image. Will be saved in header info.

Keyword Arguments

- **pixscale_in** (*float or None*) – Input image pixel scale. If None, then tries to grab info from the header.
- **dist_in** (*float*) – Input distance (parsec) of original object. If not set, then we look for the header keywords 'DISTANCE' or 'DIST'.
- **dist_out** (*float*) – Output distance (parsec) of object in image. Will be saved in header info. If not set, then assumed to be same as input distance.
- **cen_star** (*bool*) – Is the star placed in the central pixel? If so, then the stellar flux is assumed to be a single pixel that is equal to the maximum flux in the image. Rather than rebinning that pixel, the total flux is pulled out and re-added to the central pixel of the final image.
- **shape_out** (*tuple, int, or None*) – Desired size for the output array (ny,nx). If a single value, then will create a 2-element tuple of the same value.

Returns *HDUlist of the new image.*

webbpsf_ext.image_manip.make_disk_image

webbpsf_ext.image_manip.**make_disk_image**(*inst*, *disk_params*, *sp_star=None*, *pixscale_out=None*, *dist_out=None*, *shape_out=None*)

Rescale disk model flux to desired pixel scale and distance. If instrument bandpass is different from disk model, scales flux assuming a grey scattering model.

Returns image flux values in photons/sec.

Parameters

- **inst** (*mod::webbpsf_ext instrument class*) – E.g. NIRCam_ext, MIRI_ext classes
- **disk_params** (*dict*) –

  Arguments describing the necessary model information:

  – 'file' : Path to model file or an HDUList.

  – 'pixscale' : Pixel scale (arcsec/pixel).

  – 'dist' : Assumed model distance in parsecs.

  – 'wavelength' : Wavelength of observation in microns.

- 'units' : String of assumed flux units (ie., MJy/arcsec^2 or muJy/pixel)
- 'cen_star' : True/False. Is a star already placed in the central pixel?

Will Convert from [M,m,u,n]Jy/[arcsec^2,pixel] to photons/sec/pixel

Keyword Arguments

- **sp_star** (pysynphot.spectrum) – A pysynphot spectrum of central star. Used to adjust observed photon flux if filter differs from model input
- **pixscale_out** (*float*) – Desired pixelscale of returned image. If None, then use instrument's oversampled pixel scale.
- **dist_out** (*float*) – Distance to place disk at. Flux is scaled appropriately relative to the input distance specified in *disk_params*.
- **shape_out** (*tuple, int, or None*) – Desired size for the output array (ny,nx). If a single value, then will create a 2-element tuple of the same value.

webbpsf_ext.image_manip.model_to_hdulist

webbpsf_ext.image_manip.**model_to_hdulist**(*args_model*, *sp_star*, *bandpass*)

HDUList from model FITS file.

Convert disk model to an HDUList with units of photons/sec/pixel. If observed filter is different than input filter, we assume that the disk has a flat scattering, meaning it scales with stellar continuum. Pixel sizes and distances are left unchanged, and stored in header.

Parameters

- **args_model - tuple** –

  Arguments describing the necessary model information:

  - fname : Name of model file or an HDUList
  - scale0 : Pixel scale (in arcsec/pixel)
  - dist0 : Assumed model distance
  - wave_um : Wavelength of observation
  - units0 : Assumed flux units (e.g., MJy/arcsec^2 or muJy/pixel)
- **sp_star** (pysynphot.spectrum) – A pysynphot spectrum of central star. Used to adjust observed photon flux if filter differs from model input
- **bandpass** (pysynphot.obsbandpass) – Output *Pysynphot* bandpass from instrument class. This corresponds to the flux at the entrance pupil for the particular filter.

webbpsf_ext.image_manip.pad_or_cut_to_size

webbpsf_ext.image_manip.**pad_or_cut_to_size**(*array*, *new_shape*, *fill_val=0.0*, *offset_vals=None*, *shift_func=<function fshift>*, ***kwargs*)

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

Parameters

- **array** (*ndarray*) – A 1D, 2D, or 3D array. If 3D, then taken to be a stack of images that are cropped or expanded in the same fashion.

- **new_shape** (*tuple*) – 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
- **offset_vals** (*tuple*) – Option to perform image shift in the (xpix) direction for 1D, or (ypix,xpix) direction for 2D/3D prior to cropping or expansion.
- **shift_func** (*function*) – Function to use for shifting. Usually either *fshift* or *fourier_imshift*.

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

webbpsf_ext.image_manip.rotate_offset

webbpsf_ext.image_manip.**rotate_offset**(*data*, *angle*, *cen=None*, *cval=0.0*, *order=1*, *reshape=True*, *recenter=True*, *shift_func=<function fshift>*, ***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. Default rotation is clockwise direction.

Parameters

- **data** (*ndarray*) – The input array.
- **angle** (*float*) – The rotation angle in degrees (rotates in clockwise 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?
- **shift_func** (*function*) – Function to use for shifting. Usually either *fshift* or *fourier_imshift*.

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 pre-filtered 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.

webbpsf_ext.image_manip.rotate_shift_image

webbpsf_ext.image_manip.**rotate_shift_image**(*hdul*, *index=0*, *angle=0*, *delx_asec=0*, *dely_asec=0*, *shift_func=<function fshift>*, ***kwargs*)

Rotate/Shift image

Rotate then offset image by some amount. Positive angles rotate the image counter-clockwise.

Parameters

- **hdul** (*HDUList*) – Input HDUList
- **index** (*int*) – Specify HDU index, usually 0
- **angle** (*float*) – Rotate entire scene by some angle. Positive angles rotate counter-clockwise.
- **delx_asec** (*float*) – Offset in x direction (specified in arcsec). Pixel scale should be included in header keyword 'PIXELSCL'.
- **dely_asec** (*float*) – Offset in x direction (specified in arcsec). Pixel scale should be included in header keyword 'PIXELSCL'.
- **shift_func** (*function*) – Function to use for shifting. Usually either *fshift* or *fourier_imshift*.

Keyword Arguments

- **order** (*int, optional*) – The order of the spline interpolation, default is 3. The order has to be in the range 0-5.
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the input array is extended beyond its boundaries. Default is 'constant'. Behavior for each valid value is as follows:

  'reflect' (*d c b a | a b c d | d c b a*) The input is extended by reflecting about the edge of the last pixel.

  'constant' (*k k k k | a b c d | k k k k*) The input is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

  'nearest' (*a a a a | a b c d | d d d d*) The input is extended by replicating the last pixel.

  'mirror' (*d c b | a b c d | c b a*) The input is extended by reflecting about the center of the last pixel.

  'wrap' (*a b c d | a b c d | a b c d*) The input is extended by wrapping around to the opposite edge.

- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0.
- **prefilter** (*bool, optional*) – Determines if the input array is prefiltered with *spline_filter* before interpolation. The default is True, which will create a temporary *float64* array of filtered values if *order > 1*. If setting this to False, the output will be slightly blurred if *order > 1*, unless the input is prefiltered, i.e. it is the result of calling *spline_filter* on the original input.

webbpsf_ext.logging_utils

Functions

restart_logging([verbose])	Restart Logging
setup_logging([level, filename, verbose])	Setup Logging

webbpsf_ext.logging_utils.restart_logging

webbpsf_ext.logging_utils.**restart_logging**(*verbose=True*)
Restart Logging

Restart logging using the same settings as those currently stored in conf.logging_level. This function was shamelessly stolen from WebbPSF utils.py.

Parameters **verbose** (*boolean*) – Should this function print the new logging targets to standard output?

webbpsf_ext.logging_utils.setup_logging

webbpsf_ext.logging_utils.**setup_logging**(*level='INFO'*, *filename=None*, *verbose=True*)
Setup Logging

Allows selection of logging detail and output locations (screen and/or file). Shamelessly stolen from WebbPSF utils.py.

This is a convenience wrapper to Python's built-in logging package. By default, this sets up log messages to be written to the screen, but the user can also request logging to a file.

For more advanced log handling, see the Python logging module's own documentation.

Parameters

- **level** (*str*) – Name of log output to show. Defaults to 'INFO', set to 'DEBUG' for more extensive messages, or to 'WARN' or 'ERROR' for fewer.
- **filename** (*str, optional*) – Filename to write the log output to. If not set, output will just be displayed on screen. (Default: None)

Examples

```
>>> webbpsf_ext.setup_logging(filename='webbpsf_ext_log.txt')
```

This will save all log messages to 'webbpsf_ext_log.txt' in the current directory.

```
>>> webbpsf_ext.setup_logging(level='WARN')
```

This will show only WARNING or ERROR messages on screen, and not save any logs to files at all (since the filename argument is None)

Classes

FilterLevelRange(min_level, max_level)	

webbpsf_ext.logging_utils.FilterLevelRange

class webbpsf_ext.logging_utils.**FilterLevelRange**(*min_level*, *max_level*)

Bases: object

__init__(*min_level*, *max_level*)

Methods

__init__(min_level, max_level)	
filter(record)	

webbpsf_ext.maths

Functions

binned_statistic(x, values[, func, bins])	Binned statistic
find_closest(A, B)	Find closest indices
fit_bootstrap(pinit, datax, datay, function)	Bootstrap fitting routine
hist_indices(values[, bins, return_more])	Histogram indices
jl_poly(xvals, coeff[, dim_reorder, ...])	Evaluate polynomial
jl_poly_fit(x, yvals[, deg, QR, robust_fit, ...])	Fast polynomial fitting
radial_std(im_diff[, pixscale, oversample, ...])	Generate contrast curve of PSF difference

webbpsf_ext.maths.binned_statistic

webbpsf_ext.maths.**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)
```

webbpsf_ext.maths.find_closest

webbpsf_ext.maths.**find_closest**(*A*, *B*)

Find closest indices

Given two arrays, A and B, find the indices in B whose values are closest to those in A. Returns an array with size equal to A.

This is much much faster than something like, especially for large arrays: *np.argmin(np.abs(A - B[:, np.newaxis]), axis=0)*

webbpsf_ext.maths.fit_bootstrap

webbpsf_ext.maths.**fit_bootstrap**(*pinit*, *datax*, *datay*, *function*, *yerr_systematic=0.0*, *nrand=1000*, *return_more=False*)

Bootstrap fitting routine

Bootstrap fitting algorithm to determine the uncertainties on the fit parameters.

Parameters

- **pinit** (*ndarray*) – Initial guess for parameters to fit
- **datax, datay** (*ndarray*) – X and Y values of data to be fit
- **function** (*func*) – Model function
- **yerr_systematic** (*float or array_like of floats*) – Systematic uncertainites contributing to additional error in data. This is treated as independent Normal error on each data point. Can have unique values for each data point. If 0, then we just use the standard deviation of the residuals to randomize the data.
- **nrand** (*int*) – Number of random data sets to generate and fit.
- **return_more** (*bool*) – If true, then also return the full set of fit parameters for the randomized data to perform a more thorough analysis of the distribution. Otherwise, just reaturn the mean and standard deviations.

webbpsf_ext.maths.hist_indices

webbpsf_ext.maths.**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)
```

webbpsf_ext.maths.jl_poly

webbpsf_ext.maths.**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).

webbpsf_ext.maths.jl_poly_fit

webbpsf_ext.maths.**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 normally 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)
```

webbpsf_ext.maths.radial_std

webbpsf_ext.maths.**radial_std**(*im_diff*, *pixscale=None*, *oversample=None*, *supersample=False*, *func=<function std>*)

Generate contrast curve of PSF difference

Find the standard deviation within fixed radial bins of a differenced image. Returns two arrays representing the 1-sigma contrast curve at given distances.

Parameters

- **im_diff** (*ndarray*) – Differenced image of two PSFs, for instance.
- **Keywords**
- **========**
- **pixscale** (*float*) – Pixel scale of the input image
- **oversample** (*int*) – Is the input image oversampled compared to detector? If set, then the binsize will be pixscale*oversample (if supersample=False).
- **supersample** (*bool*) – If set, then oversampled data will have a binsize of pixscale, otherwise the binsize is pixscale*oversample.
- **func_std** (*func*) – The function to use for calculating the radial standard deviation.

webbpsf_ext.opds

Functions

OPDFile_to_HDUList(file[, slice])	Make a picklable HDUList for ingesting into multiprocessor WebbPSF helper function.
plot_im(im, fig, ax[, vlim, add_cbar, ...])	Plot single image on some axes
plot_opd(hdul[, index, opd0, vlim1, vlim2])	Plot OPDs images (full or delta)
slew_time(dist_asec)	Given a slew distance (arcsec), calculate telescope slew time.

webbpsf_ext.opds.OPDFile_to_HDUList

webbpsf_ext.opds.**OPDFile_to_HDUList**(*file*, *slice=0*)
Make a picklable HDUList for ingesting into multiproccessor WebbPSF helper function.

webbpsf_ext.opds.plot_im

webbpsf_ext.opds.**plot_im**(*im*, *fig*, *ax*, *vlim=None*, *add_cbar=True*, *return_ax=False*, *extent=None*, *cmap='RdBu_r'*)
Plot single image on some axes

webbpsf_ext.opds.plot_opd

webbpsf_ext.opds.**plot_opd**(*hdul*, *index=1*, *opd0=None*, *vlim1=None*, *vlim2=None*)
Plot OPDs images (full or delta)

webbpsf_ext.opds.slew_time

webbpsf_ext.opds.**slew_time**(*dist_asec*)
Given a slew distance (arcsec), calculate telescope slew time. Output is sec. Data comes from JDOX website:

https://jwst-docs.stsci.edu/jppom/visit-overheads-timing-model/slew-times.

Classes

OTE_WFE_Drift_Model(**kwargs)	OPD subclass for calculating OPD drift values over time.

webbpsf_ext.opds.OTE_WFE_Drift_Model

class webbpsf_ext.opds.**OTE_WFE_Drift_Model**(***kwargs*)
Bases: webbpsf.opds.OTE_Linear_Model_WSS

OPD subclass for calculating OPD drift values over time.

__init__(***kwargs*)

Parameters

- **opdfile** (*str or fits.HDUList*) – FITS file to load an OPD from. The OPD data must be specified in microns.
- **opd_index** (*int, optional*) – Index of a datacube to load OPD from, if the selected extension contains a datacube.
- **transmission** (*str or None*) – FITS file for pupil mask, with throughput from 0-1. If not explicitly provided, will be inferred from wherever is nonzero in the OPD file.
- **segment_mask_file** (*str*) – FITS file for pupil mask, with throughput from 0-1. If not explicitly provided, will use JWpupil_segments_RevW_npix1024.fits, or equivalent for other value of npix

- **zero** (*bool*) – Set an OPD to precisely zero everywhere.
- **rm_ptt** (*bool*) – Remove piston, tip, and tilt? This is mostly for visualizing the higher order parts of the LOM. Default: False.
- **v2v3** (*tuple of 2 astropy.Quantities*) – Tuple giving V2,v3 coordinates as quantities, typically in arcminutes, or None to default to the master chief ray location between the two NIRCam modules.
- **include_nominal_field_dependence** (*bool*) – Include the Zernike polynomial model for OTE field dependence for the nominal OTE. Note, if OPD is None, then this will be ignored and the nominal field dependence will be disabled.
- **control_point_fieldpoint** (*str*) – A parameter used in the field dependence model for a misaligned secondary mirror. Name of the field point where the OTE MIMF control point is located, on instrument defined by "control_point_instr". Default: 'nrca3_full'. The OTE control point is the field point to which the OTE has been aligned and defines the field angles for the field-dependent SM pose aberrations.
- **npix** (*int*) – Size of OPD: npix x npix

Methods

`__init__`(**kwargs)

Parameters

- **opdfile** (*str or fits.HDUList*) -- FITS file to load an OPD from. The OPD data must be specified in microns.

`apply_frill_drift`([amplitude, random, case, ...])	Apply model of segment PTT motions for the frill-induced drift.
`apply_iec_drift`([amplitude, random, case, ...])	Apply model of segment PTT motions for the drift seen at OTIS induced by the IEC (Instrument Electronics Compartment) heater resistors.
`as_fits`([include_pupil])	Return the OPD as a fits.HDUList object
`calc_rms`(arr[, segname])	Calculate RMS of input images
`copy`()	Make a copy of a wavefront object
`display`([nrows, row, what, crosshairs, ax, ...])	Display plots showing an optic's transmission and OPD.
`display_opd`([ax, labelsegs, vmax, colorbar, ...])	Draw on screen the perturbed OPD
`estimated_Strehl`(wavelength[, verbose])	Compute an estimated Strehl given a wavelength in meters
`evolve_dopd`(delta_time, slew_angles[, case, ...])	Evolve the delta OPD with multiple slews
`gen_delta_opds`(delta_time[, start_angle, ...])	Create series of delta OPDs
`gen_frill_drift`(delta_time[, start_angle, ...])	Frill WFE drift scaling
`gen_iec_series`(delta_time[, amplitude, ...])	Create a series of IEC WFE scale factors
`gen_thermal_drift`(delta_time[, start_angle, ...])	Thermal WFE drift scaling
`get_opd`(wave)	Return the optical path difference, given a wavelength.
`get_phasor`(wave)	Compute a complex phasor from an OPD, given a wavelength.

continues on next page

Table 55 – continued from previous page

get_transmission(wave)	Return the electric field amplitude transmission, given a wavelength.
header_keywords()	Return info we would like to save in FITS header of output PSFs
interp_dopds(delta_time, dopds, dt_new[, ...])	Interpolate an array of delta OPDs
label_seg(segment[, ax, show_axes, color, ...])	Annotate a plot with a text label for a particular segment
move_global_zernikes(zvector[, unit, ...])	Add one or more aberrations specified arbitrarily as Zernike polynomials.
move_seg_global(segment[, xtilt, ytilt, ...])	Move a segment in pose and/or ROC, using PM global V coordinates..
move_seg_local(segment[, xtilt, ytilt, ...])	Move a segment in pose and/or ROC, using segment-local control coordinates.
move_sm_local([xtilt, ytilt, rot_unit, ...])	Move the secondary mirror in pose, using segment-local control coordinates.
move_sur(sur_file[, group, verbose, reverse])	Move using a JWST Segment Update Request file
opds_as_hdul(delta_time, slew_angles[, ...])	Convert series of delta OPDS to HDUList
powerspectrum([max_cycles, sampling, vmax, ...])	Compute the spatial power spectrum of an aberrated wavefront.
print_state()	
ptv([segment])	return peak to valley WFE in nanometers
reset([verbose])	Reset an OPD to the state it was loaded from disk.
rms([segment])	Return RMS WFE in nanometers
slew_pos_averages(delta_time, slew_angles[, ...])	Get averages at each slew position
slew_scaling(start_angle, end_angle)	WFE scaling due to slew angle
thermal_slew(delta_time[, start_angle, ...])	Update the OPD based on presence of a pitch angle change between observations.
update_opd([display, verbose])	Update the OPD based on the current linear model values.
writeto(outname[, overwrite])	Write OPD to a FITS file on disk
zern_seg(segment[, vmax, unit])	Show the Zernike terms applied to a given segment
zero([zero_original])	Reset an OPD to precisely zero everywhere.

Attributes

pupil_diam	Diameter of the pupil (if this is a pupil plane optic)
shape	Return shape of the OpticalElement, as a tuple

apply_frill_drift(*amplitude=None*, *random=False*, *case='BOL'*, *delay_update=False*)

Apply model of segment PTT motions for the frill-induced drift.

This is additive with other WFE terms.

Parameters

- **amplitude** (*float*) – Amplitude of drift in nm rms to apply
- **random** (*bool*) – if True, choose a random amplitude from within the expected range for either the BOL or EOL cases. The assumed model is a uniform distribution between 0 and a maximum amplitude of 8.6 or 18.4 nm rms respectively.

- **case** (*string*) – either "BOL" for current best estimate at beginning of life, or "EOL" for more conservative prediction at end of life. Only relevant if random=True.
- **delay_update** (*bool*) – hold off on computing the WFE change? This is useful for computational efficiency if you're making many changes at once.

`apply_iec_drift`(*amplitude=None*, *random=False*, *case='BOL'*, *delay_update=False*)

Apply model of segment PTT motions for the drift seen at OTIS induced by the IEC (Instrument Electronics Compartment) heater resistors. This effect was in part due to non-flight-like ground support equipment mountings, and is not expected in flight at the same levels it was seen at JSC. We model it anyway, at an amplitude consistent with upper limits for flight.

This is additive with other WFE terms.

Parameters

- **amplitude** (*float*) – Amplitude of drift in nm rms to apply
- **random** (*bool*) – if True, choose a random amplitude from within the expected range for either the BOL or EOL cases. The assumed model is a sinusoidal drift between 0 and 3.5 nm, i.e. a random variate from the arcsine distribution times 3.5.
- **case** (*string*) – either "BOL" for current best estimate at beginning of life, or "EOL" for more conservative prediction at end of life. Only relevant if random=True. (Note, for IEC drift the amplitude is the same regardless.)
- **delay_update** (*bool*) – hold off on computing the WFE change? This is useful for computational efficiency if you're making many changes at once.

`as_fits`(*include_pupil=True*)

Return the OPD as a fits.HDUList object

Parameters **include_pupil** (*bool*) – Include the pupil mask as a FITS extension?

`calc_rms`(*arr*, *segname=None*)

Calculate RMS of input images

`copy`()

Make a copy of a wavefront object

`display`(*nrows=1*, *row=1*, *what='intensity'*, *crosshairs=False*, *ax=None*, *colorbar=True*, *colorbar_orientation=None*, *title=None*, *opd_vmax=<Quantity 5.e-07 m>*, *wavelength=<Quantity 1.e-06 m>*, *npix=512*, *grid_size=None*)

Display plots showing an optic's transmission and OPD.

Parameters

- **what** (*str*) – What to display: 'intensity', 'amplitude', 'phase', 'opd', or 'both' (meaning intensity and OPD in two subplots)
- **ax** (*matplotlib.Axes instance*) – Axes to display into
- **nrows, row** (*integers*) – number of rows and row index for subplot display
- **crosshairs** (*bool*) – Display crosshairs indicating the center?
- **colorbar** (*bool*) – Show colorbar?
- **colorbar_orientation** (*bool*) – Desired orientation, horizontal or vertical? Default is horizontal if only 1 row of plots, else vertical
- **opd_vmax** (*float*) – Max absolute value for OPD image display, in meters.
- **title** (*string*) – Plot label

- **wavelength** (*float, default 1 micron*) – For optics with wavelength-dependent behavior, evaluate at this wavelength for display.
- **npix** (*integer*) – For optics without a fixed pixel sampling, evaluate onto this many pixels for display.
- **grid_size** (*float*) – For optics without a fixed pixel sampling, evaluate onto this large a spatial or angular extent for display. Specify in units of arcsec for image plane optics, meters for all other optics. If unspecified, a default value will be chosen instead, possibly from the ._default_display_size attribute, if present.

`display_opd`(*ax=None*, *labelsegs=True*, *vmax=150.0*, *colorbar=True*, *clear=False*, *title=None*, *unit='nm'*, *pupil_orientation='entrance_pupil'*, *cbpad=None*, *colorbar_orientation='vertical'*, *show_axes=False*, *show_rms=True*, *show_v2v3=False*, *cmap=None*)

Draw on screen the perturbed OPD

Parameters

- **ax** (*matplotlib.Axes*) – axes instance to display into.
- **labelsegs** (*bool*) – draw segment name labels on each segment? default True.
- **show_axes** (*bool*) – Draw local control axes per each segment
- **show_rms** (*bool*) – Annotate the RMS wavefront value
- **show_v2v3** – Draw the observatory V2V3 coordinate axes
- **pupil_orientation** (*string*) – either 'entrance_pupil' or 'exit_pupil', for which orientation we should display the OPD in.
- **clear** (*bool*) – Clear plot window before display? default true
- **unit** (*str*) – Unit for WFE. default is 'nm'

`estimated_Strehl`(*wavelength*, *verbose=True*)

Compute an estimated Strehl given a wavelength in meters

Parameters

- **wavelength** (*float*) – in meters
- **verbose** (*bool*) – should I print out an informative message?

`evolve_dopd`(*delta_time*, *slew_angles*, *case='BOL'*, *return_wfe_amps=True*, *return_dopd_fin=True*, *do_thermal=True*, *do_frill=True*, *do_iec=True*, ***kwargs*)

Evolve the delta OPD with multiple slews

Input an array of *delta_time* and *slew_angles* to return the evolution of a delta_OPD image. Option to return the various WFE components, including OTE backplane (thermal), frill tensioning, and IEC heater switching.

Parameters

- **delta_time** (*astropy.units quantity object*) – An array of times assuming astropy units.
- **slew_angles** (*ndarray*) – The sun pitch angles, in degrees between -5 and +45.
- **case** (*string*) – Either "BOL" for current best estimate at beginning of life, or "EOL" for more conservative prediction at end of life.
- **do_thermal** (*bool*) – Include thermal slew component? Mostly for debugging purposes.
- **do_frill** (*bool*) – Include frill component? Mostly for debugging purposes.

- **do_iec** (*bool*) – Include IEC component? Good to exclude if calling this function repeatedly for evolution of multiple slews, then add IEC later.
- **return_wfe_amps** (*bool*) – Return a dictionary that provides the RMS WFE (nm) of each component at each time step.
- **return_dopd_fin** (*bool*) – Option to exclude calculating final delta OPD in case we only want the final RMS WFE dictionary.

Keyword Arguments

- **amplitude** (*float*) – Full amplitude of IEC arcsine distribution. Values will range from -0.5*amplitude to +0.5*amplitude.
- **period** (*float*) – Period in minutes of IEC oscillations. Usually 3-5 minutes.
- **random_seed** (*int*) – Random seed to pass to IEC generation.

gen_delta_opds(*delta_time*, *start_angle=- 5*, *end_angle=45*, *do_thermal=True*, *do_frill=True*, *do_iec=True*, *case='BOL'*, *return_wfe_amps=True*, *return_dopd_fin=True*, *random_seed=None*, ***kwargs*)

Create series of delta OPDs

Generate a series of delta OPDS, the result of which is a combination of thermal, frill, and IEC effects. The thermal and frill values are dependent on time, start/end slew angles, and case ('BOL' or 'EOL'). Delta OPD contributions from the IEC heater switching are treated as random state switches assuming an arcsine distribution.

Parameters

- **delta_time** (*astropy.units quantity object*) – An array of times assuming astropy units.
- **start_angle** (*float*) – The starting sun pitch angle, in degrees between -5 and +45.
- **end_angle** (*float*) – The ending sun pitch angle, in degrees between -5 and +45.
- **case** (*string*) – Either "BOL" for current best estimate at beginning of life, or "EOL" for more conservative prediction at end of life.
- **do_thermal** (*bool*) – Include thermal slew component? Mostly for debugging purposes.
- **do_frill** (*bool*) – Include frill component? Mostly for debugging purposes.
- **do_iec** (*bool*) – Include IEC component? Good to exclude if calling this function repeatedly for evolution of multiple slews, then add IEC later.
- **return_wfe_amps** (*bool*) – Return a dictionary that provides the RMS WFE (nm) of each component at each time step.
- **return_dopd_fin** (*bool*) – Option to exclude calculating final delta OPD in case we only want the final RMS WFE dictionary.
- **random_seed** (*int*) – Random seed to pass to IEC generation.

gen_frill_drift(*delta_time*, *start_angle=- 5*, *end_angle=45*, *case='BOL'*)

Frill WFE drift scaling

Function to determine the factor to scale the delta OPD associated with frill tensioning. Returns the RMS WFE (nm) depending on time and slew angles.

Parameters

- **delta_time** (*astropy.units quantity object*) – The time since a slew occurred.
- **start_angle** (*float*) – The starting sun pitch angle, in degrees between -5 and +45

- **end_angle** (*float*) – The ending sun pitch angle, in degrees between -5 and +45
- **case** (*string*) – either "BOL" for current best estimate at beginning of life, or "EOL" for more conservative prediction at end of life. The amplitude of the frill drift is roughly 2x lower for BOL (8.6 nm after 2 days) versus EOL (18.4 nm after 2 days).

gen_iec_series(*delta_time*, *amplitude=3.5*, *period=5.0*, *interp_kind='linear'*, *random_seed=None*)
Create a series of IEC WFE scale factors

Create a series of random IEC heater state changes based on arcsine distribution.

Parameters **delta_time** (*astropy.units quantity object array*) – Time series of atropy units to interpolate IEC amplitudes

Keyword Arguments

- **amplitude** (*float*) – Full amplitude of arcsine distribution. Values will range from -0.5*amplitude to +0.5*amplitude.
- **period** (*float*) – Period in minutes of IEC oscillations. Usually 3-5 minutes.
- **random_seed** (*int*) – Provide a random seed value between 0 and (2**32)-1 to generate reproducible random values.
- **interp_kind** (*str or int*) – Specifies the kind of interpolation as a string ('linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'previous', 'next', where 'zero', 'slinear', 'quadratic' and 'cubic' refer to a spline interpolation of zeroth, first, second or third order; 'previous' and 'next' simply return the previous or next value of the point) or as an integer specifying the order of the spline interpolator to use. Default is 'linear'.

gen_thermal_drift(*delta_time*, *start_angle=- 5*, *end_angle=45*, *case='BOL'*)
Thermal WFE drift scaling

Function to determine the factor to scale the delta OPD associated with OTE backplane thermal distortion. Returns the RMS WFE (nm) depending on time and slew angles.

Parameters

- **delta_time** (*astropy.units quantity object*) – The time since a slew occurred.
- **start_angle** (*float*) – The starting sun pitch angle, in degrees between -5 and +45
- **end_angle** (*float*) – The ending sun pitch angle, in degrees between -5 and +45
- **case** (*string*) – either "BOL" for current best estimate at beginning of life, or "EOL" for more conservative prediction at end of life. The amplitude of the frill drift is roughly 3x lower for BOL (13 nm after 14 days) versus EOL (43 nm after 14 days).

get_opd(*wave*)
Return the optical path difference, given a wavelength.

When the OPD map is defined in terms of wavelength-independent phase, as in the case of the vector apodizing phase plate coronagraph of Snik et al. (Proc. SPIE, 2012), it is converted to optical path difference in meters at the given wavelength for consistency with the rest of POPPY.

Parameters **wave** (*float or obj*) – either a scalar wavelength or a Wavefront object

Returns *ndarray giving OPD in meters*

get_phasor(*wave*)
Compute a complex phasor from an OPD, given a wavelength.

The returned value should be the complex phasor array as appropriate for multiplying by the wavefront amplitude.

Parameters **wave** (*float or obj*) – either a scalar wavelength or a Wavefront object

`get_transmission`(*wave*)
Return the electric field amplitude transmission, given a wavelength.

Parameters **wave** (*float or obj*) – either a scalar wavelength or a Wavefront object

Returns *ndarray giving electric field amplitude transmission between 0 - 1.0*

`header_keywords`()
Return info we would like to save in FITS header of output PSFs

`interp_dopds`(*delta_time*, *dopds*, *dt_new*, *wfe_dict=None*, *interp_kind='linear'*, ***kwargs*)
Interpolate an array of delta OPDs

Perform a linear interpolation on a series of delta OPDS.

Parameters

- **delta_time** (*astropy.units quantity object*) – An array of times assuming astropy units corresponding to each *dopd*.
- **dopds** (*ndarray*) – Array of delta OPD images associated with *delta_time*.
- **dt_new** (*astropy.units quantity object*) – New array to interpolate onto.

Keyword Arguments

- `wfe_dict` (*dict or None*) – If specified, then must provide a dictionary where the values for each keywords are the WFE drift components associated with each *delta_time*. Will then return a dictionary
- `interp_kind` (*str or int*) – Specifies the kind of interpolation as a string ('linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'previous', 'next', where 'zero', 'slinear', 'quadratic' and 'cubic' refer to a spline interpolation of zeroth, first, second or third order; 'previous' and 'next' simply return the previous or next value of the point) or as an integer specifying the order of the spline interpolator to use. Default is 'linear'.

`label_seg`(*segment*, *ax=None*, *show_axes=False*, *color='black'*, *pupil_orientation='entrance_pupil'*)
Annotate a plot with a text label for a particular segment

`move_global_zernikes`(*zvector*, *unit='micron'*, *absolute=False*, *delay_update=False*, *display=False*)
Add one or more aberrations specified arbitrarily as Zernike polynomials. This assumes no particular physics for the mirror motions, and allows adding any arbitrary WFE.

Parameters **zvector** (*list or ndarray*) – Zernike coefficients

Note that the Zernikes are interpreted as being with respect to the *CIRCUMSCRIBING* circle.

`move_seg_global`(*segment*, *xtilt=0.0*, *ytilt=0.0*, *clocking=0.0*, *rot_unit='urad'*, *radial=None*, *xtrans=0.0*, *ytrans=0.0*, *piston=0.0*, *roc=0.0*, *trans_unit='micron'*, *display=False*, *delay_update=False*, *absolute=False*)
Move a segment in pose and/or ROC, using PM global V coordinates..

These motions are converted into the segment-local "Control" coordinate systems, which are distinct for each segment.

Parameters

- **segment** (*str*) – Segment name, e.g. 'A1'. Use 'SM' for the secondary mirror.
- **xtilt, ytilt, clocking** (*floats*) – Tilt angle, in microradians by default. 'xtilt' means tilt *around the X axis*, and similarly for ytilt.

- **radial, ytrans,xtrans** (*floats*) – Displacement distance, in microns by default. Note the 'radial' and 'xtrans', 'ytrans' are redundant and included for convenience; the Ball WAS linear optical model uses radial translation as the control DoF, but physically that maps to either x or y translation depending on whether A, B, or C segment, and the Ball MCS algorithms expect X and Y translations. We support both ways of describing this here.
- **piston** (*float*) – Displacement distance for piston.
- **roc** (*float*) – radius of curvature mechanism adjustment, in microns.
- **trans_unit** (*str*) – Unit for translations. Can be 'micron', 'millimeter','nanometer', 'mm', 'nm', 'um'
- **rot_unit** (*str*) – Unit for rotations. Can be 'urad', 'radian', 'milliradian', 'arcsec', 'arcmin', 'milliarcsec'
- **absolute** (*bool*) – Same meaning as for JWST SURs: if true, move the segment to exactly this position. Otherwise moves are treated as incremental relative moves from the current position.
- **display** (*bool*) – Display after moving?
- **delay_update** (*bool*) – hold off on computing the WFE change? This is useful for computational efficiency if you're moving a whole bunch of segments at once. Incompatible with display=True.

`move_seg_local`(*segment*, *xtilt=0.0*, *ytilt=0.0*, *clocking=0.0*, *rot_unit='urad'*, *radial=None*, *xtrans=None*, *ytrans=None*, *piston=0.0*, *roc=0.0*, *trans_unit='micron'*, *display=False*, *delay_update=False*, *absolute=False*)

Move a segment in pose and/or ROC, using segment-local control coordinates.

These motions are always commanded in the segment-local "Control" coordinate systems, which are distinct for each segment.

Parameters

- **segment** (*str*) – Segment name, e.g. 'A1'.
- **xtilt, ytilt, clocking** (*floats*) – Tilt angle, in microradians by default. 'xtilt' means tilt *around the X axis*, and similarly for ytilt.
- **radial, ytrans,xtrans** (*floats*) – Displacement distance, in microns by default. Note the 'radial' and 'xtrans', 'ytrans' are redundant and included for convenience; the Ball WAS linear optical model uses radial translation as the control DoF, but physically that maps to either x or y translation depending on whether A, B, or C segment, and the Ball MCS algorithms expect X and Y translations. We support both ways of describing this here.
- **piston** (*float*) – Displacement distance for piston.
- **roc** (*float*) – radius of curvature mechanism adjustment, in microns.
- **trans_unit** (*str*) – Unit for translations. Can be 'meter', 'micron', 'millimeter', 'nanometer', 'm', 'mm', 'nm', 'um'
- **rot_unit** (*str*) – Unit for rotations. Can be 'urad', 'radian', 'milliradian', 'arcsec', 'arcmin', 'milliarcsec'
- **absolute** (*bool*) – Same meaning as for JWST SURs: if true, move the segment to exactly this position. Otherwise moves are treated as incremental relative moves from the current position.
- **display** (*bool*) – Display after moving?

- **delay_update** (*bool*) – hold off on computing the WFE change? This is useful for computational efficiency if you're moving a whole bunch of segments at once. Incompatible with display=True.

`move_sm_local`(*xtilt=0.0*, *ytilt=0.0*, *rot_unit='urad'*, *xtrans=0.0*, *ytrans=0.0*, *piston=0.0*, *trans_unit='micron'*, *display=False*, *delay_update=False*)

Move the secondary mirror in pose, using segment-local control coordinates.

These motions are always commanded in the segment-local "Control" coordinate systems, which are distinct for each segment. The SM is also handled a bit differently than all the PMSAs in terms of its local coordinate system, which this function handles behind the scenes.

Parameters

- **segment** (*str*) – Segment name, e.g. 'A1'. Use 'SM' for the secondary mirror.
- **xtilt, ytilt, clocking** (*floats*) – Tilt angle, in microradians by default. 'xtilt' means tilt *around the X axis*, and similarly for ytilt.
- **xtrans, ytrans, piston** (*floats*) – Displacement distance, in microns by default.
- **trans_unit** (*str*) – Unit for translations. Can be 'micron', 'millimeter','nanometer', 'mm', 'nm', 'um'
- **rot_unit** (*str*) – Unit for rotations. Can be 'urad', 'radian', 'milliradian', 'arcsec', 'arcmin', 'milliarcsec'
- **display** (*bool*) – Display after moving?
- **delay_update** (*bool*) – hold off on computing the WFE change? This is useful for computational efficiency if you're moving a whole bunch of segments at once. Incompatible with display=True.

`move_sur`(*sur_file*, *group=None*, *verbose=False*, *reverse=False*)

Move using a JWST Segment Update Request file

Parameters

- **sur_file** (*file name*) – Path to SUR XML file
- **group** (*one-based int index*) – Index to a single group to run. Default is to run all groups. Note, this index counts up from 1 (not 0) for consistency with group indexing in the SUR files themselves.
- **verbose** (*bool*) – Flag controlling whether moves are printed.
- **reverse** (*bool*) – Run this SUR "backwards", i.e. in opposite order of all groups and flipping the sign of all moves. (This can be useful for certain testing and mock data generation scenarios.)

`name`

string. Descriptive Name of this optic

`opds_as_hdul`(*delta_time*, *slew_angles*, *delta_opds=None*, *wfe_dict=None*, *case=None*, *add_main_opd=True*, *slew_averages=False*, *return_ind=None*, ***kwargs*)

Convert series of delta OPDS to HDUList

`powerspectrum`(*max_cycles=50*, *sampling=5*, *vmax=100*, *iterate=False*)

Compute the spatial power spectrum of an aberrated wavefront.

Produces nice plots on screen.

Returns an array [low, mid, high] giving the RMS spatial frequencies in the different JWST-defined spatial frequency bins:

low: <=5 cycles/aperture mid: 5 < cycles <= 30 high: 30 < cycles

ptv(*segment=None*)
return peak to valley WFE in nanometers

Parameters **segment** (*string*) – Segment name, to compute RMS for a single segment. Leave unspecified (None) to compute for the entire aperture.

property pupil_diam
Diameter of the pupil (if this is a pupil plane optic)

reset(*verbose=True*)
Reset an OPD to the state it was loaded from disk.

i.e. undo all segment moves.

rms(*segment=None*)
Return RMS WFE in nanometers

Parameters **segment** (*string*) – Segment name, to compute RMS for a single segment. Leave unspecified (None) to compute RMS WFE for the entire aperture. Segments are identified by name: A1-A6, B1-B6, C1-C6

property shape
Return shape of the OpticalElement, as a tuple

slew_pos_averages(*delta_time*, *slew_angles*, *opds=None*, *wfe_dict=None*, *mn_func=<function mean>*, *interpolate=False*, ***kwargs*)
Get averages at each slew position

Given a series of times and slew angles, calculate the average OPD and WFE RMS error within each slew angle position. Returns a tuple with new arrays of (dt_new, opds_new, wfe_dict_new).

If input both *opds* and *wfe_dict* are not specified, then we call the *evolve_dopd* function and return .

Parameters

- **delta_time** (*astropy.units quantity object*) – An array of times assuming astropy units.
- **slew_angles** (*ndarray*) – The sun pitch angles at each *delta_time*, in degrees between -5 and +45.
- **opds** (*ndarray or None*) – Cube of OPD images (or delta OPDs) associated with each *delta_time*. If set to None, then a new set of OPDs are not calculated.
- **wfe_dict** (*dict or None*) – If specified, then must provide a dictionary where the values for each keywords are the WFE drift components associated with each *delta_time*. New set of WFE dictionary is not calculated if set to None.
- **mn_func** (*function*) – Function to use for taking averages. Default: np.mean()
- **interpolate** (*bool*) – Instead of taking average, use the interpolation function *self.interp_dopds()*.

Keyword Arguments

- **case** (*string*) – Either "BOL" for current best estimate at beginning of life, or "EOL" for more conservative prediction at end of life.
- **do_thermal** (*bool*) – Include thermal slew component? Mostly for debugging purposes.
- **do_frill** (*bool*) – Include frill component? Mostly for debugging purposes.
- **do_iec** (*bool*) – Include IEC component? Good to exclude if calling this function repeatedly for evolution of multiple slews, then add IEC later.

- **amplitude** (*float*) – Full amplitude of IEC arcsine distribution. Values will range from -0.5*amplitude to +0.5*amplitude.
- **period** (*float*) – Period in minutes of IEC oscillations. Usually 3-5 minutes.
- **kind** (*str or int*) – Specifies the kind of interpolation (if specified) as a string. Default: 'linear'.

slew_scaling(*start_angle*, *end_angle*)

WFE scaling due to slew angle

Scale the WSS Hexike components based on slew pitch angles.

Parameters

- **start_angle** (*float*) – The starting sun pitch angle, in degrees between -5 and +45
- **end_angle** (*float*) – The ending sun pitch angle, in degrees between -5 and +45

thermal_slew(*delta_time*, *start_angle=- 5*, *end_angle=45*, *scaling=None*, *display=False*, *case='EOL'*, *delay_update=False*)

Update the OPD based on presence of a pitch angle change between observations.

Use a delta slew time along with the beginning and ending angles of the observatory relative to the sun (or the user can define a scaling factor) to determine the expected WFE caused by thermal variations. Note: The start_angle and end_angle are used together, but will be ignored if the scaling variable is set to somthing other than "None".

The maximum HOT to COLD pitch angles are -5 to 45 degrees. With regards to this, we make some assumptions: 1. A COLD to HOT slew is just the negative of the HOT to COLD slew 2. The scaling factor can be simplified to a simple ratio of angles (this is

a gross over-simplification due to lack of a better model)

The HOT to COLD vs COLD to HOT nature of the slew is determined by the start and end angles

Note, multiple calls in a row to this function are NOT cumulative; rather, the model internally resets to the initial starting OPD each time, and calculates a single slew. This is intentional to be more reproducible and well defined, with less hidden history state. If you need a more complex time evolution, build that yourself by summing individual delta OPDs.

Parameters

- **delta_time** (*astropy.units quantity object*) – The time between observations. Default units: "hour"
- **start_angle** (*float*) – The starting sun pitch angle, in degrees between -5 and +45
- **end_angle** (*float*) – The ending sun pitch angle, in degrees between -5 and +45
- **scaling** (*float between 0 and 1*) – Scaling factor that can be used instead of the start_angle and end_angle parameters. This directly sets the amplitude of the drift and overrides the angles and case settings.
- **display** (*bool*) – Display the updated OPD
- **case** (*string*) – either "BOL" for current best estimate at beginning of life, or "EOL" for more conservative prediction at end of life. The amplitude of the thermal drift is roughly 3x lower for BOL (13 nm after 14 days) versus EOL (43 nm after 14 days).
- **delay_update** (*bool*) – Users typically only need to call this directly if they have set the "delay_update" parameter to True in some function call to move mirrors.

update_opd(*display=False*, *verbose=False*)
Update the OPD based on the current linear model values.

Users typically only need to call this directly if they have set the "delay_update" parameter to True in some function call to move mirrors.

writeto(*outname*, *overwrite=True*, ***kwargs*)
Write OPD to a FITS file on disk

zern_seg(*segment*, *vmax=150*, *unit='nm'*)
Show the Zernike terms applied to a given segment

zero(*zero_original=False*)
Reset an OPD to precisely zero everywhere.

Parameters **zero_original** (*bool*) – should we zero out the stored copy of the original OPD? If so, then even using the reset() function won't set this back to the original value.

webbpsf_ext.psfs

Functions

create_obslist(bp, npix[, nwaves, is_grism, ...])	
create_waveset(bp, npix[, nwaves, is_grism])	
field_coeff_func(v2grid, v3grid, cf_fields, ...)	Interpolation function for PSF coefficient residuals
gen_image_from_coeff(inst, coeff, coeff_hdr)	Generate PSF
make_coeff_resid_grid(xin, yin, cf_resid, ...)	
nproc_use(fov_pix, oversample, nwavelengths)	Estimate Number of Processors

webbpsf_ext.psfs.create_obslist

webbpsf_ext.psfs.**create_obslist**(*bp*, *npix*, *nwaves=None*, *is_grism=False*, *sp_norm=None*, *use_sp_waveset=False*)

webbpsf_ext.psfs.create_waveset

webbpsf_ext.psfs.**create_waveset**(*bp*, *npix*, *nwaves=None*, *is_grism=False*)

webbpsf_ext.psfs.field_coeff_func

webbpsf_ext.psfs.**field_coeff_func**(*v2grid*, *v3grid*, *cf_fields*, *v2_new*, *v3_new*, *method='linear'*)
Interpolation function for PSF coefficient residuals

Uses *RegularGridInterpolator* to quickly determine new coefficient residulas at specified points.

Parameters

- **v2grid** (*ndarray*) – V2 values corresponding to *cf_fields*.
- **v3grid** (*ndarray*) – V3 values corresponding to *cf_fields*.

- **cf_fields** (*ndarray*) – Coefficient residuals at different field points Shape is (nV3, nV2, ncoeff, ypix, xpix)
- **v2_new** (*ndarray*) – New V2 point(s) to interpolate on. Same units as v2grid.
- **v3_new** (*ndarray*) – New V3 point(s) to interpolate on. Same units as v3grid.

webbpsf_ext.psfs.gen_image_from_coeff

webbpsf_ext.psfs.**gen_image_from_coeff**(*inst*, *coeff*, *coeff_hdr*, *sp_norm=None*, *nwaves=None*, *use_sp_waveset=False*, *return_oversample=False*)

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, then this returns a single image or list of images if sp_norm is a list of spectra.

Parameters

- **coeff** (*ndarray*) – A cube of polynomial coefficients for generating PSFs. This is generally oversampled with a shape (fov_pix*oversamp, fov_pix*oversamp, deg).
- **coeff_hdr** (*FITS header*) – Header information saved while generating coefficients.
- **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.
- **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 source.
- **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.
- **return_oversample** (*bool*) – If True, then instead returns the oversampled version of the PSF.

Keyword Arguments

- **grism_order** (*int*) – Grism spectral order (default=1).
- **ND_acq** (*bool*) – ND acquisition square in coronagraphic mask.

webbpsf_ext.psfs.make_coeff_resid_grid

webbpsf_ext.psfs.**make_coeff_resid_grid**(*xin*, *yin*, *cf_resid*, *xgrid*, *ygrid*)

webbpsf_ext.psfs.nproc_use

webbpsf_ext.psfs.**nproc_use**(*fov_pix*, *oversample*, *nwavelengths*, *coron=False*)

Estimate Number of Processors

Attempt to estimate a reasonable number of processors to use for a multi-wavelength calculation. One really does not want to end up swapping to disk with huge arrays.

NOTE: Requires `psutil` package. Otherwise defaults to `mp.cpu_count() / 2`

Parameters

- **fov_pix** (*int*) – Square size in detector-sampled pixels of final PSF image.
- **oversample** (*int*) – The optical system that we will be calculating for.
- **nwavelengths** (*int*) – Number of wavelengths.
- **coron** (*bool*) – Is the nproc recommendation for coronagraphic imaging? If so, the total RAM usage is different than for direct imaging.

webbpsf_ext.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. Function included are:

- medabsdev - median absolute deviation
- biweightMean - biweighted mean estimator
- mean - robust estimator of the mean of a data set
- mode - robust estimate of the mode of a data set using the half-sample method
- std - robust estimator of the standard deviation of a data set
- checkfit - return the standard deviation and biweights for a fit in order to determine its quality
- linefit - outlier resistant fit of a line to data
- polyfit - outlier resistant fit of a polynomial to data

For the fitting routines, the coefficients are returned in the same order as np.polyfit, i.e., with the coefficient of the highest power listed first.

For additional information about the original IDL routines, see: http://idlastro.gsfc.nasa.gov/contents.html#C17

Functions

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

webbpsf_ext.robust.biweightMean

webbpsf_ext.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()

webbpsf_ext.robust.checkfit

webbpsf_ext.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.

webbpsf_ext.robust.linefit

webbpsf_ext.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.

webbpsf_ext.robust.mean

webbpsf_ext.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.

webbpsf_ext.robust.medabsdev

webbpsf_ext.robust.**medabsdev**(*data*, *axis=None*, *keepdims=False*, *nan=True*)

Median Absolute Deviation

A "robust" version of standard deviation. Runtime is the same as *astropy.stats.funcs.mad_std*.

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.

webbpsf_ext.robust.mode

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

webbpsf_ext.robust.polyfit

webbpsf_ext.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.

webbpsf_ext.robust.std

webbpsf_ext.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.

webbpsf_ext.spectra

Functions

BOSZ_filename(Teff, metallicity, log_g, res)	Generate filename for BOSZ spectrum.
BOSZ_spectrum(Teff, metallicity, log_g[, ...])	BOSZ stellar atmospheres (Bohlin et al 2017).
bin_spectrum(sp, wave[, waveunits])	Rebin spectrum
companion_spec(bandpass[, model, atmo, ...])	Determine flux (ph/sec) of a companion
cond_filter(table, filt[, module, dist])	Given a COND table and NIRCam filter, return arrays of MJup and Vega mags.
cond_table([age, file])	Load COND Model Table
download_BOSZ_spectrum(Teff, metallicity, ...)	
jupiter_spec([dist, waveout, fluxout, base_dir])	Jupiter as an Exoplanet
linder_filter(table, filt, age[, dist, ...])	Linder Mags vs Mass Arrays
linder_table([file])	Load Linder Model Table
mag_to_counts(src_mag, bandpass[, sp_type, ...])	Convert stellar magnitudes in some bandpass to corresponding flux values (e-/sec)
sp_accr(mmdot[, rin, dist, truncated, ...])	Exoplanet accretion flux values (Zhu et al., 2015).
stellar_spectrum(sptype, *renorm_args, **kwargs)	Stellar spectrum

webbpsf_ext.spectra.BOSZ_filename

webbpsf_ext.spectra.**BOSZ_filename**(*Teff*, *metallicity*, *log_g*, *res*, *carbon=0*, *alpha=0*)

Generate filename for BOSZ spectrum.

webbpsf_ext.spectra.BOSZ_spectrum

webbpsf_ext.spectra.**BOSZ_spectrum**(*Teff*, *metallicity*, *log_g*, *res=2000*, *interpolate=True*, *carbon=0*, *alpha=0*, ***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 Angstroms to 32 microns. Teff range from 3500K to 36000K.

This function interpolates 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.

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

- **carbon** (*float*) – Carbon abundance [C/M]. Must be either [-0.75,-0.5,-0.25, 0, 0.25, 0.5].
- **alpha** (*float*) – alpha-element value [alpha/H]. Must be either [-0.25, 0, 0.25, 0.5]

- **res** (*str*) – Spectral resolution to use (instrument broadening). Valid points: [200, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100000, 300000]
- **interpolate** (*bool*) – Interpolate spectrum using a weighted average of grid points surrounding the desired input parameters.

References

https://archive.stsci.edu/prepds/bosz/

webbpsf_ext.spectra.bin_spectrum

webbpsf_ext.spectra.**bin_spectrum**(*sp*, *wave*, *waveunits='um'*)

Rebin spectrum

Rebin a pysynphot.spectrum to a different 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.

webbpsf_ext.spectra.companion_spec

webbpsf_ext.spectra.**companion_spec**(*bandpass*, *model='SB12'*, *atmo='hy3s'*, *mass=10*, *age=100*, *entropy=10*, *dist=10*, *accr=False*, *mmdot=None*, *mdot=None*, *accr_rin=2*, *truncated=False*, *sptype=None*, *renorm_args=None*, *Av=0*, ***kwargs*)

Determine flux (ph/sec) of a companion

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

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

Parameters

- **bandpass** (pysynphot.obsbandpass) – A Pysynphot bandpass object.
- **model** (*str*) – Exoplanet model to use ('sb12', 'bex', 'cond') or stellar spectrum model ('bosz', 'ck04models', 'phoenix').
- **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).
- **dist** (*float*) – Distance in pc.
- **Av** (*float*) – Extinction magnitude (assumes Rv=4.0) of the companion (e.g., due to being embedded in a disk).
- **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).

webbpsf_ext.spectra.cond_filter

webbpsf_ext.spectra.**cond_filter**(*table*, *filt*, *module='A'*, *dist=None*, ***kwargs*)

Given a COND table and NIRCam filter, return arrays of MJup and Vega mags. If distance (pc) is provided, then return the apparent magnitude, otherwise absolute magnitude at 10pc. Input table has already been filtered by age.

webbpsf_ext.spectra.cond_table

webbpsf_ext.spectra.**cond_table**(*age=None*, *file=None*, ***kwargs*)

Load COND Model Table

Function to read in the COND model tables, which have been formatted in a very specific way. Has the option to return a dictionary of astropy Tables, where each dictionary element corresponds to the specific ages within the COND table. Or, if the age keyword is specified, then this function only returns a single astropy table.

Parameters

- **age** (*float*) – Age in Myr. If set to None, then an array of ages from the file is used to generate dictionary. If set, chooses the closest age supplied in table.
- **file** (*string*) – Location and name of COND file. See isochrones stored at https://phoenix.ens-lyon.fr/Grids/. Default is model.AMES-Cond-2000.M-0.0.JWST.Vega

webbpsf_ext.spectra.download_BOSZ_spectrum

webbpsf_ext.spectra.**download_BOSZ_spectrum**(*Teff*, *metallicity*, *log_g*, *res*, *carbon=0*, *alpha=0*)

webbpsf_ext.spectra.jupiter_spec

webbpsf_ext.spectra.**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.

webbpsf_ext.spectra.linder_filter

webbpsf_ext.spectra.**linder_filter**(*table*, *filt*, *age*, *dist=10*, *cond_file=None*, ***kwargs*)
Linder Mags vs Mass Arrays

Given a Linder table, filter name, and age (Myr), return arrays of MJup and Vega mags. If distance (pc) is provided, then return the apparent magnitude, otherwise absolute magnitude at 10pc.

This function takes the isochrones tables from Linder et al 2019 and creates a irregular contour grid of filter magnitude and log(age) where the z-axis is log(mass). This is mapped onto a regular grid that is interpolated within the data boundaries and linearly extrapolated outside of the region of available data.

Parameters

- **table** (*astropy table*) – Astropy table output from *linder_table*.
- **filt** (*string*) – Name of NIRCam filter.
- **age** (*float*) – Age in Myr of planet.
- **dist** (*float*) – Distance in pc. Default is 10pc (abs mag).

webbpsf_ext.spectra.linder_table

webbpsf_ext.spectra.**linder_table**(*file=None*, ***kwargs*)
Load Linder Model Table

Function to read in isochrone models from Linder et al. 2019. Returns an astropy Table.

Parameters **file** (*string*) – Location and name of Linder et al file. Default is `BEX_evol_mags_-3_MH_0.00.dat`.

webbpsf_ext.spectra.mag_to_counts

webbpsf_ext.spectra.**mag_to_counts**(*src_mag*, *bandpass*, *sp_type='G0V'*, *mag_units='vegamag'*, ***kwargs*)
Convert stellar magnitudes in some bandpass to corresponding flux values (e-/sec)

webbpsf_ext.spectra.sp_accr

webbpsf_ext.spectra.**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).

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.

webbpsf_ext.spectra.stellar_spectrum

webbpsf_ext.spectra.**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

Classes

planets_sb12([atmo, mass, age, entropy, ...])	Exoplanet spectrum from Spiegel & Burrows (2012)
source_spectrum(name, sptype, mag_val, bp, ...)	Model source spectrum

webbpsf_ext.spectra.planets_sb12

class webbpsf_ext.spectra.**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 (increments of 1MJup).
- **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.

__init__(*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*)

Methods

__init__([atmo, mass, age, entropy, ...])	
export_pysynphot([waveout, fluxout])	Output to pysynphot.spectrum object

Attributes

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
sub_dir	
wave	Wavelength of spectrum
waveunits	Wavelength units

property age
Age in millions of years

property atmo
Atmosphere type

property distance
Assumed distance to source (pc)

property entropy
Initial entropy (8.0-13.0)

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

property flux
Spectral flux

property fluxunits
Flux units

property mass
Mass of planet (MJup)

property mdot
Accretion rate in MJup/yr

property wave
Wavelength of spectrum

property waveunits
Wavelength units

webbpsf_ext.spectra.source_spectrum

class webbpsf_ext.spectra.**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 webbpsf_ext
>>> from webbpsf_ext.spectra import source_spectrum
>>>
>>> name = 'HR8799'
>>> vot = 'votables/{}.vot'.format(name)
>>> bp_k = webbpsf_ext.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
```

__init__(*name*, *sptype*, *mag_val*, *bp*, *votable_file*, *Teff=None*, *metallicity=None*, *log_g=None*, *Av=None*, ***kwargs*)

Methods

__init__(name, sptype, mag_val, bp, votable_file)	
bb_jy(wave, T)	Blackbody function (Jy)
fit_SED([x0, robust, use_err, IR_excess, ...])	Fit a model function to photometry
func_resid(x[, IR_excess, wlim, use_err])	Calculate model residuals
model_IRexcess(x[, sp])	Model for stellar spectrum with IR excess
model_scale(x[, sp])	Simple model to scale stellar spectrum
plot_SED([ax, return_figax, xr, yr, units])	

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)

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 no 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.

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?

model_IRexcess(*x*, *sp=None*)
Model for stellar spectrum with IR excess

Model of a stellar spectrum plus IR excess, where the excess is a modified blackbody. The final model follows the form:

$$x_0 BB(\lambda, x_1)\lambda^{x_2}$$

model_scale(*x*, *sp=None*)
Simple model to scale stellar spectrum

webbpsf_ext.utils

Functions

check_fitsgz(opd_file[, inst_str])	WebbPSF FITS files can be either .fits or compressed .gz.

webbpsf_ext.utils.check_fitsgz

webbpsf_ext.utils.**check_fitsgz**(*opd_file*, *inst_str=None*)
WebbPSF FITS files can be either .fits or compressed .gz. Search for .fits by default, then .fits.gz.

Parameters

- **opd_file** (*str*) – Name of FITS file, either .fits or .fits.gz
- **inst_str** (*None or str*) – If OPD file is instrument-specific, then specify here. Will look in instrument OPD directory. If set to None, then also checks *opd_file* for an instrument-specific string to determine if to look in instrument OPD directory, otherwise assume file name is in webbpsf data base directory.

webbpsf_ext.webbpsf_ext_core

Functions

coron_grid(self, npsf_per_axis[, xoff_vals, ...])	Get grid points based on coronagraphic obseervation
nrc_mask_trans(image_mask, x, y)	Compute the amplitude transmission appropriate for a BLC for some given pixel spacing corresponding to the supplied Wavefront.

webbpsf_ext.webbpsf_ext_core.coron_grid

webbpsf_ext.webbpsf_ext_core.**coron_grid**(*self*, *npsf_per_axis*, *xoff_vals=None*, *yoff_vals=None*)

Get grid points based on coronagraphic obseervation

Returns sci pixels values around mask center.

webbpsf_ext.webbpsf_ext_core.nrc_mask_trans

webbpsf_ext.webbpsf_ext_core.**nrc_mask_trans**(*image_mask*, *x*, *y*)

Compute the amplitude transmission appropriate for a BLC for some given pixel spacing corresponding to the supplied Wavefront.

Based on the Krist et al. SPIE paper on NIRCam coronagraph design

Note : To get the actual transmission, these values should be squared.

Classes

MIRI_ext([filter, pupil_mask, image_mask, ...])	MIRI instrument PSF coefficients
NIRCam_ext([filter, pupil_mask, image_mask, ...])	NIRCam instrument PSF coefficients

webbpsf_ext.webbpsf_ext_core.MIRI_ext

class webbpsf_ext.webbpsf_ext_core.**MIRI_ext**(*filter=None*, *pupil_mask=None*, *image_mask=None*, *fov_pix=None*, *oversample=None*, ***kwargs*)

Bases: webbpsf.webbpsf_core.MIRI

MIRI instrument PSF coefficients

Subclass of WebbPSF's MIRI class for generating polynomial coefficients to cache and quickly generate PSFs for arbitrary spectral types as well as WFE variations due to field-dependent OPDs and telescope thermal drifts.

Parameters

- **filter** (*str*) – Name of input filter.
- **pupil_mask** (*str, None*) – Pupil elements such as grisms or lyot stops (default: None).
- **image_mask** (*str, None*) – Specify which coronagraphic occulter (default: None).
- **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.

__init__(*filter=None*, *pupil_mask=None*, *image_mask=None*, *fov_pix=None*, *oversample=None*, ***kwargs*)

Methods

__init__([filter, pupil_mask, image_mask, ...])	
calc_datacube(wavelengths, *args, **kwargs)	Calculate a spectral datacube of PSFs
calc_psf([add_distortion, fov_pixels, ...])	Compute a PSF
calc_psf_from_coeff([sp, return_oversample, ...])	Create PSF image from coefficients
calc_psfs_grid([sp, wfe_drift, osamp, ...])	PSF grid across an instrumnet FoV
calc_psfs_sgd(xoff_asec, yoff_asec[, use_coeff])	Calculate small grid dither PSFs
display()	Display the currently configured optical system on screen
drift_opd(wfe_drift[, opd])	A quick method to drift the pupil OPD.
gen_mask_image([npix, pixelscale, ...])	Return an image representation of the focal plane mask.
gen_psf_coeff(**kwargs)	Generate PSF coefficients
gen_save_name([wfe_drift])	Generate save name for polynomial coefficient output file.
gen_wfedrift_coeff([force, save])	Fit WFE drift coefficients
gen_wfefield_coeff([force, save])	Fit WFE field-dependent coefficients
gen_wfemask_coeff([large_grid, force, save])	Fit WFE changes in mask position
get_opd_file_full_path([opdfilename])	Return full path to the named OPD file.
get_opd_info([opd, pupil, HDUL_to_OTELM])	Parse out OPD information for a given OPD, which can be a file name, tuple (file,slice), HDUList, or OTE Linear Model.
get_optical_system([fft_oversample, ...])	Return an OpticalSystem instance corresponding to the instrument as currently configured.
interpolate_was_opd(array, newdim)	Interpolates an input 2D array to any given size.
load_was_opd(inputWasOpd[, size, save, filename])	Load and interpolate an OPD from the WAS.
psf_grid([num_psfs, all_detectors, save, ...])	Create a PSF library in the form of a grid of PSFs across the detector based on the specified instrument, filter, and detector.
set_position_from_aperture_name(aperture_name)	Set the simulated center point of the array based on a named SIAF aperture.
visualize_wfe_budget([slew_delta_time, ...])	Display a visual WFE budget showing the various terms that sum into the overall WFE for a given instrument

Attributes

aperturename	SIAF aperture name for detector pixel to sky coords transformations
bandpass	
detector	Detector selected for simulated PSF
detector_list	Detectors on which the simulated PSF could lie
detector_position	The pixel position in (X, Y) on the detector, relative to the currently-selected SIAF aperture subarray.
fastaxis	Fast readout direction in sci coords
filter	Currently selected filter name (e.g.
filter_list	List of available filter names for this instrument
fov_pix	
image_mask	Currently selected image plane mask, or None for direct imaging
is_coron	Coronagraphic observations based on pupil mask settings
is_slitspec	LRS observations based on pupil mask settings
name	
ndeg	Degree of polynomial fit
npsf	Number of wavelengths/PSFs to fit
options	A dictionary capable of storing other arbitrary options, for extensibility.
oversample	
pixelscale	Detector pixel scale, in arcsec/pixel
pupil	Filename *or* fits.HDUList for JWST pupil mask.
pupil_mask	Currently selected Lyot pupil mask, or None for direct imaging
pupilopd	Filename *or* fits.HDUList for JWST pupil OPD.
quick	Perform quicker coeff calculation over limited bandwidth?
save_dir	Coefficient save directory
save_name	Coefficient file name
siaf_ap	SIAF Aperture object
slowaxis	Slow readout direction in sci coords
telescope	
wave_fit	Wavelength range to fit

property aperturename
SIAF aperture name for detector pixel to sky coords transformations

calc_datacube(*wavelengths*, **args*, ***kwargs*)
Calculate a spectral datacube of PSFs

Parameters **wavelengths** (*iterable of floats*) – List or ndarray or tuple of floating point wavelengths in meters, such as you would supply in a call to calc_psf via the "monochromatic" option

calc_psf(*add_distortion=None*, *fov_pixels=None*, *oversample=None*, *wfe_drift=None*, *coord_vals=None*, *coord_frame='tel'*, ***kwargs*)

Compute a PSF

Slight modification of inherent WebbPSF *calc_psf* function. If add_distortion, fov_pixels, and oversample are not specified, then we automatically use the associated attributes.

Notes

More advanced PSF computation options (pupil shifts, source positions, jitter, ...) may be set by configuring the *.options* dictionary attribute of this class.

Parameters

- **sp** (`pysynphot.spectrum`) – Source input spectrum. If not specified, the default is flat in phot lam. (equal number of photons per spectral bin).
- **source** (*synphot.spectrum.SourceSpectrum or dict*) – TODO: synphot not yet implemented in webbpsf_ext!!
- **nlambda** (*int*) – How many wavelengths to model for broadband? The default depends on how wide the filter is: (5,3,1) for types (W,M,N) respectively
- **monochromatic** (*float, optional*) – Setting this to a wavelength value (in meters) will compute a monochromatic PSF at that wavelength, overriding filter and nlambda settings.
- **fov_arcsec** (*float*) – field of view in arcsec. Default=5
- **fov_pixels** (*int*) – field of view in pixels. This is an alternative to fov_arcsec.
- **outfile** (*string*) – Filename to write. If None, then result is returned as an HDUList
- **oversample, detector_oversample, fft_oversample** (*int*) – How much to oversample. Default=4. By default the same factor is used for final output pixels and intermediate optical planes, but you may optionally use different factors if so desired.
- **overwrite** (*bool*) – overwrite output FITS file if it already exists?
- **display** (*bool*) – Whether to display the PSF when done or not.
- **save_intermediates, return_intermediates** (*bool*) – Options for saving to disk or returning to the calling function the intermediate optical planes during the propagation. This is useful if you want to e.g. examine the intensity in the Lyot plane for a coronagraphic propagation.
- **normalize** (*string*) – Desired normalization for output PSFs. See doc string for OpticalSystem.calc_psf. Default is to normalize the entrance pupil to have integrated total intensity = 1.
- **add_distortion** (*bool*) – If True, will add 2 new extensions to the PSF HDUlist object. The 2nd extension will be a distorted version of the over-sampled PSF and the 3rd extension will be a distorted version of the detector-sampled PSF.
- **crop_psf** (*bool*) – If True, when the PSF is rotated to match the detector's rotation in the focal plane, the PSF will be cropped so the shape of the distorted PSF will match it's undistorted counterpart. This will only be used for NIRCam, NIRISS, and FGS PSFs.

Keyword Arguments

- **return_hdul** (*bool*) – Return PSFs in an HDUList rather than set of arrays (default: True).

- **return_oversample** (*bool*) – Returns the oversampled version of the PSF instead of detector-sampled PSF. Only valid for *reaturn_hdul=False*, otherwise full HDUList returned. Default: True.

calc_psf_from_coeff(*sp=None*, *return_oversample=True*, *return_hdul=True*, *wfe_drift=None*, *coord_vals=None*, *coord_frame='tel'*, ***kwargs*)

Create PSF image from coefficients

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).

Returns a single image or list of images if sp is a list of spectra. By default, it returns only the oversampled PSF, but setting return_oversample=False will instead return detector-sampled images.

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 mask.
- **return_oversample** (*bool*) – Returns the oversampled version of the PSF instead of detector-sampled PSF. Default: True.
- **wfe_drift** (*float or None*) – Wavefront error drift amplitude in nm.
- **coord_vals** (*tuple or None*) – Coordinates (in arcsec or pixels) to calculate field-dependent PSF. If multiple values, then this should be an array ([xvals], [yvals]).
- **coord_frame** (*str*) –

  Type of input 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.
- **return_hdul** (*bool*) – Return PSFs in an HDUList rather than set of arrays (default: True).

calc_psfs_grid(*sp=None*, *wfe_drift=0*, *osamp=1*, *npsf_per_full_fov=15*, *xsci_vals=None*, *ysci_vals=None*, *return_coords=None*, *use_coeff=True*, ***kwargs*)

PSF grid across an instrumnet FoV

Create a grid of PSFs across instrument aperture FoV. By default, imaging observations will be for full detector FoV with regularly spaced grid. Coronagraphic observations will cover nominal coronagraphic mask region (usually 10s of arcsec) and will have logarithmically spaced values where appropriate.

Keyword Arguments

- **sp** (pysynphot.spectrum) – If not specified, the default is flat in phot lam (equal number of photons per wavelength 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 suppression of coronagraphic mask. If set, then the resulting PSF image will be scaled to generate the total observed number of photons from the spectrum (ie., not scaled by unit response).
- **wfe_drift** (*float*) – Desired WFE drift value relative to default OPD.
- **osamp** (*int*) – Sampling of output PSF relative to detector sampling.

- **npsf_per_full_fov** (*int*) – Number of PSFs across one dimension of the instrument's field of view. If a coronagraphic observation, then this is for the nominal coronagrahic field of view.
- **xsci_vals** (*None or ndarray*) – Option to pass a custom grid values along x-axis in 'sci' coords. If coronagraph, this instead corresponds to coronagraphic mask axis, which has a slight rotation relative to detector axis in MIRI.
- **ysci_vals** (*None or ndarray*) – Option to pass a custom grid values along y-axis in 'sci' coords. If coronagraph, this instead corresponds to coronagraphic mask axis, which has a slight rotation relative to detector axis in MIRI.
- **return_coords** (*None or str*) – Option to also return coordinate values in desired frame ('det', 'sci', 'tel', 'idl'). Output is then xvals, yvals, hdul_psfs.
- **use_coeff** (*bool*) – If True, uses *calc_psf_from_coeff*, other WebbPSF's built-in *calc_psf*.

calc_psfs_sgd(*xoff_asec*, *yoff_asec*, *use_coeff=True*, ***kwargs*)

Calculate small grid dither PSFs

Convenience function to calculation a series of SGD PSFs. This is essentially a wrapper around the *calc_psf_from_coeff* and *calc_psf* functions. Only valid for coronagraphic observations.

Parameters

- **xoff_asec** (*float or array-like*) – Offsets in x-direction (in 'idl' coordinates).
- **yoff_asec** (*float or array-like*) – Offsets in y-direction (in 'idl' coordinates).
- **use_coeff** (*bool*) – If True, uses *calc_psf_from_coeff*, other WebbPSF's built-in *calc_psf*.

property detector

Detector selected for simulated PSF

Used in calculation of field-dependent aberrations. Must be selected from detectors in the *detector_list* attribute.

property detector_list

Detectors on which the simulated PSF could lie

property detector_position

The pixel position in (X, Y) on the detector, relative to the currently-selected SIAF aperture subarray. By default the SIAF aperture will correspond to the full-frame detector, so (X,Y) will in that case be absolute (X,Y) pixels on the detector. But if you select a subarray aperture name from the SIAF, then the (X,Y) are interpreted as (X,Y) within that subarray.

Please note, this is X,Y order - **not** a Pythonic y,x axes ordering.

display()

Display the currently configured optical system on screen

drift_opd(*wfe_drift*, *opd=None*)

A quick method to drift the pupil OPD. This function applies some WFE drift to input OPD file by breaking up the wfe_drift attribute into thermal, frill, and IEC components. If we want more realistic time evolution, then we should use the procedure in dev_utils/WebbPSF_OTE_LM.ipynb to create a time series of OPD maps, which can then be passed directly to create unique PSFs.

This outputs an OTE Linear Model. In order to update instrument class:

```
>>> opd_dict = inst.drift_opd()
>>> inst.pupilopd = opd_dict['opd']
>>> inst.pupil = opd_dict['opd']
```

property fastaxis
Fast readout direction in sci coords

property filter
Currently selected filter name (e.g. F200W)

gen_mask_image(*npix=None*, *pixelscale=None*, *detector_orientation=True*)
Return an image representation of the focal plane mask. For 4QPM, we should the phase offsets (0 or 1), whereas the Lyot and LRS slit masks return transmission.

Parameters

- **npix** (*int*) – Number of pixels in output image. If not set, then is automatically determined based on mask FoV and *pixelscale*
- **pixelscale** (*float*) – Size of output pixels in units of arcsec. If not specified, then selects nominal detector pixel scale.
- **detector_orientation** (*bool*) – Should the output image be rotated to be in detector coordinates? If set to False, then output mask is rotated along V2/V3 axes.

gen_psf_coeff(***kwargs*)
Generate PSF coefficients

Creates a set of coefficients that will generate simulated PSFs for any arbitrary wavelength. This function first simulates a number of evenly- spaced PSFs throughout the specified bandpass (or the full channel). An nth-degree polynomial is then fit to each oversampled pixel using a linear-least squares fitting routine. The final set of coefficients for each pixel is returned as an image cube. The returned set of coefficient are then used to produce PSF via *calc_psf_from_coeff*.

Useful for quickly generated imaging and dispersed PSFs for multiple spectral types.

Keyword Arguments

- **wfe_drift** (*float*) – Wavefront error drift amplitude in nm.
- **force** (*bool*) – Forces a recalculation of PSF even if saved PSF exists. (default: False)
- **save** (*bool*) – Save the resulting PSF coefficients to a file? (default: True)
- **nproc** (*bool or None*) – Manual setting of number of processor cores to break up PSF calculation. If set to None, this is determined based on the requested PSF size, number of available memory, and hardware processor cores. The automatic calculation endeavors to leave a number of resources available to the user so as to not crash the user's machine.
- **return_results** (*bool*) – By default, results are saved as object the attributes *psf_coeff* and *psf_coeff_header*. If return_results=True, results are instead returned as function outputs and will not be saved to the attributes. This is mostly used for successive coeff simulations to determine varying WFE drift or focal plane dependencies.
- **return_extras** (*bool*) – Additionally returns a dictionary of monochromatic PSFs images and their corresponding wavelengths for debugging purposes. Can be used with or without *return_results*. If *return_results=False*, then only this dictionary is returned, otherwise if *return_results=False* then returns everything as a 3-element tuple (psf_coeff, psf_coeff_header, extras_dict).

gen_save_name(*wfe_drift=0*)
Generate save name for polynomial coefficient output file.

gen_wfedrift_coeff(*force=False*, *save=True*, ***kwargs*)
Fit WFE drift coefficients

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.

It's Legendre's all the way down...

Parameters

- **force** (*bool*) – Forces a recalculation of coefficients even if saved file exists. (default: False)
- **save** (*bool*) – Save the resulting PSF coefficients to a file? (default: True)

Keyword Arguments

- **wfe_list** (*array-like*) – A list of wavefront error drift values (nm) to calculate and fit. Default is [0,1,2,5,10,20,40], which covers the most-likely scenarios (1-5nm) while also covering a range of extreme drift values (10-40nm).
- **return_results** (*bool*) – By default, results are saved in *self._psf_coeff_mod* dictionary. If return_results=True, results are instead returned as function outputs and will not be saved to the dictionary attributes.
- **return_raw** (*bool*) – Normally, we return the relation between PSF coefficients as a function of position. Instead this returns (as function outputs) the raw values prior to fitting. Final results will not be saved to the dictionary attributes.

gen_wfefield_coeff(*force=False*, *save=True*, ***kwargs*)
Fit WFE field-dependent coefficients

Find a relationship between field position and PSF coefficients for non-coronagraphic observations and when *include_si_wfe* is enabled.

Parameters

- **force** (*bool*) – Forces a recalculation of coefficients even if saved file exists. (default: False)
- **save** (*bool*) – Save the resulting PSF coefficients to a file? (default: True)

Keyword Arguments

- **return_results** (*bool*) – By default, results are saved in *self._psf_coeff_mod* dictionary. If return_results=True, results are instead returned as function outputs and will not be saved to the dictionary attributes.
- **return_raw** (*bool*) – Normally, we return the relation between PSF coefficients as a function of position. Instead this returns (as function outputs) the raw values prior to fitting. Final results will not be saved to the dictionary attributes.

gen_wfemask_coeff(*large_grid=False*, *force=False*, *save=True*, ***kwargs*)
Fit WFE changes in mask position

For coronagraphic masks, slight changes in the PSF location relative to the image plane mask can substantially alter the PSF speckle pattern. This function generates a number of PSF coefficients at a variety of positions, then fits polynomials to the residuals to track how the PSF changes across the mask's field of view. Special care is taken near the 10-20mas region in order to provide accurate sampling of the SGD offsets.

Parameters

- **large_grid** (*bool*) – Use a large number (high-density) of grid points to create coefficients. If True, then a higher fidelity PSF variations across the FoV, but could take hours to genrate on the first pass. Setting to False allows for quicker coefficient creation with a smaller memory footprint, useful for testing and debugging.

- **force** (*bool*) – Forces a recalculation of coefficients even if saved file exists. (default: False)
- **save** (*bool*) – Save the resulting PSF coefficients to a file? (default: True)

Keyword Arguments

- **return_results** (*bool*) – By default, results are saved in *self._psf_coeff_mod* dictionary. If return_results=True, results are instead returned as function outputs and will not be saved to the dictionary attributes.
- **return_raw** (*bool*) – Normally, we return the relation between PSF coefficients as a function of position. Instead this returns (as function outputs) the raw values prior to fitting. Final results will not be saved to the dictionary attributes.

get_opd_file_full_path(*opdfilename=None*)

Return full path to the named OPD file.

The OPD may be:

- a local or absolute path,
- or relative implicitly within an SI directory, e.g. $WEBBPSF_PATH/NIRCam/OPD
- or relative implicitly within $WEBBPSF_PATH

This function handles filling in the implicit path in the latter cases.

get_opd_info(*opd=None*, *pupil=None*, *HDUL_to_OTELM=True*)

Parse out OPD information for a given OPD, which can be a file name, tuple (file,slice), HDUList, or OTE Linear Model. Returns dictionary of some relevant information for logging purposes. The dictionary has an OPD version as an OTE LM.

This outputs an OTE Linear Model. In order to update instrument class:

```
>>> opd_dict = inst.get_opd_info()
>>> opd_new = opd_dict['pupilopd']
>>> inst.pupilopd = opd_new
>>> inst.pupil = opd_new
```

get_optical_system(*fft_oversample=2*, *detector_oversample=None*, *fov_arcsec=2*, *fov_pixels=None*, *options=None*)

Return an OpticalSystem instance corresponding to the instrument as currently configured.

When creating such an OpticalSystem, you must specify the parameters needed to define the desired sampling, specifically the oversampling and field of view.

Parameters

- **fft_oversample** (*int*) – Oversampling factor for intermediate plane calculations. Default is 2
- **detector_oversample** (*int, optional*) – By default the detector oversampling is equal to the intermediate calculation oversampling. If you wish to use a different value for the detector, set this parameter. Note that if you just want images at detector pixel resolution you will achieve higher fidelity by still using some oversampling (i.e. *not* setting *oversample_detector=1*) and instead rebinning down the oversampled data.
- **fov_pixels** (*float*) – Field of view in pixels. Overrides fov_arcsec if both set.
- **fov_arcsec** (*float*) – Field of view, in arcseconds. Default is 2

Returns **osys** (*poppy.OpticalSystem*) – an optical system instance representing the desired configuration.

property image_mask
Currently selected image plane mask, or None for direct imaging

image_mask_list
List of available image_masks

include_ote_field_dependence
Should calculations include the Science Instrument internal WFE?

interpolate_was_opd(*array*, *newdim*)
Interpolates an input 2D array to any given size.

Parameters

- **array** (*float*) – input array to interpolate
- **newdim** (*int*) – new size of the 2D square array (newdim x newdim)

Returns **newopd** (*new array interpolated to (newdim x newdim)*)

property is_coron
Coronagraphic observations based on pupil mask settings

property is_slitspec
LRS observations based on pupil mask settings

load_was_opd(*inputWasOpd*, *size=1024*, *save=False*, *filename='new_was_opd.fits'*)
Load and interpolate an OPD from the WAS.

Ingests a WAS OPD and interpolates it to the proper size for WebbPSF.

Parameters

- **HDUlist_or_filename** (*string*) – Either a fits.HDUList object or a filename of a FITS file on disk
- **size** (*int, optional*) – Desired size of the output OPD. Default is 1024.
- **save** (*bool, optional*) – Save the interpolated OPD if True. Default is False.
- **filename** (*string, optional*) – Filename of the output OPD, if 'save' is True. Default is 'new_was_opd.fits'.

Returns **HDUlist** (*string*) – fits.HDUList object of the interpolated OPD

property ndeg
Degree of polynomial fit

property npsf
Number of wavelengths/PSFs to fit

psf_grid(*num_psfs=16*, *all_detectors=True*, *save=False*, *outdir=None*, *outfile=None*, *overwrite=True*, *verbose=True*, *use_detsampled_psf=False*, *single_psf_centered=True*, ***kwargs*)
Create a PSF library in the form of a grid of PSFs across the detector based on the specified instrument, filter, and detector. The output GriddedPSFModel object will contain a 3D array with axes [i, y, x] where i is the PSF position on the detector grid and (y,x) is the 2D PSF.

Parameters

- **num_psfs** (*int*) – The total number of fiducial PSFs to be created and saved in the files. This number must be a square number. Default is 16. E.g. num_psfs = 16 will create a 4x4 grid of fiducial PSFs.
- **all_detectors** (*bool*) – If True, run all detectors for the instrument. If False, run for the detector set in the instance. Default is True

- **save** (*bool*) – True/False boolean if you want to save your file. Default is False.
- **outdir** (*str*) – If "save" keyword is set to True, your file will be saved in the specified directory. Default of None will save it in the current directory
- **outfile** (*str*) – If "save" keyword is set to True, your file will be saved as {outfile}_det.fits. Default of None will save it as instr_det_filt_fovp#_samp#_npsf#.fits
- **overwrite** (*bool*) – True/False boolean to overwrite the output file if it already exists. Default is True.
- **verbose** (*bool*) – True/False boolean to print status updates. Default is True.
- **use_detsampled_psf** (*bool*) – If True, the grid of PSFs returned will be detector sampled (made by binning down the oversampled PSF). If False, the PSFs will be oversampled by the factor defined by the oversample/detector_oversample/fft_oversample keywords. Default is False. This is rarely needed - if uncertain, leave this alone.
- **single_psf_centered** (*bool*) – If num_psfs is set to 1, this defines where that psf is located. If True it will be the center of the detector, if False it will be the location defined in the WebbPSF attribute detector_position (reminder - detector_position is (x,y)). Default is True This is also rarely needed.
- ****kwargs** – Any extra arguments to pass the WebbPSF calc_psf() method call.

Returns

- **gridmodel** (*photutils GriddedPSFModel object or list of objects*) – Returns a GriddedPSFModel object or a list of objects if more than one configuration is specified (1 per instrument, detector, and filter) User also has the option to save the grid as a fits.HDUlist object.
- *Use*
- *—-*
- *nir = webbpsf.NIRCam()*
- *nir.filter = "F090W"*
- *list_of_grids = nir.psf_grid(all_detectors=True, num_psfs=4)*
- *wfi = webbpsf.WFI()*
- *wfi.filter = "Z087"*
- *wfi.detector = "SCA02"*
- *grid = wfi.psf_grid(all_detectors=False, oversample=5, fov_pixels=101)*

property pupil_mask
Currently selected Lyot pupil mask, or None for direct imaging

pupil_mask_list
List of available pupil_masks

property quick
Perform quicker coeff calculation over limited bandwidth?

property save_dir
Coefficient save directory

property save_name
Coefficient file name

set_position_from_aperture_name(*aperture_name*)
Set the simulated center point of the array based on a named SIAF aperture. This will adjust the detector and detector position attributes.

property siaf_ap
SIAF Aperture object

property slowaxis
Slow readout direction in sci coords

visualize_wfe_budget(*slew_delta_time=<Quantity 14. d>*, *slew_case='EOL'*, *ptt_only=False*, *verbose=True*)
Display a visual WFE budget showing the various terms that sum into the overall WFE for a given instrument

Compares a WebbPSF instrument instance with the JWST optical budget for that instrument

Parameters

- **inst** (*webbpsf.JWInstrument*) – A JWST instrument instance
- **slew_delta_time** (*astropy.Quantity time*) – Time duration for thermal slew model
- **slew_case** (*basestring*) – 'BOL' or 'EOL' for beginning of life or end of life thermal slew model. EOL is about 3x higher amplitude
- **ptt_only** (*bool*) – When decomposing wavefront into controllable modes, use a PTT-only basis? The default is to use all controllable pose modes. (This is mostly a leftover debug option at this point, not likely useful in general)
- **verbose** (*bool*) – Be more verbose

property wave_fit
Wavelength range to fit

webbpsf_ext.webbpsf_ext_core.NIRCam_ext

class webbpsf_ext.webbpsf_ext_core.**NIRCam_ext**(*filter=None*, *pupil_mask=None*, *image_mask=None*, *fov_pix=None*, *oversample=None*, ***kwargs*)
Bases: webbpsf.webbpsf_core.NIRCam

NIRCam instrument PSF coefficients

Subclass of WebbPSF's NIRCam class for generating polynomial coefficients to cache and quickly generate PSFs for arbitrary spectral types as well as WFE variations due to field-dependent OPDs and telescope thermal drifts.

Parameters

- **filter** (*str*) – Name of input filter.
- **pupil_mask** (*str, None*) – Pupil elements such as grisms or lyot stops (default: None).
- **image_mask** (*str, None*) – Specify which coronagraphic occulter (default: None).
- **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.

__init__(*filter=None*, *pupil_mask=None*, *image_mask=None*, *fov_pix=None*, *oversample=None*, ***kwargs*)

Methods

__init__([filter, pupil_mask, image_mask, ...])	

calc_datacube(wavelengths, *args, **kwargs)	Calculate a spectral datacube of PSFs
calc_psf([add_distortion, fov_pixels, ...])	Compute a PSF
calc_psf_from_coeff([sp, return_oversample, ...])	Create PSF image from polynomial coefficients
calc_psfs_grid([sp, wfe_drift, osamp, ...])	PSF grid across an instrumnet FoV
calc_psfs_sgd(xoff_asec, yoff_asec[, use_coeff])	Calculate small grid dither PSFs
display()	Display the currently configured optical system on screen
drift_opd(wfe_drift[, opd])	A quick method to drift the pupil OPD.
gen_mask_image([npix, pixelscale, ...])	Return an image representation of the focal plane mask attenuation.
gen_psf_coeff([bar_offset])	Generate PSF coefficients
gen_save_name([wfe_drift])	Generate save name for polynomial coefficient output file.
gen_wfedrift_coeff([force, save])	Fit WFE drift coefficients
gen_wfefield_coeff([force, save])	Fit WFE field-dependent coefficients
gen_wfemask_coeff([large_grid, force, save])	Fit WFE changes in mask position
get_bar_offset([narrow])	Obtain the value of the bar offset that would be passed through to PSF calculations for bar/wedge coronagraphic masks.
get_opd_file_full_path([opdfilename])	Return full path to the named OPD file.
get_opd_info([opd, pupil, HDUL_to_OTELM])	Parse out OPD information for a given OPD, which can be a file name, tuple (file,slice), HDUList, or OTE Linear Model.
get_optical_system([fft_oversample, ...])	Return an OpticalSystem instance corresponding to the instrument as currently configured.
interpolate_was_opd(array, newdim)	Interpolates an input 2D array to any given size.
load_was_opd(inputWasOpd[, size, save, filename])	Load and interpolate an OPD from the WAS.
psf_grid([num_psfs, all_detectors, save, ...])	Create a PSF library in the form of a grid of PSFs across the detector based on the specified instrument, filter, and detector.
set_position_from_aperture_name(aperture_name)	Set the simulated center point of the array based on a named SIAF aperture.
visualize_wfe_budget([slew_delta_time, ...])	Display a visual WFE budget showing the various terms that sum into the overall WFE for a given instrument

Attributes

LONG_WAVELENGTH_MAX	
LONG_WAVELENGTH_MIN	
ND_acq	Use Coronagraphic ND acquisition square?
SHORT_WAVELENGTH_MAX	
SHORT_WAVELENGTH_MIN	
aperturename	SIAF aperture name for detector pixel to sky coords transformations
bandpass	Return bandpass throughput
channel	
coron_substrate	Include coronagraphic substrate material?
detector	Detector selected for simulated PSF
detector_list	Detectors on which the simulated PSF could lie
detector_position	The pixel position in (X, Y) on the detector, relative to the currently-selected SIAF aperture subarray.
fastaxis	Fast readout direction in sci coords
filter	Currently selected filter name (e.g.
filter_list	List of available filter names for this instrument
fov_pix	
image_mask	Currently selected image plane mask, or None for direct imaging
is_coron	Observation with coronagraphic mask (incl Lyot stop)?
is_dark	
is_grism	
is_lyot	Is a Lyot mask in the pupil wheel?
module	
name	
ndeg	
npsf	Number of wavelengths/PSFs to fit
options	A dictionary capable of storing other arbitrary options, for extensibility.
oversample	
pixelscale	Detector pixel scale, in arcsec/pixel
pupil	Filename *or* fits.HDUList for JWST pupil mask.
pupil_mask	Currently selected Lyot pupil mask, or None for direct imaging
pupilopd	Filename *or* fits.HDUList for JWST pupil OPD.

continues on next page

Table 70 – continued from previous page

quick	Perform quicker coeff calculation over limited bandwidth?
save_dir	Coefficient save directory
save_name	Coefficient file name
scaid	SCA ID (481, 482, .
siaf_ap	SIAF Aperture object
slowaxis	Slow readout direction in sci coords
telescope	
wave_fit	Wavelength range to fit

property ND_acq
Use Coronagraphic ND acquisition square?

property aperturename
SIAF aperture name for detector pixel to sky coords transformations

property bandpass
Return bandpass throughput

calc_datacube(*wavelengths*, **args*, ***kwargs*)
Calculate a spectral datacube of PSFs

Parameters **wavelengths** (*iterable of floats*) – List or ndarray or tuple of floating point wavelengths in meters, such as you would supply in a call to calc_psf via the "monochromatic" option

calc_psf(*add_distortion=None*, *fov_pixels=None*, *oversample=None*, *wfe_drift=None*, *coord_vals=None*, *coord_frame='tel'*, ***kwargs*)
Compute a PSF

Slight modification of inherent WebbPSF *calc_psf* function. If add_distortion, fov_pixels, and oversample are not specified, then we automatically use the associated attributes. Also, add ability to directly specify wfe_drift and coordinate offset values in the same fashion as *calc_psf_from_coeff*.

Notes

Additional PSF computation options (pupil shifts, source positions, jitter, ...) may be set by configuring the *.options* dictionary attribute of this class.

Parameters

- **sp** (pysynphot.spectrum) – Source input spectrum. If not specified, the default is flat in phot lam. (equal number of photons per spectral bin).
- **source** (*synphot.spectrum.SourceSpectrum or dict*) – TODO: synphot not yet implemented in webbpsf_ext!!
- **nlambda** (*int*) – How many wavelengths to model for broadband? The default depends on how wide the filter is: (5,3,1) for types (W,M,N) respectively
- **monochromatic** (*float, optional*) – Setting this to a wavelength value (in meters) will compute a monochromatic PSF at that wavelength, overriding filter and nlambda settings.
- **fov_arcsec** (*float*) – field of view in arcsec. Default=5
- **fov_pixels** (*int*) – field of view in pixels. This is an alternative to fov_arcsec.
- **outfile** (*string*) – Filename to write. If None, then result is returned as an HDUList

- **oversample, detector_oversample, fft_oversample** (*int*) – How much to oversample. Default=4. By default the same factor is used for final output pixels and intermediate optical planes, but you may optionally use different factors if so desired.
- **overwrite** (*bool*) – overwrite output FITS file if it already exists?
- **display** (*bool*) – Whether to display the PSF when done or not.
- **save_intermediates, return_intermediates** (*bool*) – Options for saving to disk or returning to the calling function the intermediate optical planes during the propagation. This is useful if you want to e.g. examine the intensity in the Lyot plane for a coronagraphic propagation.
- **normalize** (*string*) – Desired normalization for output PSFs. See doc string for OpticalSystem.calc_psf. Default is to normalize the entrance pupil to have integrated total intensity = 1.
- **add_distortion** (*bool*) – If True, will add 2 new extensions to the PSF HDUlist object. The 2nd extension will be a distorted version of the over-sampled PSF and the 3rd extension will be a distorted version of the detector-sampled PSF.
- **crop_psf** (*bool*) – If True, when the PSF is rotated to match the detector's rotation in the focal plane, the PSF will be cropped so the shape of the distorted PSF will match it's undistorted counterpart. This will only be used for NIRCam, NIRISS, and FGS PSFs.

Keyword Arguments

- **return_hdul** (*bool*) – Return PSFs in an HDUList rather than set of arrays (default: True).
- **return_oversample** (*bool*) – Returns the oversampled version of the PSF instead of detector-sampled PSF. Only valid for *reaturn_hdul=False*, otherwise full HDUList returned. Default: True.

calc_psf_from_coeff(*sp=None*, *return_oversample=True*, *wfe_drift=None*, *coord_vals=None*, *coord_frame='tel'*, *coron_rescale=False*, *return_hdul=True*, ***kwargs*)

Create PSF image from polynomial coefficients

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).

Returns a single image or list of images if sp is a list of spectra. By default, it returns only the oversampled PSF, but setting return_oversample=False will instead return detector-sampled images.

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*) – Returns the oversampled version of the PSF instead of detector-sampled PSF. Default: True.
- **wfe_drift** (*float or None*) – Wavefront error drift amplitude in nm.
- **coord_vals** (*tuple or None*) – Coordinates (in arcsec or pixels) to calculate field-dependent PSF. If multiple values, then this should be an array ([xvals], [yvals]).
- **coord_frame** (*str*) –

  Type of input coordinates.

- 'tel': arcsecs V2,V3
- 'sci': pixels, in DMS axes orientation; aperture-dependent
- 'det': pixels, in raw detector read out axes orientation
- 'idl': arcsecs relative to aperture reference location.

- **return_hdul** (*bool*) – Return PSFs in an HDUList rather than set of arrays (default: True).
- **coron_rescale** (*bool*) – Rescale off-axis coronagraphic PSF to better match analytic prediction when source overlaps coronagraphic occulting mask. Primarily used for planetary companion PSFs.

calc_psfs_grid(*sp=None*, *wfe_drift=0*, *osamp=1*, *npsf_per_full_fov=15*, *xsci_vals=None*, *ysci_vals=None*, *return_coords=None*, *use_coeff=True*, ***kwargs*)

PSF grid across an instrumnet FoV

Create a grid of PSFs across instrument aperture FoV. By default, imaging observations will be for full detector FoV with regularly spaced grid. Coronagraphic observations will cover nominal coronagraphic mask region (usually 10s of arcsec) and will have logarithmically spaced values where appropriate.

Keyword Arguments

- **sp** (pysynphot.spectrum) – If not specified, the default is flat in phot lam (equal number of photons per wavelength 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 suppression of coronagraphic mask. If set, then the resulting PSF image will be scaled to generate the total observed number of photons from the spectrum (ie., not scaled by unit response).
- **wfe_drift** (*float*) – Desired WFE drift value relative to default OPD.
- **osamp** (*int*) – Sampling of output PSF relative to detector sampling.
- **npsf_per_full_fov** (*int*) – Number of PSFs across one dimension of the instrument's field of view. If a coronagraphic observation, then this is for the nominal coronagrahic field of view (20"x20").
- **xsci_vals** (*None or ndarray*) – Option to pass a custom grid values along x-axis in 'sci' coords.
- **ysci_vals** (*None or ndarray*) – Option to pass a custom grid values along y-axis in 'sci' coords.
- **return_coords** (*None or str*) – Option to also return coordinate values in desired frame ('det', 'sci', 'tel', 'idl'). Output is then xvals, yvals, hdul_psfs.
- **use_coeff** (*bool*) – If True, uses *calc_psf_from_coeff*, other WebbPSF's built-in *calc_psf*.
- **coron_rescale** (*bool*) – Rescale off-axis coronagraphic PSF to better match analytic prediction when source overlaps coronagraphic occulting mask. Only valid for use_coeff=True.

calc_psfs_sgd(*xoff_asec*, *yoff_asec*, *use_coeff=True*, ***kwargs*)

Calculate small grid dither PSFs

Convenience function to calculation a series of SGD PSFs. This is essentially a wrapper around the *calc_psf_from_coeff* and *calc_psf* functions. Only valid for coronagraphic observations.

Parameters

- **xoff_asec** (*float or array-like*) – Offsets in x-direction (in 'idl' coordinates).

- **yoff_asec** (*float or array-like*) – Offsets in y-direction (in 'idl' coordinates).
- **use_coeff** (*bool*) – If True, uses *calc_psf_from_coeff*, other WebbPSF's built-in *calc_psf*.

property coron_substrate
Include coronagraphic substrate material?

property detector
Detector selected for simulated PSF

Used in calculation of field-dependent aberrations. Must be selected from detectors in the *detector_list* attribute.

property detector_list
Detectors on which the simulated PSF could lie

property detector_position
The pixel position in (X, Y) on the detector, relative to the currently-selected SIAF aperture subarray. By default the SIAF aperture will correspond to the full-frame detector, so (X,Y) will in that case be absolute (X,Y) pixels on the detector. But if you select a subarray aperture name from the SIAF, then the (X,Y) are interpreted as (X,Y) within that subarray.

Please note, this is X,Y order - **not** a Pythonic y,x axes ordering.

display()
Display the currently configured optical system on screen

drift_opd(*wfe_drift*, *opd=None*)
A quick method to drift the pupil OPD. This function applies some WFE drift to input OPD file by breaking up the wfe_drift attribute into thermal, frill, and IEC components. If we want more realistic time evolution, then we should use the procedure in dev_utils/WebbPSF_OTE_LM.ipynb to create a time series of OPD maps, which can then be passed directly to create unique PSFs.

This outputs an OTE Linear Model. In order to update instrument class:

```
>>> opd_dict = inst.drift_opd()
>>> inst.pupilopd = opd_dict['opd']
>>> inst.pupil = opd_dict['opd']
```

property fastaxis
Fast readout direction in sci coords

property filter
Currently selected filter name (e.g. F200W)

gen_mask_image(*npix=None*, *pixelscale=None*, *bar_offset=None*, *nd_squares=True*)
Return an image representation of the focal plane mask attenuation. Output is in 'sci' coords orientation. If no image mask is present, then returns an array of all 1s. Mask is centered in image, while actual subarray readout has a slight offset.

Parameters

- **npix** (*int*) – Number of pixels in output image. If not set, then is automatically determined based on mask FoV and *pixelscale*
- **pixelscale** (*float*) – Size of output pixels in units of arcsec. If not specified, then selects oversample pixel scale.

gen_psf_coeff(*bar_offset=0*, ***kwargs*)
Generate PSF coefficients

Creates a set of coefficients that will generate simulated PSFs for any arbitrary wavelength. This function first simulates a number of evenly- spaced PSFs throughout the specified bandpass (or the full channel).

An nth-degree polynomial is then fit to each oversampled pixel using a linear-least squares fitting routine. The final set of coefficients for each pixel is returned as an image cube. The returned set of coefficient are then used to produce PSF via *calc_psf_from_coeff*.

Useful for quickly generated imaging and dispersed PSFs for multiple spectral types.

Parameters **bar_offset** (*float*) – For wedge masks, option to set the PSF position across the bar. In this framework, we generally set the default to 0, then use the *gen_wfemask_coeff* function to determine how the PSF changes along the wedge axis as well as perpendicular to the wedge. This allows for more arbitrary PSFs within the mask, including small grid dithers as well as variable PSFs for extended objects. Default: 0.

Keyword Arguments

- **wfe_drift** (*float*) – Wavefront error drift amplitude in nm.
- **force** (*bool*) – Forces a recalculation of PSF even if saved PSF exists. (default: False)
- **save** (*bool*) – Save the resulting PSF coefficients to a file? (default: True)
- **nproc** (*bool or None*) – Manual setting of number of processor cores to break up PSF calculation. If set to None, this is determined based on the requested PSF size, number of available memory, and hardware processor cores. The automatic calculation endeavors to leave a number of resources available to the user so as to not crash the user's machine.
- **return_results** (*bool*) – By default, results are saved as object the attributes *psf_coeff* and *psf_coeff_header*. If return_results=True, results are instead returned as function outputs and will not be saved to the attributes. This is mostly used for successive coeff simulations to determine varying WFE drift or focal plane dependencies.
- **return_extras** (*bool*) – Additionally returns a dictionary of monochromatic PSFs images and their corresponding wavelengths for debugging purposes. Can be used with or without *return_results*. If *return_results=False*, then only this dictionary is returned, otherwise if *return_results=False* then returns everything as a 3-element tuple (psf_coeff, psf_coeff_header, extras_dict).

gen_save_name(*wfe_drift=0*)
Generate save name for polynomial coefficient output file.

gen_wfedrift_coeff(*force=False*, *save=True*, ***kwargs*)
Fit WFE drift coefficients

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.

It's Legendre's all the way down...

Parameters

- **force** (*bool*) – Forces a recalculation of coefficients even if saved file exists. (default: False)
- **save** (*bool*) – Save the resulting PSF coefficients to a file? (default: True)

Keyword Arguments

- **wfe_list** (*array-like*) – A list of wavefront error drift values (nm) to calculate and fit. Default is [0,1,2,5,10,20,40], which covers the most-likely scenarios (1-5nm) while also covering a range of extreme drift values (10-40nm).

- **return_results** (*bool*) – By default, results are saved in *self._psf_coeff_mod* dictionary. If return_results=True, results are instead returned as function outputs and will not be saved to the dictionary attributes.
- **return_raw** (*bool*) – Normally, we return the relation between PSF coefficients as a function of position. Instead this returns (as function outputs) the raw values prior to fitting. Final results will not be saved to the dictionary attributes.

gen_wfefield_coeff(*force=False*, *save=True*, ***kwargs*)
Fit WFE field-dependent coefficients

Find a relationship between field position and PSF coefficients for non-coronagraphic observations and when *include_si_wfe* is enabled.

Parameters

- **force** (*bool*) – Forces a recalculation of coefficients even if saved file exists. (default: False)
- **save** (*bool*) – Save the resulting PSF coefficients to a file? (default: True)

Keyword Arguments

- **return_results** (*bool*) – By default, results are saved in *self._psf_coeff_mod* dictionary. If return_results=True, results are instead returned as function outputs and will not be saved to the dictionary attributes.
- **return_raw** (*bool*) – Normally, we return the relation between PSF coefficients as a function of position. Instead this returns (as function outputs) the raw values prior to fitting. Final results will not be saved to the dictionary attributes.

gen_wfemask_coeff(*large_grid=False*, *force=False*, *save=True*, ***kwargs*)
Fit WFE changes in mask position

For coronagraphic masks, slight changes in the PSF location relative to the image plane mask can substantially alter the PSF speckle pattern. This function generates a number of PSF coefficients at a variety of positions, then fits polynomials to the residuals to track how the PSF changes across the mask's field of view. Special care is taken near the 10-20mas region in order to provide accurate sampling of the SGD offsets.

Parameters

- **large_grid** (*bool*) – Use a large number (high-density) of grid points to create coefficients. If True, then a higher fidelity PSF variations across the FoV, but could take hours to generate on the first pass. Setting to False allows for quicker coefficient creation with a smaller memory footprint, useful for testing and debugging.
- **force** (*bool*) – Forces a recalculation of coefficients even if saved file exists. (default: False)
- **save** (*bool*) – Save the resulting PSF coefficients to a file? (default: True)

Keyword Arguments

- **return_results** (*bool*) – By default, results are saved in *self._psf_coeff_mod* dictionary. If return_results=True, results are instead returned as function outputs and will not be saved to the dictionary attributes.
- **return_raw** (*bool*) – Normally, we return the relation between PSF coefficients as a function of position. Instead this returns (as function outputs) the raw values prior to fitting. Final results will not be saved to the dictionary attributes.

get_bar_offset(*narrow=False*)
Obtain the value of the bar offset that would be passed through to PSF calculations for bar/wedge coronagraphic masks.

get_opd_file_full_path(*opdfilename=None*)
Return full path to the named OPD file.

The OPD may be:

- a local or absolute path,
- or relative implicitly within an SI directory, e.g. $WEBBPSF_PATH/NIRCam/OPD
- or relative implicitly within $WEBBPSF_PATH

This function handles filling in the implicit path in the latter cases.

get_opd_info(*opd=None*, *pupil=None*, *HDUL_to_OTELM=True*)
Parse out OPD information for a given OPD, which can be a file name, tuple (file,slice), HDUList, or OTE Linear Model. Returns dictionary of some relevant information for logging purposes. The dictionary returns the OPD as an OTE LM by default.

This outputs an OTE Linear Model. In order to update instrument class:

```
>>> opd_dict = inst.get_opd_info()
>>> opd_new = opd_dict['pupilopd']
>>> inst.pupilopd = opd_new
>>> inst.pupil = opd_new
```

get_optical_system(*fft_oversample=2*, *detector_oversample=None*, *fov_arcsec=2*, *fov_pixels=None*, *options=None*)
Return an OpticalSystem instance corresponding to the instrument as currently configured.

When creating such an OpticalSystem, you must specify the parameters needed to define the desired sampling, specifically the oversampling and field of view.

Parameters

- **fft_oversample** (*int*) – Oversampling factor for intermediate plane calculations. Default is 2
- **detector_oversample** (*int, optional*) – By default the detector oversampling is equal to the intermediate calculation oversampling. If you wish to use a different value for the detector, set this parameter. Note that if you just want images at detector pixel resolution you will achieve higher fidelity by still using some oversampling (i.e. *not* setting *oversample_detector=1*) and instead rebinning down the oversampled data.
- **fov_pixels** (*float*) – Field of view in pixels. Overrides fov_arcsec if both set.
- **fov_arcsec** (*float*) – Field of view, in arcseconds. Default is 2

Returns **osys** (*poppy.OpticalSystem*) – an optical system instance representing the desired configuration.

property image_mask
Currently selected image plane mask, or None for direct imaging

image_mask_list
List of available image_masks

include_ote_field_dependence
Should calculations include the Science Instrument internal WFE?

interpolate_was_opd(*array*, *newdim*)
Interpolates an input 2D array to any given size.

Parameters

- **array** (*float*) – input array to interpolate
- **newdim** (*int*) – new size of the 2D square array (newdim x newdim)

Returns **newopd** (*new array interpolated to (newdim x newdim)*)

`property is_coron`
Observation with coronagraphic mask (incl Lyot stop)?

`property is_lyot`
Is a Lyot mask in the pupil wheel?

`load_was_opd`(*inputWasOpd*, *size=1024*, *save=False*, *filename='new_was_opd.fits'*)
Load and interpolate an OPD from the WAS.

Ingests a WAS OPD and interpolates it to the proper size for WebbPSF.

Parameters

- **HDUlist_or_filename** (*string*) – Either a fits.HDUList object or a filename of a FITS file on disk
- **size** (*int, optional*) – Desired size of the output OPD. Default is 1024.
- **save** (*bool, optional*) – Save the interpolated OPD if True. Default is False.
- **filename** (*string, optional*) – Filename of the output OPD, if 'save' is True. Default is 'new_was_opd.fits'.

Returns **HDUlist** (*string*) – fits.HDUList object of the interpolated OPD

`property npsf`
Number of wavelengths/PSFs to fit

`psf_grid`(*num_psfs=16*, *all_detectors=True*, *save=False*, *outdir=None*, *outfile=None*, *overwrite=True*, *verbose=True*, *use_detsampled_psf=False*, *single_psf_centered=True*, ***kwargs*)
Create a PSF library in the form of a grid of PSFs across the detector based on the specified instrument, filter, and detector. The output GriddedPSFModel object will contain a 3D array with axes [i, y, x] where i is the PSF position on the detector grid and (y,x) is the 2D PSF.

Parameters

- **num_psfs** (*int*) – The total number of fiducial PSFs to be created and saved in the files. This number must be a square number. Default is 16. E.g. num_psfs = 16 will create a 4x4 grid of fiducial PSFs.
- **all_detectors** (*bool*) – If True, run all detectors for the instrument. If False, run for the detector set in the instance. Default is True
- **save** (*bool*) – True/False boolean if you want to save your file. Default is False.
- **outdir** (*str*) – If "save" keyword is set to True, your file will be saved in the specified directory. Default of None will save it in the current directory
- **outfile** (*str*) – If "save" keyword is set to True, your file will be saved as {outfile}_det.fits. Default of None will save it as instr_det_filt_fovp#_samp#_npsf#.fits
- **overwrite** (*bool*) – True/False boolean to overwrite the output file if it already exists. Default is True.
- **verbose** (*bool*) – True/False boolean to print status updates. Default is True.
- **use_detsampled_psf** (*bool*) – If True, the grid of PSFs returned will be detector sampled (made by binning down the oversampled PSF). If False, the PSFs will be oversampled by the

factor defined by the oversample/detector_oversample/fft_oversample keywords. Default is False. This is rarely needed - if uncertain, leave this alone.

- **single_psf_centered** (*bool*) – If num_psfs is set to 1, this defines where that psf is located. If True it will be the center of the detector, if False it will be the location defined in the WebbPSF attribute detector_position (reminder - detector_position is (x,y)). Default is True This is also rarely needed.
- ****kwargs** – Any extra arguments to pass the WebbPSF calc_psf() method call.

Returns

- **gridmodel** (*photutils GriddedPSFModel object or list of objects*) – Returns a GriddedPSFModel object or a list of objects if more than one configuration is specified (1 per instrument, detector, and filter) User also has the option to save the grid as a fits.HDUlist object.
- *Use*
- *—-*
- *nir = webbpsf.NIRCam()*
- *nir.filter = "F090W"*
- *list_of_grids = nir.psf_grid(all_detectors=True, num_psfs=4)*
- *wfi = webbpsf.WFI()*
- *wfi.filter = "Z087"*
- *wfi.detector = "SCA02"*
- *grid = wfi.psf_grid(all_detectors=False, oversample=5, fov_pixels=101)*

property pupil_mask
Currently selected Lyot pupil mask, or None for direct imaging

pupil_mask_list
List of available pupil_masks

property quick
Perform quicker coeff calculation over limited bandwidth?

property save_dir
Coefficient save directory

property save_name
Coefficient file name

property scaid
SCA ID (481, 482, … 489, 490)

set_position_from_aperture_name(*aperture_name*)
Set the simulated center point of the array based on a named SIAF aperture. This will adjust the detector and detector position attributes.

property siaf_ap
SIAF Aperture object

property slowaxis
Slow readout direction in sci coords

visualize_wfe_budget(*slew_delta_time=<Quantity 14. d>*, *slew_case='EOL'*, *ptt_only=False*, *verbose=True*)

Display a visual WFE budget showing the various terms that sum into the overall WFE for a given instrument

Compares a WebbPSF instrument instance with the JWST optical budget for that instrument

Parameters

- **inst** (*webbpsf.JWInstrument*) – A JWST instrument instance
- **slew_delta_time** (*astropy.Quantity time*) – Time duration for thermal slew model
- **slew_case** (*basestring*) – 'BOL' or 'EOL' for beginning of life or end of life thermal slew model. EOL is about 3x higher amplitude
- **ptt_only** (*bool*) – When decomposing wavefront into controllable modes, use a PTT-only basis? The default is to use all controllable pose modes. (This is mostly a leftover debug option at this point, not likely useful in general)
- **verbose** (*bool*) – Be more verbose

property wave_fit

Wavelength range to fit

1.10 Revision History

1.10.1 v1.0.4 (Dec 28, 2021)

- check if `im_star` is int or float if not None
- set `nexposures = 1` for level1b using `NIRCam()` class function
- deprecate nghxrg.py
- add tutorial ipynb files
- update api docs auto generation
- use `webbpsf_ext` v1.0.4

1.10.2 v1.0.3 (Dec 23, 2021)

- Minor updates to seamlessly generate new releases on PyPI and new docs on readthedocs

1.10.3 v1.0.1 (Dec 14, 2021)

- Default OPD JWST_OTE_OPD_RevAA_prelaunch_predicted.fits

1.10.4 v1.0.0 (Nov 22, 2021)

- Updates to work with WebbPSF v1 release candidate
- Move PSF generation to new `webbpsf_ext` package (https://github.com/JarronL/webbpsf_ext)
- Create DMS-like level1b FITS files using pipeline data models for imaging and coronagraphy
- PSF coefficients now use Legendre polynomials by default
- Create calibration files for each SCA (darks, IPC, noise, flats, linearity, etc)
- Background roll-off at grism edges
- SIAF-aware locations

1.10.5 v0.9.0beta (no release)

- Updates to work with WebbPSF 0.9.0.
- Start working on commissioning and DMS-like data
- Add more advanced time-dependent detector effects
- BEX model isochrones for low-mass companions from Linder et al (2019)
- There was a pandemic…

1.10.6 v0.8.0beta (no release)

- Updates 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

1.10.7 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)

1.10.8 v0.6.5 (Mar 2018)

- Fixed a critical bug where the off-axis PSF size was incorrect when performing WFE drift calculations.

1.10.9 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.

1.10.10 v0.6.3 (Mar 2018)

- First PyPI release.
- Effectively the same as 0.6.2, but better documentation of packaging and distributing.

1.10.11 v0.6.2 (Mar 2018)

- Implemented coronagraphic wedges, including arbitrary offsets along bar
- Renamed `obs_coronagraphy` to `~pynrc.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.

1.10.12 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

1.10.13 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

1.10.14 v0.1.2 (Jan 2017)

- Observations subclass for coronagraphs and direct imaging

1.10.15 v0.1.1 (Sep 2016)

- Add support for LW slitless grism
- Add support for extended sources

1.10.16 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

1.10.17 Planned Updates

FoV aware positions

- Correct coronagraph field locations depending on Lyot optical wedge
- Filter location relative offsets

Detector updates in ngNRC.py

- Pixel glow (dark current) based on subarray size
- Charge diffusion (esp for saturated pixels)
- Persistence/latent image
- Optical distortions
- QE variations across a pixel's surface

PSF Related

- 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

- DHS mode

1.11 License

1.12 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:

1.12.1 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 :)

1.12.2 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 3.7, 3.8, and 3.9 for PyPy. Check https://travis-ci.org/JarronL/pynrc/pull_requests and make sure that the tests pass for all supported Python versions.

1.12.3 Tips

To run a subset of tests:

```
$ py.test tests.test_pynrc
```

1.12.4 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
$ conda install docutils=0.16
$ conda install bump2version
```

1. Add entries to HISTORY.rst. Make sure all your changes are committed to git.
2. Update version using `bumpversion`, which automatically updates `pynrc.version`. Usage: `bump2version [options] part [file]`, where "part" is either major, minor, or patch (e.g., major.minor.patch). See https://github.com/c4urself/bump2version for more details.

   ```
   $ bumpversion patch
   ```

3. Generate documentation locally:

   ```
   $ make docs
   ```

4. Push all updates to github and make sure readthedocs generates correctly before actually submitting the release.
5. Package a distribution and test upload the release to TestPyPI:

   ```
   $ make release-test
   ```

6. 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:

```
openssl dgst -sha256 path/to/package_name-0.1.1.tar.gz
```

Then, commit and push the yaml file to GitHub:

```
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*.

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).

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

w

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z