
Eureka!

Eureka! pipeline developers

Apr 19, 2022

CONTENTS:

1	Installation	3
1.1	Initial environment preparation	3
1.2	Installation methods	3
1.3	CRDS Environment Variables	4
1.4	Issues with installing the jwst dependency	5
2	Eureka! Quickstart	7
2.1	1. Installation	7
2.2	2. Download the data	7
2.3	3. Set up your run directory	7
2.4	4. Run Eureka!	9
2.5	5. Where to go next	13
3	Eureka! Control File (.ecf)	15
3.1	Stage 1	15
3.2	Stage 2	17
3.3	Stage 3	19
3.4	Stage 4	24
3.5	Stage 5	27
3.6	Stage 5 Fit Parameters	31
4	Eureka! Outputs	35
4.1	Stage 2 Outputs	35
4.2	Stage 3 Outputs	35
4.3	Stage 4 Outputs	42
4.4	Stage 5 Outputs	42
5	Contributing to Eureka!	53
5.1	Page information	53
5.2	Testing Eureka!	53
5.3	GitHub Basics	54
6	The Code (API)	61
6.1	lib	61
6.2	S1_calibrations	81
6.3	S2_calibrations	81
6.4	S3_data_reduction	81
6.5	S4_generate_lightcurves	105
6.6	S5_lightcurve_fitting	108
6.7	S6_planet_spectra	125

7	Eureka! FAQ	127
7.1	Common Errors	127
8	Copyright	129
9	Introduction	131
9.1	Goals	131
9.2	Import standard python packages and Eureka!	131
10	Indices and tables	137
	Python Module Index	139
	Index	141

Welcome to the documentation for Eureka!.

Eureka! will eventually be capable of reducing data from any JWST instrument and fitting light curves. At the moment the package is under heavy development, and currently works on NIRCam, NIRSpec, and MIRI data only. The code is not officially associated with JWST or the ERS team.

The code is separated into five parts or “Stages”:

- Stage 1: An optional step that calibrates Raw data (converts ramps to slopes). This step can be skipped if you’d rather use STScI’s JWST pipeline’s Stage 1 outputs.
- Stage 2: An optional step which calibrates Stage 1 data (performs flatfielding, unit conversion, etc.). This step can be skipped if you’d rather use STScI’s JWST pipeline’s Stage 2 outputs.
- Stage 3: Starts with Stage 2 data and reduces the data (performs background subtraction, etc.) in order to convert 2D spectra into a time-series of 1D spectra
- Stage 4: Bins the 1D Spectra and generates light curves
- Stage 5: Fits the light curves (under development)

The full code for Eureka! is available on [GitHub](#)

INSTALLATION

1.1 Initial environment preparation

It is **strongly** recommended that you install Eureka! in a new conda environment as other packages you've previously installed could have conflicting requirements with Eureka!. You can install a lightweight version of conda at [this link](#). Once conda is installed, you can create a new environment by doing:

```
conda create -n eureka python==3.9.7
conda activate eureka
```

1.2 Installation methods

1.2.1 a) With git and conda

While Eureka! is under heavy development, the most stable way of installing Eureka! is using git and conda. This can be done following:

```
git clone https://github.com/kevin218/Eureka.git
cd Eureka
conda env create --file environment.yml --force
conda activate eureka
pip install --no-deps .
```

To update your Eureka! installation to the most recent version, you can do the following within that Eureka folder

```
git pull
conda env update --file environment.yml --prune
pip install --no-deps --upgrade .
```

1.2.2 b) With pip

Once in your new conda environment, you can install the Eureka! package with pip with the following command:

```
pip install git+https://github.com/kevin218/Eureka.git#egg=eureka[jwst]
```

where specific branches can be installed using:

```
pip install git+https://github.com/kevin218/Eureka.git@mybranchname#egg=eureka[jwst]
```

If you desire any of the files in the [demos folder](#), you will have to download these from GitHub following the method described below.

To update your Eureka! installation to the most recent version, you can do then do the following

```
pip install --upgrade git+https://github.com/kevin218/Eureka.git#egg=eureka[jwst]
```

1.2.3 c) With git and pip

Once in your new conda environment, you can install Eureka! directly from source on [GitHub](#) using git and pip by running:

```
git clone https://github.com/kevin218/Eureka.git
cd Eureka
pip install .[jwst]
```

To update your Eureka! installation to the most recent version, you can do the following within that Eureka folder

```
git pull
pip install --upgrade .[jwst]
```

1.3 CRDS Environment Variables

Eureka! installs the JWST Calibration Pipeline as part of its requirements, and this also requires users to set the proper environment variables so that it can download the proper reference files needed to run the pipeline. For users not on the internal STScI network, two environment variables need to be set to enable this functionality. In your `~/ .zshrc` (for Mac users) or `~/ .bashrc` file (for bash users), or other shell initialization file, add these two lines (specifying your desired location to cache the CRDS files, e.g. `/Users/your_name/crds_cache` for Mac users or `/home/your_name/crds_cache` for Linux users):

```
export CRDS_PATH=/PATH/TO/FOLDER/crds_cache
export CRDS_SERVER_URL=https://jwst-crds.stsci.edu
```

If these environment variables are not set, Stages 1-3 of the pipeline will fail.

1.4 Issues with installing the jwst dependency

If you have issues installing the jwst dependency, check out the debugging advice related to the jwst package on our *FAQ page*.

EUREKA! QUICKSTART

Want to get up and running with Eureka!, but not really sure where to begin? Keep reading!

2.1 1. Installation

The first thing you need to do is install the package, so if you haven't already, take a break from this page and follow the [Installation](#) instructions (if you have issues be sure to visit the [FAQ](#) first).

2.2 2. Download the data

With the installation complete, you'll need some data to run Eureka! on. For now let's use some simulated data that was produced for the [Transiting Exoplanet Community ERS Data Challenge](#). Datasets for all four instruments are available on the [STScI Box site](#), however, for the rest of this quickstart guide the [NIRSpec Tiny dataset](#) will be used.

Now, I'm sure you wouldn't just leave the data in your Downloads folder, but if so, let's make a new directory to store things instead. For example:

```
mkdir /User/Data/JWST-Sim/NIRSpec/  
cd /User/Data/JWST-Sim/NIRSpec/  
unzip -j ~/Downloads/Tiny.zip -d .
```

Note that for Eureka! you do *not* need to download any ancillary data - any additional files will be downloaded automatically (if you correctly set the CRDS environment variables during installation).

2.3 3. Set up your run directory

2.3.1 3.1 Gather the demo files

We're almost there, but before you can get things running you need to set up a directory for Eureka! to store both input and output files.

```
mkdir /User/DataAnalysis/JWST/MyFirstEureka  
cd /User/DataAnalysis/JWST/MyFirstEureka
```

From here, the simplest way to set up all of the Eureka input files is to download them from the JWST demos directory on the Github repository ([direct download](#)). Then we can copy them over:

```
mkdir demos
unzip -j ~/Downloads/JWST.zip -d ./demos
```

This demos directory contains a selection of template files to run Eureka!. There are three different types of files:

- *.ecf: These are Eureka! control files and contain input parameters required to run each stage of the pipeline. For more detail on the ecf parameters for each stage, see [here](#).
- *.epf: This is a Eureka! parameter file and describes the initial guesses and priors to be used when performing light curve fitting (Stage 5).
- run_eureka.py: A script to run the Eureka! pipeline.

2.3.2 3.2 Customise the demo files

You might notice that not all of the demo files will be applicable to every dataset, either because they are tailored to a specific instrument, or because they are for a Eureka! pipeline stage that precedes the input data. This is the case for the NIRSpec data being used here, which as a `*rateints.fits` file (more information on JWST pipeline products [here](#)) has already been processed through an equivalent to Stage 1 of Eureka!.

So, let's only copy over the specific files needed to process this NIRSpec dataset further. Given that the dataset contains a transit for WASP-39b, let's also change some of the default filenames to something a little more informative:

```
cp demos/run_eureka.py .
cp demos/S2_nirspec_fs_template.ecf S2_wasp39b.ecf
cp demos/S3_nirspec_fs_template.ecf S3_wasp39b.ecf
cp demos/S4_template.ecf S4_wasp39b.ecf
cp demos/S5_template.ecf S5_wasp39b.ecf
cp demos/S5_fit_par_template.epf S5_fit_par_wasp39b.ecf
cp demos/S6_template.ecf S6_wasp39b.ecf
```

Notice that all of the *.ecf files have a common `wasp39b` string. It's useful to keep this homogenous across files as it is what Eureka! interprets as an “event label”, and is used to locate specific input files when running the pipeline. To see this more clearly, open up the `run_eureka.py` file and look at how the individual stages are being called. While you're here, modify the `eventlabel` string directly to match the chosen naming:

```
eventlabel = 'wasp39b'
```

Finally, you need to connect everything together by opening up each .ecf file and updating the `topdir`, `inputdir`, and `outputdir` parameters within. For the `S2_wasp39b.ecf`, you want something like:

```
topdir      /User
inputdir    /Data/JWST-Sim/NIRSpec
outputdir   /DataAnalysis/JWST/MyFirstEureka/Stage2
```

However, for the later stages you can use something simpler, e.g. for the `S3_wasp39b.ecf`:

```
topdir      /User/DataAnalysis/JWST/MyFirstEureka
inputdir    /Stage2
outputdir   /Stage3
```

The explicit settings for the `S4_wasp39b.ecf`, `S5_wasp39b.ecf` and `S6_wasp39b.ecf` will be skipped here for brevity (but you should still do them!). However, it is important to notice a few settings in the `S5_wasp39b.ecf`. Specifically, you need to assign the correct .epf file, and modify the number of processors you want to use during the light curve fitting.


```
ncpu          4
fit_par       S5_fit_par_wasp39b.epf
```

While editing those files you may have noticed that there are a whole range of other inputs that can be tweaked and adjusted at each different stage. For now you can ignore these, as the demo files have been specifically tailored to this simulated dataset of WASP-39b.

2.4 4. Run Eureka!

Now that everything is set up, you should now be able to run the pipeline using:

```
python run_eureka.py
```

This will start printing information to your terminal, saving a bunch of output data/figures to the `outputdir` file locations you assigned above, and depending on the number of processors you were brave enough to allocate, potentially make your laptop as noisy as the engine of a Boeing 747.

Carry on reading for more information on each individual stage in the pipeline and some of the products it produces. Alternatively, feel free to dig through the output directories and get a gauge of what each stage is doing at your own speed.

2.4.1 Stage 1: Ramp Fitting

Stage 1 takes individual ramp level images and collapses them into integration level images, alongside some other basic corrections. This Stage broadly follows the STScI JWST pipeline methodology, with a few opportunities for adjustment as detailed on the [.ecf](#) information page.

The NIRSpec data being used here has already undergone the equivalent of this Stage, and it is therefore skipped (you will also notice it is commented out in the `run_eureka.py` file).

2.4.2 Stage 2: Calibrations

Stage 2 calibrates the data by performing a range of steps such as flat fielding and photometric unit conversions. Similarly to Stage 1, this broadly follows the STScI JWST pipeline methodology. In the case of the NIRSpec dataset we are using, the Eureka! implementation of this Stage avoids any spatial trimming of the images that usually occurs with the STScI pipeline. This facilitates a more accurate correction of the background and 1/f noise during Stage 3, as more background pixels are retained.

2.4.3 Stage 3: Reduction

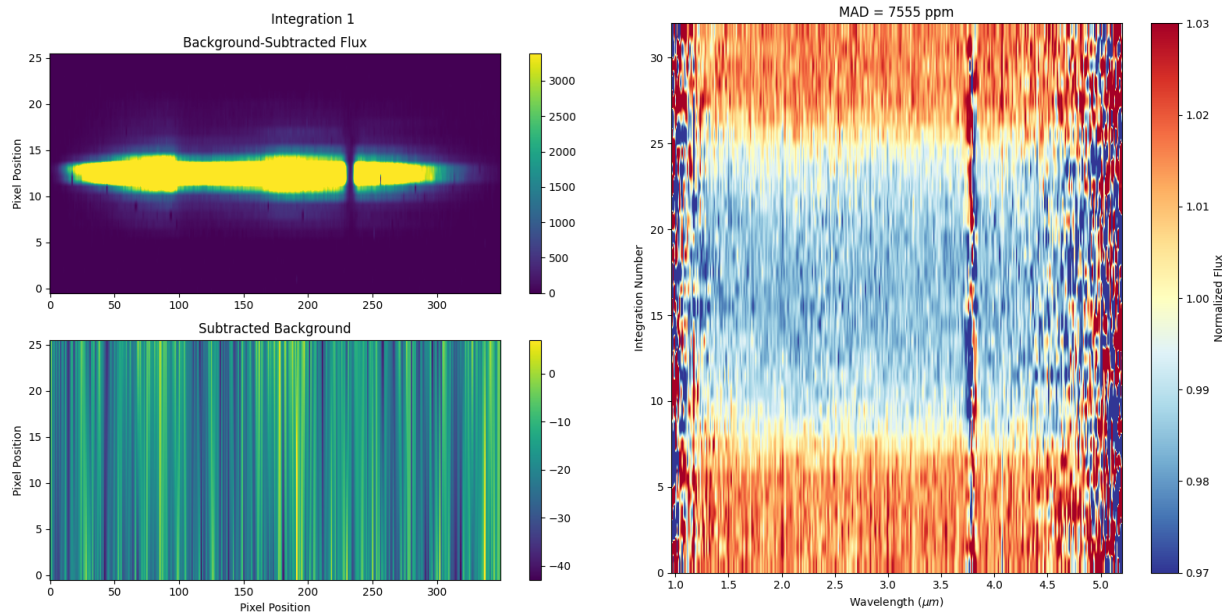
From Stage 3 onwards, Eureka! no longer makes use of the STScI pipeline and instead implements a range of custom routines to reduce the data further. It's at this stage that background subtraction and spectral extraction is performed, resulting in 1D spectra that can be used for light curve analysis and fitting.

By entering the `figs` folder you'll find a range of diagnostic figures. For example, on the left hand side of the figure copied below, the background subtracted 2D spectrum for the first integration is plotted (top) alongside a 2D image of the estimated background. Note that the distinct striping is a result of 1/f noise in the NIRSpec detector electronics, and is dominant along pixel columns as they correspond to the direction of the detector readout.

To the right you can see a 2D representation of the variation in flux between consecutive integrations as a function of wavelength. In fact, the transit of WASP-39b can be seen via the horizontal band of reduced flux between integrations

~9-25. At the top, the median absolute deviation (MAD) for the entire dataset is displayed, and is calculated by determining the flux difference between each image and the next, for each wavelength, followed by taking the overall median of these values across all wavelengths and all images.

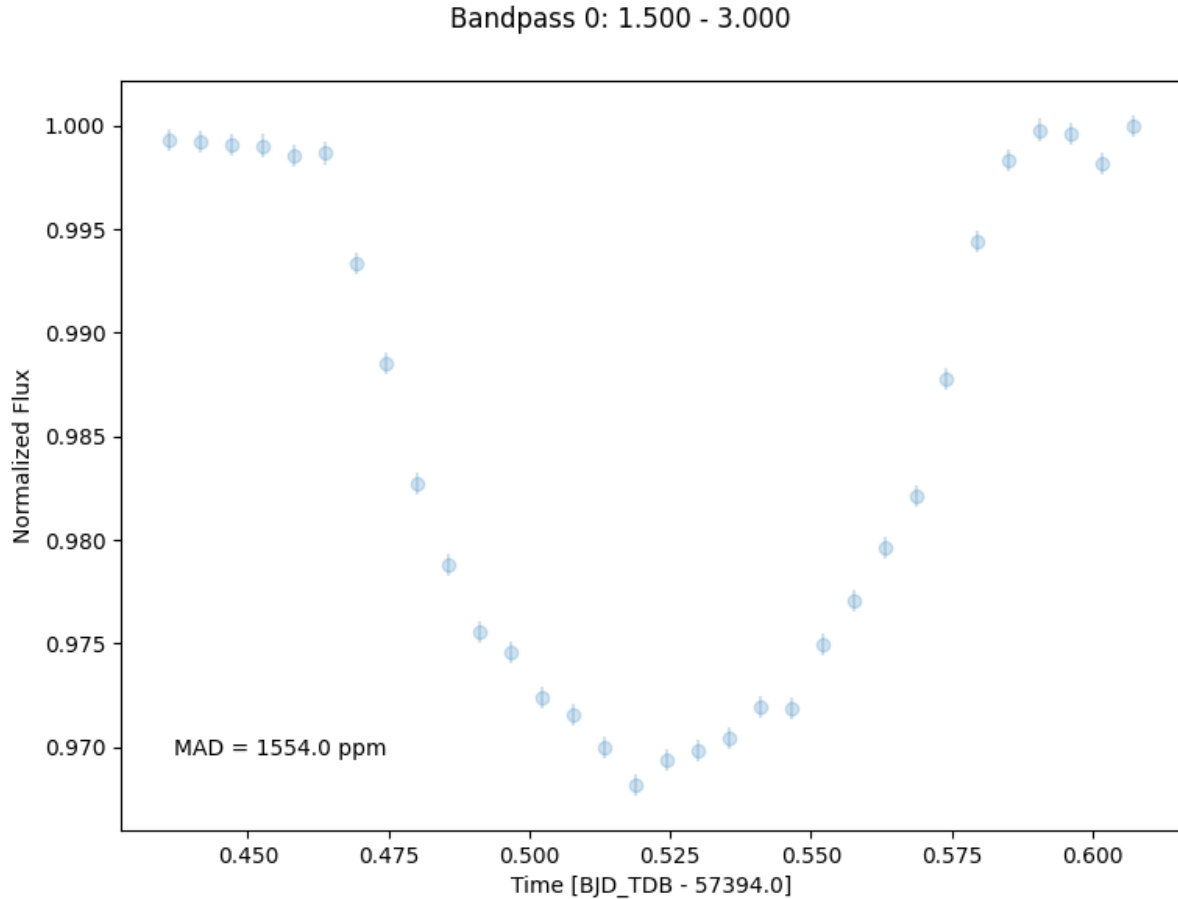
Finally, note that the actual data for these produced 1D spectra are contained in the `*Table_Save.txt` file.



2.4.4 Stage 4: Create Lightcurves

Stage 4 takes the 1D spectra produced by the previous stage and turns them into light curves. The number of wavelength channels to turn into light curves, along with the wavelength range across which they will be calculated, can be defined in the Stage 4 `.ecf` file. In the interest of reducing the computational burden of the following light curve fitting stage, only two light curves will be generated corresponding to 1.5-3.0 μm and 3.0-4.5 μm (see figure below).

Similarly to Stage 3, the actual data for the produced light curves can be found in the `*Table_Save.txt` file.



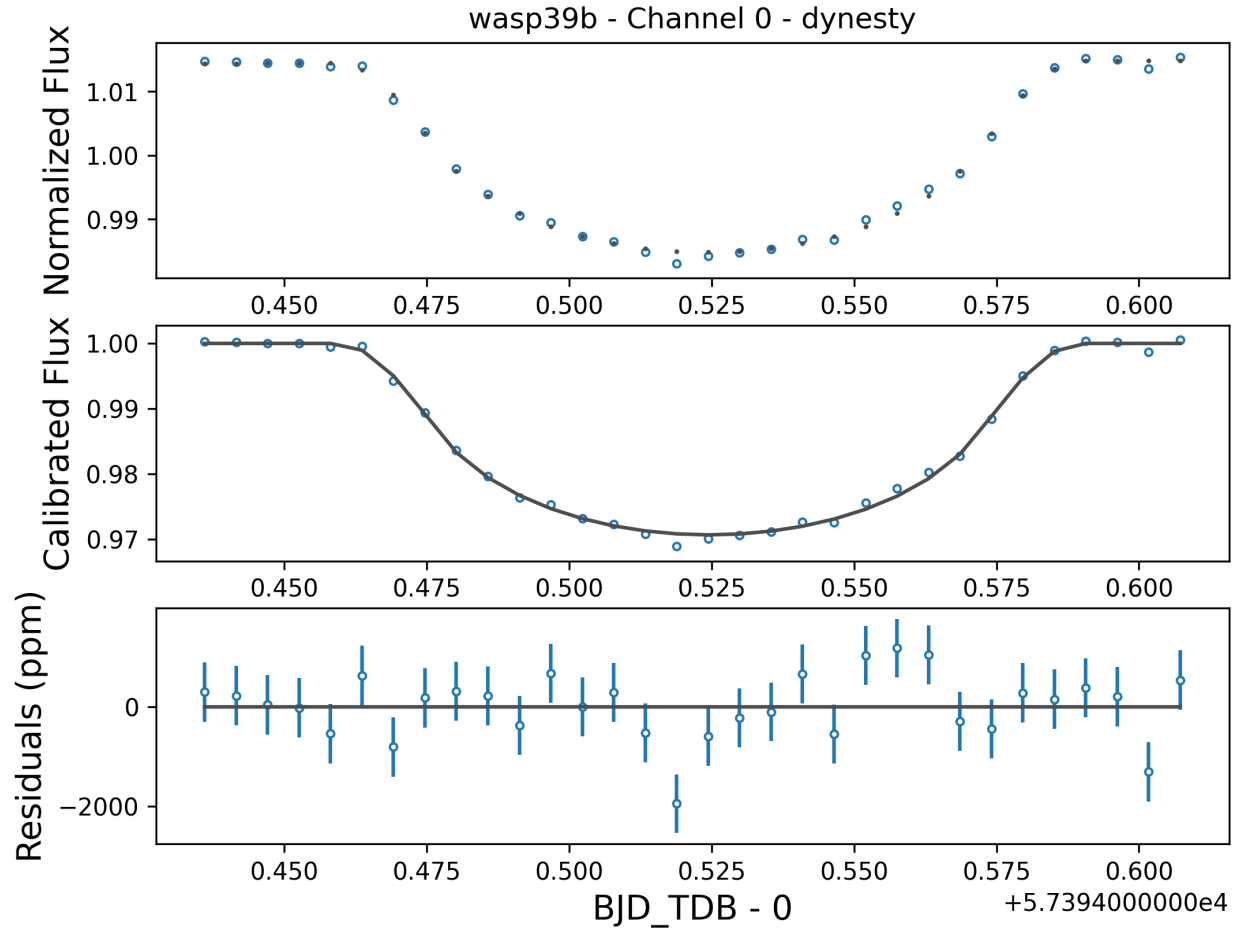
2.4.5 Stage 5: Lightcurve Fitting

Stage 5 takes all of the lightcurves produced by the previous stage and performs a variety of fitting routines to estimate specific system and planetary properties. For this quickstart, the fitting was performed using nested sampling as implemented by *dynesty*, for a model assuming a transit of WASP-39b plus an arbitrary linear polynomial trend.

As a reminder, the input initial guesses and priors for the model properties are contained within the Stage 5 `.epf` file. To facilitate this quickstart demo, input parameters applicable to WASP-39b have already been assigned. For your own reductions, you'll need to tailor this file to the system you are observing and the type of fit you want to perform.

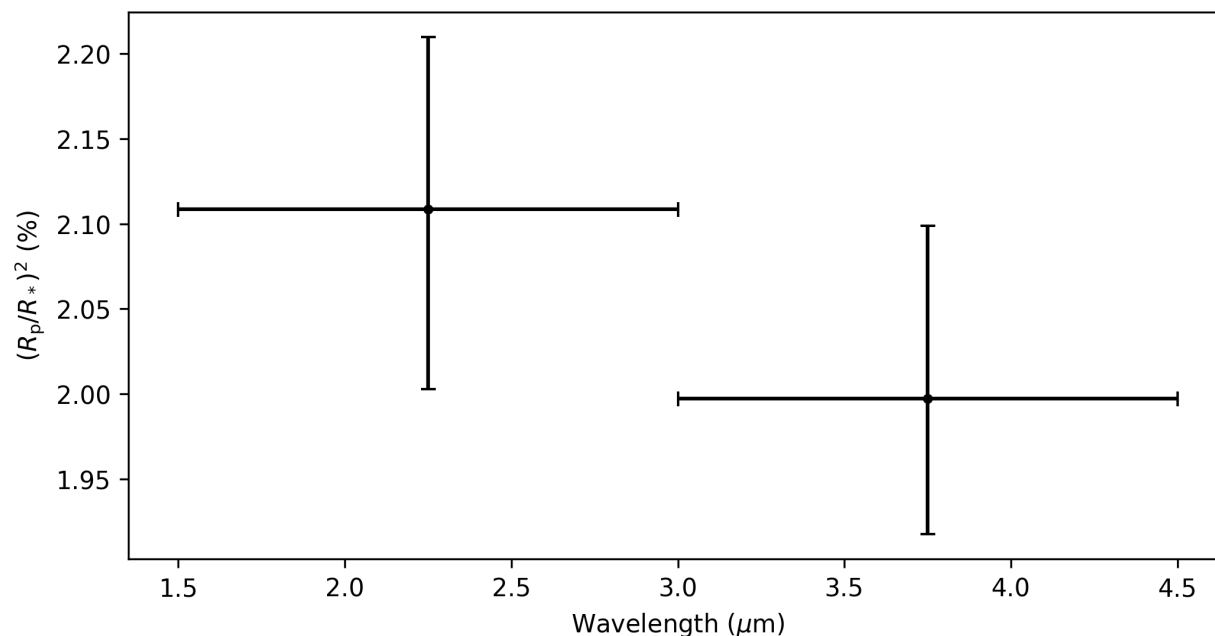
We have used nested sampling during this quickstart, however, this is not the only fitting method - both a simple least-squares minimisation as implemented by *scipy* and a full MCMC as implemented by *emcee* can also be used. Given the computational demands of running nested sampling or MCMC, it's advised that you perform initial testing with the least-squares fitter, before moving to a more advanced fitter. As the quickstart Stage 5 `.ecf` and `.epf` have already been prepared with suitable input values, we have skipped straight to a nested-sampling fit.

An example figure demonstrating the best fit model lightcurve alongside the data is shown below, and corner plot representations of the fit posteriors can be found under the `figs` directory. Once again, the actual model light curve data can be found in the `*Table_Save_ch*.txt` files.



2.4.6 Stage 6: Plot Spectra

The final Stage of Eureka!, Stage 6, takes the output data from the lightcurve fitting and produces transmission and/or emission spectra. As mentioned earlier, this quickstart only makes use of two different light curves from this dataset from 1.5-3.0 m and 3.0-4.5 m. In this case, our transmission spectrum for the transit of WASP-39b will only have two data points (see figure below). Note that the errors bars are not representative of what could be expected for true JWST data, as to reduce the computational burden this dataset has been trimmed down from 8192 integrations to only 32. Finally, the transmission spectrum data is saved in the `*Table_Save.txt` file.



2.5 5. Where to go next

You made it! Congratulations, it's time to reward yourself with a break

However, if this quickstart guide wasn't enough to sate your appetite, consider taking a look at the different parameter settings within the *.ecf files [here](#) and tweak away! If you want to explore the NIRSpec Tiny Dataset further, head back to the Stage 4 .ecf and try increasing the number of wavelength channels. Once you're comfortable, consider running things through with the [full dataset](#). Or, if you're bored with NIRSpec, maybe take a look at a simulated dataset for [NIRCam](#), [NIRISS](#), or [MIRI](#) instead.

If any bugs / errors cropped up while you were working through this quickstart, or if they turn up in the future, take a look at our [FAQ](#) or [report an issue](#) on our GitHub repository. Thanks!

EUREKA! CONTROL FILE (.ECF)

To run the different Stages of Eureka!, the pipeline requires control files (.ecf) where Stage-specific parameters are defined (e.g. aperture size, path of the data, etc.).

In the following, we look at the contents of the ecf for Stages 1, 2, 3, 4, and 5.

3.1 Stage 1

```
# Eureka! Control File for Stage 1: Detector Processing

suffix                uncal

# Control ramp fitting method
ramp_fit_algorithm    'default'    #Options are 'default', 'mean', or 'differenced'
ramp_fit_max_cores    'none'      #Options are 'none', 'quarter', 'half', 'all'

# Pipeline stages
skip_group_scale      False
skip_dq_init          False
skip_saturation        False
skip_ipc              True        #Skipped by default for all instruments
skip_superbias        False
skip_refpix           False
skip_linearity         False
skip_persistence      True        #Skipped by default for Near-IR TSO
skip_dark_current     False
skip_jump             False
skip_ramp_fitting     False
skip_gain_scale       False

# Project directory
topdir                /home/User

# Directories relative to project directory
inputdir              /Data/JWST-Sim/NIRCam/Uncalibrated
outputdir             /Data/JWST-Sim/NIRCam/Stage1

# Diagnostics
testing_S1            False
```

(continues on next page)

(continued from previous page)

```
#####

# "Default" ramp fitting settings
default_ramp_fit_weighting      default      #Options are "default", "fixed",
↪ "interpolated", "flat", or "custom"
default_ramp_fit_fixed_exponent 10          #Only used for "fixed" weighting
default_ramp_fit_custom_snr_bounds [5,10,20,50,100] # Only used for "custom"
↪ weighting, array no spaces
default_ramp_fit_custom_exponents [0.4,1,3,6,10] # Only used for "custom"
↪ weighting, array no spaces
```

3.1.1 suffix

Data file suffix (e.g. uncal).

3.1.2 ramp_fit_algorithm

Algorithm to use to fit a ramp to the frame-level images of uncalibrated files. Only default (i.e. the JWST pipeline) and mean can be used currently.

3.1.3 ramp_fit_max_cores

Fraction of processor cores to use to compute the ramp fits, options are none, quarter, half, all.

3.1.4 skip_*

If True, skip the named step.

Note: Note that some instruments and observing modes might skip a step either way! See [here](#) for the list of steps run for each instrument/mode by the STScI's JWST pipeline.

3.1.5 topdir + inputdir

The path to the directory containing the Stage 0 JWST data (uncal.fits).

3.1.6 topdir + outputdir

The path to the directory in which to output the Stage 1 JWST data and plots.

3.1.7 testing_S1

If True, only a single file will be used, outputs won't be saved, and plots won't be made. Useful for making sure most of the code can run.

3.1.8 default_ramp_fit_weighting

Define the method by which individual frame pixels will be weighted during the default ramp fitting process. The is specifically for the case where `ramp_fit_algorithm` is default. Options are default, fixed, interpolated, flat, or custom.

default: Slope estimation using a least-squares algorithm with an “optimal” weighting, see [here](#).

In short this weights each pixel, i , within a slope following $w_i = (i - i_{midpoint})^P$, where the exponent P is selected depending on the estimated signal-to-noise ratio of each pixel (see link above).

fixed: As with default, except the weighting exponent P is fixed to a precise value through the `default_ramp_fit_fixed_exponent` entry

interpolated: As with default, except the SNR to P lookup table is converted to a smooth interpolation.

flat: As with default, except the weighting equation is no longer used, and all pixels are weighted equally.

custom: As with default, except a custom SNR to P lookup table can be defined through the `default_ramp_fit_custom_snr_bounds` and `default_ramp_fit_custom_exponents` (see example .ecf file).

3.2 Stage 2

A full description of the Stage 2 Outputs is available here: [Stage 2 Output](#)

```
# Eureka! Control File for Stage 2: Data Reduction

suffix                rateints      # Data file suffix

# Controls the cross-dispersion extraction
slit_y_low            None         # Use None to rely on the default parameters
slit_y_high           None         # Use None to rely on the default parameters

# Modify the existing file to change the dispersion extraction - FIX: DOES NOT WORK,
↪ CURRENTLY
waverange_start       None         # Use None to rely on the default parameters
waverange_end         None         # Use None to rely on the default parameters

# Note: different instruments and modes will use different steps by default
skip_bkg_subtract     False        # Not run for TSO observations
skip_imprint_subtract True         # Not run for NIRCcam Wide-Field Slitless Spectroscopy
skip_msa_flagging     True         # Not run for NIRCcam Wide-Field Slitless Spectroscopy
skip_extract_2d       False
skip_srctype          True         # Not run for NIRCcam Wide-Field Slitless Spectroscopy
skip_master_background True        # Not run for NIRCcam Wide-Field Slitless Spectroscopy
skip_wavcorr          True         # Not run for NIRCcam Wide-Field Slitless Spectroscopy
skip_flat_field       False
skip_straylight       True         # Not run for NIRCcam Wide-Field Slitless Spectroscopy
```

(continues on next page)

(continued from previous page)

```

skip_fringe           True      # Not run for NIRCam Wide-Field Slitless Spectroscopy
skip_pathloss         True      # Not run for NIRCam Wide-Field Slitless Spectroscopy
skip_barshadow        True      # Not run for NIRCam Wide-Field Slitless Spectroscopy
skip_photom           True
skip_resample         True      # Not run for NIRCam Wide-Field Slitless Spectroscopy
skip_cube_build       True      # Not run for NIRCam Wide-Field Slitless Spectroscopy
skip_extract_1d       False

# Diagnostics
testing_S2           False
hide_plots           False      # If True, plots will automatically be closed rather
↳ than popping up

# Project directory
topdir                /home/User/

# Directories relative to project dir
inputdir              /Data/JWST-Sim/NIRCam/Stage1
outputdir             /Data/JWST-Sim/NIRCam/Stage2

```

3.2.1 suffix

Data file suffix (e.g. rateints).

Note: Note that other Instruments might used different suffixes!

3.2.2 slit_y_low & slit_y_high

Controls the cross-dispersion extraction. Use None to rely on the default parameters.

3.2.3 waverange_start & waverange_end

Modify the existing file to change the dispersion extraction (DOES NOT WORK). Use None to rely on the default parameters.

3.2.4 skip_*

If True, skip the named step.

Note: Note that some instruments and observing modes might skip a step either way! See [here](#) for the list of steps run for each instrument/mode by the STScI's JWST pipeline.

3.2.5 testing_S2

If True, outputs won't be saved and plots won't be made. Useful for making sure most of the code can run.

3.2.6 hide_plots

If True, plots will automatically be closed rather than popping up on the screen.

3.2.7 topdir + inputdir

The path to the directory containing the Stage 1 JWST data.

3.2.8 topdir + outputdir

The path to the directory in which to output the Stage 2 JWST data and plots.

3.3 Stage 3

```
# Eureka! Control File for Stage 3: Data Reduction

ncpu          1          # Number of CPUs
suffix        calints    # Data file suffix

# Subarray region of interest
ywindow       [5,64]     # Vertical axis as seen in DS9
xwindow       [100,1700] # Horizontal axis as seen in DS9
src_pos_type   gaussian   # Determine source position when not given in header
↳(Options: gaussian, weighted, or max)

# Background parameters
bg_hw         8          # Half-width of exclusion region for BG subtraction
↳(relative to source position)
bg_thresh     [5,5]      # Double-iteration X-sigma threshold for outlier rejection
↳along time axis
bg_deg        1          # Polynomial order for column-by-column background
↳subtraction, -1 for median of entire frame
p3thresh      5          # X-sigma threshold for outlier rejection during background
↳subtraction
save_bgsub    False      # Whether or not to save background subtracted FITS files

# Spectral extraction parameters
spec_hw       8          # Half-width of aperture region for spectral extraction
↳(relative to source position)
fittype       meddata    # Method for constructing spatial profile (Options: smooth,
↳meddata, poly, gauss, wavelet, or wavelet2D)
window_len    31         # Smoothing window length, when fittype = smooth
prof_deg      3          # Polynomial degree, when fittype = poly
p5thresh      10         # X-sigma threshold for outlier rejection while constructing
↳spatial profile
```

(continues on next page)

(continued from previous page)

```

p7thresh      10          # X-sigma threshold for outlier rejection during optimal_
↳spectral extraction

# Diagnostics
isplots_S3     3          # Generate few (1), some (3), or many (5) figures (Options:_
↳1 - 5)
testing_S3     False      # Boolean, set True to only use last file and generate_
↳select figures
hide_plots     False      # If True, plots will automatically be closed rather than_
↳popping up
save_output    True       # Save outputs for use in S4
verbose        True       # If True, more details will be printed about steps

# Project directory
topdir         /home/User/

# Directories relative to project dir
inputdir       /Data/JWST-Sim/NIRCam/Stage2/  # The folder containing the outputs from_
↳Eureka!'s S2 or JWST's S2 pipeline (will be overwritten if calling S2 and S3_
↳sequentially)
outputdir      /Data/JWST-Sim/NIRCam/Stage3/

```

3.3.1 ncpu

Sets the number of cores being used when Eureka! is executed. Currently, the only parallelized part of the code is the **background subtraction** for every individual integration and is being initialized in `s3_reduce.py` with:

```
util.BGsubtraction
```

3.3.2 suffix

If your data directory (`topdir + inputdir`, see below) contains files with different data formats, you want to consider setting this variable.

E.g.: Simulated NIRCam Data:

Stage 2 - For NIRCam, Stage 2 consists of the flat field correction, WCS/wavelength solution, and photometric calibration (counts/sec -> MJy). Note that this is specifically for NIRCam: the steps in Stage 2 change a bit depending on the instrument. The Stage 2 outputs are roughly equivalent to a “flt” file from HST.

- Stage 2 Outputs/*calints.fits - Fully calibrated images (MJy) for each individual integration. This is the one you want if you’re starting with Stage 2 and want to do your own spectral extraction.
- Stage 2 Outputs/*x1dints.fits - A FITS binary table containing 1D extracted spectra for each integration in the “calint” files.

As we want to do our own spectral extraction, we set this variable to `calints`.

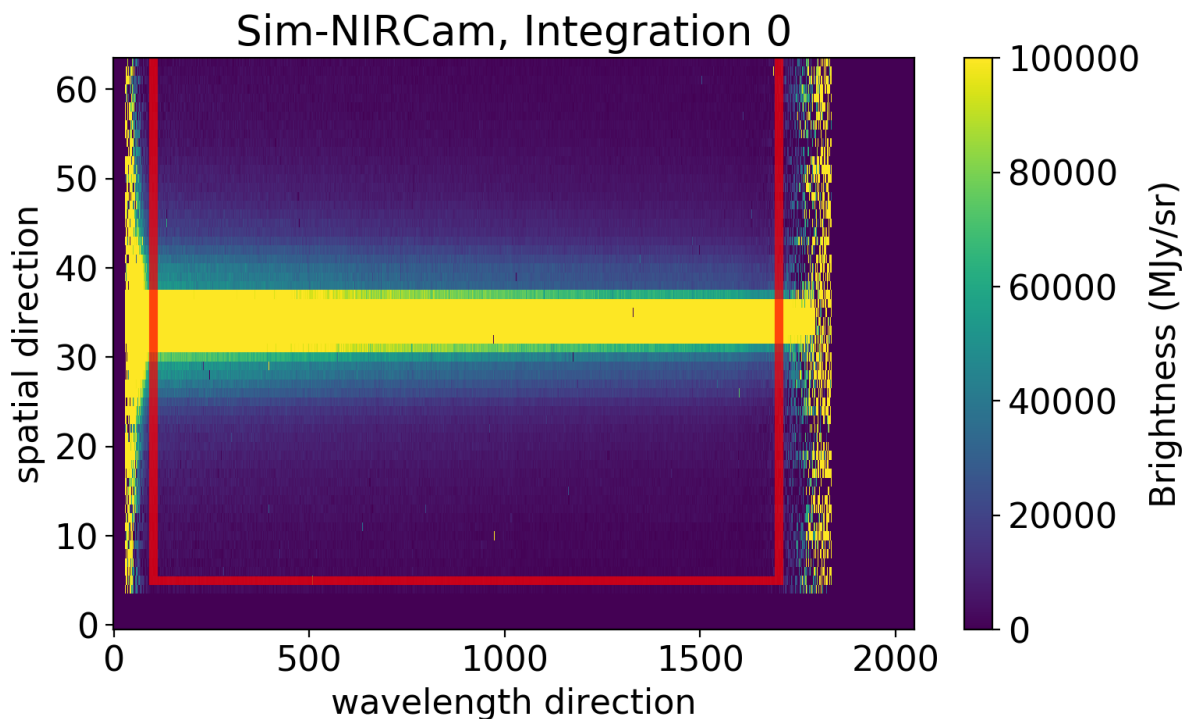
Note: Note that other Instruments might used different suffixes!

3.3.3 ywindow & xwindow

Can be set if one wants to remove edge effects (e.g.: many nans at the edges).

Below an example with the following setting:

```
ywindow    [5,64]
xwindow    [100,1700]
```



Everything outside of the box will be discarded and not used in the analysis.

3.3.4 src_pos_type

Determine the source position on the detector when not given in header (Options: gaussian, weighted, or max).

3.3.5 bg_hw & spec_hw

bg_hw and spec_hw set the background and spectrum aperture relative to the source position.

Let's look at an **example** with the following settings:

```
bg_hw      = 23
spec_hw     = 18
```

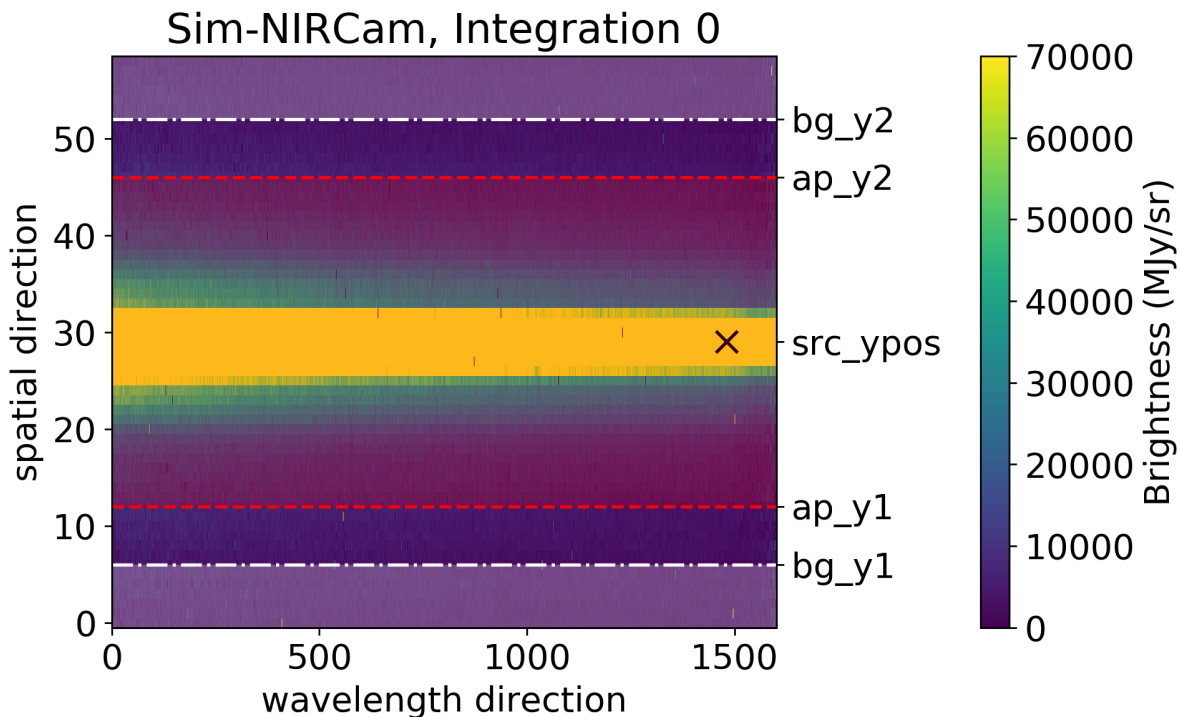
Looking at the fits file science header, we can determine the source position:

```
src_xpos = hdulist['SCI',1].header['SRCXPOS']-xwindow[0]
src_ypos = hdulist['SCI',1].header['SRCYPOS']-ywindow[0]
```

Let's assume in our example that `src_ypos = 29`.

(`xwindow[0]` and `ywindow[0]` corrects for the trimming of the data frame, as the edges were removed with the `xwindow` and `ywindow` parameters)

The plot below shows you which parts will be used for the background calculation (shaded in white; between the lower edge and `src_ypos - bg_hw`, and `src_ypos + bg_hw` and the upper edge) and which for the spectrum flux calculation (shaded in red; between `src_ypos - spec_hw` and `src_ypos + spec_hw`).



3.3.6 bg_thresh

Double-iteration X-sigma threshold for outlier rejection along time axis. The flux of every background pixel will be considered over time for the current data segment. e.g: `bg_thresh = [5,5]`: Two iterations of 5-sigma clipping will be performed in time for every background pixel. Outliers will be masked and not considered in the background flux calculation.

3.3.7 bg_deg

Sets the degree of the column-by-column background subtraction. If `bg_deg` is negative, use the median background of entire frame. Set to None for no background subtraction. Also, best to emphasize that we're performing column-by-column BG subtraction

The function is defined in `S3_data_reduction.optspex.fitbg`

Possible values:

- `bg_deg = None`: No background subtraction will be performed.
- `bg_deg < 0`: The median flux value in the background area will be calculated and subtracted from the entire 2D Frame for this particular integration.

- `bg_deg => 0`: A polynomial of degree `bg_deg` will be fitted to every background column (background at a specific wavelength). If the background data has an outlier (or several) which is (are) greater than $5 * (\text{Mean Absolute Deviation})$, this value will be not considered as part of the background. Step-by-step:

1. Take background pixels of first column
2. Fit a polynomial of degree `bg_deg` to the background pixels.
3. Calculate the residuals (`flux(bg_pixels) - polynomial_bg_deg(bg_pixels)`)
4. Calculate the MAD (Mean Absolute Deviation) of the greatest background outlier.
5. If MAD of the greatest background outlier is greater than 5, remove this background pixel from the background value calculation. Repeat from Step 2. and repeat as long as there is no $5 * \text{MAD}$ outlier in the background column.
6. Calculate the flux of the polynomial of degree `bg_deg` (calculated in Step 2) at the spectrum and subtract it.

3.3.8 p3thresh

Only important if `bg_deg => 0` (see above). # sigma threshold for outlier rejection during background subtraction which corresponds to step 3 of optimal spectral extraction, as defined by Horne (1986).

3.3.9 p5thresh

Used during Optimal Extraction. # sigma threshold for outlier rejection during step 5 of optimal spectral extraction, as defined by Horne (1986). Default is 10. For more information, see the source code of [`optspex.optimize`](#).

3.3.10 p7thresh

Used during Optimal Extraction. # sigma threshold for outlier rejection during step 7 of optimal spectral extraction, as defined by Horne (1986). Default is 10. For more information, see the source code of [`optspex.optimize`](#).

3.3.11 fittype

Used during Optimal Extraction. `fittype` defines how to construct the normalized spatial profile for optimal spectral extraction. Options are: 'smooth', 'meddata', 'wavelet', 'wavelet2D', 'gauss', or 'poly'. Using the median frame (meddata) should work well with JWST. Otherwise, using a smoothing function (smooth) is the most robust and versatile option. Default is meddata. For more information, see the source code of [`optspex.optimize`](#).

3.3.12 window_len

Used during Optimal Extraction. `window_len` is only used when `fittype = 'smooth'`. It sets the length scale over which the data are smoothed. Default is 31. For more information, see the source code of [`optspex.optimize`](#).

3.3.13 prof_deg

Used during Optimal Extraction. `prof_deg` is only used when `fittype = 'poly'`. It sets the polynomial degree when constructing the spatial profile. Default is 3. For more information, see the source code of `optspex.optimize`.

3.3.14 isplots_S3

Sets how many plots should be saved when running Stage 3. A full description of these outputs is available here: [Stage 3 Output](#)

3.3.15 testing_S3

If set to `True` only the last segment (which is usually the smallest) in the `inputdir` will be run. Also, only five integrations from the last segment will be reduced.

3.3.16 save_output

If set to `True` output will be saved as files for use in S4. Setting this to `False` is useful for quick testing

3.3.17 hide_plots

If `True`, plots will automatically be closed rather than popping up on the screen.

3.3.18 topdir + inputdir

The path to the directory containing the Stage 2 JWST data.

3.3.19 topdir + outputdir

The path to the directory in which to output the Stage 3 JWST data and plots.

3.4 Stage 4

```
# Eureka! Control File for Stage 4: Generate Lightcurves

# Number of spectroscopic channels spread evenly over given wavelength range
nspecchan      20          # Number of spectroscopic channels
wave_min       2.4         # Minimum wavelength. Set to None to use the shortest_
↳ extracted wavelength from Stage 3.
wave_max       4.0         # Maximum wavelength. Set to None to use the longest_
↳ extracted wavelength from Stage 3.
allapers       True        # Run S4 on all of the apertures considered in S3? Otherwise_
↳ will use newest output in the inputdir

# Parameters for drift correction of 1D spectra
correctDrift    False      # Set True to correct drift/jitter in 1D spectra (not_
↳ recommended for simulated data)
```

(continues on next page)

(continued from previous page)

```

drift_preclip    0      # Ignore first drift_preclip points of spectrum
drift_postclip  100     # Ignore last drift_postclip points of spectrum, None = no_
↳clipping
drift_range      11     # Trim spectra by +/-X pixels to compute valid region of cross_
↳correlation
drift_hw         5      # Half-width in pixels used when fitting Gaussian, must be_
↳smaller than drift_range
drift_iref       -1     # Index of reference spectrum used for cross correlation, -1 =_
↳last spectrum
sub_mean         True   # Set True to subtract spectrum mean during cross correlation
sub_continuum    True   # Set True to subtract the continuum from the spectra using a_
↳highpass filter
highpassWidth    10     # The integer width of the highpass filter when subtracting the_
↳continuum

# Parameters for sigma clipping
sigma_clip       False  # Whether or not sigma clipping should be performed on the 1D_
↳time series
sigma            10     # The number of sigmas a point must be from the rolling median_
↳to be considered an outlier
box_width        10     # The width of the box-car filter (used to calculated the_
↳rolling median) in units of number of data points
maxiters         5      # The number of iterations of sigma clipping that should be_
↳performed.
boundary         'fill' # Use 'fill' to extend the boundary values by the median of all_
↳data points (recommended), 'wrap' to use a periodic boundary, or 'extend' to use the_
↳first/last data points
fill_value       mask   # Either the string 'mask' to mask the outlier values_
↳(recommended), 'boxcar' to replace data with the mean from the box-car filter, or a_
↳constant float-type fill value.

# Diagnostics
isplots_S4       3      # Generate few (1), some (3), or many (5) figures (Options: 1 -_
↳5)
hide_plots       False  # If True, plots will automatically be closed rather than_
↳popping up
verbose          True   # If True, more details will be printed about steps

# Project directory
topdir           /home/User

# Directories relative to project dir
inputdir         /Data/JWST-Sim/NIRCam/Stage3 # The folder containing the outputs from_
↳Eureka!'s S3 or JWST's S3 pipeline (will be overwritten if calling S3 and S4_
↳sequentially)
outputdir        /Data/JWST-Sim/NIRCam/Stage4

```

3.4.1 nspecchan

Number of spectroscopic channels spread evenly over given wavelength range

3.4.2 wave_min & wave_max

Start and End of the wavelength range being considered. Set to None to use the shortest/longest extracted wavelength from Stage 3.

3.4.3 allapers

If True, run S4 on all of the apertures considered in S3. Otherwise the code will use the only or newest S3 outputs found in the inputdir. To specify a particular S3 save file, ensure that “inputdir” points to the procedurally generated folder containing that save file (e.g. set inputdir to /Data/JWST-Sim/NIRCam/Stage3/S3_2021-11-08_nircam_wfss_ap10_bg10_run1/).

3.4.4 correctDrift

If True, correct for drift/jitter in 1D spectra.

3.4.5 drift_preclip

Ignore first drift_preclip points of spectrum when correcting for drift/jitter in 1D spectra.

3.4.6 drift_postclip

Ignore the last drift_postclip points of spectrum when correcting for drift/jitter in 1D spectra. None = no clipping.

3.4.7 drift_range

Trim spectra by +/- drift_range pixels to compute valid region of cross correlation when correcting for drift/jitter in 1D spectra.

3.4.8 drift_hw

Half-width in pixels used when fitting Gaussian when correcting for drift/jitter in 1D spectra. Must be smaller than drift_range.

3.4.9 drift_iref

Index of reference spectrum used for cross correlation when correcting for drift/jitter in 1D spectra. -1 = last spectrum.

3.4.10 sub_mean

If True, subtract spectrum mean during cross correlation (can help with cross-correlation step).

3.4.11 isplots_S4

Sets how many plots should be saved when running Stage 4. A full description of these outputs is available here: [Stage 4 Output](#)

3.4.12 hide_plots

If True, plots will automatically be closed rather than popping up on the screen.

3.4.13 topdir + inputdir

The path to the directory containing the Stage 3 JWST data.

3.4.14 topdir + outputdir

The path to the directory in which to output the Stage 4 JWST data and plots.

3.5 Stage 5

```
# Eureka! Control File for Stage 5: Lightcurve Fitting

ncpu          1 # The number of CPU threads to use when running emcee or dynesty in_
↳parallel

allapers      True          # Run S5 on all of the apertures considered in_
↳S4? Otherwise will use newest output in the inputdir
rescale_err   False        # Rescale uncertainties to have reduced chi-
↳squared of unity
fit_par       ./S5_fit_par_template.epf # What fitting epf do you want to use?
verbose      True          # If True, more details will be printed about_
↳steps
fit_method    [dynesty]    #options are: lsq, emcee, dynesty (can list_
↳multiple types separated by commas)
run_myfuncs   [batman_tr,polynomial] #options are: batman_tr, batman_ecl, sinusoid_pc,
↳expramp, GP, and polynomial (can list multiple types separated by commas)

# Limb darkening controls (not yet implemented)
#fix_ld       False #use limb darkening file?
#ld_file      /path/to/limbdarkening/ld_outputfile.txt #location of limb darkening_
↳file
```

(continues on next page)

```

#lsq
lsq_method      'Nelder-Mead' # The scipy.optimize.minimize optimization method to use
lsq_tol         1e-6        # The tolerance for the scipy.optimize.minimize optimization.
↳method

#mcmc
old_chain       None        # Output folder relative to topdir that contains an old emcee.
↳chain to resume where you left off (set to None to start from scratch)
lsq_first       True        # Initialize with an initial lsq call (can help shorten burn-in,
↳but turn off if lsq fails). Only used if old_chain is None
run_nsteps      4000
run_nwalkers     50
run_nburn       2000

#dynesty
run_nlive       1024        # Must be > ndim * (ndim + 1) // 2
run_bound       'multi'
run_sample      'unif'
run_tol         0.1

#GP inputs
kernel_inputs   ['time','time'] #options: time
kernel_class    ['ExpSquared','Exp'] #options: ExpSquared, Matern32, Exp,
↳RationalQuadratic

# Plotting controls
interp         False       # Should astrophysical model be interpolated (useful for uneven
↳sampling like that from HST)

# Diagnostics
isplots_S5      5          # Generate few (1), some (3), or many (5) figures (Options: 1 -
↳5)
testing_S5      False      # Boolean, set True to only use the first spectral channel
testing_model   False      # Boolean, set True to only inject a model source of systematics
hide_plots      False      # If True, plots will automatically be closed rather than
↳popping up

# Project directory
topdir          /home/User/

# Directories relative to project dir
inputdir        /Data/JWST-Sim/NIRCam/Stage4/ # The folder containing the outputs from
↳Eureka!'s S4 pipeline (will be overwritten if calling S4 and S5 sequentially)
outputdir       /Data/JWST-Sim/NIRCam/Stage5/

```

3.5.1 ncpu

Integer. Sets the number of CPUs to use for multiprocessing Stage 5 fitting.

3.5.2 allapers

Boolean to determine whether Stage 5 is run on all the apertures considered in Stage 4. If False, will just use the most recent output in the input directory.

3.5.3 rescale_err

Boolean to determine whether the uncertainties will be rescaled to have a reduced chi-squared of 1

3.5.4 fit_par

Path to Stage 5 priors and fit parameter file.

3.5.5 run_verbose

Boolean to determine whether Stage 5 prints verbose output.

3.5.6 fit_method

Fitting routines to run for Stage 5 lightcurve fitting. Can be one or more of the following: [lsq, emcee, dynesty]

3.5.7 run_myfuncs

Determines the transit and systematics models used in the Stage 5 fitting. Can be one or more of the following: [batman_tr, batman_ecl, sinusoid_pc, expramp, polynomial]

3.5.8 Least-Squares Fitting Parameters

The following set the parameters for running the least-squares fitter.

3.5.9 lsq_method

Least-squares fitting method: one of any of the `scipy.optimize.minimize` least-squares methods.

3.5.10 `lsq_tolerance`

Float to determine the tolerance of the `scipy.optimize.minimize` method.

3.5.11 `Emcee Fitting Parameters`

The following set the parameters for running `emcee`.

3.5.12 `old_chain`

Output folder containing previous `emcee` chains to resume previous runs. To start from scratch, set to `None`.

3.5.13 `lsq_first`

Boolean to determine whether to run least-squares fitting before MCMC. This can shorten burn-in but should be turned off if least-squares fails. Only used if `old_chain` is `None`.

3.5.14 `run_nsteps`

Integer. The number of steps for `emcee` to run.

3.5.15 `run_nwalkers`

Integer. The number of walkers to use.

3.5.16 `run_nburn`

Integer. The number of burn-in steps to run.

3.5.17 `Dynesty Fitting Parameters`

The following set the parameters for running `dynesty`. These options are described in more detail in: <https://dynesty.readthedocs.io/en/latest/api.html?highlight=unif#module-dynesty.dynesty>

3.5.18 `run_nlive`

Integer. Number of live points for `dynesty` to use. Should be at least greater than $(\text{ndim} * (\text{ndim} + 1)) / 2$, where `ndim` is the total number of fitted parameters. For shared fits, multiply the number of free parameters by the number of wavelength bins specified in Stage 4.

3.5.19 run_bound

The bounding method to use. Options are: ['none', 'single', 'multi', 'balls', 'cubes']

3.5.20 run_sample

The sampling method to use. Options are ['auto', 'unif', 'rwalk', 'rstagger', 'slice', 'rslice', 'hslice']

3.5.21 run_tol

Float. The tolerance for the dynesty run. Determines the stopping criterion. The run will stop when the estimated contribution of the remaining prior volume to the total evidence falls below this threshold.

3.5.22 interp

Boolean to determine whether the astrophysical model is interpolated when plotted. This is useful when there is uneven sampling in the observed data.

3.5.23 isplots_S5

Sets how many plots should be saved when running Stage 5. A full description of these outputs is available here: [Stage 5 Output](#)

3.5.24 hide_plots

If True, plots will automatically be closed rather than popping up on the screen.

3.5.25 topdir + inputdir

The path to the directory containing the Stage 4 JWST data.

3.5.26 topdir + outputdir

The path to the directory in which to output the Stage 5 JWST data and plots.

3.6 Stage 5 Fit Parameters

Warning: The Stage 5 fit parameter file has the file extension .epf, not .ecf. These have different formats, and are not interchangeable.

This file describes the transit/eclipse and systematics parameters and their prior distributions. Each line describes a new parameter, with the following basic format:

```
Name Value Free PriorPar1 PriorPar2 PriorType
```

Name defines the specific parameter being fit for. Available options are:

- **Transit and Eclipse Parameters**

- rp - planet-to-star radius ratio, for the transit models.
- fp - planet/star flux ratio, for the eclipse models.

- **Orbital Parameters**

- per - orbital period (in days)
- t0 - transit time (in days)
- time_offset - (optional), the absolute time offset of your time-series data (in days)
- inc - orbital inclination (in degrees)
- a - a/R^* , the ratio of the semimajor axis to the stellar radius
- ecc - orbital eccentricity
- w - argument of periapsis (degrees)

- **Phase Curve Parameters - the phase curve model allows for the addition of up to four sinusoids into a single phase curve**

- AmpCos1 - Amplitude of the first cosine
- AmpSin1 - Amplitude of the first sine
- AmpCos2 - Amplitude of the second cosine
- AmpSin2 - Amplitude of the second sine

- **Limb Darkening Parameters**

- limb_dark - The limb darkening model to be used. Options are: ['uniform', 'linear', 'quadratic', 'kipping2013', 'square-root', 'logarithmic', 'exponential', '4-parameter']
- uniform limb-darkening has no parameters, linear has a single parameter u1, quadratic, kipping2013, square-root, logarithmic, and exponential have two parameters u1, u2, 4-parameter has four parameters u1, u2, u3, u4

- **Systematics Parameters** - Depending on the model specified in the Stage 5 ECF, set either polynomial model coefficients c0--c9 for 0th to 3rd order polynomials. The polynomial coefficients are numbered as increasing powers (i.e. c0 a constant, c1 linear, etc.). The x-values of the polynomial are the time with respect to the mean of the time of the lightcurve time array. Polynomial fits should include at least c0 for usable results. The exponential ramp model is defined as follows: $r0 \cdot \text{np.exp}(-r1 \cdot \text{time_local} + r2) + r3 \cdot \text{np.exp}(-r4 \cdot \text{time_local} + r5) + 1$, where r0--r2 describe the first ramp, and r3--r5 the second. time_local is the time relative to the first frame of the dataset. If you only want to fit a single ramp, you can omit r3--r5 or set them to 0.
- **White Noise Parameters** - options are scatter_mult for a multiplier to the expected noise from Stage 3 (recommended), or scatter_ppm to directly fit the noise level in ppm

Free determines whether the parameter is fixed, free, independent, or shared. fixed parameters are fixed in the fitting routine and not fit for. free parameters are fit for according to the specified prior distribution, independently for each wavelength channel. shared parameters are fit for according to the specified prior distribution, but are common to all wavelength channels. independent variables set auxiliary functions needed for the fitting routines.

The PriorType can be U (Uniform), LU (Log Uniform), or N (Normal). If U/LU, then PriorPar1 and PriorPar2 are the lower and upper limits of the prior distribution. If N, then PriorPar1 is the mean and PriorPar2 is the standard deviation of the Gaussian prior.

Here's an example fit parameter file:

```
#Name      Value      Free?      PriorPar1  PriorPar2  PriorType
# PriorType can be U (Uniform), LU (Log Uniform), or N (Normal).
# If U/LU, PriorPar1 and PriorPar2 represent upper and lower limits of the parameter/
↳log(the parameter).
# If N, PriorPar1 is the mean and PriorPar2 is the standard deviation of a Gaussian.
↳prior.
#-----
↳-----
#
# -----
# ** Transit/eclipse parameters **
# -----
rp          0.16        'free'      0.05        0.3         U
#fp          0.008      'free'      0           0.5         U
# -----
# ** Phase curve parameters **
# -----
#AmpCos1     0.4        'free'      0           1           U
#AmpSin1     0.01       'free'     -1          1           U
#AmpCos2     0.01       'free'     -1          1           U
#AmpSin2     0.01       'free'     -1          1           U
# -----
# ** Orbital parameters **
# -----
per          0.813473978 'free'     0.813473978 0.000000035 N
t0           55528.353027 'free'     55528.34    55528.36    U
time_offset  0          'independent'
inc          82.109     'free'     82.109      0.088       N
a            4.97       'free'     4.97        0.14        N
ecc          0.0        'fixed'    0           1           U
w            90.        'fixed'    0           180         U
# -----
# ** Limb darkening parameters **
# Choose limb_dark from ['uniform', 'linear', 'quadratic', 'kipping2013', 'square-root',
↳'logarithmic', 'exponential', '4-parameter']
# -----
limb_dark    'kipping2013' 'independent'
u1           0.3        'free'     0           1           U
u2           0.1        'free'     0           1           U
# -----
# ** Systematic variables **
# polynomial model variables (c0--c9 for 0th--3rd order polynomials in time); Fitting at
↳least c0 is very strongly recommended!
# expramp model variables (r0--r2 for one exponential ramp, r3--r5 for a second
↳exponential ramp)
# GP model variables (A, WN, m_1, m_2)
# -----
c0           1          'free'     0.95        1.05        U
# -----
# ** White noise **
# Use scatter_mult to fit a multiplier to the expected noise level from Stage 3.
↳(recommended)
```

(continues on next page)

(continued from previous page)

```
# Use scatter_ppm to fit the noise level in ppm
# -----
scatter_mult 1          'free'          1          0.1          N
```

EUREKA! OUTPUTS

Stage 2 through Stage 5 of Eureka! can be configured to output plots of the pipeline's interim results as well as the data required to run further stages.

4.1 Stage 2 Outputs

If `skip_extract_1d` is set in the Stage 2 ECF, the 1-dimensional spectrum will not be extracted, and no plots will be made. Otherwise, Stage 2 will extract the 1-dimensional spectrum from the calibrated images, and will plot the spectrum.

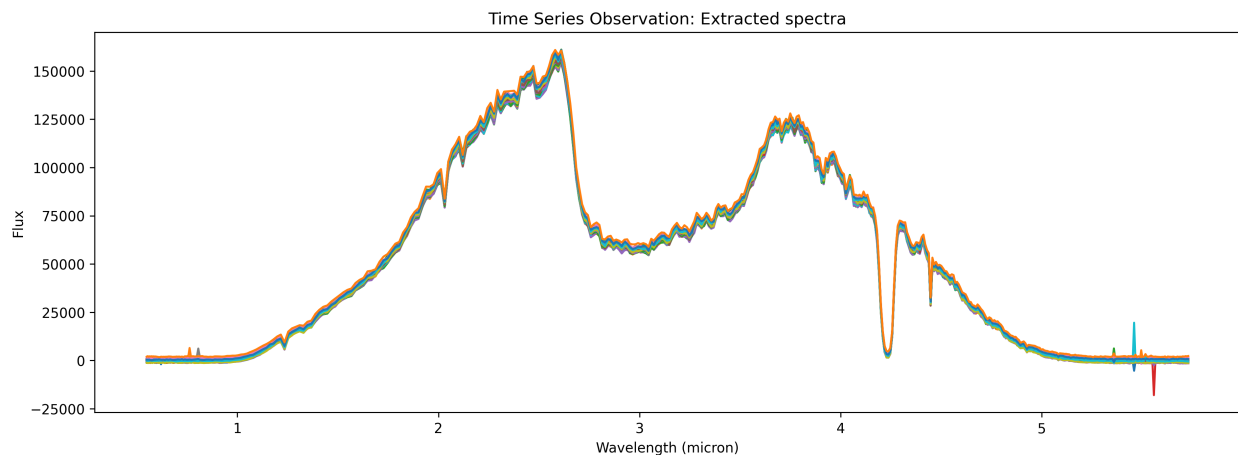


Fig. 1: Stage 2 output: 1-Dimensional Spectrum Plot

4.2 Stage 3 Outputs

In Stage 3 through Stage 5, output plots are controlled with the `isplots_SX` parameter. The resulting plots are cumulative: setting `isplots_S3 = 5` will also create the plots specified in `isplots_S3 = 3` and `isplots_S3 = 1`.

In Stage 3:

- If `isplots_S3 = 1`: Eureka! will plot the 2-dimensional, non-drift-corrected light curve.
- If `isplots_S3 = 3`: Eureka! will plot the results of the background and optimal spectral extraction steps for each exposure in the observation, as well as the source position on the detector.
- If `isplots_S3 = 5`: Eureka! will plot the Subdata plots from the optimal spectral extraction step.

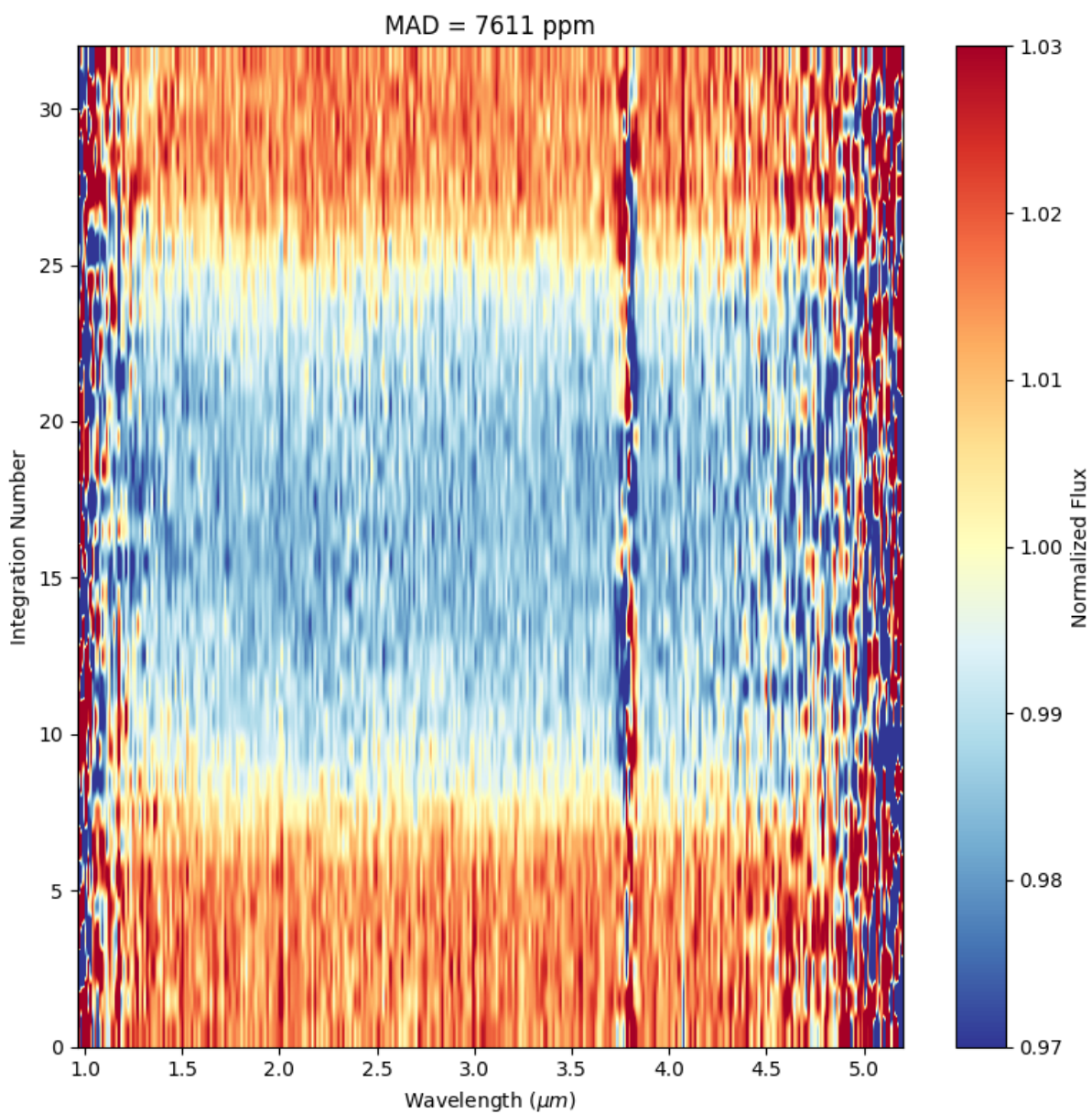


Fig. 2: Stage 3 output: 2-Dimensional Spectrum Plot

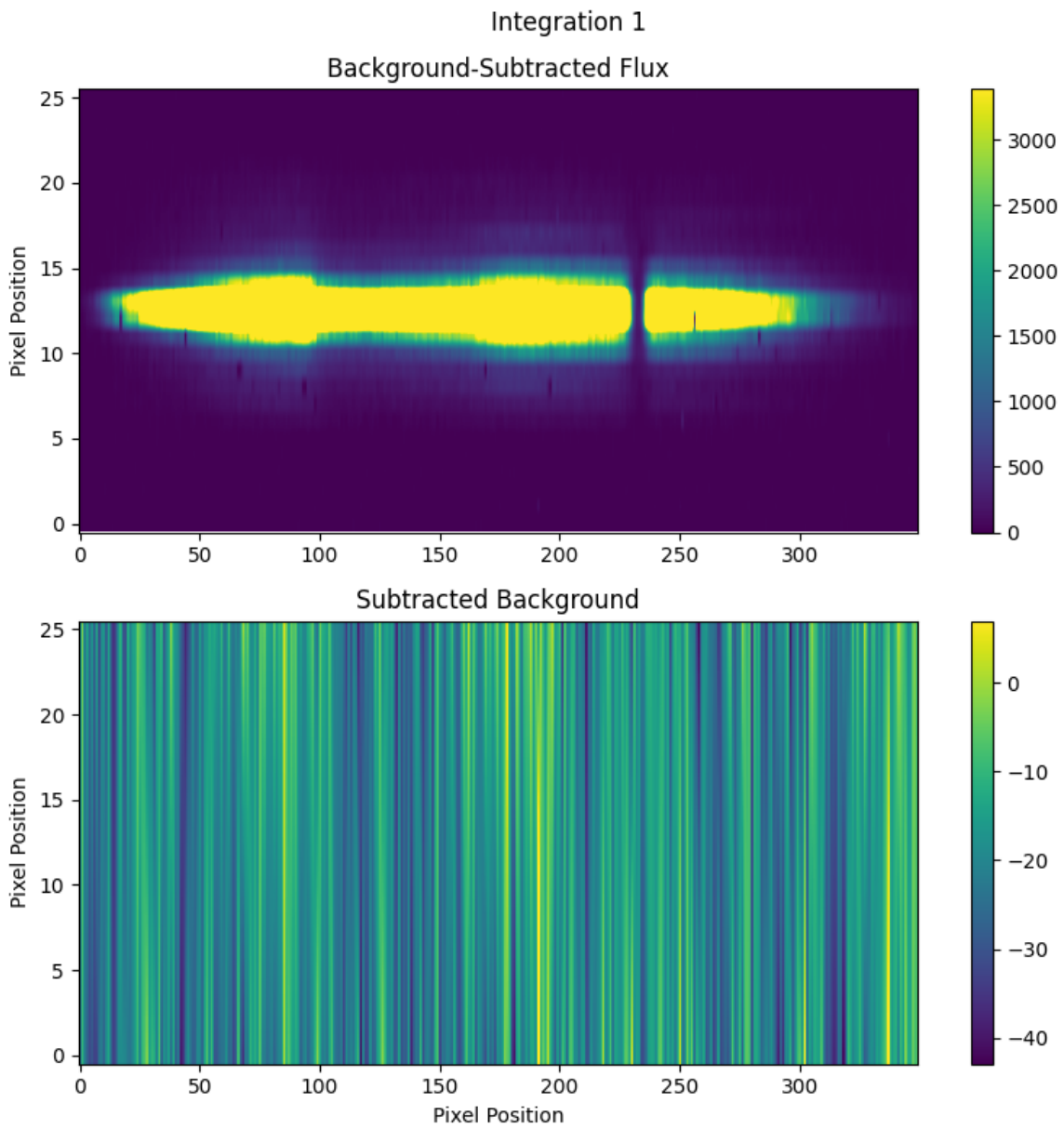


Fig. 3: Stage 3 output: Background Subtracted Flux Plot

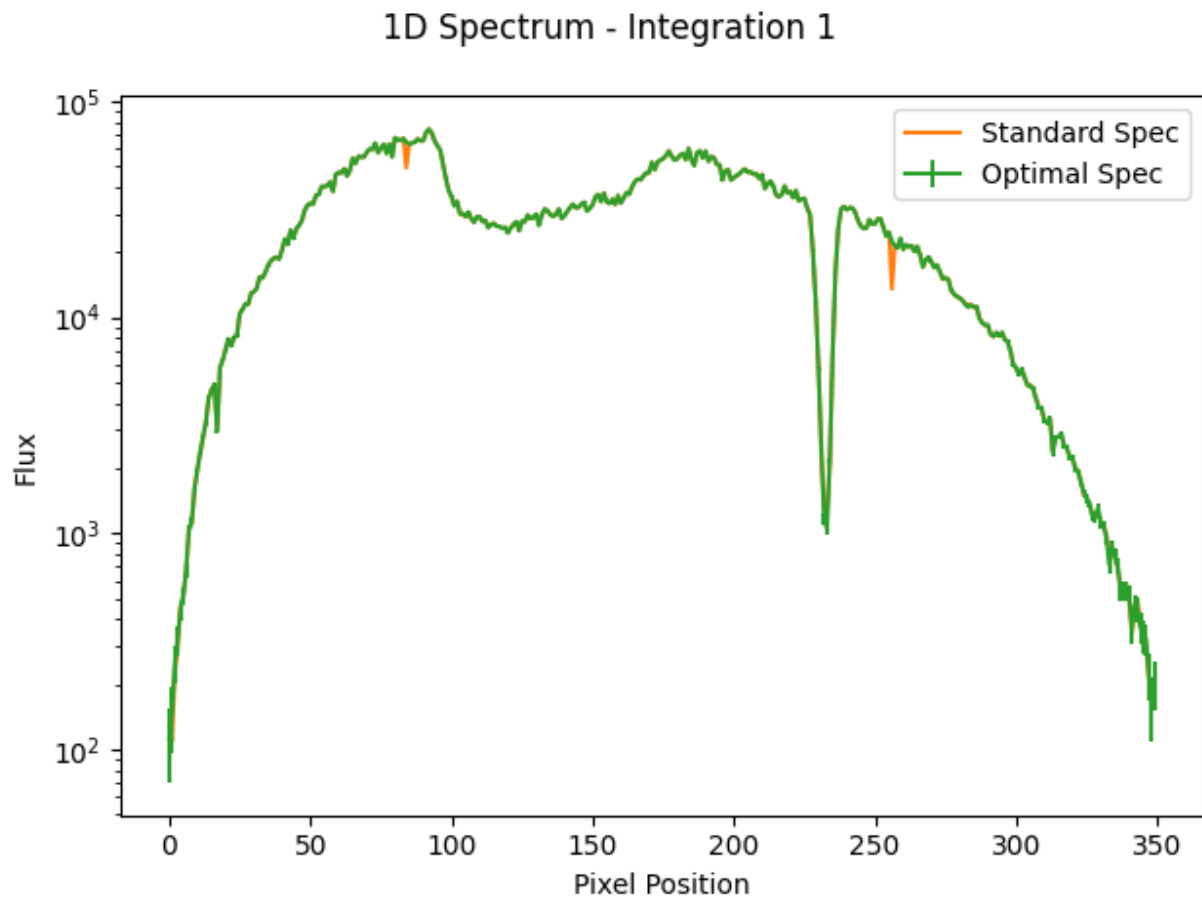


Fig. 4: Stage 3 output: 1-Dimensional Spectrum Plot

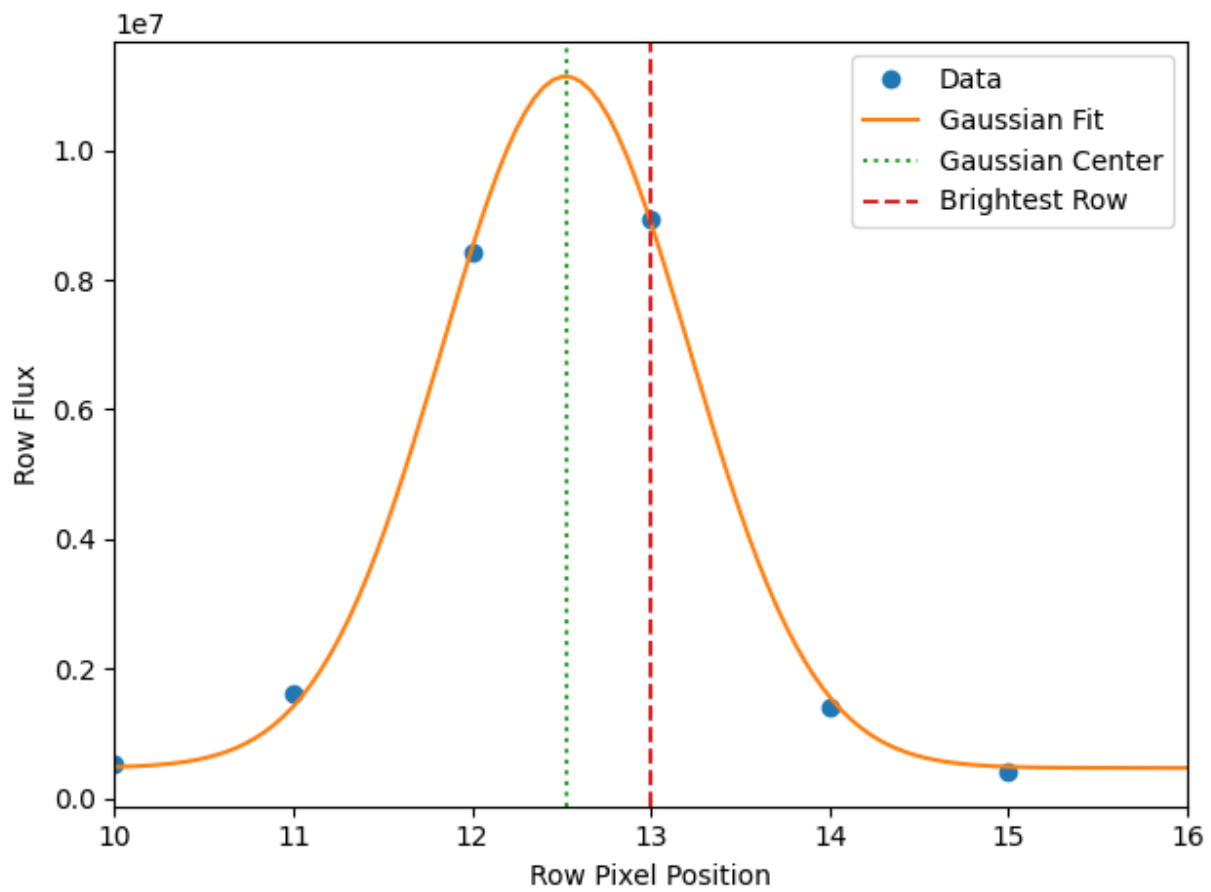


Fig. 5: Stage 3 output: Source Position Fit Plot

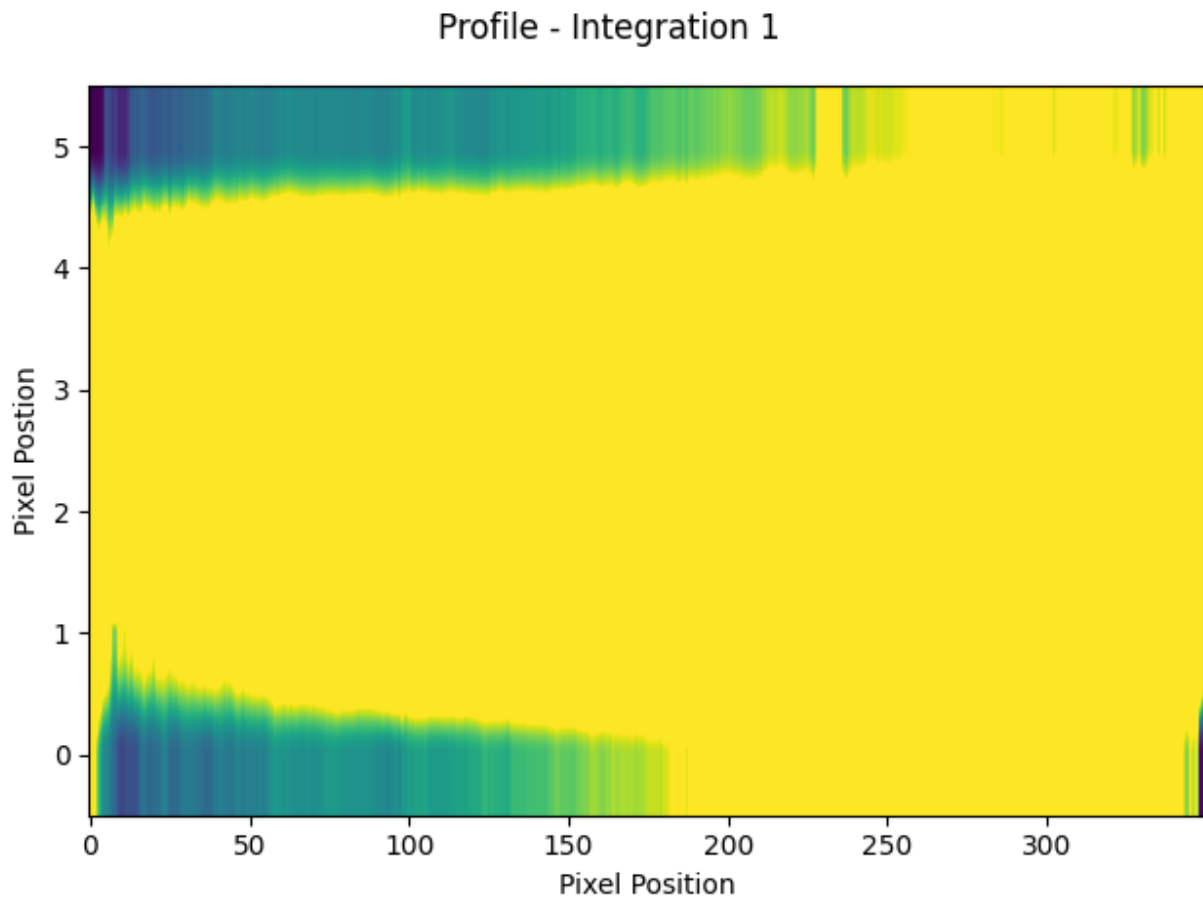


Fig. 6: Stage 3 output: Weighting Profile Plot

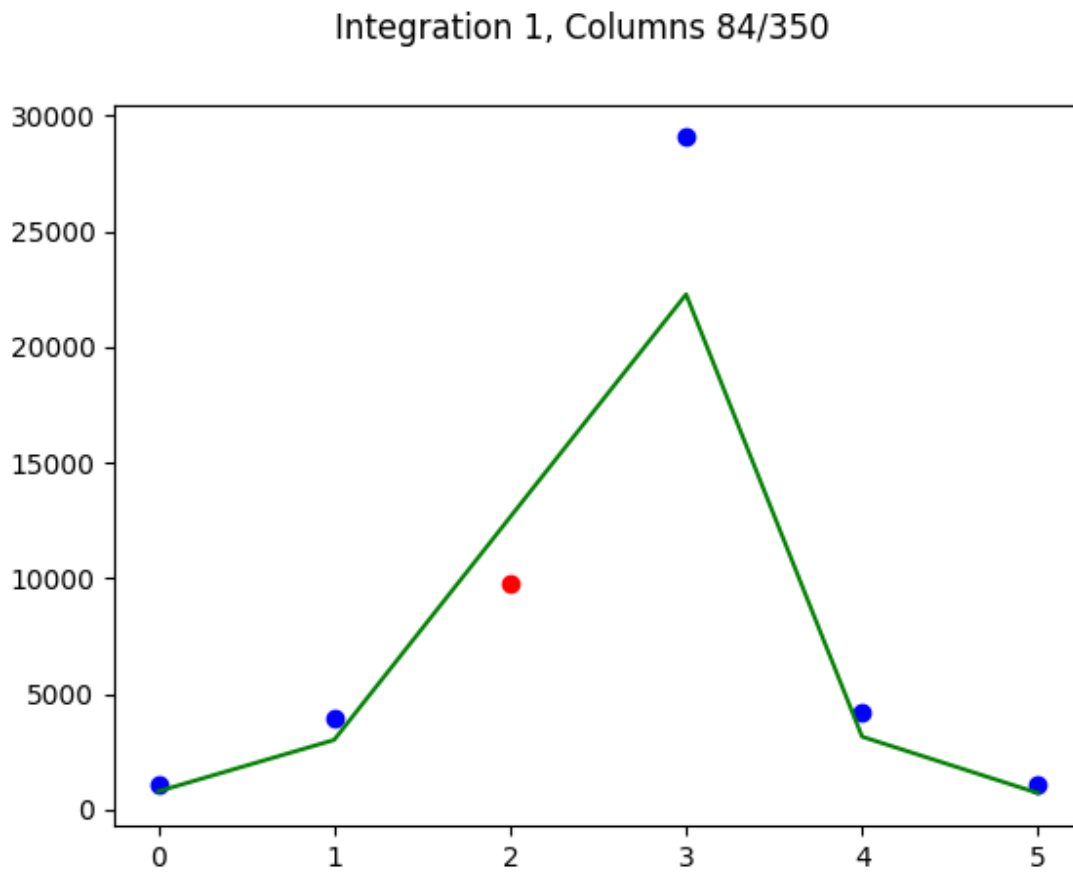


Fig. 7: Stage 3 output: Spectral Extraction Subdata Plot

4.3 Stage 4 Outputs

In Stage 4:

- If `isplots_S4 = 1`: Eureka! will plot the spectral drift per exposure, and the drift-corrected 2-dimensional lightcurve with extracted bins overlaid.

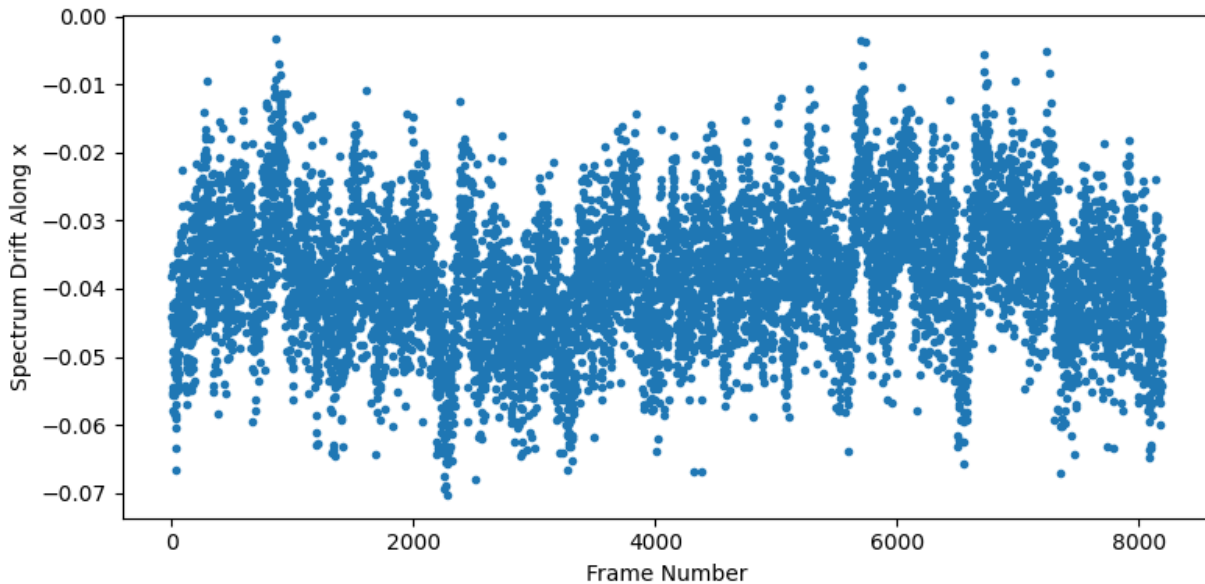


Fig. 8: Stage 4 output: Spectral Drift Plot

- If `isplots_S4 = 3`: Eureka! will plot the spectroscopic lightcurves for each wavelength bin.
- If `isplots_S4 = 5`: Eureka! will plot the cross-correlated reference spectrum with the current spectrum for each integration, and the cross-correlation strength for each integration.

4.4 Stage 5 Outputs

In Stage 5:

- If `isplots_S5 = 1`: Eureka! will plot the fitted lightcurve model over the data in each channel.
- If `isplots_S5 = 3`: Eureka! will plot an RMS deviation plot for each channel to help check for correlated noise, plot the normalized residual distribution, and plot the fitting chains for each channel.
- If `isplots_S5 = 5`, and if `emcee` or `dynesty` were used as the fitter: Eureka! will plot a corner plot for each channel.
- If a GP model was used in the fit, then Eureka! will plot the lightcurve, the GP model, and the residuals.

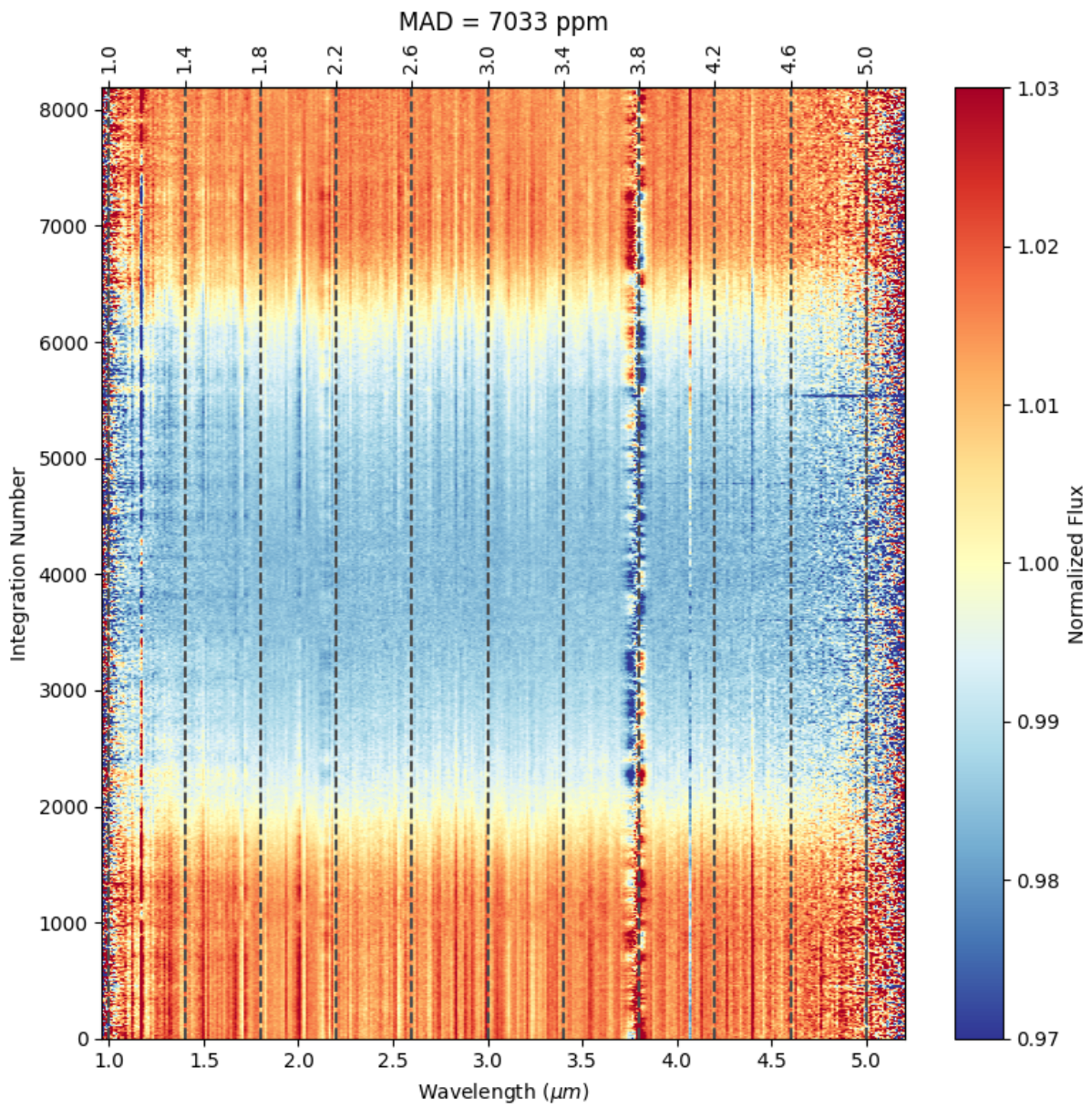


Fig. 9: Stage 4 output: 2-Dimensional Binned Spectrum

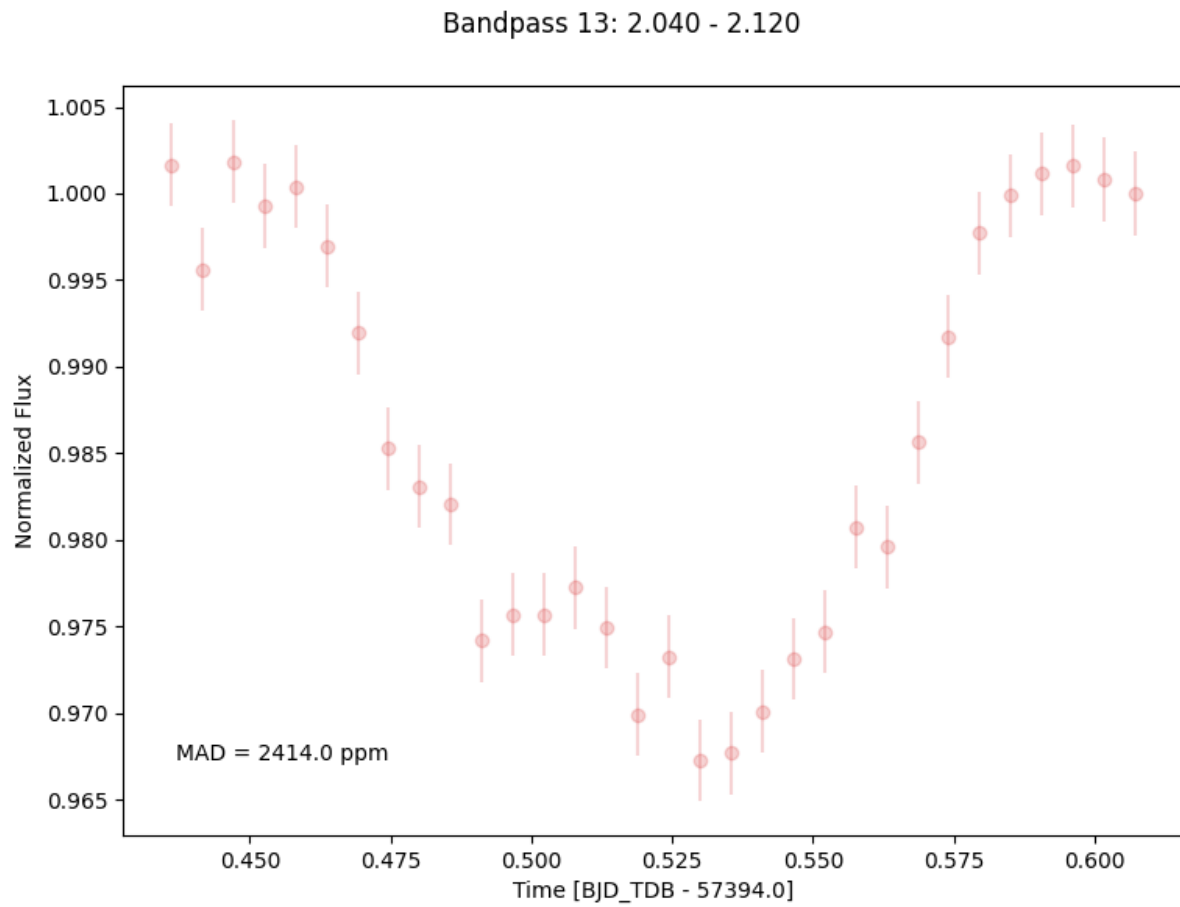


Fig. 10: Stage 4 output: Spectroscopic Lightcurve

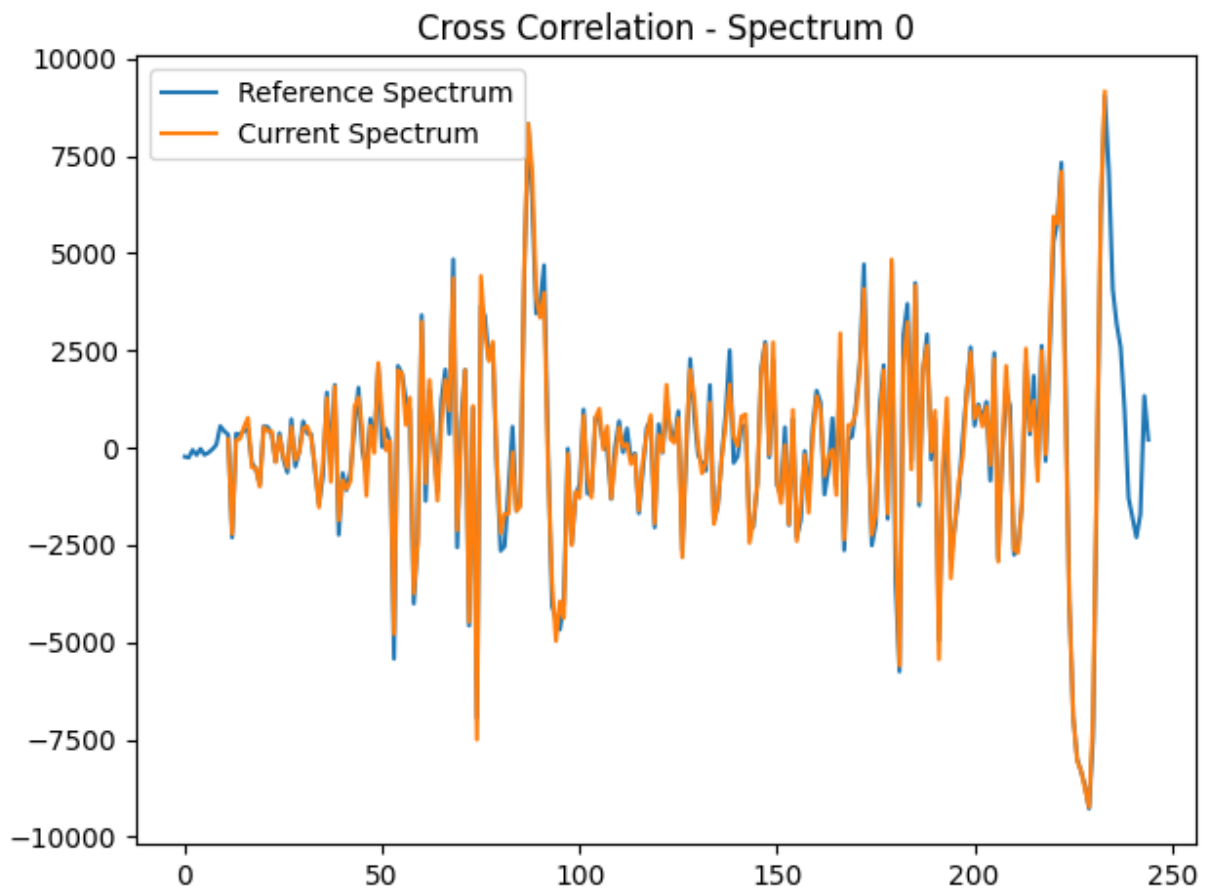


Fig. 11: Stage 4 output: Cross-Correlated Reference Spectrum

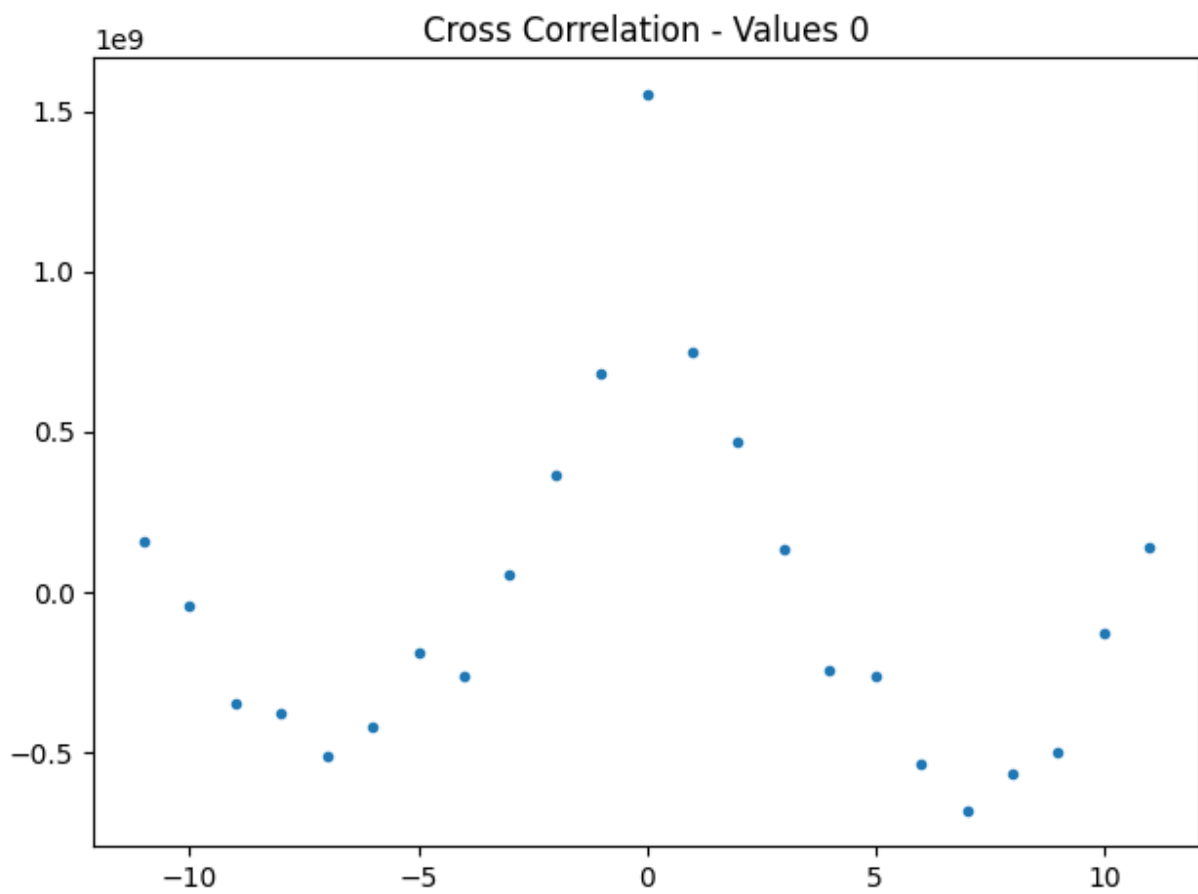


Fig. 12: Stage 4 output: Cross-Correlation Strength

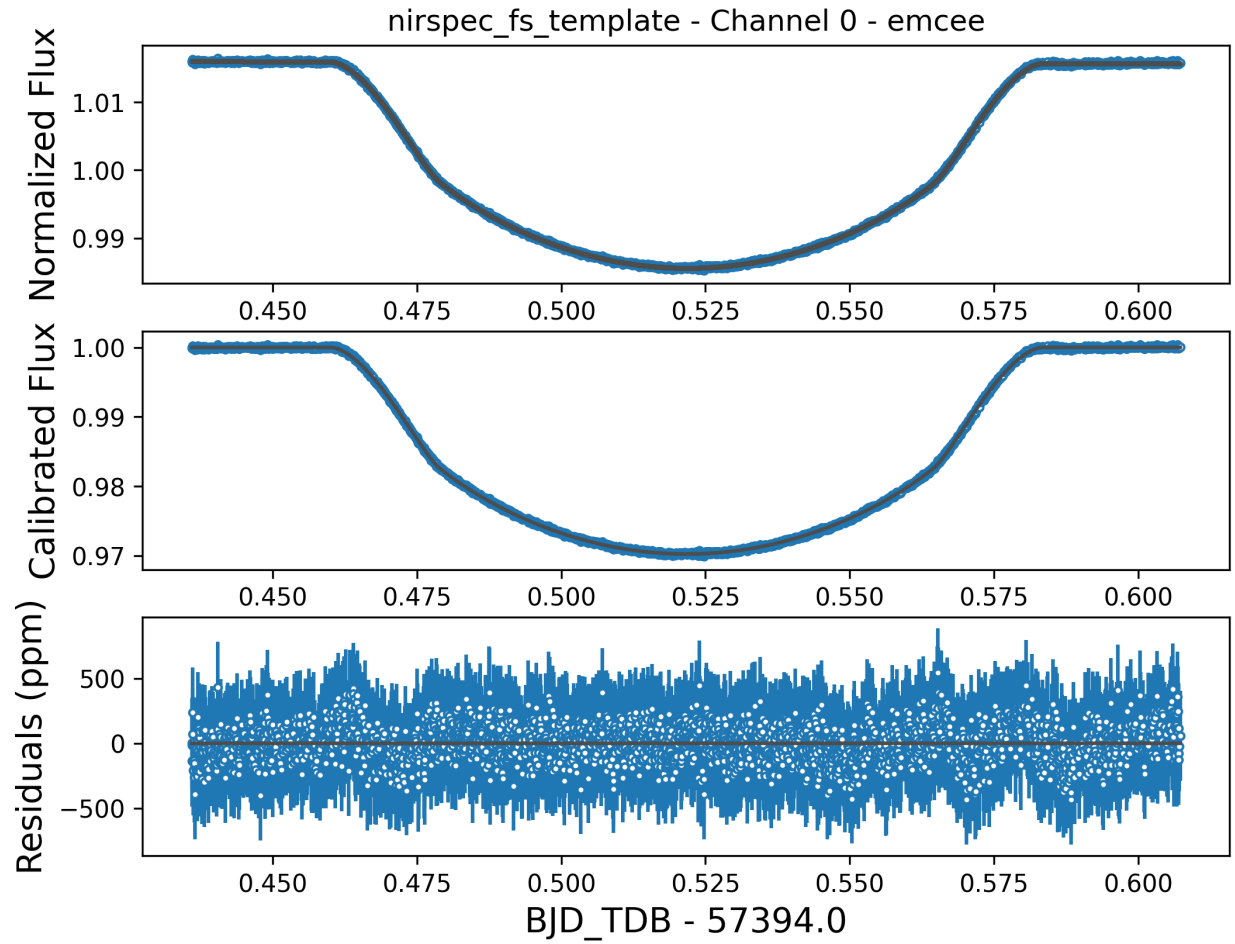


Fig. 13: Stage 5 output: Fitted lightcurve

Correlated Noise

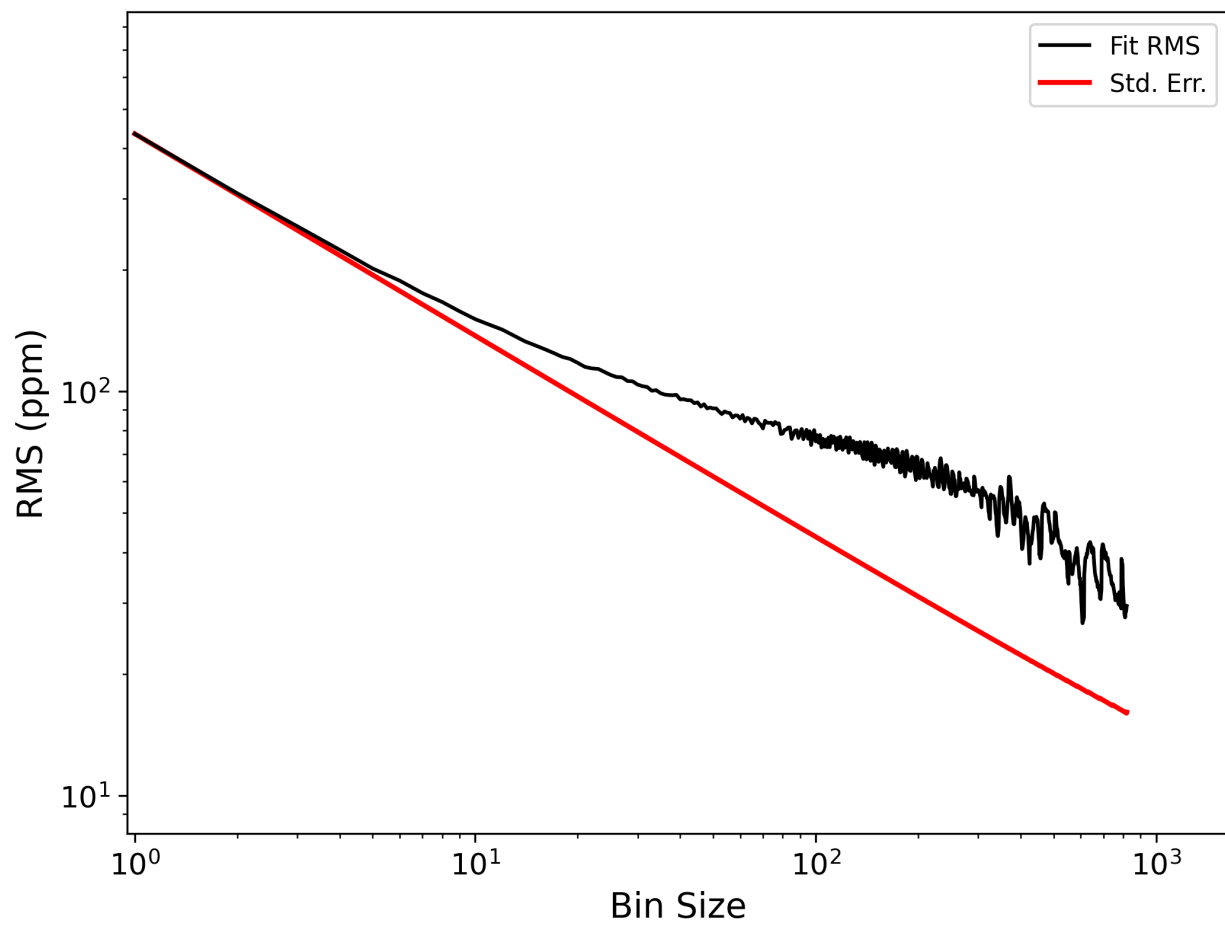


Fig. 14: Stage 5 output: RMS Deviation Plot

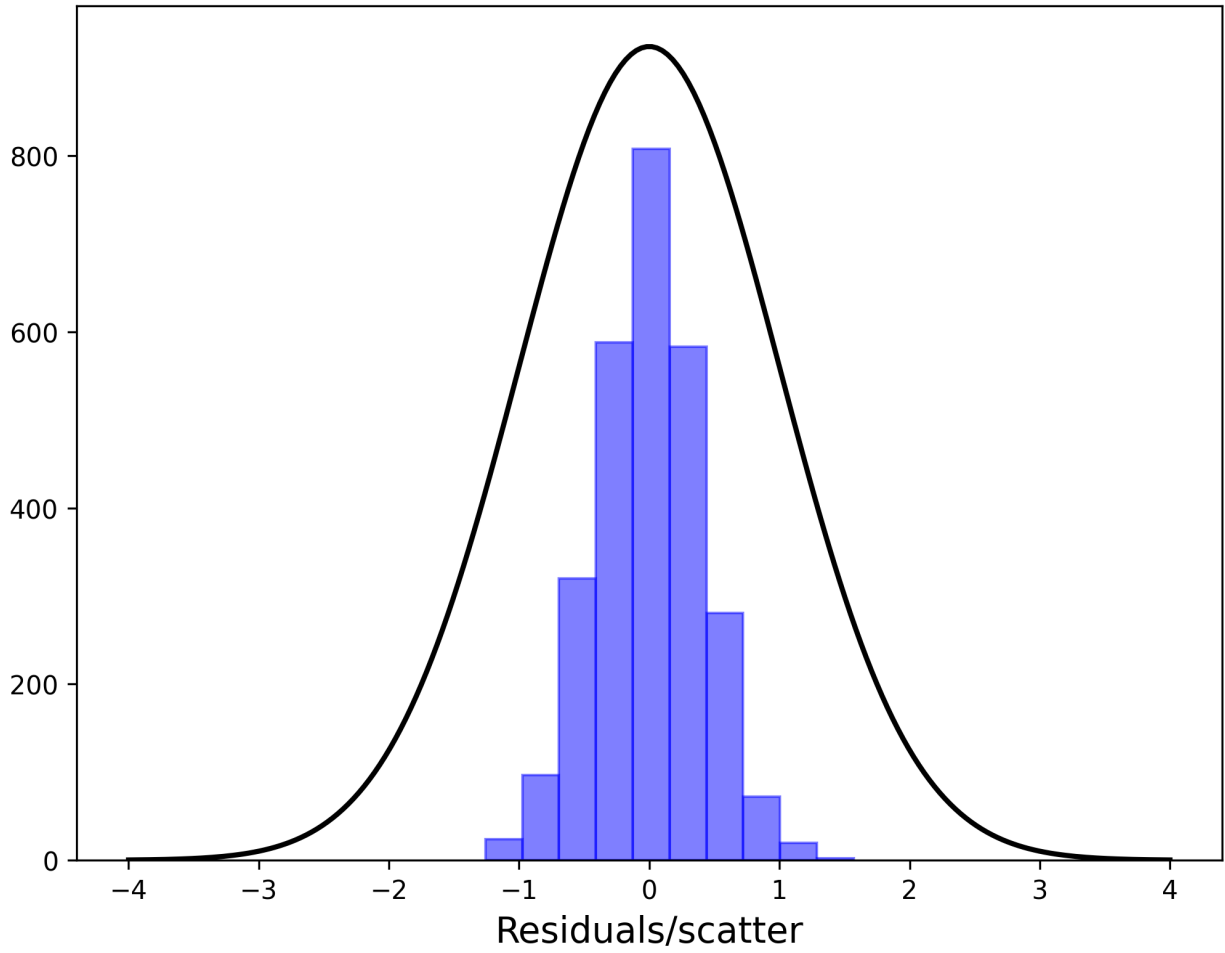
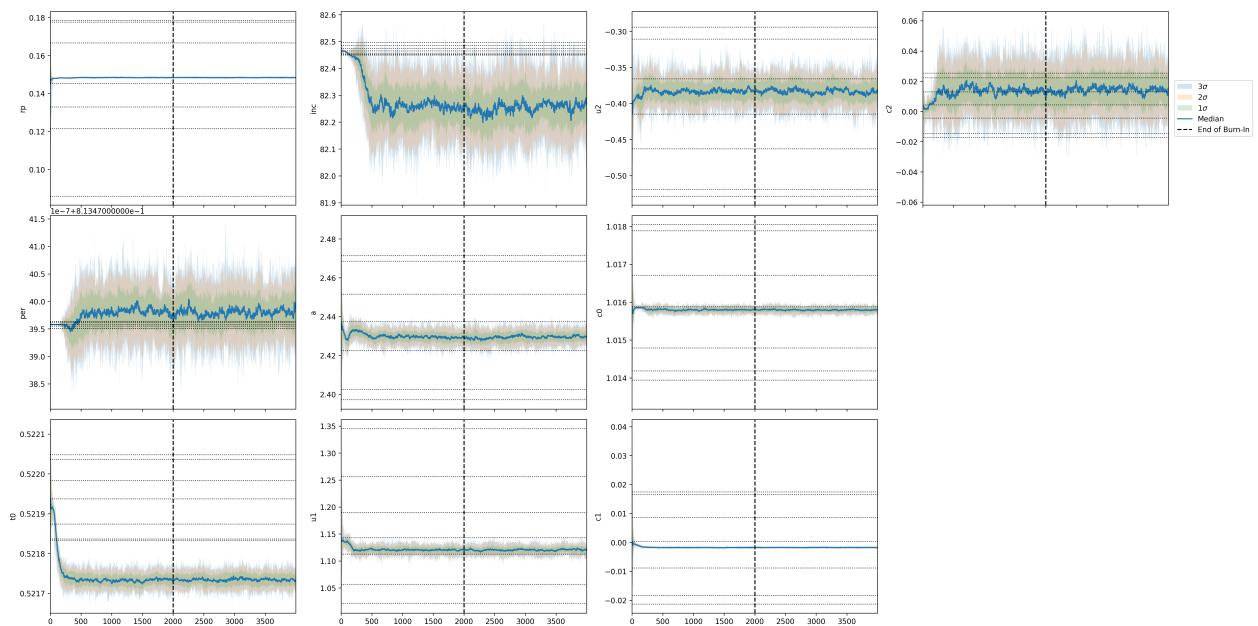


Fig. 15: Stage 5 output: Residual Distribution



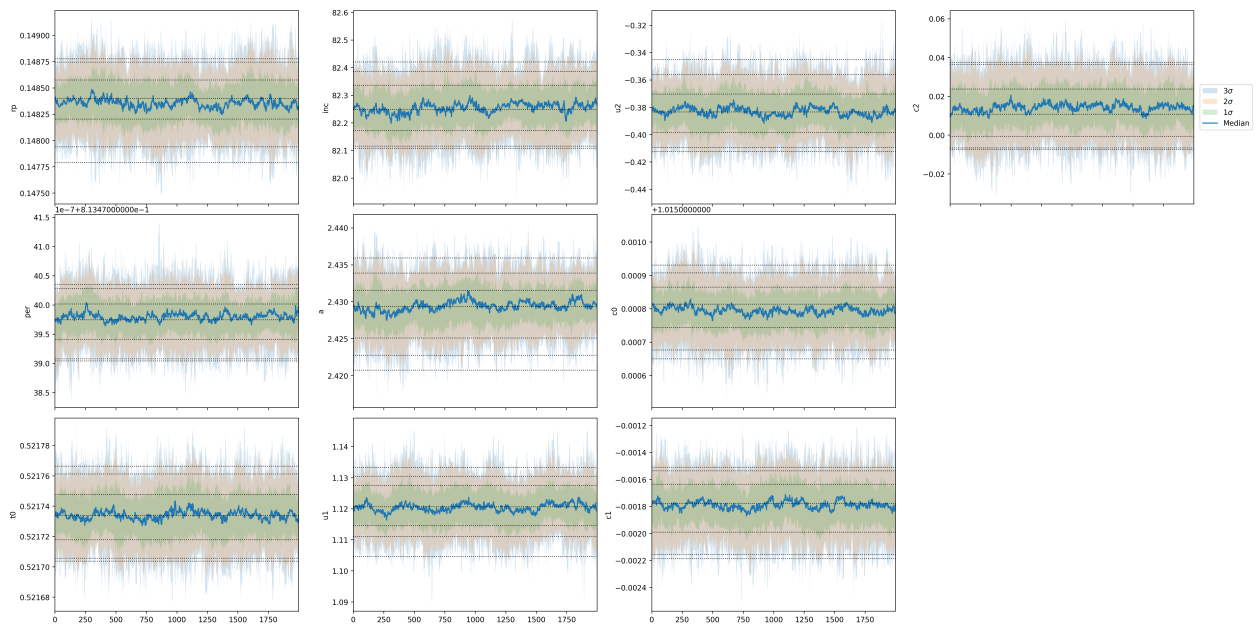


Fig. 16: Stage 5 output: Fitting Chains. Only made for emcee runs. Two version of the plot will be saved, one including the burn in steps and one without the burn in steps.

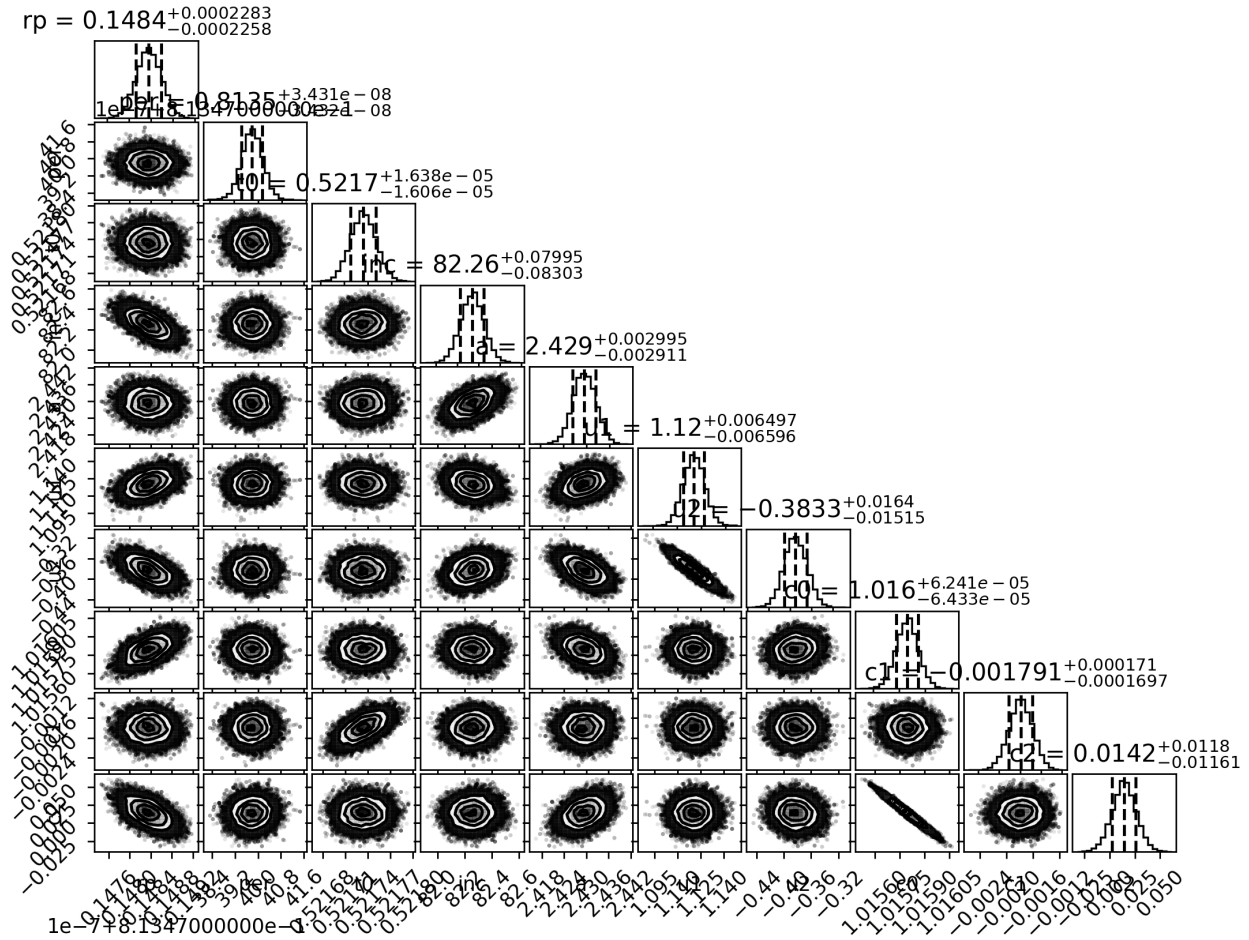


Fig. 17: Stage 5 output: Corner Plot

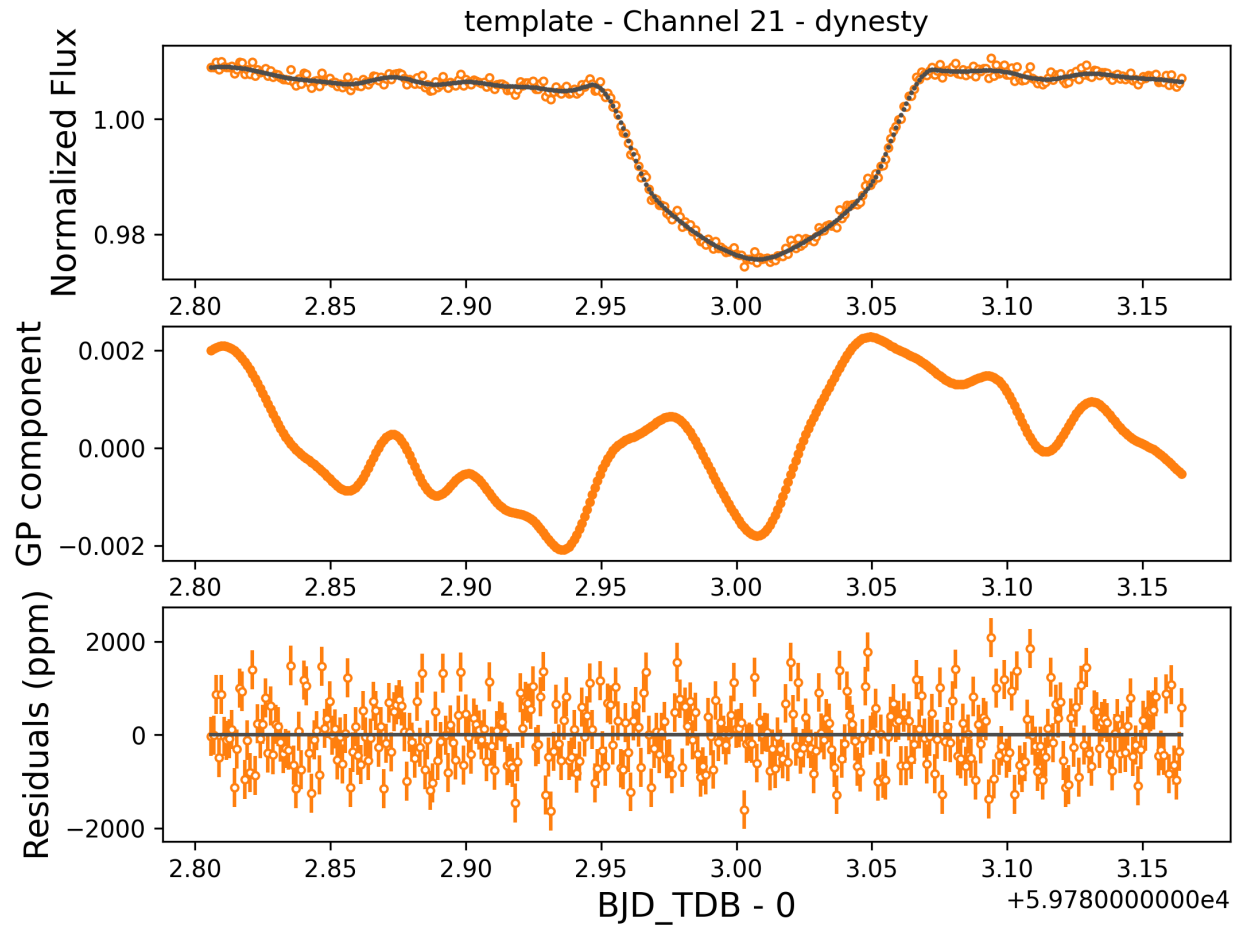


Fig. 18: Stage 5 output: Lightcurve, GP model, and Residual Plot

CONTRIBUTING TO EUREKA!

5.1 Page information

Here you will find information on how to contribute to Eureka! Which includes the importance of testing as well as some GitHub basics

5.2 Testing Eureka!

As of 2/23/2022, Eureka has working unit tests for NIRCcam (S3-S5) and NIRSpec (S2-S5). Currently, the end-to-end test for each instrument only performs least-squares fitting on the bare transit model for NIRCcam and NIRSpec, to test the minimum viable functionality. Test data for MIRI and NIRISS has not yet been created, so these instruments will not have unit tests until further in the future. Future testing will test the functionality of different fitting methods and different systematics models. **It is required for all contributors of Eureka! to run these tests locally before opening a pull request with their new code.** By running the tests locally, the contributor will be able to see whether the functionality of the main code is still intact. This requirement is common in software development teams and is meant to encourage smooth collaboration by helping track small bugs or changes that affect the basic functionality of the code so that colleagues won't have to.

For these tests, the `pytest` package ([link](#)) will be used. Once installed, the user submitting a pull request may navigate to the tests folder in Eureka! (`eureka/tests`) and run the following command:

```
pytest
```

Which will run the entire suite of tests found within the folder. To run a specific instrument test, the following command can be used:

```
pytest -k test_NIRCcam
```

or

```
pytest -k test_NIRSpec
```

All the tests should pass and a result similar to the following picture should be seen.

```

===== test session starts =====
platform linux -- Python 3.8.6, pytest-6.2.4, py-1.10.0, pluggy-0.
13.1
rootdir: /home/gguzmanc/Desktop/Eureka
plugins: asdf-2.8.1
collected 10 items

test_general.py .

IMPORTANT: Make sure that any changes to the ecf files are include
d in demo ecf files and documentation (docs/source/ecf.rst)

NIRCam test:
. [ 20%]
test_lightcurve_fitting.py ..... [100%]

===== 10 passed in 15.62s =====

```

If this isn't the case, tracking the error will be necessary. Some common errors regarding basic functionality of the code are:

- Renaming functions or packages with no follow-through in code applications of the function (or in test code)
- Added ecf file parameters with no follow-through in code applications of the parameter (or test ecf files)
- Bugs - spelling errors, indentation errors, escape sequence errors

It is therefore the responsibility of the contributor to update the tests and the code until local tests run correctly. Of course, there will be times where this might be harder than expected, and in those cases we welcome contributors to speak thoroughly on the issues within their pull request so other team members may help them and be aware of the problem.

5.3 GitHub Basics

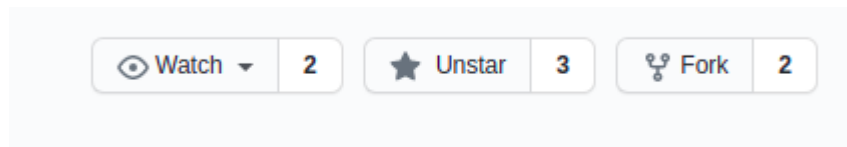
This section will go over how to best contribute to the Eureka project utilizing GitHub tools such as forking, and will be dedicated to Eureka-specific examples. If you are a beginner to GitHub, a comprehensive introduction to GitHub can be found on the following page.

Because Eureka is a repository where contributions are by invitation, cloning the repository locally and trying to push to it will not work due to lack of permissions. Yet, this does not mean people are excluded from contributing to Eureka! The way to contribute is by creating a GitHub “fork” of the original repository. A fork creates a copy of the repository on your own GitHub account that you can then push to freely without disrupting the original repository's workflow. Once you have finished the feature you are working on in your fork of Eureka you can then open a “pull request” in the original Eureka repository, where the changes you made can be reviewed and perhaps even integrated by the repository owners!

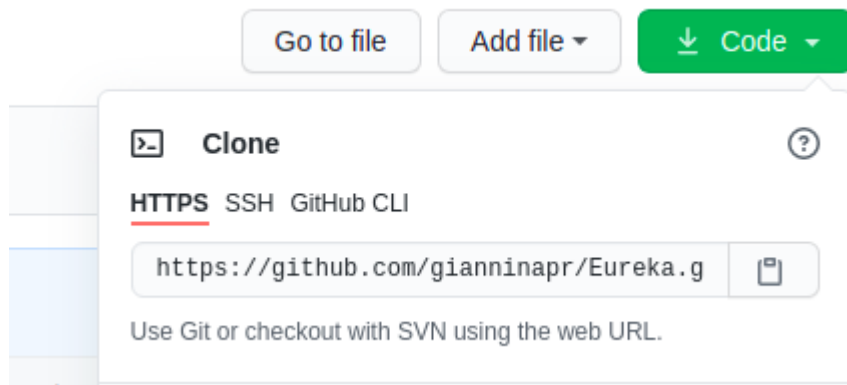
The details of this process are described below.

5.3.1 Creating a Fork of Eureka from Scratch

To create a fork of the Eureka repository all you have to do is go on the repository website and press the “fork” button that can be seen on the top right.



Once you have forked the repository you should be able to find the link of your own copy of Eureka by accessing the Eureka repository now found in your own personal GitHub page.



You can then copy the link and run

```
git clone [insert link here]
```

Now you have both a copy in your online GitHub repository (what we’ll call “remote”) and a copy you can work on in your local machine (what we’ll call “local”) that can push changes to your online copy of Eureka! To link the remote and local copies do:

```
git remote add origin [insert same link here]
```

This sets up tracking to “origin” which is your remote copy of Eureka, so that you can push to it.

5.3.2 Branching in Order to Keep a Clean Workflow

It’s not enough to just fork the repository and clone it, because Eureka is a project in which there are many contributions happening at the same time, you want to keep a clean copy of Eureka as your main branch! This will allow for you to easily download changes that have happened in the original Eureka repository and make sure that none of your work conflicts with them. The way to open a development branch within your own local copy of Eureka is:

```
git checkout -b name_of_branch
```

This will not only create the new branch with your name choice, it will also switch you to the branch. If you ever want to make sure which branch you’re on:

```
git branch
```

This will show you a list of branches. You should be seeing two branches, your development branch `name_of_branch` should have a star next to it to show that it is the active branch you’re on. The other branch is the main branch.

To switch between branches use:

```
git checkout branch_name
```

You should be doing all of your work in the development branch, and leave the main branch intact for updates. How to update your local repository will be discussed in detail in the sections below.

5.3.3 Committing and Pushing to your Fork

Once you have worked on all your changes, the way to make them available on the remote (online) version of the Eureka repository is to commit your changes and then push to your fork. To check the changes you have made do:

```
git status
```

This will give you a list of all your changes, and you should review that the files that have been added or modified are indeed the ones you worked on. Once you have that clear, you can stage all your changes for commit by doing:

```
git add --all
```

If you made a change you do not want to push yet or don't want to include yet, you could add the changes you are ready to commit one by one by doing

```
git add file_path/file.py
```

(as it shows up on the status list), and avoid adding the ones you don't want to stage yet.

If you want to discard an entire set of changes to a file, you can do:

```
git checkout -- file_path/file.py
```

Yet keep in mind this will delete **all** changes made to a file.

Once you have all the changes staged for commit it is time to commit. You can make sure the changes you are committing are as expected by doing

```
git status
```

once more. Anything that is staged for commit should appear under "Changes to be committed". If you need to unstage a file, you can do:

```
git reset HEAD file_path/file
```

This will prevent that file from being included in the commit unless you stage it again by doing

```
git add
```

Once you are sure you have all the changes you want to commit you can do:

```
git commit -m "Add a commit message here"
```

The commit message should be descriptive enough to track the changes but not too long.

Once the changes are committed you push them to your online repository by doing:

```
git push
```


Once you've done this your changes and branch should appear on your online version of the Eureka repository. You can go to your GitHub page and make sure this is correct.

5.3.4 Updating your Local Fork with Any Updates from the Original Eureka Repository

Because Eureka is a collaborative project, there will be other people working on their own features at the same time as you. If any changes are implemented to the original Eureka repository, this can become a conflict on your future pull request. That is why it is imperative to update your local fork with any updates from the original Eureka repository before attempting to open a pull request.

First, we need to set up a connection between your local copy of the Eureka repository and the remote **original** Eureka repository. To see the current remote repositories linked to your local repository you can do:

```
git remote -v
```

This should currently show you something like the following

```
origin      https://github.com/your_username/Eureka.git (fetch)
origin      https://github.com/your_username/Eureka.git (push)
```

What this is showing us is that your local branch is only currently connected to the remote copy of Eureka. Yet, in order to update your code with updates from the original Eureka repository, you need to establish a connection to it. Utilizing the standard nomenclature “upstream” (for the original repository):

```
git remote add upstream https://github.com/kevin218/Eureka.git
```

Now, if you run

```
git remote -v
```

again, you should see the new links to the original Eureka repository:

```
origin      https://github.com/your_username/Eureka.git (fetch)
origin      https://github.com/your_username/Eureka.git (push)
upstream    https://github.com/kevin218/Eureka.git (fetch)
upstream    https://github.com/kevin218/Eureka.git (push)
```

You'll also want to setup your main branch to track from the remote repository automatically, since that branch will be dedicated to importing the updates done to the original Eureka repository. The way to do this is to first make sure you are in the main branch:

```
git checkout main
```

OR

```
git checkout main_branch_name
```

Then set the upstream as the original Eureka repository

```
git branch --set-upstream upstream/main
```

This has now set up your main local branch to track changes done in the original Eureka remote repository (upstream) as opposed to your own copy of Eureka (origin). It is imperative then that you make sure all changes you make are in your development branch. An advice is to constantly double check what branch you're working

on. If you have made changes in the main branch by mistake, see how to resolve this in the section *Common Mistakes* below.

It is also worth mentioning that the remotes can be called anything, “origin” and “upstream” are just the standard nomenclature. They can be called anything as long as you are able to keep track of which one is which, which is always possible to check by doing:

```
git remote -v
```

Once all of this is setup you are ready to check whether any changes made to the original repository conflict with your own changes.

```
git checkout main # (double check you're on your main branch)
git pull
```

This will give you any changes that have been done to Eureka on your main branch. This process should go smoothly as long as you have not made any changes to the main branch and done all your work in the development branch. Once you have pulled, you need to switch back to your development branch and merge the changes from main.

```
git checkout development_branch
```

Make sure you don’t have any changes staged for commit (either commit and push them or unstage them). Then do:

```
git merge main
```

OR

```
git merge main_branch_name
```

This will merge all the changes done on the main branch into your feature branch. Git will then proceed to tell you if this merge can be done smoothly or not. If it can it will simply pop up a text editor with a commit message along the lines of ‘Merged main to feature branch’, once you commit that merge, you can push it up to your remote repository by doing:

```
git push # **Don't forget this step!**
```

If there is a merge conflict, git will tell you. Merge conflicts are not typically hard to fix although they might seem scary, usually what it means is that while you were working on your feature someone else did work on the same lines of code, and your version and the original Eureka version are in conflict. Depending on the editor you use these merge conflicts can be easy to track down and resolve. If your editor doesn’t point you to the main conflicts automatically, git should tell you the files in which the merge conflicts occurred. You can then open the file and find lines that look like this:

```
<<<<<<< HEAD
current change
=====
change that would be merged
>>>>>>>>>>
```

Pick which one you’d like to keep by deleting the other one. Once you have resolved all conflicts you can finish the merge by doing the standard commit process: stage your changes and commit.

```
git add file_path/filename.py
```

OR

```
git add --all
```

and then

```
git commit -m "Commit message"  
git push
```

5.3.5 Opening a Pull Request with Eureka

It is good practice that before opening any pull request you should have finished the following checklist:

- [x] Made changes to the code
- [x] Committed those changes
- [x] Pushed the changes to your remote repository
- [x] Checked for any updates to the original Eureka
- [] Open Pull Request

Once you have taken care of these things, the next step is done through the GitHub web interface. Go to your remote copy of Eureka and you will see a button that says “Compare and Open Pull Request”. Press this button to open the pull request, you should then see a new page that shows all the changes done that are included in the pull request as well as a text box to include details and information about the changes done to the code in your pull request. It is always good to be as detailed as possible with anything that might help the reviewers understand your code better. Once you’ve reviewed the changes and described your feature you can open the pull request. Congratulations!

Important Note: Even after opening a pull request you can continue to work on your code, every change you push into the code will show up on the pull request. This is meant to be a feature since people can review and comment on your code and you can make the changes directly. Yet, if you are going to work on another feature separate from the one you opened a pull request for, then it is good to create a new development branch for that other feature and work it from there. This of course, as long as your new feature is standalone and does not depend on code submitted in your first pull request. If you started working on changes in your first development branch but you actually want them on a new branch, refer to the troubleshooting section below.

5.3.6 Troubleshooting

Didn’t fork and started working on a local version of the original Eureka

This is not an issue at all! Just make sure you have your origin and upstream setup correctly by doing

```
git remote -v
```

If origin is pointing to the original Eureka repository (<https://github.com/kevin218/Eureka.git>) and you want to keep the standard nomenclature discussed above then you can rename origin to upstream the following way:

```
git remote rename origin upstream
```

Once that’s done you can add your own fork as origin by forking the repository on GitHub as shown at the beginning of the tutorial, getting the link, and then doing:

```
git remote add origin https://github.com/your_username/Eureka.git # (fork link)
```

Eureka!

Now if you do

```
git remote -v
```

You should see something like

```
origin      https://github.com/your_username/Eureka.git (fetch)
origin      https://github.com/your_username/Eureka.git (push)
upstream    https://github.com/kevin218/Eureka.git (fetch)
upstream    https://github.com/kevin218/Eureka.git (push)
```

You're good to go!

Made Changes in Main Branch instead of Development Branch

Let's say you worked on changes in your main branch instead of your development branch. If they are not yet committed you can do

```
git add --all # (stage your changes)
git stash
```

Switch to your development branch by doing

```
git checkout branch_name
```

and then do

```
git stash pop
```

If the changes have been committed then it is a little more complicated, but not too much to worry about. You should create a new branch while being on the main branch that has the committed changes

```
git checkout -b new_development_branch
```

This will create a new branch with the committed changes included, and it should then become your new development branch. Then checkout to the main branch

```
git checkout main_branch_name
```

and reset the committed changes by doing

```
git reset --hard HEAD^1
```

If you have committed to the main branch more than once, then the number should be however many commits back is the original main branch's last commit, for example:

```
----- 1 ----- 2 ----- 3 ----- 4 ----- 5 ----- 6
              ^                               ^
            original                         master
          master commit
```

Would be:

```
git reset --hard HEAD^4
```

THE CODE (API)

6.1 lib

6.1.1 lib.astropytable

`eureka.lib.astropytable.readtable(filename)`

`eureka.lib.astropytable.savetable_S3(filename, time, wave_id, stdspec, stdvar, optspec, opterr)`

Saves data in an event as .txt using astropy

Parameters

event [An Event instance.]

Returns

.txt file

`eureka.lib.astropytable.savetable_S4(filename, time, wavelength, bin_width, lcdata, lcerr)`

Saves data in an event as .txt using astropy

Parameters

event [An Event instance.]

Returns

.txt file

`eureka.lib.astropytable.savetable_S5(filename, time, wavelength, bin_width, lcdata, lcerr, model, residuals)`

Saves data in an event as .txt using astropy

Parameters

event [An Event instance.]

Returns

.txt file

`eureka.lib.astropytable.savetable_S6(filename, wavelength, bin_width, tr_depth, tr_depth_err, ecl_depth, ecl_depth_err)`

Saves data in an event as .txt using astropy

Parameters

event [An Event instance.]

Returns**.txt file****6.1.2 lib.centroid****eureka.lib.centroid.ctrgauss**(*data, guess=None, mask=None, indarr=None, trim=None*)

Finds and records the stellar centroid of a set of images by fitting a two dimensional Gaussian function to the data.

It does not find the average centroid, but instead records the centroid of each image in the supplied frame parameters array at the supplied indices. The frame parameters array is assumed to have the same number of rows as the number of frames in the data cube.

Parameters

data [ndarray (2D)] The stellar image.

guess [array_like] The initial guess of the position of the star. Has the form (y, x) of the guess center.

mask [ndarray (2D)] The stellar image.

indarr [array_like] The indices of the x and y center columns of the frame parameters and the width index. Defaults to 4, 5, and 6 respectively.

trim [Scalar (positive)] If trim!=0, trims the image in a box of 2*trim pixels around the guess center. Must be !=0 for 'col' method.

Returns

center [y, x] The updated frame parameters array. Contains the centers of each star in each image and their average width.

eureka.lib.centroid.ctrguess(*data, mask=None, guess=None*)

Calculates crude initial guesses of parameters required for Gaussian centroiding of stellar images.

Specifically, this function guesses the flux of the center of a star, the array indices of this location, and a rough guess of the width of the associated PSF. This method is not robust to bad pixels or any other outlying values.

Parameters

data [ndarray (2D)] The image in the form of a 2D array containing the star to be centroided. Works best if this is a small subarray of the actual data image.

mask [ndarray (2D)] The image in the form of a 2D array containing the star to be centroided. Works best if this is a small subarray of the actual data image.

Returns

ght [scalar] The rough estimate of the height (or max flux) of the stars PSF.

gwd [tuple] The guessed width of the PSF, in the form (gwdy, gwdx) where *gwdy* and *gwdx* are the y and x widths respectively.

gct [tuple] The guessed center of the PSF, in the form (gcty, gctx) where *gcty* and *gctx* are the y and x center indices respectively.

Notes

Logic adapted from *gaussian.py*

6.1.3 lib.clipping

```
eureka.lib.clipping.clip_outliers(data, log, wavelength, sigma=10, box_width=5, maxiters=5,
                                   boundary='extend', fill_value='mask', verbose=False)
```

Find outliers in 1D time series.

Be careful when using this function on a time-series with known astrophysical variations. The variable `box_width` should be set to be significantly smaller than any astrophysical variation timescales otherwise these signals may be clipped.

Parameters

- data: ndarray (1D, float)** The input array in which to identify outliers
- log: logedit.Logedit** The open log in which notes from this step can be added.
- wavelength: float** The wavelength currently under consideration.
- sigma: float** The number of sigmas a point must be from the rolling mean to be considered an outlier
- box_width: int** The width of the box-car filter (used to calculate the rolling median) in units of number of data points
- maxiters: int** The number of iterations of sigma clipping that should be performed.
- fill_value: string or float** Either the string 'mask' to mask the outlier values, 'boxcar' to replace data with the mean from the box-car filter, or a constant float-type fill value.

Returns

- data: ndarray (1D, boolean)** An array with the same dimensions as the input array with outliers replaced with `fill_value`.

Notes

History:

- **Jan 29-31, 2022 Taylor Bell** Initial version, added logging

```
eureka.lib.clipping.gauss_removal(img, mask, linspace, where='bkg')
```

An additional step to remove cosmic rays. This fits a Gaussian to the background (or a skewed Gaussian to the orders) and masks data points which are above a certain sigma.

Parameters

- img** [np.ndarray] Single exposure image.
- mask** [np.ndarray] An approximate mask for the orders.
- linspace** [array] Sets the lower and upper bin bounds for the pixel values. Should be of length = 2.
- where** [str, optional] Sets where the mask is covering. Default is *bkg*. Other option is *order*.

Returns

- img** [np.ndarray] The same input image, now masked for newly identified outliers.

6.1.4 lib.disk

`eureka.lib.disk.disk(r, ctr, size, status=False)`

6.1.5 lib.gaussian

Name

gaussian

File

gaussian.py

Description

Routines for evaluating, estimating parameters of, and fitting Gaussians.

Package Contents

N-dimensional functions:

gaussian(x, width=1., center=0., height=None, params=None) Evaluate the Gaussian function with given parameters at x (n-dimensional).

fitgaussian(y, x) Calculates a Gaussian fit to (y, x) data, returns (width, center, height).

1-dimensional functions:

gaussianguess(y, x=None) Crudely estimates the parameters of a Gaussian that fits the (y, x) data.

Examples:

See fitgaussian() example.

Revisions

2007-09-17 0.1 jh@physics.ucf.edu Initial version 0.01, portions adapted from <http://www.scipy.org/Cookbook/FittingData>.

2007-10-02 0.2 jh@physics.ucf.edu Started making N-dimensional, put width before center in args.

2007-11-13 0.3 jh@physics.ucf.edu Made N-dimensional. 2008-12-02 0.4 nlust@physics.ucf.edu Made fit gaussian return errors, and

fixed a bug generating initial guesses

2009-10-25 0.5 jh@physics.ucf.edu Standardized all headers, fixed an error in a fitgaussian example, added example ">>>"s and plot labels.

`eureka.lib.gaussian.fitgaussian(y, x=None, bgpars=None, fitbg=0, guess=None, mask=None, weights=None, maskg=False, yxguess=None)`

Fits an N-dimensional Gaussian to (value, coordinate) data.

Parameters

- y** [ndarray] Array giving the values of the function.
- x** [ndarray] (optional) Array (any shape) giving the abscissas of y (if missing, uses `np.indices(y)`). The highest dimension must be equal to the number of other dimensions (i.e., if x has 6 dimensions, the highest dimension must have length 5). The rest of the dimensions must have the same shape as y. Must be sorted ascending (which is not checked), if guess is not given.
- bgpars** [ndarray or tuple, 3-elements] Background parameters, the elements determine a X- and Y-linearly dependant level, of the form: $f = Y \cdot \text{bgparam}[0] + X \cdot \text{bgparam}[1] + \text{bgparam}[2]$ (Not tested for 1D yet).
- fitbg** [Integer] This flag indicates the level of background fitting: `fitbg=0`: No fitting, estimate the bg as `median(data)`. `fitbg=1`: Fit a constant to the bg ($\text{bg} = c$). `fitbg=2`: Fit a plane as bg ($\text{bg} = a \cdot x + b \cdot y + c$).
- guess** [tuple, (width, center, height)] Tuple giving an initial guess of the Gaussian parameters for the optimizer. If supplied, x and y can be any shape and need not be sorted. See `gaussian()` for meaning and format of this tuple.
- mask** [ndarray] Same shape as y. Values where its corresponding mask value is 0 are disregarded for the minimization. Only values where the mask value is 1 are considered.
- weights** [ndarray] Same shape as y. This array defines weights for the minimization, for scientific data the weights should be $1/\sqrt{\text{variance}}$.

Returns

- params** [ndarray] This array contains the best fitting values parameters: width, center, height, and if requested, bgpars. with:
 width : The fitted Gaussian widths in each dimension. center : The fitted Gaussian center coordinate in each dimension. height : The fitted height.
- err** [ndarray] An array containing the concatenated uncertainties corresponding to the values of params. For example, 2D input gives `np.array([widthyerr, widthxerr, centeryerr, centerxerr, heighterr])`.

Notes

If the input does not look anything like a Gaussian, the result might not even be the best fit to that.

Method: First guess the parameters (if no guess is provided), then call a Levenberg-Marquardt optimizer to finish the job.

Examples

```
>>> import matplotlib.pyplot as plt
>>> import gaussian as g
```

```
>>> # parameters for X
>>> lx = -3. # low end of range
>>> hx = 5. # high end of range
>>> dx = 0.05 # step
```

```
>>> # parameters of the noise
>>> nc = 0.0 # noise center
>>> ns = 1.0 # noise width
>>> na = 0.2 # noise amplitude
```

```
>>> # 1D Example
```

```
>>> # parameters of the underlying Gaussian
>>> wd = 1.1 # width
>>> ct = 1.2 # center
>>> ht = 2.2 # height
```

```
>>> # x and y data to fit
>>> x = np.arange(lx, hx + dx / 2., dx)
>>> x += na * np.random.normal(nc, ns, x.size)
>>> y = g.gaussian(x, wd, ct, ht) + na * np.random.normal(nc, ns, x.size)
>>> s = x.argsort() # sort, in case noise violated order
>>> xs = x[s]
>>> ys = y[s]
```

```
>>> # calculate guess and fit
>>> (width, center, height) = g.gaussianguess(ys, xs)
>>> (fw, fc, fh, err) = g.fitgaussian(ys, xs)
```

```
>>> # plot results
>>> plt.clf()
>>> plt.plot(xs, ys)
>>> plt.plot(xs, g.gaussian(xs, wd, ct, ht))
>>> plt.plot(xs, g.gaussian(xs, width, center, height))
>>> plt.plot(xs, g.gaussian(xs, fw, fc, fh))
>>> plt.title('Gaussian Data, Guess, and Fit')
>>> plt.xlabel('Abcissa')
>>> plt.ylabel('Ordinate')
>>> # plot residuals
>>> plt.clf()
>>> plt.plot(xs, ys - g.gaussian(xs, fw, fc, fh))
>>> plt.title('Gaussian Fit Residuals')
>>> plt.xlabel('Abcissa')
>>> plt.ylabel('Ordinate')
```

```
>>> # 2D Example
```

```
>>> # parameters of the underlying Gaussian
>>> wd = (1.1, 3.2) # width
>>> ct = (1.2, 3.1) # center
>>> ht = 2.2        # height
```

```
>>> # x and y data to fit
>>> nx = (hx - lx) / dx + 1
>>> x = np.indices((nx, nx)) * dx + lx
>>> y = g.gaussian(x, wd, ct, ht) + na * np.random.normal(nc, ns, x.shape[1:])
```

```
>>> # calculate guess and fit
>>> #(width, center, height) = g.gaussianguess(y, x) # not in 2D yet...
>>> (fw, fc, fh, err) = g.fitgaussian(y, x, (wd, ct, ht))
```

```
>>> # plot results
>>> plt.clf()
>>> plt.title('2D Gaussian Given')
>>> plt.xlabel('X')
>>> plt.ylabel('Y')
>>> plt.imshow(g.gaussian(x, wd, ct, ht))
>>> plt.clf()
>>> plt.title('2D Gaussian With Noise')
>>> plt.xlabel('X')
>>> plt.ylabel('Y')
>>> plt.imshow(y)
>>> #plt.imshow(g.gaussian(x, width, center, height)) # not in 2D yet...
>>> plt.clf()
>>> plt.title('2D Gaussian Fit')
>>> plt.xlabel('X')
>>> plt.ylabel('Y')
>>> plt.imshow(g.gaussian(x, fw, fc, fh))
>>> plt.clf()
>>> plt.title('2D Gaussian Fit Residuals')
>>> plt.xlabel('X')
>>> plt.ylabel('Y')
>>> plt.imshow(y - g.gaussian(x, fw, fc, fh))
```

```
>>> # All cases benefit from...
```

```
>>> # show difference between fit and underlying Gaussian
>>> # Random data, your answers WILL VARY.
>>> np.array(fw) - np.array(wd)
array([ 0.00210398, -0.00937687])
>>> np.array(fc) - np.array(ct)
array([-0.00260803,  0.00555011])
>>> np.array(fh) - np.array(ht)
0.0030143371034774269
```

```

>>> Last Example:
>>> x = np.indices((30,30))
>>> g1 = g.gaussian(x, width=(1.2, 1.15), center=(13.2,15.75), height=1e4,
>>>                bgpars=[0.0, 0.0, 100.0])
>>> error = np.sqrt(g1) * np.random.randn(30,30)
>>> y = g1 + error
>>> var = g1
>>>
>>> plt.figure(1)
>>> plt.clf()
>>> plt.imshow(y, origin='lower_left', interpolation='nearest')
>>> plt.colorbar()
>>> plt.title('2D Gaussian')
>>> plt.xlabel('X')
>>> plt.ylabel('Y')
>>>
>>> guess = ((1.2,1.2),(13,16.),1e4)
>>> reload(g)
>>> fit = g.fitgaussian(y, x, bgpars=[0.0, 0.0, 110.], fitbg=1, guess=guess,
>>>                    mask=None, weights=1/np.sqrt(var))
>>> print(fit[0])

```

`eureka.lib.gaussian.fitgaussians(y, x=None, guess=None, sigma=1.0)`

Fit position and flux of a data image with gaussians, same sigma is applied to all dispersions. Parameters:

——— `y`: array_like

Array giving the values of the function.

`x` [array_like] (optional) Array (any shape) giving the abscissas of `y` (if missing, uses `np.indices(y)`).

`guess` [2D-tuple, [[width1, center1, height1],

[width2, center2, height2], ...]

Tuple giving an initial guess of the Gaussian parameters for the optimizer. If supplied, `x` and `y` can be any shape and need not be sorted. See `gaussian()` for meaning and format of this tuple.

`eureka.lib.gaussian.gaussian(x, width=1.0, center=0.0, height=None, bgpars=[0.0, 0.0, 0.0])`

Evaluates the Gaussian and a background with given parameters at locations in `x`.

Parameters

x [ndarray (any shape)] Abcissa values. Arranged as the output of `np.indices()` but may be float. The highest dimension must be equal to the number of other dimensions (i.e., if `x` has 6 dimensions, the highest dimension must have length 5, and each of those must give the coordinate along the respective axis). May also be 1-dimensional. Default: `np.indices(y.shape)`.

width [array_like] The width of the Gaussian function, sometimes called sigma. If scalar, assumed constant for all dimensions. If array, must be linear and the same length as the first dimension of `x`. In this case, each element gives the width of the function in the corresponding dimension. Default: `[1.]`.

center [array_like] The mean value of the Gaussian function, sometimes called `x0`. Same scalar/array behavior as `width`. Default: `[0.]`.

height [scalar] The height of the Gaussian at its center. If not set, initialized to the value that makes the Gaussian integrate to 1. If you want it to integrate to another number, leave height

alone and multiply the result by that other number instead. Must be scalar. Default: `[product(1./sqrt(2 * pi * width**2))]`.

bgpars [ndarray or tuple, 3-element] Background parameters, the elements determine a X- and Y-linearly dependant level, of the form: $f = Y \cdot \text{bgparam}[0] + X \cdot \text{bgparam}[1] + \text{bgparam}[2]$ (Not tested for 1D yet).

Returns

results [ndarray, same shape as x (or first element of x if multidimensional)] This function returns the Gaussian function of the given width(s), center(s), and height applied to its input plus a linear background level. The Gaussian function is: $f(x) = 1./\sqrt{2 * \pi * \text{width}^2} * \exp(-0.5 * ((x - \text{center}) / \text{width})^2)$. It is defined in multiple dimensions as the product of orthogonal, single-dimension Gaussians.

Examples

```
>>> import matplotlib.pyplot as plt
>>> import gaussian as g
```

```
>>> x = np.arange(-10., 10.005, 0.01)
>>> plt.plot(x, g.gaussian(x))
>>> plt.title('Gaussian')
>>> plt.xlabel('Abcissa')
>>> plt.ylabel('Ordinate')
```

```
>>> # use an array [3] as a single parameter vector
>>> z = np.array([2., 2, 3])
>>> plt.plot(x, g.gaussian(x, *z))
```

```
>>> # Test that it integrates to 1.
>>> a = np.indices([100, 100]) - 50
>>> print(np.sum(g.gaussian(a, 3, 3)))
0.999999999999997
>>> print(np.sum(g.gaussian(a, np.array([1,2]), np.array([2,3]))))
1.00000000107
```

```
>>> plt.clf()
>>> plt.imshow(g.gaussian(a, [3,5], [7,3]))
>>> plt.title('2D Gaussian')
>>> plt.xlabel('X')
>>> plt.ylabel('Y')
```

```
>>> A gaussian + a linear background level:
>>> g2 = g.gaussian(x, width=(1.2, 1.15), center=(13.2,15.75), height=4.3,
>>>                 bgpars=[0.05, 0.01, 1.0])
>>> plt.figure(1)
>>> plt.clf()
>>> plt.imshow(g2, origin='lower_left', interpolation='nearest')
>>> plt.colorbar()
>>> plt.title('2D Gaussian')
>>> plt.xlabel('X')
>>> plt.ylabel('Y')
```

```
>>> plt.figure(2)
>>> plt.clf()
>>> plt.plot(g2[13,:])
>>> plt.title('X slice of 2D Gaussian')
>>> plt.xlabel('X')
>>> plt.ylabel('Z')
```

```
>>> plt.figure(3)
>>> plt.clf()
>>> plt.plot(g2[:,16])
>>> plt.title('Y slice of 2D Gaussian')
>>> plt.xlabel('Y')
>>> plt.ylabel('Z')
```

eureka.lib.gaussian.**gaussianguess**(*data*, *mask=None*, *yxguess=None*)

eureka.lib.gaussian.**gaussians**(*x*, *param*)

Evaluate more than 1 gaussian.

eureka.lib.gaussian.**resids**(*param*, *x*, *ngauss*, *y*)

eureka.lib.gaussian.**residuals**(*params*, *x*, *data*, *mask*, *weights*, *bgpars*, *fitbg*)

Calculates the residuals between data and a gaussian model determined by the rest of the parameters. Used in fitgaussian.

Parameters

params [1D ndarray] This array contains the parameters desired to fit with fitgaussian, depending on fitbg, the number of elements varies.

x [ndarray] Array (any shape) giving the abscissas of data.

data [ndarray] Array giving the values of the function.

mask [ndarray] Same shape as data. Values where its corresponding mask value is 0 are disregarded for the minimization. Only values where the mask value is 1 are considered.

weights [ndarray] Same shape as data. This array defines weights for the minimization, for scientific data the weights should be $1/\sqrt{\text{variance}}$.

bgpars [ndarray or tuple, 3-elements] Background parameters, the elements determine a X- and Y-linearly dependant level, of the form: $\text{background} = Y \cdot \text{bgparam}[0] + X \cdot \text{bgparam}[1] + \text{bgparam}[2]$

fitbg [Integer] This flag indicates the level of background fitting: fitbg=0: No fitting, estimate the bg as median(data). fitbg=1: Fit a constant to the bg ($\text{bg} = c$). fitbg=2: Fit a plane as bg ($\text{bg} = a \cdot x + b \cdot y + c$).

Returns

residuals [1D ndarray] An array of the (unmasked) weighted residuals between data and a gaussian model determined by params (and bgpars when necessary).

6.1.6 lib.gelmanrubin

eureka.lib.gelmanrubin.convergetest(*pars*, *nchains*)

Driver routine for gelmanrubin.

Perform convergence test of Gelman & Rubin (1992) on a MCMC chain.

Parameters

pars [ndarray] A 2D array containing a separate parameter MCMC chain per row.

nchains [scalar] The number of chains to split the original chain into. The length of each chain MUST be evenly divisible by *nchains*.

Returns

psrf [ndarray] The potential scale reduction factors of the chain. If the chain has converged, each value should be close to unity. If they are much greater than 1, the chain has not converged and requires more samples. The order of psrfs in this vector are in the order of the free parameters.

meanpsrf [scalar] The mean of *psrf*. This should be ~1 if your chain has converged.

Notes

History:

2010-08-20 ccampo Initial version.

Examples

Consider four MCMC runs that has already been loaded. The individual fits are located in the *fit* list. These are for channels 1-4.

```
>>> import gelmanrubin
>>> import numpy as np
>>> # channels 1/3 free parameters
>>> ch13pars = np.concatenate((fit[0].allparams[fit[0].freepars],
>>>                             fit[2].allparams[fit[2].freepars]))
```

```
>>> # channels 2/4 free parameters
>>> ch24pars = np.concatenate((fit[1].allparams[fit[1].freepars],
>>>                             fit[3].allparams[fit[3].freepars]))
```

```
>>> # number of chains to split into
>>> nchains = 4
```

```
>>> # test for convergence
>>> ch13conv = gelmanrubin.convergetest(ch13pars, nchains)
>>> ch24conv = gelmanrubin.convergetest(ch24pars, nchains)
```

```
>>> # show results
>>> print(ch13conv)
(array([ 1.02254252,  1.00974035,  1.04838778,  1.0017869 ,  1.7869707 ,
```

(continues on next page)

(continued from previous page)

```

2.15683239, 1.00506215, 1.00235165, 1.06784124, 1.04075207,
1.01452032]), 1.1960716427734874)
>>> print(ch24conv)
(array([ 1.01392515,  1.00578357,  1.03285576,  1.13138702,  1.0001787 ,
        3.52118005,  1.10592542,  1.05514509,  1.00101459]),
        1.3185994837687156)

```

`eureka.lib.gelmanrubin.gelmanrubin(chain, nchains)`

Perform convergence test of Gelman & Rubin (1992) on a MCMC chain.

Parameters

chain [ndarray] A vector of parameter samples from a MCMC routine.

nchains [scalar] The number of chains to split the original chain into. The length of *chain* WILL BE MODIFIED if NOT evenly divisible by *nchains*.

Returns

psrf [scalar] The potential scale reduction factor of the chain. If the chain has converged, this should be close to unity. If it is much greater than 1, the chain has not converged and requires more samples.

Notes

History:

2010-08-20 ccampo Initial version.

2011-07-07 kevin Removed chain length constraint

6.1.7 lib.logedit

`class eureka.lib.logedit.Logedit(logname, read=None)`

Bases: object

This object handles writing text outputs into a log file and to the screen as well.

Methods

<code>closelog()</code>	Closes an existing log file.
<code>writetclose(message[, mute, end])</code>	Print message in terminal and log, then close log.
<code>writellog(message[, mute, end])</code>	Prints message in the terminal and stores it in the log file.

`closelog()`

Closes an existing log file.

`writetclose(message, mute=False, end='\n')`

Print message in terminal and log, then close log.

`writellog(message, mute=False, end='\n')`

Prints message in the terminal and stores it in the log file.

6.1.8 lib.manageevent

`eureka.lib.manageevent.loadevent(filename, load=[], loadfilename=None)`

Loads an event stored in .dat and .h5 files.

Parameters

filename [String] The string contains the name of the event file.

load [String tuple] The elements of this tuple contain the parameters to read. We usually use the values: 'data', 'uncd', 'head', 'bdmskd', 'brmskd' or 'mask'.

Returns

This function return an Event instance.

Notes

The input filename should not have the .dat nor the .h5 extentions.

Examples

See package example.

`eureka.lib.manageevent.saveevent(event, filename, save=[], delete=[], protocol=3)`

Saves an event in .dat (using cpickle) and .h5 (using h5py) files.

Parameters

event [An Event instance.]

filename [String] The string contains the name of the event file.

save [String tuple] The elements of this tuple contain the parameters to save. We usually use the values: 'data', 'uncd', 'head', 'bdmskd', 'brmskd' or 'mask'.

delete [String tuple] Parameters to be deleted.

Returns

Notes

The input filename should not have the .dat nor the .h5 extentions. Side effect: This routine deletes all parameters except 'event'

after saving it.

Examples

See package example.

`eureka.lib.manageevent.updateevent(event, filename, add)`

Adds parameters given by add from filename to event.

Parameters

event [An Event instance.]

filename [String] The string contains the name of the event file.

add [String tuple] The elements of this tuple contain the parameters to add. We usually use the values: ‘data’, ‘uncd’, ‘head’, ‘bdmskd’, ‘brmaskd’ or ‘mask’.

Returns

This function return an Event instance.

Notes

The input filename should not have the .dat nor the .h5 extentions.

Examples

See package example.

6.1.9 lib.medstddev

`eureka.lib.medstddev.medstddev(data, mask=None, medi=False, axis=0)`

Compute the stddev with respect to the median.

This is rather than the standard method of using the mean.

6.1.10 lib.plots

`eureka.lib.plots.set_rc(style='preserve', usetex=False, from_scratch=False, **kwargs)`

Function to adjust matplotlib rcParams for plotting procedures.

Parameters

style [str, optional] Your plotting style from (“custom”, “eureka”, “preserve”, or “default”). Custom passes all kwargs to the ‘font’ rcParams group at the moment. Eureka sets some nicer rcParams settings recommended by the Eureka team. Preserve leaves all rcParams as-is and can be used to toggle the usetex parameter. By default uses ‘preserve’.

usetex [bool, optional] Do you want to use LaTeX fonts (which requires LaTeX to be installed), by default False

from_scratch [bool, optional] Should the rcParams first be set to rcdefaults? By default False

****kwargs** [dict, optional] Any additional parameters to passed to the ‘font’ rcParams group.

Raises

ValueError Ensures that usetex and from_scratch arguments are boolean

ValueError Ensures that input style is one of: “custom”, “eureka”, “preserve”, or “default”

6.1.11 lib.readECF

class eureka.lib.readECF.MetaClass(*folder='.', file=None, **kwargs*)

Bases: object

A class to hold Eureka! metadata.

Methods

<code>copy_ecf()</code>	Copy an ECF file to the output directory to ensure reproducibility.
<code>read(folder, file)</code>	A function to read ECF files
<code>write(folder)</code>	A function to write an ECF file based on the current MetaClass settings.

`copy_ecf()`

Copy an ECF file to the output directory to ensure reproducibility.

NOTE: This will update the inputdir of the ECF file to point to the exact inputdir used to avoid ambiguity later and ensure that the ECF could be used to make the same outputs.

Parameters

None

Returns

None

Notes

History: - Mar 2022 Taylor J Bell

Initial Version based on old readECF code.

read(*folder, file*)

A function to read ECF files

Parameters

folder: str The folder containing an ECF file to be read in.

file: str The ECF filename to be read in.

Returns

None

Notes

History: - Mar 2022 Taylor J Bell

Initial Version based on old readECF code.

write(folder)

A function to write an ECF file based on the current MetaClass settings.

NOTE: For now this only rewrites the input ECF file to a new ECF file in the requested folder. In the future this function should make a full ECF file based on any adjusted parameters.

Parameters

folder: str The folder where the ECF file should be written.

Returns

None

Notes

History: - Mar 2022 Taylor J Bell

Initial Version.

6.1.12 lib.readEPF

```
class eureka.lib.readEPF.Parameter(name, value, ptype, priorpar1=None, priorpar2=None, prior=None)
```

Bases: object

A generic parameter class

Attributes

ptype Getter for the ptype

values Return all values for this parameter

property ptype

Getter for the ptype

property values

Return all values for this parameter

```
class eureka.lib.readEPF.Parameters(param_path='.', param_file=None, **kwargs)
```

Bases: object

A class to hold the Parameter instances

Methods

<code>read(folder, file)</code>	A function to read EPF files
<code>write(folder)</code>	A function to write an EPF file based on the current Parameters settings.

`read(folder, file)`

A function to read EPF files

Parameters

folder: str The folder containing an EPF file to be read in.

file: str The EPF filename to be read in.

Returns

None

Notes

History: - Mar 2022 Taylor J Bell

Initial Version based on old readECF code.

`write(folder)`

A function to write an EPF file based on the current Parameters settings.

NOTE: For now this only rewrites the input EPF file to a new EPF file in the requested folder. In the future this function should make a full EPF file based on any adjusted parameters.

Parameters

folder: str The folder where the EPF file should be written.

Returns

None

Notes

History: - Mar 2022 Taylor J Bell

Initial Version.

6.1.13 lib.smooth

`eureka.lib.smooth.medfilt(x, window_len)`

Apply a length-k median filter to a 1D array x. Boundaries are extended by repeating endpoints.

`eureka.lib.smooth.smooth(x, window_len=10, window='hanning')`

smooth the data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the beginning and end part of the output signal.

input: x: the input signal window_len: the dimension of the smoothing window window: str

The type of window from ‘flat’, ‘hanning’, ‘hamming’, ‘bartlett’, or ‘blackman’. flat window will produce a moving average smoothing.

output: the smoothed signal

example:

```
t=linspace(-2,2,0.1) x=sin(t)+randn(len(t))*0.1 y=smooth(x)
```

see also:

numpy.hanning, numpy.hamming, numpy.bartlett, numpy.blackman, numpy.convolve, scipy.signal.lfilter

TODO: the window parameter could be the window itself if an array instead of a string

Source: <http://www.scipy.org/Cookbook/SignalSmooth> 2009-03-13

6.1.14 lib.smoothing

eureka.lib.smoothing.**gauss_kernel_mask2**(ny_nx, sy_sx, j_i, mask)

summary

Parameters

ny_nx [list/ndarray] The ny and nx values.

sy_sx [list/ndarray] The sy and sx values.

j_i [list/ndarray] The j and i values

mask [ndarray] The current mask

Returns

ndarray The Gaussian kernel.

6.1.15 lib.sort_nicely

eureka.lib.sort_nicely.**alphanum_key**(s)

Turn a string into a list of string and number chunks. “z23a” -> [“z”, 23, “a”]

eureka.lib.sort_nicely.**sort_nicely**(listl)

Sort the given list in the way that humans expect.

eureka.lib.sort_nicely.**tryint**(s)

6.1.16 lib.splinterp

eureka.lib.splinterp.**splinterp**(x2, x, y)

This function implements the methods splrep and splev of the module scipy.interpolate

Parameters

X2: 1D array_like array of points at which to return the value of the smoothed spline or its derivatives

X, Y: array_like The data points defining a curve $y = f(x)$.

Returns

an array of values representing the spline function or curve.

If `tck` was returned from `splrep`, then this is a list of arrays

representing the curve in N-dimensional space.

Examples

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

```
>>> x = np.arange(21)/20.0 * 2.0 * np.pi
>>> y = np.sin(x)
>>> x2 = np.arange(41)/40.0 * 2.0 * np.pi
```

```
>>> y2 = splinterp(x2, x, y)
>>> plt.plot(x2,y2)
```

6.1.17 lib.suntimecorr

`eureka.lib.suntimecorr.getcoords(file)`

Use regular expressions to extract X,Y,Z, and time values from the horizons file.

`eureka.lib.suntimecorr.suntimecorr(ra, dec, obst, coordtable, verbose=False)`

This function calculates the light-travel time correction from observer to a standard location. It uses the 2D coordinates (RA and DEC) of the object being observed and the 3D position of the observer relative to the standard location. The latter (and the former, for solar-system objects) may be gotten from JPL's Horizons system.

6.1.18 lib.utc_tt

`eureka.lib.utc_tt.leapdates(rundir)`

Generates an array of leap second dates which are automatically updated every six months. Uses local leap second file, but retrieves a leap second file from NIST if the current file is out of date. Last update: 2011-03-17

`eureka.lib.utc_tt.leapseconds(jd_utc, dates)`

Computes the difference between UTC and TT for a given date. `jd_utc = (float)` UTC Julian date `dates = (array-like)` an array of Julian dates on which leap seconds occur

`eureka.lib.utc_tt.utc_tdb(jd_utc, leapdir)`

Converts UTC Julian dates to Barycentric Dynamical Time (TDB). Formula taken from USNO Circular 179, based on that found in Fairhead and Bretagnon (1990). Accurate to 10 microseconds. `jd_utc = (array-like)` UTC Julian date

`eureka.lib.utc_tt.utc_tt(jd_utc, leapdir)`

Converts UTC Julian dates to Terrestrial Time (TT). `jd_utc = (array-like)` UTC Julian date

6.1.19 lib.util

`eureka.lib.util.check_nans(data, mask, log, name="")`

Checks where a data array has NaNs

Parameters

data: ndarray a data array (e.g. data, err, dq, ...)

mask: ndarray input mask

log: logedit.Logedit The open log in which NaNs will be mentioned if existent.

name: str, optional The name of the data array passed in (e.g. SUBDATA, SUBERR, SUBV0)

Returns

mask: ndarray output mask where 0 will be written where the input data array has NaNs

`eureka.lib.util.get_mad(meta, wave_1d, optspec, wave_min=None, wave_max=None)`

Computes variation on median absolute deviation (MAD) using ediff1d.

Parameters

meta: MetaClass The metadata object.

wave_1d: ndarray Wavelength array (nx) with trimmed edges depending on xwindow and ywindow which have been set in the S3 ecf

optspec: ndarray Optimally extracted spectra, 2D array (time, nx)

wave_min: float Minimum wavelength for binned lightcurves, as given in the S4 .ecf file

wave_max: float Maximum wavelength for binned lightcurves, as given in the S4 .ecf file

Returns: Single MAD value in ppm

`eureka.lib.util.makedirectory(meta, stage, **kwargs)`

Creates a directory for the current stage

Parameters

meta: MetaClass The metadata object.

stage: str 'S#' string denoting stage number (i.e. 'S3', 'S4')

****kwargs**

Returns

run: int The run number

`eureka.lib.util.pathdirectory(meta, stage, run, old_datetime=None, **kwargs)`

Finds the directory for the requested stage, run, and datetime (or old_datetime)

Parameters

meta: MetaClass The metadata object.

stage: str 'S#' string denoting stage number (i.e. 'S3', 'S4')

run: int run #, output from makedirectory function

old_datetime: str The date that a previous run was made (for looking up old data)

****kwargs**

Returns

path: str Directory path for given parameters

`eureka.lib.util.readfiles(meta)`

Reads in the files saved in todir + inputdir and saves them into a list

Parameters

meta: MetaClass The metadata object.

Returns

meta: MetaClass The metadata object with added segment_list containing the sorted data fits files.

`eureka.lib.util.trim(data, meta)`

Removes the edges of the data arrays

Parameters

data: DataClass The data object.

meta: MetaClass The metadata object.

Returns

data: DataClass The data object with added subdata arrays with trimmed edges depending on xwindow and ywindow which have been set in the S3 ecf.

meta: MetaClass The metadata object.

6.2 S1_calibrations

6.2.1 S1_calibrations.s1_process.rampfitJWST

6.2.2 S1_calibrations.s1_process.EurekaS1Pipeline

6.2.3 S1_calibrations.ramp_fitting

6.3 S2_calibrations

6.3.1 S2_calibrations.s2_calibrate.calibrateJWST

6.3.2 S2_calibrations.s2_calibrate.EurekaSpec2Pipeline

6.3.3 S2_calibrations.s2_calibrate.Eurekalmage2Pipeline

6.4 S3_data_reduction

6.4.1 S3_data_reduction.background

`eureka.S3_data_reduction.background.BGsubtraction(data, meta, log, isplots)`

Does background subtraction using inst.fit_bg & background.fitbg

Parameters

data: DataClass Data object containing data, uncertainty, and variance arrays in units of MJy/sr or DN/s.

meta: MetaClass The metadata object.

log: logedit.Logedit The open log in which notes from this step can be added.

isplots: int The amount of plots saved; set in ecf.

Returns

data: DataClass Data object containing background subtracted data.

Notes

History:

- **Dec 10, 2021 Taylor Bell** Edited to pass the full DataClass object into `inst.fit_bg`
`eureka.S3_data_reduction.background.fitbg(dataim, meta, mask, x1, x2, deg=1, threshold=5, isrotate=False, isplots=0)`

Fit sky background with out-of-spectra data.

Parameters

dataim: ndarray The data array

meta: MetaClass The metadata object.

mask: ndarray A mask array

x1: ndarray

x2: ndarray

deg: int, optional Polynomial order for column-by-column background subtraction Default is 1.

threshold: int, optional Sigma threshold for outlier rejection during background subtraction. Default is 5.

isrotate: bool, optional Default is False.

isplots: int, optional The amount of plots saved; set in ecf. Default is 0.

Notes

History:

- **May 2013** Removed `[:-1]` for LDSS3
 - **Feb 2014** Modified `x1` and `x2` to allow for arrays
- `eureka.S3_data_reduction.background.fitbg2(dataim, meta, mask, bgmask, deg=1, threshold=5, isrotate=False, isplots=0)`

Fit sky background with out-of-spectra data.

`fitbg2` uses `bgmask`, a mask for the background region which enables fitting more complex background regions than simply above or below a given distance from the trace. This will help mask the 2nd and 3rd orders of NIRISS.

Parameters

dataim: `ndarray` The data array

meta: `MetaClass` The metadata object.

mask: `ndarray` A mask array

bgmask: `ndarray` A background mask array.

deg: `int, optional` Polynomial order for column-by-column background subtraction. Default is 1.

threshold: `int, optional` Sigma threshold for outlier rejection during background subtraction. Default is 5.

isrotate: `bool, optional` Default is False.

isplots: `int, optional` The amount of plots saved; set in `ecf`. Default is 0.

Notes

History:

- **September 2016 Kevin Stevenson** Initial version

```
eureka.S3_data_reduction.background.fitbg3(data, order_mask, readnoise=11, sigclip=[4, 2, 3],
                                           isplots=0)
```

Fit sky background with out-of-spectra data. Optimized to remove the 1/f noise in the NIRISS spectra (works in the y-direction).

Parameters

isplots [`bool`, `optional`] Plots intermediate steps for the background fitting routine. Default is False.

Returns

data [`object`] data object now contains new attribute `bkg_removed`.

6.4.2 S3_data_reduction.bright2flux

```
eureka.S3_data_reduction.bright2flux.bright2dn(data, meta, mjd=False)
```

This function converts the data, uncertainty, and variance arrays from brightness units (MJy/sr) or (MJy) to raw units (DN).

Parameters

data: `DataClass` Data object containing data, uncertainty, and variance arrays in units of MJy/sr.

meta: `MetaClass` The metadata object.

Returns

data: `DataClass` Data object containing data, uncertainty, and variance arrays in units of DN.

Notes

The photometry files can be downloaded from CRDS (https://jwst-crds.stsci.edu/browse_db/)

History:

- **2021-05-28 kbs** Initial version
- **2021-07-21 sz** Added functionality for MIRI

`eureka.S3_data_reduction.bright2flux.bright2flux(data, pixel_area)`

This function converts the data and uncertainty arrays from brightness units (MJy/sr) to flux units (Jy/pix).

Parameters

data: `DataClass` Data object containing data, uncertainty, and variance arrays in units of MJy/sr.

pixel_area: `ndarray` Pixel area (arcsec/pix)

Returns

data: `DataClass` Data object containing data, uncertainty, and variance arrays in units of Jy/pix.

Notes

The input arrays Data and Uncd are changed in place.

History:

- 2005-06-20 Statia Luszcz, Cornell (shl35@cornell.edu).
- **2005-10-13 jh** Renamed, modified doc, removed posmed, fixed nimpos default bug (was float rather than int).
- **2005-10-28 jh** Updated header to give units being converted from/to, made srperas value a calculation rather than a constant, added Allen reference.
- **2005-11-24 jh** Eliminated NIMPOS.
- **2008-06-28 jh** Allow npos=1 case.
- **2010-01-29 patricio** (pcubillos@fulbrightmail.org) Converted to python.
- **2010-11-01 patricio** Documented, and incorporated scipy.constants.
- **2021-05-28 kbs** Updated for JWST
- **2021-12-09 TJB** Updated to account for the new DataClass object

`eureka.S3_data_reduction.bright2flux.convert_to_e(data, meta, log)`

This function converts the data object to electrons from MJy/sr or DN/s.

Parameters

data: `DataClass` Data object containing data, uncertainty, and variance arrays in units of MJy/sr or DN/s.

meta: `MetaClass` The metadata object.

log: `logedit.Logedit` The open log in which notes from this step can be added.

Returns

data: `DataClass` Data object containing data, uncertainty, and variance arrays in units of electrons.

meta: MetaClass The metadata object.

`eureka.S3_data_reduction.bright2flux.dn2electrons(data, meta)`

This function converts the data, uncertainty, and variance arrays from raw units (DN) to electrons.

Parameters

data: DataClass Data object containing data, uncertainty, and variance arrays in units of DN.

meta: MetaClass The metadata object.

Returns

data: DataClass Data object containing data, uncertainty, and variance arrays in units of electrons.

Notes

The gain files can be downloaded from CRDS (https://jwst-crds.stsci.edu/browse_db/)

History:

- **Jun 2021 Kevin Stevenson** Initial version
- **Jul 2021** Added gainfile rotation

`eureka.S3_data_reduction.bright2flux.rate2count(data)`

This function converts the data, uncertainty, and variance arrays from rate units (#/s) to counts (#).

Parameters

data: DataClass Data object containing data, uncertainty, and variance arrays in rate units (#/s).

Returns

data: DataClass Data object containing data, uncertainty, and variance arrays in count units (#).

Notes

History:

- **Mar 7, 2022 Taylor J Bell** Initial version

`eureka.S3_data_reduction.bright2flux.retrieve_ancil(fitsname)`

Use crds package to find/download the needed ancilliary files.

This code requires that the CRDS_PATH and CRDS_SERVER_URL environment variables be set in your .bashrc file (or equivalent, e.g. .bash_profile or .zshrc)

Parameters

fitsname: The filename of the file currently being analyzed.

Returns

phot_filename: str The full path to the photom calibration file.

gain_filename: str The full path to the gain calibration file.

Notes

History:

- **2022-03-04 Taylor J Bell** Initial code version.
- **2022-03-28 Taylor J Bell** Removed jwst dependency, using crds package now instead.

6.4.3 S3_data_reduction.hst_scan

`eureka.S3_data_reduction.hst_scan.calcDrift2D(im1, im2, m, n, n_files)`

`eureka.S3_data_reduction.hst_scan.calcTrace(x, centroid, grism)`

Calculates the WFC3 trace given the position of the direct image in physical pixels.

Parameters

- x** [physical pixel values along dispersion direction over which the trace is calculated]
- centroid** [[y,x] pair describing the centroid of the direct image]

Returns

- y** [computed trace]

`eureka.S3_data_reduction.hst_scan.calc_slitshift(wavegrid, xrng, refwave=None, width=3, deg=2)`

Calculates horizontal shift to correct tilt in data using wavelength.

Parameters

Returns

`eureka.S3_data_reduction.hst_scan.calc_slitshift2(spectrum, xrng, ywindow, xwindow, width=5, deg=1)`

Calculate horizontal shift to correct tilt in data using spectrum.

`eureka.S3_data_reduction.hst_scan.calibrateLambda(x, centroid, grism)`

Calculates coefficients for the dispersion solution

Parameters

- x** [physical pixel values along dispersion direction over which the wavelength is calculated]
- centroid** [[y,x] pair describing the centroid of the direct image]

Returns

- y** [computed wavelength values]

`eureka.S3_data_reduction.hst_scan.correct_slitshift2(data, slitshift, mask=None, isreverse=False)`

Applies horizontal shift to correct tilt in data.

Parameters

Returns

`eureka.S3_data_reduction.hst_scan.drift_fit2D(ev, data, validRange=9)`

Measures the spectrum drift over all frames and all non-destructive reads.

Parameters

- ev** [Event object]
- data** [4D data frames]
- preclip** [Ignore first preclip values of spectrum]
- postclip** [Ignore last postclip values of spectrum]
- width** [Half-width in pixels used when fitting Gaussian]
- deg** [Degree of polynomial fit]
- validRange** [Trim spectra by +/- pixels to compute valid region of cross correlation]

Returns

- drift** [Array of measured drift values]
- model** [Array of model drift values]

`eureka.S3_data_reduction.hst_scan.groupFrames(dates)`

Group frames by orbit and batch number

Parameters

- dates** [Time in days]
- exptime** [exposure time in seconds]

`eureka.S3_data_reduction.hst_scan.imageCentroid(filenamees, guess, trim, ny, CRPIX1, CRPIX2, POSTARG1, POSTARG2)`

Calculate centroid for a list of direct images.

Parameters

- filenames** [List of direct image filenames]
- guess** [Paired list, centroid guess]
- trim** [Trim image when calculating centroid]
- ny** [The value of NAXIS2]
- CRPIX1:** The value of CRPIX1 in the main FITS header
- CRPIX2:** The value of CRPIX2 in the main FITS header
- POSTARG1:** The value of POSTARG1 in the science FITS header
- POSTARG2:** The value of POSTARG2 in the science FITS header

Returns

- center** [Centroids]

`eureka.S3_data_reduction.hst_scan.makeBasicFlats(flatfile, xwindow, ywindow, flatoffset, ny, nx, sigma=5, isplots=0)`

Makes master flatfield image (with no wavelength correction) and new mask for WFC3 data.

Parameters

- flatfile** [List of files containing flatfiles images]

xwindow [Array containing image limits in wavelength direction]

ywindow [Array containing image limits in spatial direction]

n_spec [Number of spectra]

sigma [Sigma rejection level]

Returns

flat_master [Single master flatfield image]

mask_master [Single bad-pixel mask image]

`eureka.S3_data_reduction.hst_scan.makeflats(flatfile, wave, xwindow, ywindow, flatoffset, n_spec, ny, nx, sigma=5, isplots=0)`

Makes master flatfield image and new mask for WFC3 data.

Parameters

flatfile [List of files containing flatfiles images]

wave [wavelengths]

xwindow [Array containing image limits in wavelength direction]

ywindow [Array containing image limits in spatial direction]

n_spec [Number of spectra]

sigma [Sigma rejection level]

Returns

flat_master [Single master flatfield image]

mask_master [Single bad-pixel mask image]

`eureka.S3_data_reduction.hst_scan.replacePixels(shiftdata, shiftmask, m, n, i, j, k, ktot, ny, nx, sy, sx)`

6.4.4 S3_data_reduction.miri

`eureka.S3_data_reduction.miri.fit_bg(data, meta, n, isplots=False)`

Fit for a non-uniform background.

Uses the code written for NIRCcam and untested for MIRI, but likely to still work (as long as MIRI data gets rotated)

`eureka.S3_data_reduction.miri.flag_bg(data, meta)`

Outlier rejection of sky background along time axis.

Uses the code written for NIRCcam and untested for MIRI, but likely to still work (as long as MIRI data gets rotated)

Parameters

data: DataClass The data object in which the fits data will stored

meta: MetaData The metadata object

Returns

data: DataClass The updated data object with outlier background pixels flagged.

`eureka.S3_data_reduction.miri.read(filename, data, meta)`

Reads single FITS file from JWST's MIRI instrument.

Parameters

filename: `str` Single filename to read

data: `DataClass` The data object in which the fits data will stored

meta: `MetaData` The metadata object

Returns

data: `DataClass` The updated data object with the fits data stored inside

Notes

History:

- **Nov 2012 Kevin Stevenson** Initial Version
- **May 2021 Kevin Stevenson** Updated for NIRCam
- **Jun 2021 Taylor Bell** Updated docs for MIRI
- **Jun 2021 Sebastian Zieba** Updated for MIRI
- **Apr 2022 Sebastian Zieba** Updated wavelength array

`eureka.S3_data_reduction.miri.wave_MIRI_hardcoded()`

This code contains the wavelength array for MIRI data. It was generated by using the `jwst` and `gwcs` packages to get the wavelength information out of the WCS.

Returns

lam_x_full: `list` A list of the wavelengths

Notes

History:

- **Apr 2022 Sebastian Zieba** Initial Version

6.4.5 S3_data_reduction.nircam

`eureka.S3_data_reduction.nircam.fit_bg(data, meta, n, isplots=False)`

Fit for a non-uniform background.

`eureka.S3_data_reduction.nircam.flag_bg(data, meta)`

Outlier rejection of sky background along time axis.

Parameters

data: `DataClass` The data object in which the fits data will stored

meta: `MetaClass` The metadata object

Returns

data: `DataClass` The updated data object with outlier background pixels flagged.

`eureka.S3_data_reduction.nircam.read(filename, data, meta)`

Reads single FITS file from JWST's NIRC*am* instrument.

Parameters

filename: `str` Single filename to read

data: `DataClass` The data object in which the fits data will stored

meta: `MetaClass` The metadata object

Returns

data: `DataClass` The updated data object with the fits data stored inside

Notes

History:

- **November 2012 Kevin Stevenson** Initial version
- **May 2021 KBS** Updated for NIRC*am*
- **July 2021** Moved `bjtdb` into here

6.4.6 `S3_data_reduction.niriss_profiles`

A library of custom weighted profiles to fit to the NIRISS orders to complete the optimal extraction of the data.

`eureka.S3_data_reduction.niriss_profiles.gaussian_1poly_piecewise(args, x)`

A piece-wise function consisting of 2 generalized normal distribution profiles connected with a 1D polynomial to mimic the bat-shaped profile of NIRISS.

Parameters

args [`np.ndarray`] A list or array of parameters for the fits.

x [`np.ndarray`] X values to evaluate the shape over.

`eureka.S3_data_reduction.niriss_profiles.gaussian_2poly_piecewise(args, x)`

A piece-wise function consisting of 2 generalized normal distribution profiles connected with a 2D polynomial to mimic the bat-shaped profile of NIRISS.

Parameters

args [`np.ndarray`] A list or array of parameters for the fits.

x [`np.ndarray`] X values to evaluate the shape over.

`eureka.S3_data_reduction.niriss_profiles.generalized_normal(x, mu, alpha, beta, scale)`

Generalized normal distribution.

Parameters

x [`np.ndarray`] X values to evaluate the distribution. over.

mu [`float`] Mean/center value of the distribution.

alpha [`float`] Sets the scale/standard deviation of the distribution.

beta [`float`] Sets the shape of the distribution. Beta > 2 becomes boxy. Beta < 2 becomes peaky. Beta = 2 is a normal Gaussian.

scale [float] A value to scale the distribution by.

`eureka.S3_data_reduction.niriss_profiles.moffat_1poly_pieewise(args, x)`

A piece-wise function consisting of 2 Moffat profiles connected with a 1D polynomial to mimic the bat-shaped profile of NIRISS.

Parameters

args [np.ndarray] A list or array of parameters for the fits.

x [np.ndarray] X values to evaluate the shape over.

`eureka.S3_data_reduction.niriss_profiles.moffat_2poly_pieewise(args, x)`

A piece-wise function consisting of 2 Moffat profiles connected with a 2D polynomial to mimic the bat-shaped profile of NIRISS.

Parameters

args [np.ndarray] A list or array of parameters for the fits.

x [np.ndarray] X values to evaluate the shape over.

6.4.7 S3_data_reduction.niriss

`eureka.S3_data_reduction.niriss.f277_mask(data, isplots=0)`

Marks the overlap region in the f277w filter image.

Parameters

data [object]

isplots [int, optional] Level of plots that should be created in the S3 stage. This is set in the .ecf control files. Default is 0. This stage will plot if isplots >= 5.

Returns

mask [np.ndarray] 2D mask for the f277w filter.

mid [np.ndarray] (x,y) anchors for where the overlap region is located.

`eureka.S3_data_reduction.niriss.fit_bg(data, meta, n_iters=3, readnoise=11, sigclip=[4, 4, 4], isplots=0)`

Subtracts background from non-spectral regions.

Parameters

data [object]

meta [object]

n_iters [int, optional] The number of iterations to go over and remove cosmic rays. Default is 3.

readnoise [float, optional] An estimation of the readnoise of the detector. Default is 5.

sigclip [list, array, optional] A list or array of len(n_iters) corresponding to the sigma-level which should be clipped in the cosmic ray removal routine. Default is [4,2,3].

isplots [int, optional] The level of output plots to display. Default is 0 (no plots).

Returns

data [object]

`eureka.S3_data_reduction.niriss.fit_orders(data, meta, which_table=2)`

Creates a 2D image optimized to fit the data. Currently runs with a Gaussian profile, but will look into other more realistic profiles at some point. This routine is a bit slow, but fortunately, you only need to run it once per observations.

Parameters

data [object]

meta [object]

which_table [int, optional] Sets with table of initial y-positions for the orders to use. Default is 2.

Returns

meta [object] Adds two new attributes: *order1_mask* and *order2_mask*.

`eureka.S3_data_reduction.niriss.fit_orders_fast(data, meta, which_table=2)`

A faster method to fit a 2D mask to the NIRISS data. Very similar to *fit_orders*, but works with *scipy.optimize.leastsq*.

Parameters

data [object]

meta [object]

which_table [int, optional] Sets with table of initial y-positions for the orders to use. Default is 2.

Returns

meta [object]

`eureka.S3_data_reduction.niriss.image_filtering(img, radius=1, gf=4)`

Does some simple image processing to isolate where the spectra are located on the detector. This routine is optimized for NIRISS S2 processed data and the F277W filter.

Parameters

img [np.ndarray] 2D image array.

radius [np.float, optional] Default is 1.

gf [np.float, optional] The standard deviation by which to Gaussian smooth the image. Default is 4.

Returns

img_mask [np.ndarray] A mask for the image that isolates where the spectral orders are.

`eureka.S3_data_reduction.niriss.mask_method_one(data, meta, isplots=0, save=True)`

There are some hard-coded numbers in here right now. The idea is that once we know what the real data looks like, nobody will have to actually call this function and we'll provide a CSV of a good initial guess for each order. This method uses some fun image processing to identify the boundaries of the orders and fits the edges of the first and second orders with a 4th degree polynomial.

Parameters

data [object]

meta [object]

isplots [int, optional] Level of plots that should be created in the S3 stage. This is set in the .ecf control files. Default is 0. This stage will plot if *isplots* \geq 5.

save [bool, optional] An option to save the polynomial fits to a CSV. Default is True. Output table is saved under *niriss_order_guesses.csv*.

Returns

meta [object]

`eureka.S3_data_reduction.niriss.mask_method_two(data, meta, isplots=0, save=False)`

A second method to extract the masks for the first and second orders in NIRISS data. This method uses the vertical profile of a summed image to identify the borders of each order.

“” :param data: :type data: object :param meta: :type meta: object :param isplots: Level of plots that should be created in the S3 stage.

This is set in the .ecf control files. Default is 0. This stage will plot if isplots >= 5.

Parameters **save** (*bool*, *optional*) – Has the option to save the initial guesses for the location of the NIRISS orders. This is set in the .ecf control files. Default is False.

Returns

meta [object]

`eureka.S3_data_reduction.niriss.read(filename, f277_filename, data, meta)`

Reads a single FITS file from JWST’s NIRISS instrument. This takes in the Stage 2 processed files.

Parameters

filename [str] Single filename to read. Should be a *.fits* file.

data [object] Data object in which the fits data will be stored.

Returns

data [object] Data object now populated with all of the FITS file information.

meta [astropy.table.Table] Metadata stored in the FITS file.

`eureka.S3_data_reduction.niriss.simplify_niriss_img(data, meta, isplots=False)`

Creates an image to map out where the orders are in the NIRISS data.

Parameters

data [object]

meta [object]

isplots [int, optional] Level of plots that should be created in the S3 stage. This is set in the .ecf control files. Default is 0.

Returns

g [np.ndarray] A 2D array that marks where the NIRISS first and second orders are.

`eureka.S3_data_reduction.niriss.wave_NIRISS(wavefile, meta)`

Adds the 2D wavelength solutions to the meta object.

Parameters

wavefile [str] The name of the .FITS file with the wavelength solution.

meta [object]

Returns

meta [object]

6.4.8 S3_data_reduction.nirspec

`eureka.S3_data_reduction.nirspec.fit_bg(data, meta, n, isplots=False)`

Fit for a non-uniform background.

Uses the code written for NIRCcam and untested for NIRSpec, but likely to still work

`eureka.S3_data_reduction.nirspec.flag_bg(data, meta)`

Outlier rejection of sky background along time axis.

Parameters

data: DataClass The data object in which the fits data will stored

meta: MetaClass The metadata object

Returns

data: DataClass The updated data object with outlier background pixels flagged.

`eureka.S3_data_reduction.nirspec.read(filename, data, meta)`

Reads single FITS file from JWST's NIRCcam instrument.

Parameters

filename: str Single filename to read

data: DataClass The data object in which the fits data will stored

meta: MetaClass The metadata object

Returns

data: DataClass The updated data object with the fits data stored inside

Notes

History:

- **November 2012 Kevin Stevenson** Initial version
- **June 2021 Aarynn Carter/Eva-Maria Ahrer** Updated for NIRSpec

6.4.9 S3_data_reduction.optspex

`eureka.S3_data_reduction.optspex.optimize(subdata, mask, bg, spectrum, Q, v0, p5thresh=10, p7thresh=10, fitype='smooth', window_len=21, deg=3, windowtype='hanning', n=0, isplots=0, eventdir='.', meddata=None, hide_plots=False)`

Extract optimal spectrum with uncertainties.

Parameters

subdata: ndarray Background subtracted data.

mask: ndarray Outlier mask.

bg: ndarray Background array.

spectrum: ndarray Standard spectrum.

Q: float The gain factor.

v0: ndarray Variance array for data.

p5thresh: float Sigma threshold for outlier rejection while constructing spatial profile.

p7thresh: float Sigma threshold for outlier rejection during optimal spectral extraction.

fitttype: {'smooth', 'meddata', 'wavelet2D', 'wavelet', 'gauss', 'poly'} The type of profile fitting you want to do.

window_len: int The dimension of the smoothing window.

deg: int Polynomial degree.

windowtype: {'flat', 'hanning', 'hamming', 'bartlett', 'blackman'} UNUSED. The type of window. A flat window will produce a moving average smoothing.

n: int Integration number.

isplots: int The amount of plots saved; set in ecf.

eventdir: str Directory in which to save outputs.

meddata: ndarray The median of all data frames.

hide_plots: If True, plots will automatically be closed rather than popping up.

Returns

spectrum: ndarray The optimally extracted spectrum.

specunc: ndarray The standard deviation on the spectrum.

submask: ndarray The mask array.

`eureka.S3_data_reduction.optspex.profile_gauss(subdata, mask, threshold=10, guess=None, isplots=0)`
Construct normalized spatial profile using a Gaussian smoothing function.

Parameters

subdata: ndarray Background subtracted data.

mask: ndarray Outlier mask.

threshold: float Sigma threshold for outlier rejection while constructing spatial profile.

guess: list UNUSED. The initial guess for the Gaussian parameters.

isplots: int The amount of plots saved; set in ecf.

Returns

profile: ndarray Fitted profile in the same shape as the input data array.

`eureka.S3_data_reduction.optspex.profile_meddata(data, mask, meddata, threshold=10, isplots=0)`
Construct normalized spatial profile using median of all data frames.

Parameters

data: ndarray UNUSED. Image data.

mask: ndarray UNUSED. Outlier mask.

meddata: ndarray The median of all data frames.

threshold: float UNUSED. Sigma threshold for outlier rejection while constructing spatial profile.

isplots: int UNUSED. The amount of plots saved; set in ecf.

Returns

profile: ndarray Fitted profile in the same shape as the input data array.

`eureka.S3_data_reduction.optspex.profile_poly(subdata, mask, deg=3, threshold=10, isplots=0)`

Construct normalized spatial profile using polynomial fits along the wavelength direction.

Parameters

subdata: ndarray Background subtracted data.

mask: ndarray Outlier mask.

deg: int Polynomial degree.

threshold: float Sigma threshold for outlier rejection while constructing spatial profile.

isplots: int The amount of plots saved; set in `ecf`.

Returns

profile: ndarray Fitted profile in the same shape as the input data array.

`eureka.S3_data_reduction.optspex.profile_smooth(subdata, mask, threshold=10, window_len=21, windowtype='hanning', isplots=False)`

Construct normalized spatial profile using a smoothing function.

Parameters

subdata: ndarray Background subtracted data.

mask: ndarray Outlier mask.

threshold: float Sigma threshold for outlier rejection while constructing spatial profile.

window_len: int The dimension of the smoothing window.

windowtype: {'flat', 'hanning', 'hamming', 'bartlett', 'blackman'} UNUSED. The type of window. A flat window will produce a moving average smoothing.

isplots: int The amount of plots saved; set in `ecf`.

Returns

profile: ndarray Fitted profile in the same shape as the input data array.

`eureka.S3_data_reduction.optspex.profile_wavelet(subdata, mask, wavelet, numlvl, isplots=0)`

This function performs 1D image denoising using BayesShrink soft thresholding.

Parameters

subdata: ndarray Background subtracted data.

mask: ndarray Outlier mask.

wavelet: Wavelet object or name string qWavelet to use

numlvl: int Decomposition levels to consider (must be ≥ 0).

isplots: int The amount of plots saved; set in `ecf`.

Returns

profile: ndarray Fitted profile in the same shape as the input data array.

References

Chang et al. “Adaptive Wavelet Thresholding for Image Denoising and Compression”, 2000

`eureka.S3_data_reduction.optspex.profile_wavelet2D(subdata, mask, wavelet, numlvls, isplots=0)`

This function performs 2D image denoising using BayesShrink soft thresholding.

Parameters

subdata: `ndarray` Background subtracted data.

mask: `ndarray` Outlier mask.

wavelet: `Wavelet object or name string` qWavelet to use

numlvls: `int` Decomposition levels to consider (must be ≥ 0).

isplots: `int` The amount of plots saved; set in `ecf`.

Returns

profile: `ndarray` Fitted profile in the same shape as the input data array.

References

Chang et al. “Adaptive Wavelet Thresholding for Image Denoising and Compression”, 2000

6.4.10 S3_data_reduction.plots_s3

`eureka.S3_data_reduction.plots_s3.image_and_background(data, meta, n)`

Make image+background plot.

Parameters

data: `DataClass` The data object.

meta: `MetaClass` The metadata object.

n: `int` The integration number.

Returns

`None`

`eureka.S3_data_reduction.plots_s3.lc_nodriftcorr(meta, wave_1d, optspec)`

Plot a 2D light curve without drift correction.

Parameters

meta: `MetaClass` The metadata object.

wave_1d: Wavelength array with trimmed edges depending on `xwindow` and `ywindow` which have been set in the S3 `ecf`

optspec: The optimally extracted spectrum.

Returns

`None`

`eureka.S3_data_reduction.plots_s3.optimal_spectrum(data, meta, n)`

Make optimal spectrum plot.

Parameters

data: **DataClass** The data object.

meta: **MetaClass** The metadata object.

n: **int** The integration number.

Returns

None

`eureka.S3_data_reduction.plots_s3.profile(eventdir, profile, submask, n, hide_plots=False)`

Plot weighting profile from optimal spectral extraction routine

Parameters

eventdir: **str** Directory in which to save outputs.

profile: **ndarray** Fitted profile in the same shape as the data array.

submask: **ndarray** Outlier mask.

n: **int** The current integration number.

hide_plots: If True, plots will automatically be closed rather than popping up.

Returns

None

`eureka.S3_data_reduction.plots_s3.source_position(meta, x_dim, pos_max, m, isgauss=False, x=None, y=None, popt=None, isFWM=False, y_pixels=None, sum_row=None, y_pos=None)`

Plot source position for MIRI data.

Parameters

meta: **MetaClass** The metadata object.

x_dim: **int** The number of pixels in the y-direction in the image.

pos_max: **float** The brightest row.

m: **int** The file number.

y_pixels: **1darray** The indices of the y-pixels.

sum_row: **1darray** The sum over each row.

isgauss: **bool** Used a gaussian centring method.

popt: **list** The fitted Gaussian terms.

isFWM: **bool** Used a flux-weighted mean centring method.

y_pos: **float** The FWM central position of the star.

Returns

None

Notes

History:

- **2021-07-14: Sebastian Zieba** Initial version.
- **Oct 15, 2021: Taylor Bell** Tidied up the code a bit to reduce repeated code.

6.4.11 S3_data_reduction.s3_reduce

class eureka.S3_data_reduction.s3_reduce.DataClass

Bases: object

A class to hold Eureka! image data.

class eureka.S3_data_reduction.s3_reduce.MetaClass

Bases: object

A class to hold Eureka! metadata.

eureka.S3_data_reduction.s3_reduce.load_general_s2_meta_info(*meta*, *ecf_path*, *s2_meta*)

Loads in the S2 meta save file and adds in attributes from the S3 ECF.

Parameters

meta: MetaClass The new meta object for the current S3 processing.

ecf_path: The absolute path to where the S3 ECF is stored.

Returns

meta: MetaClass The S2 metadata object with attributes added by S3.

Notes

History:

- **March 2022 Taylor Bell** Initial version.

eureka.S3_data_reduction.s3_reduce.read_s2_meta(*meta*)

Loads in an S2 meta file.

Parameters

meta: MetaClass The new meta object for the current S3 processing.

Returns

s2_meta: MetaClass The S2 metadata object.

Notes

History:

- **March 2022 Taylor Bell** Initial version.

`eureka.S3_data_reduction.s3_reduce.reduceJWST(eventlabel, ecf_path='.', s2_meta=None)`

Reduces data images and calculates optimal spectra.

Parameters

eventlabel: str The unique identifier for these data.

ecf_path: str The absolute or relative path to where ecfs are stored

s2_meta: MetaClass The metadata object from Eureka!’s S2 step (if running S2 and S3 sequentially).

Returns

meta: MetaClass The metadata object with attributes added by S3.

Notes

History:

- **May 2021 Kevin Stevenson** Initial version
- **October 2021 Taylor Bell** Updated to allow for inputs from S2

6.4.12 S3_data_reduction.sigrej

`eureka.S3_data_reduction.sigrej.sigrej(data, sigma, mask=None, estsig=None, ival=False, axis=0, fmean=False, fstddev=False, fmedian=False, fmedstddev=False)`

This function flags outlying points in a data set using sigma rejection.

Parameters

data: ndarray Array of points to apply sigma rejection to.

sigma: ndarray (1D) 1D array of sigma values for each iteration of sigma rejection. Number of elements determines number of iterations.

mask: (optional) byte array Same shape as Data, where 1 indicates the corresponding element in Data is good and 0 indicates it is bad. Only rejection of good-flagged data will be further considered. This input mask is NOT modified in the caller.

estsig: ndarray [nsig] array of estimated standard deviations to use instead of calculated ones in each iteration. This is useful in the case of small datasets with outliers, in which case the calculated standard deviation can be large if there is an outlier and small if there is not, leading to rejection of good elements in a clean dataset and acceptance of all elements in a dataset with one bad element. Set any element of estsig to a negative value to use the calculated standard deviation for that iteration.

ival: ndarray (2D) (returned) 2D array giving the median and standard deviation (with respect to the median) at each iteration.

axis: int The axis along which to compute the mean/median.

fmean: ndarray (returned) the mean of the accepted data.

fstdddev: ndarray (returned) the standard deviation of the accepted data with respect to the mean.

fmedian: ndarray (returned) the median of the accepted data.

fmedstddev: ndarray (returned) the standard deviation of the accepted data with respect to the median.

Returns

ret: tuple This function returns a mask of accepted values in the data. The mask is a byte array of the same shape as Data. In the mask, 1 indicates good data, 0 indicates an outlier in the corresponding location of Data. fmean, fstdddev, fmedian, and fmedstddev will also be updated and returned if they were passed in. All of these will be packaged together into a tuple.

Notes

SIGREJ flags as outliers points a distance of σ * the standard deviation from the median. Unless given as a positive value in ESTSIG, standard deviation is calculated with respect to the median, using MEDSTDDEV. For each successive iteration and value of sigma SIGREJ recalculates the median and standard deviation from the set of 'good' (not masked) points, and uses these new values in calculating further outliers. The final mask contains a value of 1 for every 'inlier' and 0 for every outlying data point.

History:

- **2005-01-18 statia Statia Luszcz, Cornell.** (shl35@cornell.edu) Initial version
- **2005-01-19 statia** Changed function to return mask, rather than a list of outlying and inlying points, added final statistics keywords
- **2005-01-20 jh Joe Harrington, Cornell,** (jh@oobleck.astro.cornell.edu) Header update. Added example.
- **2005-05-26 jh** Fixed header typo.
- **2006-01-10 jh** Moved definition, added test to see if all elements rejected before last iteration (e.g., dataset is all NaN). Added input mask, estsig.
- **2010-11-01 patricio** (pcubillos@fulbrightmail.org) Converted to python.

Examples

Define the N-element vector of sample data.

```
>>> print(mean(x), stddev(x), median(x), medstddev(x))
1438.47      5311.67      67.0000      5498.10
>>> sr.sigrej(x, [9,3]), ival=ival, fmean=fmean, fmedian=fmedian)
```

```
>>> x = np.array([65., 667, 84, 968, 62, 70, 66, 78, 47, 71, 56, 65, 60])
>>> q,w,e,r,t,y = sr.sigrej(x, [2,1], ival=True, fmean=True,
>>>                        fstdddev=True, fmedian=True, fmedstddev=True)
```

```
>>> print(q)
[ True False  True False  True  True  True  True  True  True  True  True
 True]
>>> print(w)
```

(continues on next page)

(continued from previous page)

```

[[ 66.          65.5          ]
 [ 313.02675604 181.61572819]]
>>> print(e)
65.8181818182
>>> print(r)
10.1174916043
>>> print(t)
65.0
>>> print(y)
10.1538170163
>>> print(fmean, fmedian)
67.00000      67.00000

```

6.4.13 S3_data_reduction.source_pos

`eureka.S3_data_reduction.source_pos.gauss(x, a, x0, sigma, off)`

A function to find the source location using a Gaussian fit.

Parameters

- x:** `ndarray` The positions at which to evaluate the Gaussian.
- a:** `float` The amplitude of the Gaussian.
- x0:** `float` The centre point of the Gaussian.
- sigma:** `float` The standard deviation of the Gaussian.
- off:** `float` A vertical offset in the Gaussian.

Returns

- gaussian:** `ndarray` The 1D Gaussian evaluated at the points *x*, in the same shape as *x*.

Notes

History:

- **2021-07-14 Sebastian Zieba** Initial version
- **2021-10-15 Taylor Bell** Separated this into its own function to allow it to be used elsewhere.

`eureka.S3_data_reduction.source_pos.source_pos(data, meta, m, header=False)`

Make image+background plot.

Parameters

- data:** `DataClass` The data object.
- meta:** `MetaClass` The metadata object.
- m:** `int` The file number.
- header:** `bool` If True, use the source position in the FITS header.

Returns

- src_ypos:** `int` The central position of the star.

`eureka.S3_data_reduction.source_pos.source_pos_FWM(data, meta, m)`

An alternative function to find the source location using a flux-weighted mean approach

Parameters

data: `DataClass` The data object.

meta: `MetaClass` The metadata object.

m: `int` The file number.

Returns

y_pos: `int` The central position of the star.

Notes

History:

- **2021-06-24 Taylor Bell** Initial version
- **2021-07-14 Sebastian Zieba** Modified

`eureka.S3_data_reduction.source_pos.source_pos_gauss(data, meta, m)`

A function to find the source location using a gaussian fit.

Parameters

data: `DataClass` The data object.

meta: `MetaClass` The metadata object.

m: `int` The file number.

Returns

y_pos: `int` The central position of the star.

Notes

History:

- **2021-07-14 Sebastian Zieba** Initial version
- **2021-10-15 Taylor Bell** Tweaked to allow for cleaner plots_s3.py

`eureka.S3_data_reduction.source_pos.source_pos_max(data, meta, m, plot=True)`

A simple function to find the brightest row for source location

Parameters

data: `DataClass` The data object.

meta: `MetaClass` The metadata object.

m: `int` The file number.

plot: `bool` If true, plot the source position determination.

Returns

y_pos: `int` The central position of the star.

Notes

History:

- **6/24/21 Megan Mansfield** Initial version
- **2021-07-14 Sebastian Zieba** Modified

6.4.14 S3_data_reduction.wfc3

`eureka.S3_data_reduction.wfc3.conclusion_step(meta, log)`

`eureka.S3_data_reduction.wfc3.correct_drift2D(data, meta, m)`

Parameters

data: DataClass The data object in which the fits data will stored

meta: MetaClass The metadata object

m: int The current file number

`eureka.S3_data_reduction.wfc3.difference_frames(data, meta)`

`eureka.S3_data_reduction.wfc3.fit_bg(data, meta, n, isplots=False)`

Fit for a non-uniform background.

Uses the code written for NIRCcam, but adds on some extra steps

`eureka.S3_data_reduction.wfc3.flag_bg(data, meta)`

Outlier rejection of sky background along time axis.

Uses the code written for NIRCcam and untested for MIRI, but likely to still work (as long as MIRI data gets rotated)

Parameters

data: DataClass The data object in which the fits data will stored

meta: MetaData The metadata object

Returns

data: DataClass The updated data object with outlier background pixels flagged.

`eureka.S3_data_reduction.wfc3.flatfield(data, meta)`

`eureka.S3_data_reduction.wfc3.preparation_step(meta, log)`

`eureka.S3_data_reduction.wfc3.read(filename, data, meta)`

Reads single FITS file from HST's WFC3 instrument.

Parameters

filename: str Single filename to read

data: DataClass The data object in which the fits data will stored

meta: MetaClass The metadata object

Returns

data: DataClass The updated data object with the fits data stored inside

Notes

History:

- **January 2017 Kevin Stevenson** Initial code as implemented in the WFC3 pipeline
- **18-19 Nov 2021 Taylor Bell** Edited and decomposed WFC3 code to integrate with Eureka!

`eureka.S3_data_reduction.wfc3.separate_direct(meta, log)`

`eureka.S3_data_reduction.wfc3.separate_scan_direction(obstimes, postarg2, meta, log)`

6.5 S4_generate_lightcurves

6.5.1 S4_generate_lightcurves.drift

`eureka.S4_generate_lightcurves.drift.highpassfilt(signal, highpassWidth)`

Run a signal through a highpass filter to remove high frequency signals.

This function can be used to compute the continuum of a signal to be subtracted.

Parameters

signal: `ndarray (1D)` 1D array of values

highpassWidth: `int` The width of the boxcar filter to use.

Returns

smoothed_signal: `ndarray (1D)` An array containing the smoothed signal.

Notes

History:

- **14 Feb 2018 Lisa Dang** Written for early version of SPCA
- **23 Sep 2019 Taylor Bell** Generalized upon the code
- **02 Nov 2021 Taylor Bell** Added to Eureka!

`eureka.S4_generate_lightcurves.drift.spec1D(spectra, meta, log)`

Measures the 1D spectrum drift over all integrations.

Parameters

spectra: `ndarray` 2D array of flux values (nint, nx).

meta: `MetaClass` The metadata object.

log: `logedit.Logedit` The open log in which notes from this step can be added.

Returns

meta: `MetaClass` The updated metadata object.

Notes

History:

- **Dec 2013 KBS** Written for HST.
- **Jun 2021 KBS** Updated for JWST.
- **Oct 18, 2021 Taylor Bell** Minor tweak to `cc_spec` inputs.
- **Nov 02, 2021 Taylor Bell** Added option for subtraction of continuum using a highpass filter before cross-correlation.

6.5.2 `S4_generate_lightcurves.plots_s4`

`eureka.S4_generate_lightcurves.plots_s4.binned_lightcurve(meta, time, i)`

Plot each spectroscopic light curve. (Fig 4300)

Parameters

- meta:** `MetaClass` The metadata object.
- time:** `ndarray (1D)` The time in `meta.time_units` of each data point.
- i:** `int` The current bandpass number.

Returns

None

`eureka.S4_generate_lightcurves.plots_s4.cc_spec(meta, ref_spec, fit_spec, n)`

Compare the spectrum used for cross-correlation with the current spectrum (Fig 4400).

Parameters

- meta:** `MetaClass` The metadata object.
- ref_spec:** `ndarray (1D)` The reference spectrum used for cross-correlation.
- fit_spec:** `ndarray (1D)` The extracted spectrum for the current integration.
- n:** `int` The current integration number.

Returns

None

`eureka.S4_generate_lightcurves.plots_s4.cc_vals(meta, vals, n)`

Make the cross-correlation strength plot (Fig 4500).

Parameters

- meta:** `MetaClass` The metadata object.
- vals:** `ndarray (1D)` The cross-correlation strength.
- n:** `int` The current integration number.

Returns

None

```
eureka.S4_generate_lightcurves.plots_s4.drift1d(meta)
```

Plot the 1D drift/jitter results. (Fig 4100)

Parameters

meta: **MetaClass** The metadata object.

Returns

None

```
eureka.S4_generate_lightcurves.plots_s4.lc_driftcorr(meta, wave_1d, optspec)
```

Plot a 2D light curve with drift correction. (Fig 4200)

Parameters

meta: **MetaClass** The metadata object.

wave_1d: Wavelength array with trimmed edges depending on xwindow and ywindow which have been set in the S3 ecf

optspec: The optimally extracted spectrum.

Returns

None

6.5.3 S4_generate_lightcurves.s4_genLC

```
class eureka.S4_generate_lightcurves.s4_genLC.MetaClass
```

Bases: object

A class to hold Eureka! metadata.

```
eureka.S4_generate_lightcurves.s4_genLC.lcJWST(eventlabel, ecf_path='./', s3_meta=None)
```

Compute photometric flux over specified range of wavelengths.

Parameters

eventlabel: **str** The unique identifier for these data.

ecf_path: **str** The absolute or relative path to where ecfs are stored

s3_meta: **MetaClass** The metadata object from Eureka!'s S3 step (if running S3 and S4 sequentially).

Returns

meta: **MetaClass** The metadata object with attributes added by S4.

Notes

History:

- **June 2021 Kevin Stevenson** Initial version
- **October 2021 Taylor Bell** Updated to allow for inputs from new S3

```
eureka.S4_generate_lightcurves.s4_genLC.load_general_s3_meta_info(meta, ecf_path, s3_meta)
```

```
eureka.S4_generate_lightcurves.s4_genLC.load_specific_s3_meta_info(meta, ecf_path, run_i,  
                                                                    spec_hw_val, bg_hw_val)
```

```
eureka.S4_generate_lightcurves.s4_genLC.read_s3_meta(meta)
```

6.6 S5_lightcurve_fitting

6.6.1 S5_lightcurve_fitting.fitters

```
eureka.S5_lightcurve_fitting.fitters.demcfitter(lc, model, meta, log, **kwargs)
```

Perform sampling using Differential Evolution Markov Chain.

This is an empty placeholder function to be filled later.

Parameters

- lc** [eureka.S5_lightcurve_fitting.lightcurve.LightCurve] The lightcurve data object
- model** [eureka.S5_lightcurve_fitting.models.CompositeModel] The composite model to fit
- meta** [MetaClass] The metadata object
- log** [logedit.Logedit] The open log in which notes from this step can be added.
- **kwargs** [dict] Arbitrary keyword arguments.

Returns

- best_model** [eureka.S5_lightcurve_fitting.models.CompositeModel] The composite model after fitting

Notes

History:

- **December 29, 2021 Taylor Bell** Updated documentation and arguments

```
eureka.S5_lightcurve_fitting.fitters.dynestyfitter(lc, model, meta, log, **kwargs)
```

Perform sampling using dynesty.

Parameters

- lc:** eureka.S5_lightcurve_fitting.lightcurve.LightCurve The lightcurve data object
- model:** eureka.S5_lightcurve_fitting.models.CompositeModel The composite model to fit
- meta:** MetaClass The metadata object
- log:** logedit.Logedit The open log in which notes from this step can be added.
- **kwargs:** Arbitrary keyword arguments.

Returns

- best_model:** eureka.S5_lightcurve_fitting.models.CompositeModel The composite model after fitting

Notes

History:

- **December 29, 2021 Taylor Bell** Updated documentation. Reduced repeated code.
- **January 7-22, 2022 Megan Mansfield** Adding ability to do a single shared fit across all channels
- **February 23-25, 2022 Megan Mansfield** Added log-uniform and Gaussian priors.
- **February 28-March 1, 2022 Caroline Piaulet** Adding scatter_ppm parameter.

`eureka.S5_lightcurve_fitting.fitters.emceefitter(lc, model, meta, log, **kwargs)`

Perform sampling using emcee.

Parameters

- lc:** `eureka.S5_lightcurve_fitting.lightcurve.LightCurve` The lightcurve data object
- model:** `eureka.S5_lightcurve_fitting.models.CompositeModel` The composite model to fit
- meta:** `MetaClass` The metadata object
- log:** `logedit.Logedit` The open log in which notes from this step can be added.
- **kwargs:** Arbitrary keyword arguments.

Returns

- best_model:** `eureka.S5_lightcurve_fitting.models.CompositeModel` The composite model after fitting

Notes

History:

- **December 29, 2021 Taylor Bell** Updated documentation. Reduced repeated code.
- **January 7-22, 2022 Megan Mansfield** Adding ability to do a single shared fit across all channels
- **February 23-25, 2022 Megan Mansfield** Added log-uniform and Gaussian priors.
- **February 28-March 1, 2022 Caroline Piaulet** Adding scatter_ppm parameter. Added statements to avoid some initial state issues.

`eureka.S5_lightcurve_fitting.fitters.group_variables(model)`

Group variables into fitted and frozen.

Parameters

- model:** `eureka.S5_lightcurve_fitting.models.CompositeModel` The composite model to fit

Returns

- freenames:** `np.array` The names of fitted variables.
- freepars:** `np.array` The fitted variables.
- prior1:** `np.array` The lower bound for constrained variables with uniform/log uniform priors, or mean for constrained variables with Gaussian priors.
- prior2:** `np.array` The upper bound for constrained variables with uniform/log uniform priors, or mean for constrained variables with Gaussian priors.
- priortype:** `np.array` Keywords indicating the type of prior for each free parameter.

indep_vars: dict The frozen variables.

Notes

History:

- **December 29, 2021 Taylor Bell** Moved code to separate function to reduce repeated code.
- **January 11, 2022 Megan Mansfield** Added ability to have shared parameters
- **February 23-25, 2022 Megan Mansfield** Added log-uniform and Gaussian priors.

`eureka.S5_lightcurve_fitting.fitters.group_variables_lmfit(model)`

Group variables into fitted and frozen for lmfit fitter.

Parameters

model: `eureka.S5_lightcurve_fitting.models.CompositeModel` The composite model to fit

Returns

paramlist: list The fitted variables.

freenames: np.array The names of fitted variables.

indep_vars: dict The frozen variables.

Notes

History:

- **December 29, 2021 Taylor Bell** Moved code to separate function to look similar to other fitters.

`eureka.S5_lightcurve_fitting.fitters.initialize_emcee_walkers(meta, log, ndim, lsq_sol, freepars, prior1, prior2, priortype)`

`eureka.S5_lightcurve_fitting.fitters.lmfitter(lc, model, meta, log, **kwargs)`

Perform a fit using lmfit.

Parameters

lc: `eureka.S5_lightcurve_fitting.lightcurve.LightCurve` The lightcurve data object

model: `eureka.S5_lightcurve_fitting.models.CompositeModel` The composite model to fit

meta: MetaClass The metadata object

log: `logedit.Logedit` The open log in which notes from this step can be added.

****kwargs:** Arbitrary keyword arguments.

Returns

best_model: `eureka.S5_lightcurve_fitting.models.CompositeModel` The composite model after fitting

Notes

History:

- **December 29, 2021 Taylor Bell** Updated documentation. Reduced repeated code.
- **February 28-March 1, 2022 Caroline Piaulet** Adding scatter_ppm parameter.

```
eureka.S5_lightcurve_fitting.fitters.load_old_fitparams(meta, log, channel, freenames)
```

```
eureka.S5_lightcurve_fitting.fitters.lsqfitter(lc, model, meta, log, calling_function='lsq', **kwargs)
```

Perform least-squares fit.

Parameters

- lc:** `eureka.S5_lightcurve_fitting.lightcurve.LightCurve` The lightcurve data object
- model:** `eureka.S5_lightcurve_fitting.models.CompositeModel` The composite model to fit
- meta:** `MetaClass` The metadata object
- log:** `logedit.Logedit` The open log in which notes from this step can be added.
- **kwargs:** Arbitrary keyword arguments.

Returns

- best_model:** `eureka.S5_lightcurve_fitting.models.CompositeModel` The composite model after fitting

Notes

History:

- **December 29-30, 2021 Taylor Bell** Updated documentation and arguments. Reduced repeated code. Also saving covariance matrix for later estimation of sampler step size.
- **January 7-22, 2022 Megan Mansfield** Adding ability to do a single shared fit across all channels
- **February 28-March 1, 2022 Caroline Piaulet** Adding scatter_ppm parameter

```
eureka.S5_lightcurve_fitting.fitters.save_fit(meta, lc, model, fitter, fit_params, freenames,
                                              samples=[], upper_errs=[], lower_errs=[])
```

```
eureka.S5_lightcurve_fitting.fitters.start_from_oldchain_emcee(meta, log, ndim, channel,
                                                                freenames)
```

6.6.2 S5_lightcurve_fitting.lightcurve

Base and child classes to handle light curve fitting

Author: Joe Filippazzo Email: jfilippazzo@stsci.edu

```
class eureka.S5_lightcurve_fitting.lightcurve.LightCurve(time, flux, channel, nchannel, log,
                                                         longparamlist, unc=None,
                                                         parameters=None, time_units='BJD',
                                                         name='My Light Curve', share=False)
```

Bases: `eureka.S5_lightcurve_fitting.models.Model.Model`

Attributes

flux A getter for the flux

parameters A getter for the parameters

time A getter for the time

units A getter for the units

Methods

<code>fit(model, meta, log[, fitter])</code>	Fit the model to the lightcurve
<code>interp(new_time, **kwargs)</code>	Evaluate the model over a different time array
<code>plot(meta[, fits])</code>	Plot the light curve with all available fits
<code>reset()</code>	Reset the results
<code>update(newparams, names, **kwargs)</code>	Update parameter values

fit(*model*, *meta*, *log*, *fitter*='lsq', ***kwargs*)

Fit the model to the lightcurve

Parameters

model: `eureka.S5_lightcurve_fitting.models.CompositeModel` The model to fit to the data

meta: `MetaClass` The metadata object

log: `logedit.Logedit` The open log in which notes from this step can be added.

fitter: `str` The name of the fitter to use

****kwargs:** Arbitrary keyword arguments.

Returns

`None`

Notes

History: - Dec 29, 2021 Taylor Bell

Updated documentation and reduced repeated code

plot(*meta*, *fits*=`True`)

Plot the light curve with all available fits

Parameters

fits: `bool` Plot the fit models

draw: `bool` Show the figure, else return it

Returns

`None`

reset()

Reset the results

6.6.3 S5_lightcurve_fitting.likelihood

`eureka.S5_lightcurve_fitting.likelihood.GP_loglikelihood(model, fit)`

Compute likelihood, when model fit includes GP

Parameters

model: `CompositeModel` object The model including the GP model

fit: `ndarray` the evaluated model without the GP

Returns

likelihood of the model

Notes

History:

- **March 11, 2022 Eva-Maria Ahrer** moved code from `Model.py`

`eureka.S5_lightcurve_fitting.likelihood.computeRMS(data, maxnbins=None, binstep=1, isrmserr=False)`

Compute the root-mean-squared and standard error of data for various bin sizes.

Parameters

data: `ndarray` The residuals after fitting.

maxnbins: `int, optional` The maximum number of bins. Use `None` to default to 10 points per bin.

binstep: `int, optional` Bin step size.

isrmserr: `bool` True if return rmserr, else False.

Returns

rms: `ndarray` The RMS for each bin size.

stderr: `ndarray` The standard error for each bin size.

binsz: `ndarray` The different bin sizes.

rmserr: `ndarray, optional` The uncertainty in the RMS.

Notes

History:

- **December 29-30, 2021 Taylor Bell** Moved code to separate file, added documentation.

`eureka.S5_lightcurve_fitting.likelihood.computeRedChiSq(lc, log, model, meta, freenames)`

Compute the reduced chi-squared value.

Parameters

lc: `eureka.S5_lightcurve_fitting.lightcurve.LightCurve` The lightcurve data object

log: `logedit.Logedit` The open log in which notes from this step can be added.

model: `eureka.S5_lightcurve_fitting.models.CompositeModel` The composite model to fit

meta: `MetaObject` The metadata object.

freenames: iterable The names of the fitted parameters.

log: logedit.Logedit The open log in which notes from this step can be added.

Returns

chi2red: float The reduced chi-squared value.

Notes

History:

- **December 29-30, 2021 Taylor Bell** Moved code to separate file, added documentation.
- **February, 2022 Eva-Maria Ahrer** Added GP functionality

`eureka.S5_lightcurve_fitting.likelihood.ln_like(theta, lc, model, freenames)`

Compute the log-likelihood.

Parameters

theta: ndarray The current estimate of the fitted parameters

lc: eureka.S5_lightcurve_fitting.lightcurve.LightCurve The lightcurve data object

model: eureka.S5_lightcurve_fitting.models.CompositeModel The composite model to fit

freenames: iterable The names of the fitted parameters.

Returns

ln_like_val: ndarray The log-likelihood value at the position theta.

Notes

History:

- **December 29-30, 2021 Taylor Bell** Moved code to separate file, added documentation.
- **January 22, 2022 Megan Mansfield** Adding ability to do a single shared fit across all channels
- **February, 2022 Eva-Maria Ahrer** Adding GP likelihood

`eureka.S5_lightcurve_fitting.likelihood.lnprior(theta, prior1, prior2, priortype)`

Compute the log-prior.

Parameters

theta: ndarray The current estimate of the fitted parameters

prior1: ndarray The lower-bound for uniform/log uniform priors, or mean for normal priors.

prior2: ndarray The upper-bound for uniform/log uniform priors, or std. dev. for normal priors.

priortype: ndarray Keywords indicating the type of prior for each free parameter.

Returns

lnprior_prob: ndarray The log-prior probability value at the position theta.

Notes

History:

- **December 29-30, 2021 Taylor Bell** Moved code to separate file, added documentation.
- **February 23-25, 2022 Megan Mansfield** Added log-uniform and Gaussian priors.

`eureka.S5_lightcurve_fitting.likelihood.lnprob(theta, lc, model, prior1, prior2, priortype, freenames)`

Compute the log-probability.

Parameters

- theta: ndarray** The current estimate of the fitted parameters
- lc: eureka.S5_lightcurve_fitting.lightcurve.LightCurve** The lightcurve data object
- model: eureka.S5_lightcurve_fitting.models.CompositeModel** The composite model to fit
- prior1: ndarray** The lower-bound for uniform/log uniform priors, or mean for normal priors.
- prior2: ndarray** The upper-bound for uniform/log uniform priors, or std. dev. for normal priors.
- priortype: ndarray** Keywords indicating the type of prior for each free parameter.
- freenames:** The names of the fitted parameters.

Returns

- ln_prob_val: ndarray** The log-probability value at the position theta.

Notes

History:

- **December 29-30, 2021 Taylor Bell** Moved code to separate file, added documentation.
- **February 23-25, 2022 Megan Mansfield** Added log-uniform and Gaussian priors.

`eureka.S5_lightcurve_fitting.likelihood.ptform(theta, prior1, prior2, priortype)`

Compute the prior transform for nested sampling.

Parameters

- theta: ndarray** The current estimate of the fitted parameters
- prior1: ndarray** The lower-bound for uniform/log uniform priors, or mean for normal priors.
- prior2: ndarray** The upper-bound for uniform/log uniform priors, or std. dev. for normal priors.
- priortype: ndarray** Keywords indicating the type of prior for each free parameter.
- freenames:** The names of the fitted parameters.

Returns

- p: ndarray** The prior transform.

Notes

History:

- **February 23-25, 2022 Megan Mansfield** Added log-uniform and Gaussian priors.

`eureka.S5_lightcurve_fitting.likelihood.transform_log_uniform(x, a, b)`

`eureka.S5_lightcurve_fitting.likelihood.transform_normal(x, mu, sigma)`

`eureka.S5_lightcurve_fitting.likelihood.transform_uniform(x, a, b)`

6.6.4 S5_lightcurve_fitting.limb_darkening

6.6.5 S5_lightcurve_fitting.modelgrid

A module for creating and managing grids of model spectra

```
class eureka.S5_lightcurve_fitting.modelgrid.ModelGrid(model_directory, bibcode='2013A &
A...553A...6H', names={'FeH': 'PHXM_H',
'Lbol': 'PHXLUM', 'Teff': 'PHXTEFF',
'logg': 'PHXLOGG', 'mass': 'PHXMASS'},
resolution=None, wave_units=Unit('um'),
**kwargs)
```

Bases: object

Creates a ModelGrid object which contains a multi-parameter grid of model spectra and its references

Attributes

path: str The path to the directory of FITS files used to create the ModelGrid

refs: list, str The references for the data contained in the ModelGrid

teff_rng: tuple The range of effective temperatures [K]

logg_rng: tuple The range of surface gravities [dex]

FeH_rng: tuple The range of metallicities [dex]

wave_rng: array-like The wavelength range of the models [um]

n_bins: int The number of bins for the ModelGrid wavelength array

data: astropy.table.Table The table of parameters for the ModelGrid

inv_file: str An inventory file to more quickly load the database

Methods

<code>customize([Teff_rng,logg_rng,FeH_rng,...])</code>	Trims the model grid by the given ranges in effective temperature, surface gravity, and metallicity.
<code>export(filepath,**kwargs)</code>	Export the model with the given parameters to a FITS file at the given filepath
<code>get(Teff,logg,FeH[,resolution,interp])</code>	Retrieve the wavelength, flux, and effective radius for the spectrum of the given parameters
<code>grid_interp(Teff,logg,FeH[,plot])</code>	Interpolate the grid to the desired parameters
<code>info()</code>	Print a table of info about the current ModelGrid
<code>load_flux([reset])</code>	Retrieve the flux arrays for all models and load into the ModelGrid.array attribute with shape (Teff,logg,FeH,mu,wavelength)
<code>reset()</code>	Reset the current grid to the original state
<code>set_units([wave_units])</code>	Set the wavelength and flux units

customize(*Teff_rng*=(2300, 8000), *logg_rng*=(0, 6), *FeH_rng*=(-2, 1), *wave_rng*=(*<Quantity 0. um>*, *<Quantity 40. um>*), *n_bins*="")

Trims the model grid by the given ranges in effective temperature, surface gravity, and metallicity. Also sets the wavelength range and number of bins for retrieved model spectra.

Parameters

Teff_rng: array-like The lower and upper inclusive bounds for the effective temperature (K)

logg_rng: array-like The lower and upper inclusive bounds for the logarithm of the surface gravity (dex)

FeH_rng: array-like The lower and upper inclusive bounds for the logarithm of the ratio of the metallicity and solar metallicity (dex)

wave_rng: array-like The lower and upper inclusive bounds for the wavelength (microns)

n_bins: int The number of bins for the wavelength axis

export(*filepath*, ***kwargs*)

Export the model with the given parameters to a FITS file at the given filepath

Parameters

filepath: str The path to the target FITS file

get(*Teff*, *logg*, *FeH*, *resolution=None*, *interp=True*)

Retrieve the wavelength, flux, and effective radius for the spectrum of the given parameters

Parameters

Teff: int The effective temperature (K)

logg: float The logarithm of the surface gravity (dex)

FeH: float The logarithm of the ratio of the metallicity and solar metallicity (dex)

resolution: int (optional) The desired wavelength resolution (λ/d_λ)

interp: bool Interpolate the model if possible

Returns

dict A dictionary of arrays of the wavelength, flux, and mu values and the effective radius for the given model

grid_interp(*Teff*, *logg*, *FeH*, *plot=False*)

Interpolate the grid to the desired parameters

Parameters

Teff: **int** The effective temperature (K)

logg: **float** The logarithm of the surface gravity (dex)

FeH: **float** The logarithm of the ratio of the metallicity and solar metallicity (dex)

plot: **bool** Plot the interpolated spectrum along with the 8 neighboring grid spectra

Returns

dict A dictionary of arrays of the wavelength, flux, and mu values and the effective radius for the given model

info()

Print a table of info about the current ModelGrid

load_flux(*reset=False*)

Retrieve the flux arrays for all models and load into the ModelGrid.array attribute with shape (Teff, logg, FeH, mu, wavelength)

reset()

Reset the current grid to the original state

set_units(*wave_units=Unit('um')*)

Set the wavelength and flux units

Parameters

wave_units: **str**, **astropy.units.core.PrefixUnit/CompositeUnit** The wavelength units

6.6.6 S5_lightcurve_fitting.plots_s5

eureka.S5_lightcurve_fitting.plots_s5.plot_GP_components(*lc*, *model*, *meta*, *fitter*, *isTitle=True*)

Plot the lightcurve + GP model + residuals (Fig 5600)

Parameters

lc: **eureka.S5_lightcurve_fitting.lightcurve.LightCurve** The lightcurve data object

model: **eureka.S5_lightcurve_fitting.models.CompositeModel** The fitted composite model

meta: **MetaClass** The metadata object

fitter: **str** The name of the fitter (for plot filename)

Returns

None

Notes

History:

- **February 28, 2022 Eva-Maria Ahrer** Written function
- **March 9, 2022 Eva-Maria Ahrer** Adapted with shared parameters

```
eureka.S5_lightcurve_fitting.plots_s5.plot_chain(samples, lc, meta, freenames, fitter='emcee',
                                                burnin=False, nburn=0, nrows=3, ncols=4,
                                                nthin=1)
```

Plot the evolution of the chain to look for temporal trends (Fig 5400)

Parameters

- samples: ndarray** The samples produced by the sampling algorithm
- lc: eureka.S5_lightcurve_fitting.lightcurve.LightCurve** The lightcurve data object
- freenames: iterable** The names of the fitted parameters
- meta: MetaClass** The metadata object
- fitter: str** The name of the fitter (for plot filename)
- burnin: bool** Whether or not the samples include the burnin phase
- nburn: int** The number of burn-in steps that are discarded later
- nrows: int** The number of rows to make per figure
- ncols: int** The number of columns to make per figure
- nthin: int** If >1, the plot will use every nthin point to help speed up computation and reduce clutter on the plot.

Returns

None

Notes

History:

- **December 29, 2021 Taylor Bell** Moved plotting code to a separate function.

```
eureka.S5_lightcurve_fitting.plots_s5.plot_corner(samples, lc, meta, freenames, fitter)
```

Plot a corner plot. (Fig 5300)

Parameters

- samples: ndarray** The samples produced by the sampling algorithm
- lc: eureka.S5_lightcurve_fitting.lightcurve.LightCurve** The lightcurve data object
- freenames: iterable** The names of the fitted parameters
- meta: MetaClass** The metadata object
- fitter: str** The name of the fitter (for plot filename)

Returns

None

Notes

History:

- **December 29, 2021 Taylor Bell** Moved plotting code to a separate function.

`eureka.S5_lightcurve_fitting.plots_s5.plot_fit(lc, model, meta, fitter, isTitle=True)`

Plot the fitted model over the data. (Fig 5100)

Parameters

- lc:** `eureka.S5_lightcurve_fitting.lightcurve.LightCurve` The lightcurve data object
- model:** `eureka.S5_lightcurve_fitting.models.CompositeModel` The fitted composite model
- meta:** `MetaClass` The metadata object
- fitter:** `str` The name of the fitter (for plot filename)

Returns

None

Notes

History:

- **December 29, 2021 Taylor Bell** Moved plotting code to a separate function.
- **January 7-22, 2022 Megan Mansfield** Adding ability to do a single shared fit across all channels
- **February 28-March 1, 2022 Caroline Piaulet** Adding `scatter_ppm` parameter

`eureka.S5_lightcurve_fitting.plots_s5.plot_res_distr(lc, model, meta, fitter)`

Plot the normalized distribution of residuals + a Gaussian. (Fig 5500)

Parameters

- lc:** `eureka.S5_lightcurve_fitting.lightcurve.LightCurve` The lightcurve data object
- model:** `eureka.S5_lightcurve_fitting.models.CompositeModel` The fitted composite model
- meta:** `MetaClass` The metadata object
- fitter:** `str` The name of the fitter (for plot filename)

Returns

None

Notes

History:

- **February 18, 2022 Caroline Piaulet** Created function

`eureka.S5_lightcurve_fitting.plots_s5.plot_rms(lc, model, meta, fitter)`

Plot an Allan plot to look for red noise. (Fig 5200)

Parameters

- lc:** `eureka.S5_lightcurve_fitting.lightcurve.LightCurve` The lightcurve data object
- model:** `eureka.S5_lightcurve_fitting.models.CompositeModel` The fitted composite model

meta: **MetaClass** The metadata object

fitter: **str** The name of the fitter (for plot filename)

Returns

None

Notes

History:

- **December 29, 2021 Taylor Bell** Moved plotting code to a separate function.
- **January 7-22, 2022 Megan Mansfield** Adding ability to do a single shared fit across all channels

6.6.7 S5_lightcurve_fitting.s5_fit

class eureka.S5_lightcurve_fitting.s5_fit.**MetaClass**

Bases: object

A class to hold Eureka! metadata.

eureka.S5_lightcurve_fitting.s5_fit.**fitJWST**(*eventlabel*, *ecf_path*='./', *s4_meta*=None)

Fits 1D spectra with various models and fitters.

Parameters

eventlabel: **str** The unique identifier for these data.

ecf_path: **str** The absolute or relative path to where ecfs are stored

s4_meta: **MetaClass** The metadata object from Eureka!'s S4 step (if running S4 and S5 sequentially).

Returns

meta: **MetaClass** The metadata object with attributes added by S5.

Notes

History:

- **November 12-December 15, 2021 Megan Mansfield** Original version
- **December 17-20, 2021 Megan Mansfield** Connecting S5 to S4 outputs
- **December 17-20, 2021 Taylor Bell** Increasing connectedness of S5 and S4
- **January 7-22, 2022 Megan Mansfield** Adding ability to do a single shared fit across all channels
- **January - February, 2022 Eva-Maria Ahrer** Adding GP functionality

eureka.S5_lightcurve_fitting.s5_fit.**fit_channel**(*meta*, *time*, *flux*, *chan*, *flux_err*, *eventlabel*, *sharedp*, *params*, *log*, *longparamlist*, *time_units*, *paramtitles*, *chanrng*)

eureka.S5_lightcurve_fitting.s5_fit.**load_general_s4_meta_info**(*meta*, *ecf_path*, *s4_meta*)

```
eureka.S5_lightcurve_fitting.s5_fit.load_specific_s4_meta_info(meta, ecf_path, run_i,
                                                             spec_hw_val, bg_hw_val)

eureka.S5_lightcurve_fitting.s5_fit.make_longparamlist(meta, params, chanrng)

eureka.S5_lightcurve_fitting.s5_fit.read_s4_meta(meta)
```

6.6.8 S5_lightcurve_fitting.simulations

Functions to simulate lightcurve data from ExoMAST parameters

Author: Joe Filippazzo Email: jfilippazzo@stsci.edu

```
eureka.S5_lightcurve_fitting.simulations.simulate_lightcurve(target, snr=1000.0, npts=1000,
                                                             nbins=10, radius=None,
                                                             ldc=('quadratic', [0.1, 0.1]),
                                                             plot=False)
```

Simulate lightcurve data for the given target exoplanet

Parameters

target: str The name of the target to simulate

snr: float The signal to noise to use

npts: int The number of points in each lightcurve

nbins: int The number of lightcurves

radius: array-like, float (optional) The radius or radii value(s) to use

ldcs: sequence The limb darkening profile name and coefficients

plot: bool Plot the figure

Returns

tuple The time, flux, uncertainty, and transit parameters

6.6.9 S5_lightcurve_fitting.utils

A module for utility functions

```
eureka.S5_lightcurve_fitting.utils.build_target_url(target_name)
```

Build restful api url based on target name.

Parameters

target_name [string] The name of the target transit.

Returns

target_url [string]

```
eureka.S5_lightcurve_fitting.utils.calc_zoom(R_f, arr)
```

Calculate the zoom factor required to make the given array into the given resolution

Parameters

R_f: int The desired final resolution of the wavelength array

arr: array-like The array to zoom

`eureka.S5_lightcurve_fitting.utils.color_gen(colormap='viridis', key=None, n=10)`

Color generator for Bokeh plots

Parameters

colormap: **str, sequence** The name of the color map

Returns

generator A generator for the color palette

`eureka.S5_lightcurve_fitting.utils.download_exoctk_data(download_location='/home/docs')`

Retrieves the exoctk_data materials from Box, downloads them to the user's local machine, uncompresses the files, and arranges them into an exoctk_data directory.

Parameters

download_location [string] The path to where the ExoCTK data package will be downloaded. The default setting is the user's \$HOME directory.

`eureka.S5_lightcurve_fitting.utils.filter_table(table, **kwargs)`

Retrieve the filtered rows

Parameters

table: **astropy.table.Table, pandas.DataFrame** The table to filter

param: **str** The parameter to filter by, e.g. 'Teff'

value: **str, float, int, sequence** The criteria to filter by, which can be single valued like 1400 or a range with operators [$<$, $<=$, $>$, $>=$], e.g. (' >1200 ', ' $<=1400$ ')

Returns

astropy.table.Table, pandas.DataFrame The filtered table

`eureka.S5_lightcurve_fitting.utils.find_closest(axes, points, n=1, values=False)`

Find the n-neighboring elements of a given value in an array

Parameters

axes: **list, np.array** The array(s) to search

points: **array-like, float** The point(s) to search for

n: **int** The number of values to the left and right of the points

Returns

——

np.ndarray The n-values to the left and right of 'points' in 'axes'

`eureka.S5_lightcurve_fitting.utils.get_canonical_name(target_name)`

Get ExoMAST preferred name for exoplanet.

Parameters

target_name [string] The name of the target transit.

Returns

canonical_name [string]

`eureka.S5_lightcurve_fitting.utils.get_env_variables()`

Returns a dictionary containing various environment variable information.

Returns

env_variables [dict] A dictionary containing various environment variable data

`eureka.S5_lightcurve_fitting.utils.get_target_data(target_name)`

Send request to exomast restful api for target information.

Parameters

target_name [string] The name of the target transit

Returns

target_data: json: json object with target data.

`eureka.S5_lightcurve_fitting.utils.interp_flux(mu, flux, params, values)`

Interpolate a cube of synthetic spectra for a given index of mu

Parameters

mu: int The index of the (Teff,logg,FeH, *mu*, wavelength) data cube to interpolate

flux: np.ndarray The 5D data array

params: list A list of each free parameter range

values: list A list of each free parameter values

Returns

tu The array of new flux values

`eureka.S5_lightcurve_fitting.utils.rebin_spec(spec, wavnew, oversamp=100, plot=False)`

Rebin a spectrum to a new wavelength array while preserving the total flux

Parameters

spec: array-like The wavelength and flux to be binned

wavnew: array-like The new wavelength array

Returns

np.ndarray The rebinned flux

`eureka.S5_lightcurve_fitting.utils.writeFITS(filename, extensions, headers=())`

Write some data to a new FITS file

Parameters

filename: str The filename of the output FITS file

extensions: dict The extension name and associated data to include in the file

headers: array-like The (keyword, value, comment) groups for the PRIMARY header extension

6.7 S6_planet_spectra

6.7.1 S6_planet_spectra.plots_s6

```
eureka.S6_planet_spectra.plots_s6.plot_spectrum(meta, model_x=None, model_y=None, y_scalar=1,
                                                ylabel='$R_{\rm p}/R_{\rm *}$',
                                                xlabel='Wavelength ($\mu$m)', scaleHeight=None,
                                                planet_R0=None)
```

6.7.2 S6_planet_spectra.s6_spectra

```
class eureka.S6_planet_spectra.s6_spectra.MetaClass
```

Bases: object

A class to hold Eureka! metadata.

```
eureka.S6_planet_spectra.s6_spectra.load_general_s5_meta_info(meta, ecf_path, s5_meta)
```

```
eureka.S6_planet_spectra.s6_spectra.load_specific_s5_meta_info(meta, ecf_path, run_i,
                                                                spec_hw_val, bg_hw_val)
```

```
eureka.S6_planet_spectra.s6_spectra.parse_s5_saves(meta, fit_methods, y_param,
                                                    channel_key='shared')
```

```
eureka.S6_planet_spectra.s6_spectra.plot_spectra(eventlabel, ecf_path='./', s5_meta=None)
```

Gathers together different wavelength fits and makes transmission/emission spectra.

Parameters

eventlabel: str The unique identifier for these data.

ecf_path: str The absolute or relative path to where ecfs are stored

s5_meta: MetaClass The metadata object from Eureka!'s S5 step (if running S5 and S6 sequentially).

Returns

meta: MetaClass The metadata object with attributes added by S6.

Notes

History:

- **Feb 14, 2022 Taylor Bell** Original version

```
eureka.S6_planet_spectra.s6_spectra.read_s5_meta(meta)
```


EUREKA! FAQ

In this section you will find frequently asked questions about Eureka! as well as fixes for common problems

7.1 Common Errors

7.1.1 Missing packages during installation

If you are encountering errors when installing Eureka! like missing packages (e.g. `extension_helpers`), be sure that you are following the instructions on the [Installation](#) page. If you are trying to directly call `setup.py` using the `python setup.py install` command, you should instead be using a `pip` or `conda` installation command which helps to make sure that all required dependencies are installed in the right order and checks for implicit dependencies. If you still encounter issues, you should be sure that you are using a new `conda` environment as other packages you've previously installed could have conflicting requirements with Eureka!.

If you are following the installation instructions and still encounter an error, please open a new Issue on [GitHub](#) and paste the full error message you are getting along with details about which python version and operating system you are using.

7.1.2 Issues installing or importing batman

Be sure that you are installing (or have installed) `batman-package` (not `batman`) from `pip`. If you have accidentally installed the wrong package you can try `pip uninstall` it, but you may just need to make a whole new environment. In general, we strongly recommend you closely follow the instructions on the [Installation](#) page.

7.1.3 Issues installing or importing jwst

As a first step, make sure that you were following the instructions on the [Installation](#) page and were either using the `conda environment.yml` installation method or a `pip` installation command that included “[jwst]” like `pip install .[jwst]`. If you are getting error messages on linux that mention `gcc`, you likely need to first install `gcc` using `sudo apt install gcc`. If you are getting messages on macOS that mention `clang`, `X-Code`, and/or `CommandLineTools`, you need to make sure that `CommandLineTools` is first manually installed. `CommandLineTools` is installed whenever you first manually use a command that requires it like `git`, so one very simple way to install it would be to run `git status` in a terminal which should give a pop-up saying that command line developer tools must be installed first. Alternatively, you can instead run `xcode-select --install` which will install `CommandLineTools`.

If you were doing that and you are still receiving error messages, it is possible that something about your installation environment does not play well with `jwst`. You should open or comment on an already open issue on the Eureka! [GitHub](#) page and tell us as many details as you can about every step you took to finally get to your error message as well as details about your operating system, python version, and conda version. You should also consider opening an

issue on the [jwst GitHub](#) page as there may not be much we can do to help troubleshoot a package we have no control over.

Finally, if you simply cannot get jwst to install and still want to use later stages of the Eureka! pipeline, then you can install Eureka! using `pip install .` instead of `pip install .[jwst]` which will not install the jwst package. Note, however, that this means that Stages 1 and 2 will not work at all as Eureka's Stages 1 and 2 simply offer ways of editing the behaviour of the jwst package's Stages 1 and 2.

7.1.4 Matplotlib RuntimeError() whenever Eureka is imported and plt.show() is called

The importing of Eureka! sometimes causes a runtime error with Matplotlib. The error is related to latex and reads as the following

`RuntimeError: Failed to process string with tex because latex could not be found`

There are several workarounds to this problem. The first is to insert these lines prior to calling `plt.show()`

```
from matplotlib import rc
rc('text', usetex=False)
```

Another solution would be to catch it as an exception:

```
try:
    plt.show()
except RuntimeError:
    from matplotlib import rc
    rc('text', usetex=False)
```

Some more permanent solutions would be to:

- Install the following `sudo apt install cm-super`, although this won't always work
- Identify where your TeX installation is and manually add it to PATH in your `bashrc` or `bash_profile`. An example of this is to change `export PATH=~/.anaconda3/bin:$PATH` in your `~/.bashrc` file to `export PATH=~/.anaconda3/bin:~/Library/TeX/texbin:$PATH`. For anyone using Ubuntu or an older version of Mac this might be found in `/usr/bin` instead. Make sure you run `source ~/.bash_profile` or `source ~/.bashrc` to apply the changes.

7.1.5 My question isn't listed here!

First check to see if your question/concern is already addressed in an open or closed issue on the Eureka! [GitHub](#) page. If not, please open a new issue and paste the full error message you are getting along with details about which python version and operating system you are using, and ideally the `ecf` you used to get your error (ideally copy-paste it into the issue in a quote block).

COPYRIGHT

© Copyright 2021, [Eureka! pipeline developers](#)

The current main developers of Eureka! are (ordered alphabetically by first name):

- Aarynn Carter
- Adina Feinstein
- Eva-Maria Ahrer
- Giannina Guzman
- Kevin Stevenson
- Laura Kreidberg
- Megan Mansfield
- Sebastian Zieba
- Taylor Bell

INTRODUCTION

Welcome to the Eureka! tutorial - today we will learn how to run Eureka!'s S3 data reduction module, which takes 2D JWST data and reduces it to 1D spectra.

Eureka! is an open-source python package available for download at <https://github.com/kevin218/Eureka> (lead developers are Sebastian Zieba, Kevin Stevenson, and Laura Kreidberg).

9.1 Goals

- walk through all the major steps in data reduction
- get comfortable with the Eureka! structure and syntax
- most importantly, make sure none of the steps are a black box.

9.2 Import standard python packages and Eureka!

```
[16]: import sys, os, time
import numpy as np
import matplotlib.pyplot as plt
from importlib import reload
import eureka.S3_data_reduction.s3_reduce as s3
from eureka.lib import readECF as rd
from eureka.lib import logedit
from eureka.lib import readECF as rd
from eureka.lib import manageevent as me
from eureka.S3_data_reduction import optspex
from eureka.lib import astropytable
from eureka.lib import util
from eureka.S3_data_reduction import plots_s3
```

9.2.1 Step 0: Initialization

```
[17]: # Starts timer to monitor how long data reduction takes
t0      = time.time()

# Names the event (has to match the event name used for the *.ecf files)
eventlabel = 'wasp43b'

# Initialize metadata object to store all extra information
# related to the event and the data reduction

meta      = s3.Metadata()
meta.eventlabel = eventlabel

# Initialize data object to store data from the observation
dat       = s3.Data()
```

Try printing how much time has passed since the timer was initialized. Run the cell again. Do you see the time change?

```
[19]: print(time.time() - t0)  #time elapsed since the timer start

7.827232122421265
```

```
[20]: # Load Eureka! control file and store values in Metadata object
ecffile = 'S3_' + eventlabel + '.ecf'
ecf      = rd.read_ecf(ecffile)
rd.store_ecf(meta, ecf)
```

Information from the ECF (“Eureka control file”) is now stored in a Metadata object. This includes all the high level information about the data reduction (which JWST instrument was used? do we want to display plots? where is the data stored? what size is the extraction window? etc.)

To see the current contents of the Metadata object, type `meta.__dict__.keys`.

What is the value of `meta.bg_deg`? Can you change it?

9.2.2 Step 1: Make directories to store reduced data, create log file, read in data

```
[21]: # Create directories for Stage 3 processing
datetime= time.strftime('%Y-%m-%d_%H-%M-%S')
meta.outputdir = 'S3_' + datetime + '_' + meta.eventlabel
if not os.path.exists(meta.outputdir):
    os.makedirs(meta.outputdir)
if not os.path.exists(meta.outputdir+"/figs"):
    os.makedirs(meta.outputdir+"/figs")

# Load instrument module
exec('from eureka.S3_data_reduction import ' + meta.inst + ' as inst', globals())
reload(inst)

# Open new log file
meta.logname = './'+meta.outputdir + '/S3_' + meta.eventlabel + ".log"
log          = logedit.Logedit(meta.logname)
```

(continues on next page)

(continued from previous page)

```
log.writelog("\nStarting Stage 3 Reduction")

# Create list of file segments
meta = util.readfiles(meta)
num_data_files = len(meta.segment_list)
log.writelog(f'\nFound {num_data_files} data file(s) ending in {meta.suffix}.fits')

stdspec = np.array([])
```

Starting Stage 3 Reduction

Found 21 data file(s) ending in calints.fits

Important check! Were the correct files read in? They are stored in `meta.segment_list`.

9.2.3 Step 2: read the data (and look at it!)

```
[22]: # pick a single file to read and reduce as a test
m = 17

# Read in data frame and header
log.writelog(f'Reading file {m+1} of {num_data_files}')
dat = inst.read(meta.segment_list[m], dat, returnHdr=True)
```

Reading file 18 of 21

What data are we using?

The full description of the data is available [here](#)). To quickly summarize, we are using simulated NIRCcam grism time series data from the ERS Simulated Spectra Team. The simulation assumes a WASP-43 b-like planet with physically realistic spectral features added. The simulated data are based on the following observational design:

- GRISMR+F322W2 pupil and filter
- RAPID readout mode
- 19 Groups per integrations
- 1287 integrations
- 1 Exposure
- 4 Output amplifiers The data themselves are divided into “segments,” with each individual segment (seg001, seg002, etc.) containing a subset of the overall dataset. This is how flight data will be delivered. The segments are numbered in their order of observation.

We will use the Stage 2 Output from the [JWST data reduction pipeline](#). For NIRCcam, Stage 2 consists of the flat field correction, WCS/wavelength solution, and photometric calibration (counts/sec -> MJy). Note that this is specifically for NIRCcam: the steps in Stage 2 change a bit depending on the instrument. The Stage 2 outputs are roughly equivalent to a “flt” file from HST.

The files have the suffix `/*calints.fits` which contain fully calibrated images (MJy) for each individual integration. This is the one you want if you’re starting with Stage 2 and want to do your own spectral extraction.

9.2.4 Let's take a look at the data!

What is stored in the data object?

```
[23]: print(dat.__dict__.keys())

dict_keys(['mhdr', 'shdr', 'intstart', 'intend', 'data', 'err', 'dq', 'wave', 'v0', 'int_
↪times'])
```

The calibrated 2D data, error array, data quality are stored in `data`, `err`, and `dq`. `wave` is the wavelength.

The header information is stored in `mhdr` (main header) and `shdr` (science header). Use the headers to check whether the data is really from NIRCcam.

```
[28]: dat.mhdr['INSTRUME']
```

```
[28]: 'NIRCAM'
```

What units are the data stored in?

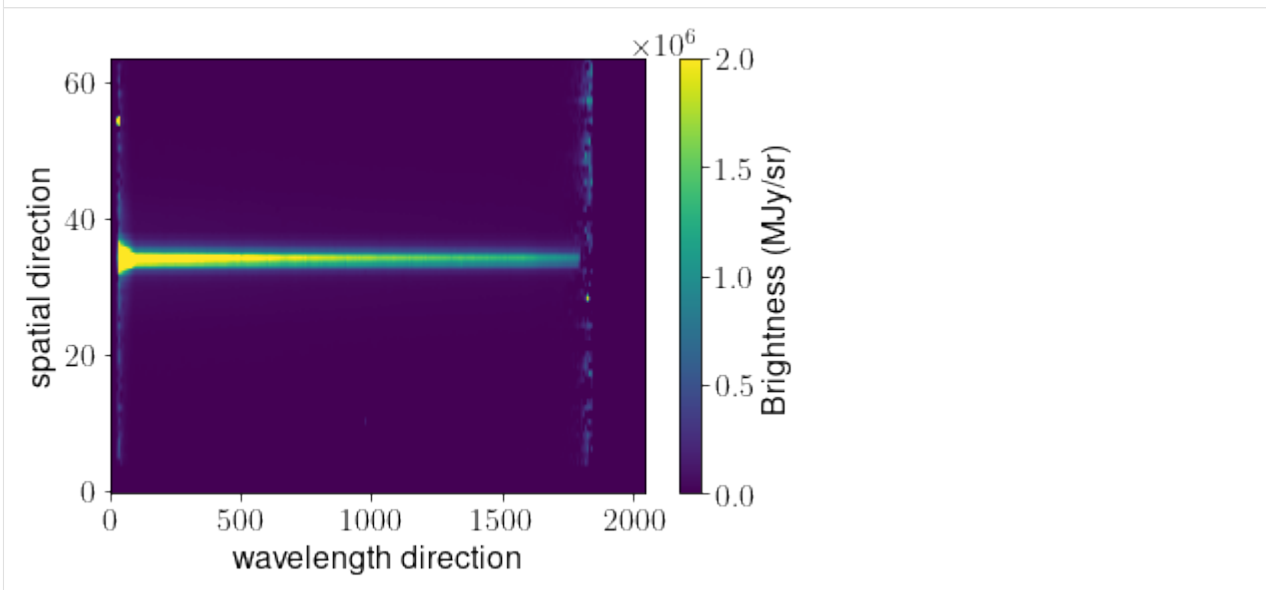
```
[29]: dat.shdr['BUNIT']
```

```
[29]: 'MJy/sr'
```

What does the data look like??

```
[11]: plt.imshow(dat.data[0], origin = 'lower', aspect='auto', vmin=0, vmax=2e6)
ax = plt.gca()
plt.colorbar(label='Brightness (MJy/sr)')
ax.set_xlabel('wavelength direction')
ax.set_ylabel('spatial direction')
```

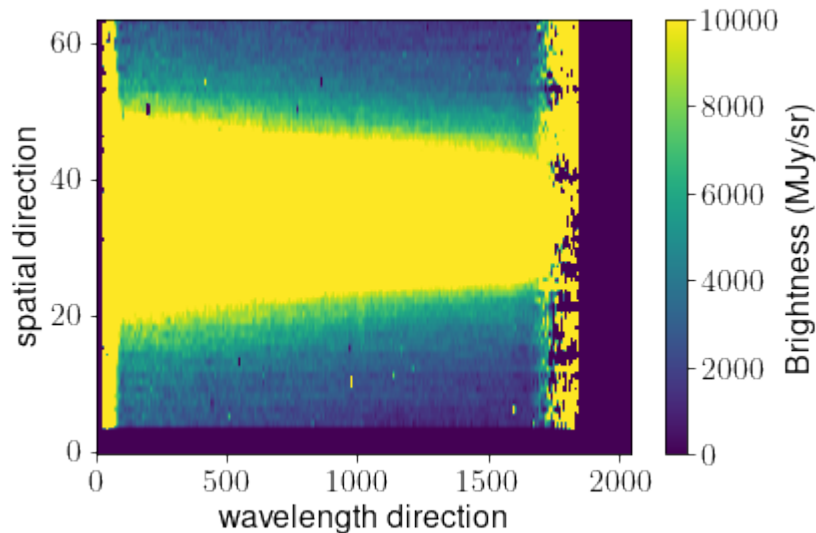
```
[11]: Text(0, 0.5, 'spatial direction')
```



What happens if we change the contrast with the `vmax` parameter? what is the approximate background level?

```
[12]: plt.imshow(dat.data[0], origin = 'lower', aspect='auto', vmin=0, vmax=100000)
      ax = plt.gca()
      plt.colorbar(label='Brightness (MJy/sr)')
      ax.set_xlabel('wavelength direction')
      ax.set_ylabel('spatial direction')
```

```
[12]: Text(0, 0.5, 'spatial direction')
```

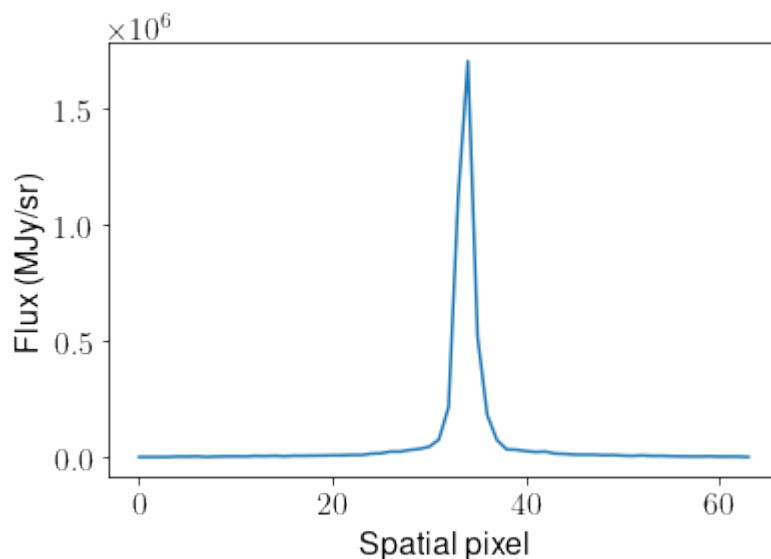


How big should the extraction window be? Should it be symmetric? (Hint: we want to capture all the flux from the target star, but minimize the background)

Let's plot the spatial profile to see how wide the PSF is.

```
[13]: plt.plot(dat.data[0][:,1000]) #plots column 1000
      plt.xlabel("Spatial pixel")
      plt.ylabel("Flux (MJy/sr)")
```

```
[13]: Text(0, 0.5, 'Flux (MJy/sr)')
```

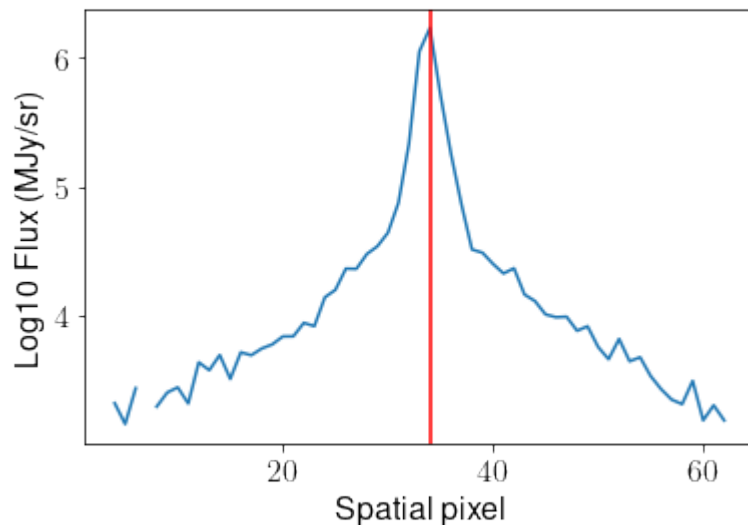


Flux is mostly concentrated over a few pixels. But the wings are pretty wide! This is easier to see in log space:

```
[14]: plt.plot(np.log10(dat.data[0][:,1000])) #plots log10 of column 1000
      ind_max = np.argmax(dat.data[0][:,1000]) #finds row where counts peak
      plt.axvline(ind_max, color = 'red') #plots peak counts
      plt.xlabel("Spatial pixel")
      plt.ylabel("Log10 Flux (MJy/sr)")

<ipython-input-14-fd8aa0f2ba34>:1: RuntimeWarning: divide by zero encountered in log10
  plt.plot(np.log10(dat.data[0][:,1000])) #plots log10 of column 1000
<ipython-input-14-fd8aa0f2ba34>:1: RuntimeWarning: invalid value encountered in log10
  plt.plot(np.log10(dat.data[0][:,1000])) #plots log10 of column 1000

[14]: Text(0, 0.5, 'Log10 Flux (MJy/sr)')
```



9.2.5 Decide which parts we want to use for the background and for the spectrum¶

```
[30]: # Get number of integrations and frame dimensions
      meta.n_int, meta.ny, meta.nx = dat.data.shape

      # Locate source position
      meta.src_xpos = dat.shdr['SRCXPOS']-meta.xwindow[0]
      meta.src_ypos = dat.shdr['SRCYPOS']-meta.ywindow[0]
```

TBC...

```
[ ]:
```


INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

e

- eureka.lib.astropytable, 61
- eureka.lib.centroid, 62
- eureka.lib.clipping, 63
- eureka.lib.disk, 64
- eureka.lib.gaussian, 64
- eureka.lib.gelmanrubin, 71
- eureka.lib.logedit, 72
- eureka.lib.manageevent, 73
- eureka.lib.medstddev, 74
- eureka.lib.plots, 74
- eureka.lib.readECF, 75
- eureka.lib.readEPF, 76
- eureka.lib.smooth, 77
- eureka.lib.smoothing, 78
- eureka.lib.sort_nicely, 78
- eureka.lib.splinterp, 78
- eureka.lib.suntimecorr, 79
- eureka.lib.utc_tt, 79
- eureka.lib.util, 80
- eureka.S3_data_reduction.background, 81
- eureka.S3_data_reduction.bright2flux, 83
- eureka.S3_data_reduction.hst_scan, 86
- eureka.S3_data_reduction.miri, 88
- eureka.S3_data_reduction.nircam, 89
- eureka.S3_data_reduction.niriss, 91
- eureka.S3_data_reduction.niriss_profiles, 90
- eureka.S3_data_reduction.nirspec, 94
- eureka.S3_data_reduction.optspex, 94
- eureka.S3_data_reduction.plots_s3, 97
- eureka.S3_data_reduction.s3_reduce, 99
- eureka.S3_data_reduction.sigrej, 100
- eureka.S3_data_reduction.source_pos, 102
- eureka.S3_data_reduction.wfc3, 104
- eureka.S4_generate_lightcurves.drift, 105
- eureka.S4_generate_lightcurves.plots_s4, 106
- eureka.S4_generate_lightcurves.s4_genLC, 107
- eureka.S5_lightcurve_fitting.fitters, 108
- eureka.S5_lightcurve_fitting.lightcurve, 111
- eureka.S5_lightcurve_fitting.likelihood, 113
- eureka.S5_lightcurve_fitting.modelgrid, 116
- eureka.S5_lightcurve_fitting.plots_s5, 118
- eureka.S5_lightcurve_fitting.s5_fit, 121
- eureka.S5_lightcurve_fitting.simulations, 122
- eureka.S5_lightcurve_fitting.utils, 122
- eureka.S6_planet_spectra.plots_s6, 125
- eureka.S6_planet_spectra.s6_spectra, 125

A

`alphanum_key()` (in module `eureka.lib.sort_nicely`), 78

B

`BGsubtraction()` (in module `reka.S3_data_reduction.background`), 81

`binned_lightcurve()` (in module `reka.S4_generate_lightcurves.plots_s4`), 106

`bright2dn()` (in module `reka.S3_data_reduction.bright2flux`), 83

`bright2flux()` (in module `reka.S3_data_reduction.bright2flux`), 84

`build_target_url()` (in module `reka.S5_lightcurve_fitting.utils`), 122

C

`calc_slitshift()` (in module `reka.S3_data_reduction.hst_scan`), 86

`calc_slitshift2()` (in module `reka.S3_data_reduction.hst_scan`), 86

`calc_zoom()` (in module `reka.S5_lightcurve_fitting.utils`), 122

`calcDrift2D()` (in module `reka.S3_data_reduction.hst_scan`), 86

`calcTrace()` (in module `reka.S3_data_reduction.hst_scan`), 86

`calibrateLambda()` (in module `reka.S3_data_reduction.hst_scan`), 86

`cc_spec()` (in module `reka.S4_generate_lightcurves.plots_s4`), 106

`cc_vals()` (in module `reka.S4_generate_lightcurves.plots_s4`), 106

`check_nans()` (in module `eureka.lib.util`), 80

`clip_outliers()` (in module `eureka.lib.clipping`), 63

`close_log()` (`eureka.lib.logedit.Logedit` method), 72

`color_gen()` (in module `reka.S5_lightcurve_fitting.utils`), 122

`computeRedChiSq()` (in module `reka.S5_lightcurve_fitting.likelihood`), 113

`computeRMS()` (in module `reka.S5_lightcurve_fitting.likelihood`), 113

`conclusion_step()` (in module `reka.S3_data_reduction.wfc3`), 104

`convergetest()` (in module `eureka.lib.gelmanrubin`), 71

`convert_to_e()` (in module `reka.S3_data_reduction.bright2flux`), 84

`copy_ecf()` (`eureka.lib.readECF.MetaClass` method), 75

`correct_drift2D()` (in module `reka.S3_data_reduction.wfc3`), 104

`correct_slitshift2()` (in module `reka.S3_data_reduction.hst_scan`), 86

`ctr_gauss()` (in module `eureka.lib.centroid`), 62

`ctr_guess()` (in module `eureka.lib.centroid`), 62

`customize()` (`eureka.S5_lightcurve_fitting.modelgrid.ModelGrid` method), 117

D

`DataClass` (class in module `reka.S3_data_reduction.s3_reduce`), 99

`demcfitter()` (in module `reka.S5_lightcurve_fitting.fitters`), 108

`difference_frames()` (in module `reka.S3_data_reduction.wfc3`), 104

`disk()` (in module `eureka.lib.disk`), 64

`dn2electrons()` (in module `reka.S3_data_reduction.bright2flux`), 85

`download_exoctl_data()` (in module `reka.S5_lightcurve_fitting.utils`), 123

`drift1d()` (in module `reka.S4_generate_lightcurves.plots_s4`), 106

`drift_fit2D()` (in module `reka.S3_data_reduction.hst_scan`), 86

`dynestyfitter()` (in module `reka.S5_lightcurve_fitting.fitters`), 108

E

`emcee_fitter()` (in module `reka.S5_lightcurve_fitting.fitters`), 109

```

eureka.lib.astropytable
    module, 61
eureka.lib.centroid
    module, 62
eureka.lib.clipping
    module, 63
eureka.lib.disk
    module, 64
eureka.lib.gaussian
    module, 64
eureka.lib.gelmanrubin
    module, 71
eureka.lib.logedit
    module, 72
eureka.lib.manageevent
    module, 73
eureka.lib.medstddev
    module, 74
eureka.lib.plots
    module, 74
eureka.lib.readECF
    module, 75
eureka.lib.readEPF
    module, 76
eureka.lib.smooth
    module, 77
eureka.lib.smoothing
    module, 78
eureka.lib.sort_nicely
    module, 78
eureka.lib.splinterp
    module, 78
eureka.lib.suntimecorr
    module, 79
eureka.lib.utc_tt
    module, 79
eureka.lib.util
    module, 80
eureka.S3_data_reduction.background
    module, 81
eureka.S3_data_reduction.bright2flux
    module, 83
eureka.S3_data_reduction.hst_scan
    module, 86
eureka.S3_data_reduction.miri
    module, 88
eureka.S3_data_reduction.nircam
    module, 89
eureka.S3_data_reduction.niriss
    module, 91
eureka.S3_data_reduction.niriss_profiles
    module, 90
eureka.S3_data_reduction.nirspec
    module, 94
eureka.S3_data_reduction.optspex
    module, 94
eureka.S3_data_reduction.plots_s3
    module, 97
eureka.S3_data_reduction.s3_reduce
    module, 99
eureka.S3_data_reduction.sigrej
    module, 100
eureka.S3_data_reduction.source_pos
    module, 102
eureka.S3_data_reduction.wfc3
    module, 104
eureka.S4_generate_lightcurves.drift
    module, 105
eureka.S4_generate_lightcurves.plots_s4
    module, 106
eureka.S4_generate_lightcurves.s4_genLC
    module, 107
eureka.S5_lightcurve_fitting.fitters
    module, 108
eureka.S5_lightcurve_fitting.lightcurve
    module, 111
eureka.S5_lightcurve_fitting.likelihood
    module, 113
eureka.S5_lightcurve_fitting.modelgrid
    module, 116
eureka.S5_lightcurve_fitting.plots_s5
    module, 118
eureka.S5_lightcurve_fitting.s5_fit
    module, 121
eureka.S5_lightcurve_fitting.simulations
    module, 122
eureka.S5_lightcurve_fitting.utils
    module, 122
eureka.S6_planet_spectra.plots_s6
    module, 125
eureka.S6_planet_spectra.s6_spectra
    module, 125
export() (eureka.S5_lightcurve_fitting.modelgrid.ModelGrid
    method), 117

```

F

```

f277_mask()          (in module eureka.S3_data_reduction.niriss), 91
filter_table()       (in module eureka.S5_lightcurve_fitting.utils), 123
find_closest()       (in module eureka.S5_lightcurve_fitting.utils), 123
fit() (eureka.S5_lightcurve_fitting.lightcurve.LightCurve
    method), 112
fit_bg() (in module eureka.S3_data_reduction.miri), 88
fit_bg() (in module eureka.S3_data_reduction.nircam), 89

```

- `fit_bg()` (in module *eureka.S3_data_reduction.niriss*), 91
- `fit_bg()` (in module *eureka.S3_data_reduction.nirspec*), 94
- `fit_bg()` (in module *eureka.S3_data_reduction.wfc3*), 104
- `fit_channel()` (in module *eureka.S5_lightcurve_fitting.s5_fit*), 121
- `fit_orders()` (in module *eureka.S3_data_reduction.niriss*), 91
- `fit_orders_fast()` (in module *eureka.S3_data_reduction.niriss*), 92
- `fitbg()` (in module *eureka.S3_data_reduction.background*), 82
- `fitbg2()` (in module *eureka.S3_data_reduction.background*), 82
- `fitbg3()` (in module *eureka.S3_data_reduction.background*), 83
- `fitgaussian()` (in module *eureka.lib.gaussian*), 64
- `fitgaussians()` (in module *eureka.lib.gaussian*), 68
- `fitJWST()` (in module *eureka.S5_lightcurve_fitting.s5_fit*), 121
- `flag_bg()` (in module *eureka.S3_data_reduction.miri*), 88
- `flag_bg()` (in module *eureka.S3_data_reduction.nircam*), 89
- `flag_bg()` (in module *eureka.S3_data_reduction.nirspec*), 94
- `flag_bg()` (in module *eureka.S3_data_reduction.wfc3*), 104
- `flatfield()` (in module *eureka.S3_data_reduction.wfc3*), 104
- ## G
- `gauss()` (in module *eureka.S3_data_reduction.source_pos*), 102
- `gauss_kernel_mask2()` (in module *eureka.lib.smoothing*), 78
- `gauss_removal()` (in module *eureka.lib.clipping*), 63
- `gaussian()` (in module *eureka.lib.gaussian*), 68
- `gaussian_1poly_piecewise()` (in module *eureka.S3_data_reduction.niriss_profiles*), 90
- `gaussian_2poly_piecewise()` (in module *eureka.S3_data_reduction.niriss_profiles*), 90
- `gaussianguess()` (in module *eureka.lib.gaussian*), 70
- `gaussians()` (in module *eureka.lib.gaussian*), 70
- `gelmanrubin()` (in module *eureka.lib.gelmanrubin*), 72
- `generalized_normal()` (in module *eureka.S3_data_reduction.niriss_profiles*), 90
- `get()` (*eureka.S5_lightcurve_fitting.modelgrid.ModelGrid* method), 117
- `get_canonical_name()` (in module *eureka.S5_lightcurve_fitting.utils*), 123
- `get_env_variables()` (in module *eureka.S5_lightcurve_fitting.utils*), 123
- `get_mad()` (in module *eureka.lib.util*), 80
- `get_target_data()` (in module *eureka.S5_lightcurve_fitting.utils*), 124
- `getcoords()` (in module *eureka.lib.suntimecorr*), 79
- `GP_loglikelihood()` (in module *eureka.S5_lightcurve_fitting.likelihood*), 113
- `grid_interp()` (*eureka.S5_lightcurve_fitting.modelgrid.ModelGrid* method), 117
- `group_variables()` (in module *eureka.S5_lightcurve_fitting.fitters*), 109
- `group_variables_lmfit()` (in module *eureka.S5_lightcurve_fitting.fitters*), 110
- `groupFrames()` (in module *eureka.S3_data_reduction.hst_scan*), 87
- ## H
- `highpassfilt()` (in module *eureka.S4_generate_lightcurves.drift*), 105
- ## I
- `image_and_background()` (in module *eureka.S3_data_reduction.plots_s3*), 97
- `image_filtering()` (in module *eureka.S3_data_reduction.niriss*), 92
- `imageCentroid()` (in module *eureka.S3_data_reduction.hst_scan*), 87
- `info()` (*eureka.S5_lightcurve_fitting.modelgrid.ModelGrid* method), 118
- `initialize_emcee_walkers()` (in module *eureka.S5_lightcurve_fitting.fitters*), 110
- `interp_flux()` (in module *eureka.S5_lightcurve_fitting.utils*), 124
- ## L
- `lc_driftcorr()` (in module *eureka.S4_generate_lightcurves.plots_s4*), 107
- `lc_nodriftcorr()` (in module *eureka.S3_data_reduction.plots_s3*), 97
- `lcJWST()` (in module *eureka.S4_generate_lightcurves.s4_genLC*), 107
- `leapdates()` (in module *eureka.lib.utc_tt*), 79
- `leapseconds()` (in module *eureka.lib.utc_tt*), 79
- `LightCurve` (class in *eureka.S5_lightcurve_fitting.lightcurve*), 111
- `lmfitter()` (in module *eureka.S5_lightcurve_fitting.fitters*), 110
- `ln_like()` (in module *eureka.S5_lightcurve_fitting.likelihood*), 114
- `lnprior()` (in module *eureka.S5_lightcurve_fitting.likelihood*), 114

`lnprob()` (in module *eureka.S5_lightcurve_fitting.likelihood*), 115
`load_flux()` (*eureka.S5_lightcurve_fitting.modelgrid.ModelGrid* method), 118
`load_general_s2_meta_info()` (in module *eureka.S3_data_reduction.s3_reduce*), 99
`load_general_s3_meta_info()` (in module *eureka.S4_generate_lightcurves.s4_genLC*), 107
`load_general_s4_meta_info()` (in module *eureka.S5_lightcurve_fitting.s5_fit*), 121
`load_general_s5_meta_info()` (in module *eureka.S6_planet_spectra.s6_spectra*), 125
`load_old_fitparams()` (in module *eureka.S5_lightcurve_fitting.fitters*), 111
`load_specific_s3_meta_info()` (in module *eureka.S4_generate_lightcurves.s4_genLC*), 107
`load_specific_s4_meta_info()` (in module *eureka.S5_lightcurve_fitting.s5_fit*), 121
`load_specific_s5_meta_info()` (in module *eureka.S6_planet_spectra.s6_spectra*), 125
`loadevent()` (in module *eureka.lib.manageevent*), 73
`Logedit` (class in *eureka.lib.logedit*), 72
`lsqfitter()` (in module *eureka.S5_lightcurve_fitting.fitters*), 111

M

`make_longparamlist()` (in module *eureka.S5_lightcurve_fitting.s5_fit*), 122
`makeBasicFlats()` (in module *eureka.S3_data_reduction.hst_scan*), 87
`makedirectory()` (in module *eureka.lib.util*), 80
`makeflats()` (in module *eureka.S3_data_reduction.hst_scan*), 88
`mask_method_one()` (in module *eureka.S3_data_reduction.niriss*), 92
`mask_method_two()` (in module *eureka.S3_data_reduction.niriss*), 93
`medfilt()` (in module *eureka.lib.smooth*), 77
`medstddev()` (in module *eureka.lib.medstddev*), 74
`MetaClass` (class in *eureka.lib.readECF*), 75
`MetaClass` (class in *eureka.S3_data_reduction.s3_reduce*), 99
`MetaClass` (class in *eureka.S4_generate_lightcurves.s4_genLC*), 107
`MetaClass` (class in *eureka.S5_lightcurve_fitting.s5_fit*), 121
`MetaClass` (class in *eureka.S6_planet_spectra.s6_spectra*), 125
`ModelGrid` (class in *eureka.S5_lightcurve_fitting.modelgrid*), 116
module

eureka.lib.astropytable, 61
eureka.lib.centroid, 62
eureka.lib.clipping, 63
eureka.lib.disk, 64
eureka.lib.gaussian, 64
eureka.lib.gelmanrubin, 71
eureka.lib.logedit, 72
eureka.lib.manageevent, 73
eureka.lib.medstddev, 74
eureka.lib.plots, 74
eureka.lib.readECF, 75
eureka.lib.readEPF, 76
eureka.lib.smooth, 77
eureka.lib.smoothing, 78
eureka.lib.sort_nicely, 78
eureka.lib.splinterp, 78
eureka.lib.suntimecorr, 79
eureka.lib.utc_tt, 79
eureka.lib.util, 80
eureka.S3_data_reduction.background, 81
eureka.S3_data_reduction.bright2flux, 83
eureka.S3_data_reduction.hst_scan, 86
eureka.S3_data_reduction.miri, 88
eureka.S3_data_reduction.nircam, 89
eureka.S3_data_reduction.niriss, 91
eureka.S3_data_reduction.niriss_profiles, 90
eureka.S3_data_reduction.nirspec, 94
eureka.S3_data_reduction.optspex, 94
eureka.S3_data_reduction.plots_s3, 97
eureka.S3_data_reduction.s3_reduce, 99
eureka.S3_data_reduction.sigrej, 100
eureka.S3_data_reduction.source_pos, 102
eureka.S3_data_reduction.wfc3, 104
eureka.S4_generate_lightcurves.drift, 105
eureka.S4_generate_lightcurves.plots_s4, 106
eureka.S4_generate_lightcurves.s4_genLC, 107
eureka.S5_lightcurve_fitting.fitters, 108
eureka.S5_lightcurve_fitting.lightcurve, 111
eureka.S5_lightcurve_fitting.likelihood, 113
eureka.S5_lightcurve_fitting.modelgrid, 116
eureka.S5_lightcurve_fitting.plots_s5, 118
eureka.S5_lightcurve_fitting.s5_fit, 121
eureka.S5_lightcurve_fitting.simulations, 122
eureka.S5_lightcurve_fitting.utils, 122
eureka.S6_planet_spectra.plots_s6, 125
eureka.S6_planet_spectra.s6_spectra, 125

`moffat_1poly_pieewise()` (in module `reka.S3_data_reduction.niriss_profiles`), 91

`moffat_2poly_pieewise()` (in module `reka.S3_data_reduction.niriss_profiles`), 91

O

`optimal_spectrum()` (in module `reka.S3_data_reduction.plots_s3`), 97

`optimize()` (in module `reka.S3_data_reduction.optspex`), 94

P

`Parameter` (class in `eureka.lib.readEPF`), 76

`Parameters` (class in `eureka.lib.readEPF`), 76

`parse_s5_saves()` (in module `reka.S6_planet_spectra.s6_spectra`), 125

`pathdirectory()` (in module `eureka.lib.util`), 80

`plot()` (`eureka.S5_lightcurve_fitting.lightcurve.LightCurve` method), 112

`plot_chain()` (in module `reka.S5_lightcurve_fitting.plots_s5`), 119

`plot_corner()` (in module `reka.S5_lightcurve_fitting.plots_s5`), 119

`plot_fit()` (in module `reka.S5_lightcurve_fitting.plots_s5`), 120

`plot_GP_components()` (in module `reka.S5_lightcurve_fitting.plots_s5`), 118

`plot_res_distr()` (in module `reka.S5_lightcurve_fitting.plots_s5`), 120

`plot_rms()` (in module `reka.S5_lightcurve_fitting.plots_s5`), 120

`plot_spectra()` (in module `reka.S6_planet_spectra.s6_spectra`), 125

`plot_spectrum()` (in module `reka.S6_planet_spectra.plots_s6`), 125

`preparation_step()` (in module `reka.S3_data_reduction.wfc3`), 104

`profile()` (in module `reka.S3_data_reduction.plots_s3`), 98

`profile_gauss()` (in module `reka.S3_data_reduction.optspex`), 95

`profile_meddata()` (in module `reka.S3_data_reduction.optspex`), 95

`profile_poly()` (in module `reka.S3_data_reduction.optspex`), 96

`profile_smooth()` (in module `reka.S3_data_reduction.optspex`), 96

`profile_wavelet()` (in module `reka.S3_data_reduction.optspex`), 96

`profile_wavelet2D()` (in module `reka.S3_data_reduction.optspex`), 97

`ptform()` (in module `reka.S5_lightcurve_fitting.likelihood`), 115

`pptype` (`eureka.lib.readEPF.Parameter` property), 76

R

`rate2count()` (in module `reka.S3_data_reduction.bright2flux`), 85

`read()` (`eureka.lib.readECF.MetaClass` method), 75

`read()` (`eureka.lib.readEPF.Parameters` method), 77

`read()` (in module `eureka.S3_data_reduction.miri`), 88

`read()` (in module `eureka.S3_data_reduction.nircam`), 89

`read()` (in module `eureka.S3_data_reduction.niriss`), 93

`read()` (in module `eureka.S3_data_reduction.nirspec`), 94

`read()` (in module `eureka.S3_data_reduction.wfc3`), 104

`read_s2_meta()` (in module `reka.S3_data_reduction.s3_reduce`), 99

`read_s3_meta()` (in module `reka.S4_generate_lightcurves.s4_genLC`), 107

`read_s4_meta()` (in module `reka.S5_lightcurve_fitting.s5_fit`), 122

`read_s5_meta()` (in module `reka.S6_planet_spectra.s6_spectra`), 125

`readfiles()` (in module `eureka.lib.util`), 81

`readtable()` (in module `eureka.lib.astropytable`), 61

`rebin_spec()` (in module `reka.S5_lightcurve_fitting.utils`), 124

`reduceJWST()` (in module `reka.S3_data_reduction.s3_reduce`), 100

`replacePixels()` (in module `reka.S3_data_reduction.hst_scan`), 88

`reset()` (`eureka.S5_lightcurve_fitting.lightcurve.LightCurve` method), 112

`reset()` (`eureka.S5_lightcurve_fitting.modelgrid.ModelGrid` method), 118

`resids()` (in module `eureka.lib.gaussian`), 70

`residuals()` (in module `eureka.lib.gaussian`), 70

`retrieve_ancil()` (in module `reka.S3_data_reduction.bright2flux`), 85

S

`save_fit()` (in module `reka.S5_lightcurve_fitting.fitters`), 111

`saveevent()` (in module `eureka.lib.manageevent`), 73

`savetable_S3()` (in module `eureka.lib.astropytable`), 61

`savetable_S4()` (in module `eureka.lib.astropytable`), 61

`savetable_S5()` (in module `eureka.lib.astropytable`), 61

`savetable_S6()` (in module `eureka.lib.astropytable`), 61

`separate_direct()` (in module `reka.S3_data_reduction.wfc3`), 105

`separate_scan_direction()` (in module `reka.S3_data_reduction.wfc3`), 105

set_rc() (in module *eureka.lib.plots*), 74
 set_units() (*eureka.S5_lightcurve_fitting.modelgrid.ModelGrid* method), 118
 sigrej() (in module *eureka.S3_data_reduction.sigrej*), 100
 simplify_niriss_img() (in module *eureka.S3_data_reduction.niriss*), 93
 simulate_lightcurve() (in module *eureka.S5_lightcurve_fitting.simulations*), 122
 smooth() (in module *eureka.lib.smooth*), 77
 sort_nicely() (in module *eureka.lib.sort_nicely*), 78
 source_pos() (in module *eureka.S3_data_reduction.source_pos*), 102
 source_pos_FWM() (in module *eureka.S3_data_reduction.source_pos*), 102
 source_pos_gauss() (in module *eureka.S3_data_reduction.source_pos*), 103
 source_pos_max() (in module *eureka.S3_data_reduction.source_pos*), 103
 source_position() (in module *eureka.S3_data_reduction.plots_s3*), 98
 spec1D() (in module *eureka.S4_generate_lightcurves.drift*), 105
 splinterp() (in module *eureka.lib.splinterp*), 78
 start_from_oldchain_emcee() (in module *eureka.S5_lightcurve_fitting.fitters*), 111
 suntimecorr() (in module *eureka.lib.suntimecorr*), 79

T

transform_log_uniform() (in module *eureka.S5_lightcurve_fitting.likelihood*), 116
 transform_normal() (in module *eureka.S5_lightcurve_fitting.likelihood*), 116
 transform_uniform() (in module *eureka.S5_lightcurve_fitting.likelihood*), 116
 trim() (in module *eureka.lib.util*), 81
 tryint() (in module *eureka.lib.sort_nicely*), 78

U

updateevent() (in module *eureka.lib.manageevent*), 73
 utc_tdb() (in module *eureka.lib.utc_tt*), 79
 utc_tt() (in module *eureka.lib.utc_tt*), 79

V

values (*eureka.lib.readEPF.Parameter* property), 76

W

wave_MIRI_hardcoded() (in module *eureka.S3_data_reduction.miri*), 89
 wave_NIRISS() (in module *eureka.S3_data_reduction.niriss*), 93
 write() (*eureka.lib.readECF.MetaClass* method), 76
 write() (*eureka.lib.readEPF.Parameters* method), 77
 writeclose() (*eureka.lib.logedit.Logedit* method), 72
 writeFITS() (in module *eureka.S5_lightcurve_fitting.utils*), 124
 writelog() (*eureka.lib.logedit.Logedit* method), 72