ExoTiC-MIRI

Release 1.0.0

David Grant

Aug 03, 2023

CONTENTS

Custom steps for JWST MIRI LRS data reduction from raw data to light curves. Interoperable with the STScI pipeline.

The Space Telescope pipeline for JWST defines two main stages for the processing of raw observational data to a time series of 1D spectra.

- Stage 1: starts from the _uncal.fits files and performs basic detector-level corrections at the group level. This is followed by ramp fitting to make _rateints.fits files, or rate-images. The dimensions of the data are transformed from [ints, groups, rows, cols] to [ints, rows, cols].
- Stage 2: picks up the _rateints.fits files and performs mode-specific corrections on these rate-images. This is followed by extraction of the spectra to make light curves. The dimensions of the data are transformed from [ints, rows, cols] to [ints, wavelength].

In this package, we make available custom steps that can be swapped in and out of both stage 1 and stage 2 data processing. The custom steps provided are built specifically for reducing time-series observations from the Mid-Infrared Instrument's low resolution spectrometer (MIRI LRS). This mode is of particular use to transiting exoplanet observations, and as such the algorithms are designed with precise relative fluxes in mind.

CHAPTER

ONE

INSTALLATION

ExoTiC-MIRI is installed with pip (python >=3.8 is required). First, install the JWST pipeline version you wish to interoperate with (compatible with 1.8.2<= jwst <= x.x.x), and then install ExoTiC-MIRI:

```
pip install jwst==1.8.2
pip install exotic-miri
```

Alternatively, you can clone the repository from GitHub.

CHAPTER

TWO

QUICK START

After installing the code (see *installation*) you are ready to reduce some data. Below is a minimal demo of how the ExoTiC-MIRI step may be interleaved with the default JWST pipeline steps.

First, set your CRDS config and then import the JWST pipeline. Note that the CRDS config must be set before importing the JWST pipeline.

```
import os

os.environ["CRDS_SERVER_URL"] = "https://jwst-crds.stsci.edu"
os.environ["CRDS_PATH"] = "path/to/your/local/crds_dir"
os.environ["CRDS_CONTEXT"] = "jwst_1100.pmap"

from jwst import datamodels
from jwst.pipeline import calwebb_detector1
from jwst.pipeline import calwebb_spec2


stsci_dark_current = calwebb_detector1.dark_current_step.DarkCurrentStep()
stsci_jump = calwebb_detector1.jump_step.JumpStep()
stsci_ramp_fit = calwebb_detector1.ramp_fit_step.RampFitStep()
```

Next, import the ExoTiC-MIRI steps you wish to implement. In this example, let's say you want additional functionality to mask certain groups before jump detection and ramp fitting.

```
from exotic_miri.stage_1 import DropGroupsStep


custom_drop_groups = DropGroupsStep()
```

You may now simply apply a mixture of default and custom steps to your data. The ExoTiC-MIRI steps work seamlessly with the default steps.

```
proc = datamodels.RampModel("path/to/your/data/jwxyz-segnnn_mirimage_uncal.fits")

proc = stsci_dark_current.call(proc)
proc = custom_drop_groups.call(proc, drop_groups=[0, 1, 2, 3, 4, -1])  # <-- custom step
proc = stsci_jump.call(proc)
_, proc = stsci_ramp_fit.call(proc)
```

That's it. For more information on the custom steps available, please see the *API*. There are also *complete pipelines* which have been employed on real JWST datasets which you may find helpful.

CHAPTER

THREE

STAGE 1 STEPS

The stage 1 steps perform basic detector-level corrections at the group level, followed by ramp fitting. Below we detail the workings of several steps that you may find useful for improving the quality of your reduction beyond the default JWST stage 1. For a complete list of ExoTiC-MIRI steps and their workings, see the *API*.

The following examples assume you have set up the pipelines and loaded in an _uncal.fits data segment. For example:

```
import os

os.environ["CRDS_SERVER_URL"] = "https://jwst-crds.stsci.edu"
os.environ["CRDS_PATH"] = "path/to/your/local/crds_dir"
os.environ["CRDS_CONTEXT"] = "jwst_1100.pmap"

from jwst import datamodels
from jwst.pipeline import calwebb_detector1

# Load data segment.
proc = datamodels.RampModel("path/to/your/data/jwxyz-segnnn_mirimage_uncal.fits")
```

3.1 Drop groups

The MIRI detector is adversely affected by several effects such as Reset Switch Charge Decay and dragging down of the final frame. As such, you may wish to mask certain groups from being included in subsequent steps, such as jump detection or ramp fitting. This can be achieved with *DropGroupsStep*.

```
from exotic_miri.stage_1 import DropGroupsStep


custom_drop_groups = DropGroupsStep()
proc = custom_drop_groups.call(proc, drop_groups=[0, 1, 2, 3, 4, -1])
```

Here, we have masked the first 5 groups and the final group. The number of groups to mask will depend primarily on the brightness of your target, but in general at least the final group should always be masked (assuming FASTR1 readout mode).

3.2 Group-level background subtraction

If you would like to experiment with subtracting the background/sky at the group level, to clean up the data before ramp fitting, then you can make use of *GroupBackgroundSubtractStep*.

```
from exotic_miri.stage_1 import GroupBackgroundSubtractStep


custom_group_bkg_subtract = GroupBackgroundSubtractStep()
proc = custom_group_bkg_subtract.call(proc, method="row_wise")
```

Here, we have used the default background regions either side of the spectral trace and applied a row-wise background subtraction. You can adjust the method and background region as required.

3.3 Custom gain

The default gain value for MIRI may not be the most accurate. This can have effects on other steps which require the gain to estimate statistical information. You can create a custom gain model with *SetCustomGain*.

```
from exotic_miri.reference import SetCustomGain


custom_set_gain = SetCustomGain()
stsci_jump = calwebb_detector1.jump_step.JumpStep()
stsci_ramp_fit = calwebb_detector1.ramp_fit_step.RampFitStep()
stsci_gain_scale = calwebb_detector1.gain_scale_step.GainScaleStep()

gain_model = custom_set_gain.call(proc, gain_value=3.1)
proc = stsci_jump.call(proc, override_gain=gain_model)
_, proc = stsci_ramp_fit.call(proc, override_gain=gain_model)
proc = stsci_gain_scale.call(proc, override_gain=gain_model)
```

Here we have created a custom gain model with a value of 3.1 (see Bell et al. 2023) and then passed this to the jump detection, ramp fitting, and gain scale steps.

3.4 Custom linearity correction

Determining a linearity correction, the model which accounts for the decrease in gain as pixels become increasingly full, is challenging for MIRI given all the nuances to this Si:As detector. It may be worth generating a custom linearity correction which is self-calibrated from your dataset, if you have a sufficient number of groups, using *SetCustomLinearity*.

```
from exotic_miri.reference import SetCustomLinearity


custom_set_linearity = SetCustomLinearity()
stsci_linearity = calwebb_detector1.linearity_step.LinearityStep()

linearity_model = custom_linearity.call(proc, group_idx_start_fit=5, group_idx_end_
↪fit=40,
                                        group_idx_start_derive=5, group_idx_end_
```

(continues on next page)

(continued from previous page)

```
↪derive=100,
                                        row_idx_start_used=350, row_idx_end_used=386)
proc = stsci_linearity.call(proc, override_linearity=linearity_model)
```

This correction involves extrapolating a linear fit to an assumed linear, or well-behaved section of the ramps. In this case, this is between groups 5 and 40. A polynomial is then fit to the ramps for data between groups 5 and 100 and for rows 350 to 386. The polynomial has the constant- and linear-term coefficients fixed at 0 and 1, respectively. This polynomial may then serve as the correction for all ramps in your dataset.

CHAPTER

FOUR

STAGE 2 STEPS

The stage 2 steps perform mode-specific corrections on the rate-images, followed by spectral extraction. Below we detail the workings of several steps that you may find useful for improving the quality of your reduction beyond the default JWST stage 2. For a complete list of ExoTiC-MIRI steps and their workings, see the *API*.

The following examples assume you have setup the pipelines and loaded in a _rateints.fits data segment. For example:

```
import os

os.environ["CRDS_SERVER_URL"] = "https://jwst-crds.stsci.edu"
os.environ["CRDS_PATH"] = "path/to/your/local/crds_dir"
os.environ["CRDS_CONTEXT"] = "jwst_1100.pmap"

from jwst import datamodels
from jwst.pipeline import calwebb_spec2

# Load data segment.
proc = datamodels.CubeModel("path/to/your/data/jwxyz-segnnn_mirimage_rateints.fits")
```

4.1 Get the wavelength map

To get the MIRI LRS wavelength map you can run *GetWavelengthMap*. Note that you must run jwst.calwebb_spec2.assign_wcs_step and jwst.calwebb_spec2.srctype_step before this step.

```
from exotic_miri.reference import GetWavelengthMap


custom_get_wavelength_map = GetWavelengthMap()
stsci_assign_wcs = calwebb_spec2.assign_wcs_step.AssignWcsStep()
stsci_srctype = calwebb_spec2.srctype_step.SourceTypeStep()

proc = stsci_srctype.call(proc)
sproc = stsci_assign_wcs.call(proc)
wavelength_map = custom_get_wavelength_map.call(proc)
```

Here we have generated a wavelength_map which represent the mapping from detector pixels (row_idx, col_idx) to wavelength (lambda).

4.2 Outlier cleaning

If outliers remain in your rate-images, then this step offers a method for cleaning them without having to use any time-domain information, and proceeds frame-by-frame. Outlier cleaning is performed by finding deviations from an estimated spatial profile of the spectral trace, following Horne 1986. Outliers can also be optionally found by specifying specific data quality flags you wish to expunge. See *CleanOutliersStep*.

```
from exotic_miri.stage_2 import CleanOutliersStep


custom_clean_outliers = CleanOutliersStep()
proc, P, O = custom_clean_outliers.call(proc, dq_bits_to_mask=[0, 11],
                                        window_heights=[40],
                                        poly_order=4, outlier_threshold=5.0)
```

Here, a spatial profile is constructed from fourth-order polynomials, using windows 40 pixels long in the dispersion direction. Outlying pixels are replaced if their values are >5 sigma from the spatial profile or if they have data quality flags 2**0 (do_not_use) or 2**11 (hot pixel). See the data quality flags table in the docs.

Also returned is a 3D array of the fitted spatial profiles, and a count of the number of outliers cleaned within 0-4 pixels of the spectral trace (column index 36). You can try setting draw_cleaning_col=True to make some interactive plots and get a better feel for the cleaning process, and help tailor the parameters to your dataset.

4.3 Background subtraction

To perform background subtraction from your rate-images, you can make use of *BackgroundSubtractStep*.

```
from exotic_miri.stage_2 import BackgroundSubtractStep


custom_bkg_subtract = BackgroundSubtractStep()
proc = custom_bkg_subtract.call(proc, method="row_wise")
```

Here, we have used the default background regions either side of the spectral trace and applied a row-wise background subtraction. There more options for estimating the background as a linear function of detector column, or for smoothing over the background. You can adjust also background region as required. NB. the constant method is not recommended for MIRI LRS data.

4.4 Extract 1D spectra

To extract a time-series of 1D stellar spectra, using a box aperture, you can make use of *Extract1DBoxStep*.

```
from exotic_miri.stage_2 import Extract1DBoxStep


custom_extract1d_box = Extract1DBoxStep()
wv, spec, spec_unc, trace_sigmas = custom_extract1d_box.call(
    proc, wavelength_map, trace_position=36,
    aperture_center=36, aperture_left_width=4, aperture_right_width=4)
```

Here, a box aperture (top-hat function) is centred on column 36 (nominal for MIRI LRS) and extends 4 pixels in each direction. The total aperture is therefore 9 pixels wide. Note that you must have run the GetWavelengthMap step, so that you may pass the wavelength map as an input. Note that this step returns four outputs: the wavelengths, the time-series spectra, the uncertainties, and a measure of the point source function (PSF) widths.

ExoTiC-MIRI also has an implementation of optimal extraction (Horne 1986). See *Extract1DOptimalStep* for details.

4.5 Align spectra

Often the pointing stability corresponds to the flux stability in your light curves. The x and y position of the spectral trace through time may be used as a diagnostic or decorrelator. To measure the positions, and optionally re-align the spectra, you can use *AlignSpectraStep*.

```
from exotic_miri.stage_2 import AlignSpectraStep


custom_align_spectra = AlignSpectraStep()
spec, spec_unc, x_shifts, y_shifts = custom_align_spectra.call(
    proc, spec, spec_unc, align_spectra=False)
```

Note that this step requires the outputs from Extract1DBoxStep or Extract1DOptimalStep as inputs.

CHAPTER

FIVE

COMPLETE PIPELINES

Here we provide complete pipelines that have been applied on real JWST datasets, to help fast-track your ExoTiC-MIRI implementation. Some of the steps in the pipeline depend on the target, mainly the brightness, and therefore the number of groups. Below we show a couple of pipeline options for datasets with more or less than 40 groups, although please note that this number is somewhat experimental at this time.

To get these up and running, you will have to take a look through the code and update the paths specific to your machine, as well as any of the args specific to the number of groups etc. The processed data is written out at the end of stage 1, as rate-image fits files, and at the end of stage 2, as an xarray.

5.1 For observations with <40 groups

Here is an example pipeline for bright targets, e.g., 7 groups.

```
import os
import numpy as np
import xarray as xr

os.environ["CRDS_SERVER_URL"] = "https://jwst-crds.stsci.edu"
os.environ["CRDS_PATH"] = "path/to/your/local/crds_dir"
os.environ["CRDS_CONTEXT"] = "jwst_1100.pmap"

from jwst import datamodels
from jwst.pipeline import calwebb_detector1, calwebb_spec2

from exotic_miri.reference import SetCustomGain, GetWavelengthMap
from exotic_miri.stage_1 import DropGroupsStep
from exotic_miri.stage_2 import CleanOutliersStep, BackgroundSubtractStep, \
    Extract1DBoxStep, AlignSpectraStep


# Data and reduction config.
seg_names = ["jwxyz-seg001_mirimage", "jwxyz-seg002_mirimage", "jwxyz-seg003_mirimage",
             "jwxyz-seg004_mirimage", "jwxyz-seg005_mirimage", "jwxyz-seg006_mirimage"]
data_dir = "/path/to/your/uncal_data_dir"
reduction_dir = "/path/to/your/reduction_dir"
stage_1_dir = os.path.join(reduction_dir, "stage_1")
stage_2_dir = os.path.join(reduction_dir, "stage_2")
for _dir in [reduction_dir, stage_1_dir, stage_2_dir]:
    if not os.path.exists(_dir):
```

(continues on next page)

(continued from previous page)

```
        os.mkdir(_dir)

# Instantiate STScI steps for stage 1.
stsci_group_scale = calwebb_detector1.group_scale_step.GroupScaleStep()
stsci_dq_init = calwebb_detector1.dq_init_step.DQInitStep()
stsci_saturation = calwebb_detector1.saturation_step.SaturationStep()
stsci_reset = calwebb_detector1.reset_step.ResetStep()
stsci_linearity = calwebb_detector1.linearity_step.LinearityStep()
stsci_dark_current = calwebb_detector1.dark_current_step.DarkCurrentStep()
stsci_jump = calwebb_detector1.jump_step.JumpStep()
stsci_ramp_fit = calwebb_detector1.ramp_fit_step.RampFitStep()
stsci_gain_scale = calwebb_detector1.gain_scale_step.GainScaleStep()

# Instantiate custom steps for stage 1.
custom_set_gain = SetCustomGain()
custom_drop_groups = DropGroupsStep()

# Instantiate STScI steps for stage 2.
stsci_assign_wcs = calwebb_spec2.assign_wcs_step.AssignWcsStep()
stsci_srctype = calwebb_spec2.srctype_step.SourceTypeStep()
stsci_flat_field = calwebb_spec2.flat_field_step.FlatFieldStep()

# Instantiate custom steps for stage 2.
custom_get_wavelength_map = GetWavelengthMap()
custom_clean_outliers = CleanOutliersStep()
custom_background_subtract = BackgroundSubtractStep()
custom_extract1d_box = Extract1DBoxStep()
custom_align_spectra = AlignSpectraStep()

# Make custom gain datamodel (using the final segment).
uncal_last = datamodels.RampModel(os.path.join(data_dir, "{}_uncal.fits".format(seg_
↪names[-1])))
gain_model = custom_set_gain.call(uncal_last, gain_value=3.1)
del uncal_last

# Iterate data chunks.
all_spec = []
all_spec_unc = []
all_bkg = []
all_x_shifts = []
all_y_shifts = []
all_trace_sigmas = []
all_outliers = []
for seg in seg_names:
    print("\n========= Working on {} =========\n".format(seg))

    # Read in segment.
    proc = datamodels.RampModel(os.path.join(data_dir, "{}_uncal.fits".format(seg)))

    # Stage 1 reduction.
    proc = stsci_group_scale.call(proc)
    proc = stsci_dq_init.call(proc)
```

(continues on next page)

(continued from previous page)

```
    proc = stsci_saturation.call(proc, n_pix_grow_sat=1)
    proc = stsci_reset.call(proc)
    proc = custom_drop_groups.call(proc, drop_groups=[6])
    proc = stsci_linearity.call(proc)
    proc = stsci_dark_current.call(proc)
    proc = stsci_jump.call(
        proc,
        rejection_threshold=15.,
        four_group_rejection_threshold=15.,
        three_group_rejection_threshold=15.,
        flag_4_neighbors=False,
        min_jump_to_flag_neighbors=15.,
        expand_large_events=False,
        skip=False, override_gain=gain_model)
    _, proc = stsci_ramp_fit.call(proc, override_gain=gain_model)
    proc = stsci_gain_scale.call(proc, override_gain=gain_model)
    proc.save(path=os.path.join(stage_1_dir, "{}_rateints.fits".format(seg)))

    # Stage 2 reduction, part 1: auxiliary data.
    proc = stsci_assign_wcs.call(proc)
    proc = stsci_srctype.call(proc)
    wavelength_map = custom_get_wavelength_map.call(proc)

    # Stage 2 reduction, part 2: cleaning.
    proc = stsci_flat_field.call(proc)
    temp, _, _ = custom_clean_outliers.call(
        proc, dq_bits_to_mask=[0, ],
        window_heights=[150, 100, 50, 40, 30, 24], poly_order=4, outlier_threshold=5.0)
    _, bkg = custom_background_subtract.call(
        temp, method="row_wise", smoothing_length=None,
        bkg_col_left_start=12, bkg_col_left_end=22,
        bkg_col_right_start=50, bkg_col_right_end=68)
    proc.data -= bkg
    proc, P, O = custom_clean_outliers.call(
        proc, dq_bits_to_mask=[0, ],
        window_heights=[150, 100, 50, 40, 30, 24], poly_order=4, outlier_threshold=5.0)

    # Stage 2 reduction, part 3: extraction.
    proc.err[~np.isfinite(proc.err)] = 0.
    wv, spec, spec_unc, trace_sigmas = custom_extract1d_box.call(
        proc, wavelength_map,
        trace_position="constant", aperture_center=36,
        aperture_left_width=4, aperture_right_width=4)
    spec, spec_unc, x_shifts, y_shifts = custom_align_spectra.call(
        proc, spec, spec_unc, align_spectra=False)

    all_spec.append(spec)
    all_spec_unc.append(spec_unc)
    all_bkg.append(np.median(bkg, axis=2))
    all_x_shifts.append(x_shifts)
    all_y_shifts.append(y_shifts)
    all_trace_sigmas.append(trace_sigmas)
```

(continues on next page)

(continued from previous page)

```
    all_outliers.append(0)

# Build xarray for all data products.
ds = xr.Dataset(
    data_vars=dict(
        flux=(["integration_number", "wavelength"], np.concatenate(all_spec), {"units":
↪"DN/s"}),
        flux_error=(["integration_number", "wavelength"], np.concatenate(all_spec_unc), {
↪"units": "DN/s"}),
        background=(["integration_number", "wavelength"], np.concatenate(all_bkg), {
↪"units": "DN/s"}),
        x_shift=(["integration_number"], np.concatenate(all_x_shifts), {"units": "pixel"}
↪),
        y_shift=(["integration_number"], np.concatenate(all_y_shifts), {"units": "pixel"}
↪),
        psf_sigma=(["integration_number"], np.concatenate(all_trace_sigmas), {"units":
↪"pixel"}),
        n_outliers=(["integration_number", "wavelength", "region_width"], np.
↪concatenate(all_outliers), {"units": ""}),
    ),
    coords=dict(
        integration_number=(["integration_number"], np.arange(1, 1 + np.concatenate(all_
↪spec).shape[0], 1), {"units": ""}),
        wavelength=(["wavelength"], wv, {"units": "microns"}),
        region_width=(["region_width"], np.arange(0, 5, 1), {"units": "pixel"}),
    ),
    attrs=dict(author="Your name",
               contact="Your email",
               code="ExoTiC-MIRI interoperating with the STScI pipeline"
               )
)
res_path = os.path.join(
    stage_2_dir, "stage_2_output.nc")
ds.to_netcdf(res_path)
```

5.2 For observations with >40 groups

Here is an example pipeline for a dimmer target, e.g., 100 groups. In this pipeline we implement the self-calibrated linearity correction.

```
import os
import numpy as np
import xarray as xr

os.environ["CRDS_SERVER_URL"] = "https://jwst-crds.stsci.edu"
os.environ["CRDS_PATH"] = "path/to/your/local/crds_dir"
os.environ["CRDS_CONTEXT"] = "jwst_1100.pmap"

from jwst import datamodels
from jwst.pipeline import calwebb_detector1, calwebb_spec2
```

(continues on next page)

(continued from previous page)

```
from exotic_miri.reference import SetCustomGain, SetCustomLinearity, GetWavelengthMap
from exotic_miri.stage_1 import DropGroupsStep
from exotic_miri.stage_2 import CleanOutliersStep, BackgroundSubtractStep, \
    Extract1DBoxStep, AlignSpectraStep


# Data and reduction config.
seg_names = ["jwxyz-seg001_mirimage", "jwxyz-seg002_mirimage"]
data_dir = "/path/to/your/uncal_data_dir"
reduction_dir = "/path/to/your/reduction_dir"
stage_1_dir = os.path.join(reduction_dir, "stage_1")
stage_2_dir = os.path.join(reduction_dir, "stage_2")
for _dir in [reduction_dir, stage_1_dir, stage_2_dir]:
    if not os.path.exists(_dir):
        os.mkdir(_dir)

# Instantiate STScI steps for stage 1.
stsci_group_scale = calwebb_detector1.group_scale_step.GroupScaleStep()
stsci_dq_init = calwebb_detector1.dq_init_step.DQInitStep()
stsci_saturation = calwebb_detector1.saturation_step.SaturationStep()
stsci_reset = calwebb_detector1.reset_step.ResetStep()
stsci_linearity = calwebb_detector1.linearity_step.LinearityStep()
stsci_dark_current = calwebb_detector1.dark_current_step.DarkCurrentStep()
stsci_jump = calwebb_detector1.jump_step.JumpStep()
stsci_ramp_fit = calwebb_detector1.ramp_fit_step.RampFitStep()
stsci_gain_scale = calwebb_detector1.gain_scale_step.GainScaleStep()

# Instantiate custom steps for stage 1.
custom_set_gain = SetCustomGain()
custom_set_linearity = SetCustomLinearity()
custom_drop_groups = DropGroupsStep()

# Instantiate STScI steps for stage 2.
stsci_assign_wcs = calwebb_spec2.assign_wcs_step.AssignWcsStep()
stsci_srctype = calwebb_spec2.srctype_step.SourceTypeStep()
stsci_flat_field = calwebb_spec2.flat_field_step.FlatFieldStep()

# Instantiate custom steps for stage 2.
custom_get_wavelength_map = GetWavelengthMap()
custom_clean_outliers = CleanOutliersStep()
custom_background_subtract = BackgroundSubtractStep()
custom_extract1d_box = Extract1DBoxStep()
custom_align_spectra = AlignSpectraStep()

# Make custom gain datamodel (using the final segment).
uncal_last = datamodels.RampModel(os.path.join(data_dir, "{}_uncal.fits".format(seg_
↪names[-1])))
gain_model = custom_set_gain.call(uncal_last, gain_value=3.1)

# Make custom linearity model (using the final segment).
linearity_model = custom_set_linearity.call(
```

(continues on next page)

(continued from previous page)

```
            uncal_last, group_idx_start_fit=10, group_idx_end_fit=40,
            group_idx_start_derive=10, group_idx_end_derive=99,
            row_idx_start_used=300, row_idx_end_used=380)
del uncal_last

# Iterate data chunks.
all_spec = []
all_spec_unc = []
all_bkg = []
all_x_shifts = []
all_y_shifts = []
all_trace_sigmas = []
all_outliers = []
for seg in seg_names:
    print("\n========= Working on {} =========\n".format(seg))

    # Read in segment.
    proc = datamodels.RampModel(os.path.join(data_dir, "{}_uncal.fits".format(seg)))

    # Stage 1 reduction.
    proc = stsci_group_scale.call(proc)
    proc = stsci_dq_init.call(proc)
    proc = stsci_saturation.call(proc, n_pix_grow_sat=1)
    proc = stsci_reset.call(proc)
    proc = custom_drop_groups.call(proc, drop_groups=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 99])
    proc = stsci_linearity.call(proc, override_linearity=linearity_model)
    proc = stsci_dark_current.call(proc)
    proc = stsci_jump.call(
        proc,
        rejection_threshold=15.,
        four_group_rejection_threshold=15.,
        three_group_rejection_threshold=15.,
        flag_4_neighbors=False,
        min_jump_to_flag_neighbors=15.,
        expand_large_events=False,
        skip=False, override_gain=gain_model)
    _, proc = stsci_ramp_fit.call(proc, override_gain=gain_model)
    proc = stsci_gain_scale.call(proc, override_gain=gain_model)
    proc.save(path=os.path.join(stage_1_dir, "{}_rateints.fits".format(seg)))

    # Stage 2 reduction, part 1: auxiliary data.
    proc = stsci_assign_wcs.call(proc)
    proc = stsci_srctype.call(proc)
    wavelength_map = custom_get_wavelength_map.call(proc)

    # Stage 2 reduction, part 2: cleaning.
    proc = stsci_flat_field.call(proc)
    temp, _, _ = custom_clean_outliers.call(
        proc, dq_bits_to_mask=[0, ],
        window_heights=[150, 100, 50, 40, 30, 24], poly_order=4, outlier_threshold=5.0)
    _, bkg = custom_background_subtract.call(
        temp, method="row_wise", smoothing_length=None,
```

(continues on next page)

(continued from previous page)

```
        bkg_col_left_start=12, bkg_col_left_end=22,
        bkg_col_right_start=50, bkg_col_right_end=68)
    proc.data -= bkg
    proc, P, O = custom_clean_outliers.call(
        proc, dq_bits_to_mask=[0, ],
        window_heights=[150, 100, 50, 40, 30, 24], poly_order=4, outlier_threshold=5.0)

    # Stage 2 reduction, part 3: extraction.
    proc.err[~np.isfinite(proc.err)] = 0.
    wv, spec, spec_unc, trace_sigmas = custom_extract1d_box.call(
        proc, wavelength_map,
        trace_position="constant", aperture_center=36,
        aperture_left_width=3, aperture_right_width=3)
    spec, spec_unc, x_shifts, y_shifts = custom_align_spectra.call(
        proc, spec, spec_unc, align_spectra=False)

    all_spec.append(spec)
    all_spec_unc.append(spec_unc)
    all_bkg.append(np.median(bkg, axis=2))
    all_x_shifts.append(x_shifts)
    all_y_shifts.append(y_shifts)
    all_trace_sigmas.append(trace_sigmas)
    all_outliers.append(O)

# Build xarray for all data products.
ds = xr.Dataset(
    data_vars=dict(
        flux=(["integration_number", "wavelength"], np.concatenate(all_spec), {"units":
↪"DN/s"}),
        flux_error=(["integration_number", "wavelength"], np.concatenate(all_spec_unc), {
↪"units": "DN/s"}),
        background=(["integration_number", "wavelength"], np.concatenate(all_bkg), {
↪"units": "DN/s"}),
        x_shift=(["integration_number"], np.concatenate(all_x_shifts), {"units": "pixel"}
↪),
        y_shift=(["integration_number"], np.concatenate(all_y_shifts), {"units": "pixel"}
↪),
        psf_sigma=(["integration_number"], np.concatenate(all_trace_sigmas), {"units":
↪"pixel"}),
        n_outliers=(["integration_number", "wavelength", "region_width"], np.
↪concatenate(all_outliers), {"units": ""}),
    ),
    coords=dict(
        integration_number=(["integration_number"], np.arange(1, 1 + np.concatenate(all_
↪spec).shape[0], 1), {"units": ""}),
        wavelength=(["wavelength"], wv, {"units": "microns"}),
        region_width=(["region_width"], np.arange(0, 5, 1), {"units": "pixel"}),
    ),
    attrs=dict(author="Your name",
               contact="Your email",
               code="ExoTiC-MIRI interoperating with the STScI pipeline"
               )
```

(continues on next page)

(continued from previous page)

```
)
res_path = os.path.join(
    stage_2_dir, "stage_2_output.nc")
ds.to_netcdf(res_path)
```

CHAPTER

SIX

API

6.1 exotic_miri.reference

class exotic_miri.reference.**GetIntegrationTimes**(**args: Any*, ***kwargs: Any*)

Bases: Step

Get the integration times.

process(*input*)

Get and save the integration times for a data segment, and compute the integration duration in seconds.

Parameters
input (*jwst.datamodels.RampModel*) – This is an uncal.fits loaded data segment.

Returns
timing_data and integration_duration – Table of integration times data and duration of an integration in seconds.

Return type
tuple(fits.table, float)

class exotic_miri.reference.**SetCustomGain**(**args: Any*, ***kwargs: Any*)

Bases: Step

Set a custom gain.

process(*input*)

Set a custom gain value. This step creates a gain model which can be passed to other steps for processing. It should be passed to all steps that make use of the gain, and passed via the arg 'override_gain'. This includes stage 1 steps such as jwst.calwebb_detector1.jump_step, jwst.calwebb_detector1.ramp_fit_step, and jwst.calwebb_detector1.gain_scale_step.

Parameters

- **input** (*jwst.datamodels.RampModel*) – This is an uncal.fits loaded data segment.
- **gain_value** (*float*) – The gain value to set. Default is 3.1.

Returns
gain – The gain model which can be passed to other steps.

Return type
jwst.datamodels.GainModel

class exotic_miri.reference.**SetCustomLinearity**(**args: Any*, ***kwargs: Any*)

Bases: Step

Set a custom linearity correction.

process(*input*)

Make self-calibrated linearity corrections per amplifier. This step uses the uncal.fits data to create a new linearity model. This model can then be passed to the jwst.calwebb_detector1.linearity_step via the arg 'override_linearity'.

The correction involves extrapolating a linear fit to an assumed linear /"well-behaved" section of the ramps, and then fitting a polynomial to the residuals. The polynomial has the constant- and linear-term coefficients fixed at 0 and 1 respectively. Recommended usage requires a large number of groups, >~40, although this is still experimental.

Parameters

- **input** (*jwst.datamodels.RampModel*) – This is an uncal.fits loaded data segment.
- **group_idx_start_fit** (*integer*) – The first group index included in the linear fit. This corresponds to the start of the section of the ramp which is assumed to be well behaved. Default is 10.
- **group_idx_end_fit** (*integer*) – The last group index included in the linear fit. This corresponds to the end of the section of the ramp which is assumed to be well behaved. Default is 40.
- **group_idx_start_derive** (*integer*) – The first group index included in the derived linearity correction. Default is 10.
- **group_idx_end_derive** (*integer*) – The last group index included in the derived linearity correction. Default is -1.
- **row_idx_start_used** (*integer*) – The first row index included in the derived linearity correction. Default is 350.
- **row_idx_end_used** (*integer*) – The last row index included in the derived linearity correction. Default is 386.
- **draw_corrections** (*boolean*) – Plot the derived linearity correction.

Returns

linearity – The linearity model which can be passed to other steps.

Return type

jwst.datamodels.LinearityModel

class exotic_miri.reference.**GetWavelengthMap**(**args: Any*, ***kwargs: Any*)

Bases: Step

Get the wavelength map.

process(*input*)

Get and save the wavelength map data. This is the mapping from the detector pixels (row_idx, col_idx) to wavelength (lambda). To run this step the input must first have had the jwst.calwebb_spec2.assign_wcs_step and jwst.calwebb_spec2.srctype_step already run.

Parameters

input (*jwst.datamodels.CubeModel*) – This is an rateints.fits loaded data segment.

Returns

wavelength_map – The wavelength map with shape (n_rows, n_cols).

Return type
np.ndarray

class exotic_miri.reference.**GetDefaultGain**(**args: Any*, ***kwargs: Any*)

Bases: Step

Get the default gain.

process(*input*)

Get and save gain data from the default CRDS files.

Parameters

- **input** (*jwst.datamodels.RampModel or jwst.datamodels.CubeModel*) – This is either an uncal.fits or rateints.fits loaded data segment. The gain will be the same no matter which data segment you pass in.
- **median_value** (*boolean*) – If True only return the median value rather than the gain model. Default is False.
- **save** (*boolean*) – If True save the gain model to disc. Default is False.
- **save_path** (*string*) – If save==True save the gain model to this path. Default is None.

Returns

- *if median_value == False* –

  gain
  [jwst.datamodels.GainModel] The gain model which can be passed to other steps.

- *elif median_value == True* –

  gain
  [float] The median gain value on the entire detector.

class exotic_miri.reference.**GetDefaultReadNoise**(**args: Any*, ***kwargs: Any*)

Bases: Step

Get the default readnoise.

process(*input*)

Get and save readnoise data from the default CRDS files.

Parameters

- **input** (*jwst.datamodels.RampModel or jwst.datamodels.CubeModel*) – This is either an uncal.fits or rateints.fits loaded data segment. The readnoise will be the same no matter which data segment you pass in.
- **median_value** (*boolean*) – If True only return the median value rather than the readnoise model. Default is False.
- **save** (*boolean*) – If True save the readnoise model to disc. Default is False.
- **save_path** (*string*) – If save==True save the readnoise model to this path. Default is None.

Returns

- *if median_value == False* –

  gain
  [jwst.datamodels.ReadnoiseModel] The readnoise model which can be passed to other steps.

- *elif median_value == True –*

  gain
  [float] The median readnoise value on the entire detector.

6.2 exotic_miri.stage_1

class exotic_miri.stage_1.**DropGroupsStep**(**args: Any*, ***kwargs: Any*)

Bases: Step

Drop groups step.

process(*input*)

Drop groups which may be adversely affecting the ramps. This may be due to detector effects such RSCD and/or the last frame effect. This step simply marks groups as do_not_use and are thus ignored in subsequent processing, such as by jwst.calwebb_detector1.ramp_fit_step.

Parameters

- **input** (*jwst.datamodels.RampModel*) – This is an uncal.fits loaded data segment.
- **drop_groups** (*list of integers*) – These integers are the groups to be dropped. The integers are zero-indexed such that 0 is the first group.

Returns
output – A RampModel with groupdq flags set as do_not_use (2**0).

Return type
jwst.datamodels.RampModel

class exotic_miri.stage_1.**ReferencePixelStep**(**args: Any*, ***kwargs: Any*)

Bases: Step

Reference pixel step.

process(*input*)

Reference pixel correction at the group level, using the reference pixels available to the MIRI LRS subarray. The corrections can be made with a variety of options for smoothing the values and/or separating odd and even rows.

The default pipeline, at the time of programming, does not have this option for subarrays. It assumes subarrays have no available reference pixels, but the MIRI LRS subarray is against the edge of the detector.

Parameters

- **input** (*jwst.datamodels.RampModel*) – This is an uncal.fits loaded data segment.
- **smoothing_length** (*integer*) – If not None, the number of rows to median smooth the estimated reference pixel values over. Default is no smoothing.
- **odd_even_rows** (*boolean*) – Treat the correction separately for odd and even rows. Default is True.
- **draw_correction** (*boolean*) – Plot the correction images.

Returns
output – A RampModel with the reference pixel correction applied.

Return type
jwst.datamodels.RampModel

class exotic_miri.stage_1.**GroupBackgroundSubtractStep**(**args: Any*, ***kwargs: Any*)

Bases: Step

Group-level background subtraction step.

process(*input*)

Subtract the background at the group level.

Parameters

- **input** (*jwst.datamodels.RampModel*) – This is an uncal.fits loaded data segment.
- **method** (*string*) – The background subtraction method: constant, the background is estimated as a median over the entire background region; row_wise, the background is estimated as a median per row; col_wise; the background is estimated within a row as a linear function of column number. Default is row_wise.
- **bkg_col_left_start** (*integer*) – The column index of the start of the background region on the left side of the spectral trace. Default is 8.
- **bkg_col_left_end** (*integer*) – The column index of the end of the background region on the left side of the spectral trace. Default is 17.
- **bkg_col_right_start** (*integer*) – The column index of the start of the background region on the right side of the spectral trace. Default is 56.
- **bkg_col_right_end** (*integer*) – The column index of the end of the background region on the right side of the spectral trace. Default is 72.
- **smoothing_length** (*integer*) – If not None, the number of rows to median smooth the estimated background values over. Default is no smoothing.

Returns

output – A RampModel with the background subtracted from each group.

Return type

jwst.datamodels.RampModel

class exotic_miri.stage_1.**DropIntegrationsStep**(**args: Any*, ***kwargs: Any*)

Bases: Step

Drop integrations step.

process(*input*)

Drop integrations from a data segment. The use of this step is most likely because these integrations are too severely affected by systematics to be worth processing, or this step may also be useful if the user wants to test pipelines on only a small subset of data.

Parameters

- **input** (*jwst.datamodels.RampModel*) – This is an uncal.fits loaded data segment.
- **drop_integrations** (*list of integers*) – These integers are the integrations to be dropped. The integers are zero-indexed such that 0 is the first integration.

Returns

output – A RampModel with [drop_integrations] removed.

Return type

jwst.datamodels.RampModel

class exotic_miri.stage_1.**RegroupStep**(**args: Any*, ***kwargs: Any*)

Bases: Step

Regroup long integrations into more shorter integrations.

process(*input*)

Regroup integrations, comprised of m groups, into several smaller integrations, comprised of n groups, where m is a multiple of n. This may not be helpful but it is here if you want to try it. TODO: add support for non-multiples.

Parameters

- **input** (*jwst.datamodels.RampModel*) – This is an uncal.fits loaded data segment.
- **n_groups** (*integer*) – The new number of groups per integration.

Returns
output – A RampModel reshaped into n_groups.

Return type
jwst.datamodels.RampModel

6.3 exotic_miri.stage_2

class exotic_miri.stage_2.**InspectDQFlagsStep**(**args: Any*, ***kwargs: Any*)

Bases: Step

Inspect DQ flags step.

process(*input*)

Inspect the data quality flags present in your rate-images. This step does not alter any data, it simply computes the number of each type of DQ flag that are present, and optionally display where they are on the detector. It is quite decent, would recommend some visual inspection with this step.

NB. DQ array bit values as per: *https://jwst-pipeline.readthedocs.io/en/latest/jwst/references_general/references_general.html#data-quality-flags*.

Parameters

- **input** (*jwst.datamodels.CubeModel*) – This is an rateints.fits loaded data segment.
- **draw_dq_flags** (*boolean*) – Plot the DQ flags on the detector for each integration.

Returns
input – The same model, unaltered from input.

Return type
jwst.datamodels.CubeModel

class exotic_miri.stage_2.**CleanOutliersStep**(**args: Any*, ***kwargs: Any*)

Bases: Step

Clean outliers step.

process(*input*)

Clean outliers using deviations from a spatial profile, following Horne 1986, and/or values from the data quality array.

NB. DQ array bit values as per: *https://jwst-pipeline.readthedocs.io/en/latest/jwst/references_general/references_general.html#data-quality-flags*.

Parameters

- **input** (*jwst.datamodels.CubeModel*) – This is a rateints.fits loaded data segment.
- **window_heights** (*list of integers*) – The size of the windows in pixels, in the dispersion direction, to use when fitting polynomials to the spatial profile. The size of the window iterates cyclically through the list until the total height of the detector is reached. Recommended to use smaller window sizes at the shorter wavelengths (larger row indexes) as the throughput * stellar spectra show larger variations here. For example, [150, 100, 50, 50, 20, 20, 20].
- **dq_bits_to_mask** (*list of integers*) – A list of data quality flags to clean. These pixels are replaced by the spatial profile values. See link above for definitions of the DQ bit values. For example, [0,] cleans pixes marked as 2**0 (do_not_use) in the DQ array.
- **poly_order** (*integer*) – Polynomial order for fitting to the windows of data. Default is 4.
- **outlier_threshold** (*float*) – Number of standard deviations away from the spatial profile for a pixel to be determined as an outlier. Default is 4.0.
- **spatial_profile_left_idx** (*integer*) – Start index of the columns which should be included in the spatial profile. Default is 26.
- **spatial_profile_right_idx** (*integer*) – End index of the columns which should be included in the spatial profile. Default is 47.
- **draw_cleaning_col** (*boolean*) – Plot the cleaning interactively. Useful for understanding the process and getting a feel for the hyperparams.
- **draw_spatial_profiles** (*boolean*) – Plot the spatial profiles after each integration is cleaned.
- **no_clean** (*boolean*) – Override, and just remove any nans. This is for quick tests.

Returns

output, spatial profile cube, outlier counts cube – A CubeModel with outliers cleaned, a 3D array of the fitted spatial profiles, and a count of the number of outliers cleaned within 0-4 pixels of the spectral trace (column index 36).

Return type

tuple(CubeModel, np.ndarray, np.ndarray)

class exotic_miri.stage_2.**BackgroundSubtractStep**(**args: Any*, ***kwargs: Any*)

Bases: Step

Background subtraction step.

process(*input*)

Subtract the background from the rate-images.

Parameters

- **input** (*jwst.datamodels.CubeModel*) – This is a rateints.fits loaded data segment.
- **method** (*string*) – The background subtraction method: constant, the background is estimated as a median over the entire background region; row_wise, the background is estimated as a median per row; col_wise, the background is estimated within a row as a linear function of column number. Default is row_wise.
- **bkg_col_left_start** (*integer*) – The column index of the start of the background region on the left side of the spectral trace. Default is 8.

- **bkg_col_left_end** (*integer*) – The column index of the end of the background region on the left side of the spectral trace. Default is 17.
- **bkg_col_right_start** (*integer*) – The column index of the start of the background region on the right side of the spectral trace. Default is 56.
- **bkg_col_right_end** (*integer*) – The column index of the end of the background region on the right side of the spectral trace. Default is 72.
- **smoothing_length** (*integer*) – If not None, the number of rows to median smooth the estimated background values over. Default is no smoothing.

Returns
output – A CubeModel with the background subtracted from each integration.

Return type
jwst.datamodels.CubeModel

class exotic_miri.stage_2.**Extract1DBoxStep**(**args: Any*, ***kwargs: Any*)

Bases: Step

Box extraction step.

process(*input*, *wavelength_map*)

Extract time-series 1D stellar spectra using a box aperture.

Parameters

- **input** (*jwst.datamodels.CubeModel*) – This is a rateints.fits loaded data segment.
- **wavelength_map** (*np.ndarray*) – The wavelength map. This is output from exotic_miri.reference.GetWavelengthMap.
- **trace_position** (*string*) – The method for locating the spectral trace per detector row. constant: uses the value specified by aperture_center. gaussian_fits: fit a Gaussian to each row to find the centre.
- **aperture_center** (*integer*) – The defined centre of the spectral trace in terms of column index. Default is 36.
- **aperture_left_width** (*integer*) – The half-width of the box aperture in pixels away from the aperture_center to the left. Default is 4, and so this aperture would include the aperture_center, say column 36, and 4 columns to the left of this.
- **aperture_right_width** (*integer*) – The half-width of the box aperture in pixels away from the aperture_center to the right. Default is 4, and so this aperture would include the aperture_center, say column 36, and 4 columns to the right of this.
- **draw_psf_fits** (*boolean*) – Plot Gaussina fits to the PSF.
- **draw_aperture** (*boolean*) – Plot the defined aperture.
- **draw_spectra** (*boolean*) – Plot the extracted 1D spectra.

Returns
wavelengths, spectra, spectra_uncertainties, trace_widths – Arrays of wavelengths (n_rows,), spectra (n_ints, n_rows), spectra_uncertainties (n_ints, n_rows), trace_widths (n_ints,).

Return type
tuple(np.ndarray, np.ndarray, np.ndarray, np.ndarray)

class exotic_miri.stage_2.**Extract1DOptimalStep**(**args: Any*, ***kwargs: Any*)

Bases: Step

Optimal extraction step.

process(*input*, *wavelength_map*, *P*, *readnoise*)

Extract time-series 1D stellar spectra using optimal extraction as detailed in Horne 1986. The spatial profile must be pre-computed and input.

Parameters

- **input** (*jwst.datamodels.CubeModel*) – This is a rateints.fits loaded data segment.
- **wavelength_map** (*np.ndarray*) – The wavelength map. This is output from exotic_miri.reference.GetWavelengthMap.
- **P** (*np.ndarray*) – A cube of spatial profiles, one for each integration, of shape (n_ints, n_rows, n_cols). The should be normalised within each row (cross-disperion direction). These data can be made using exotic_miri.stage_2.CleanOutliersStep.
- **readnoise** (*np.ndarray*) – A cube of readnoise values, one for each integration, of shape (n_ints, n_rows, n_cols).
- **median_spatial_profile** (*boolean*) – If True median the spatial profiles through time. Default is False.
- **trace_position** (*string*) – The method for locating the spectral trace per detector row. constant: uses the value specified by aperture_center. gaussian_fits: fit a Gaussian to each row to find the centre.
- **aperture_center** (*integer*) – The defined centre of the spectral trace in terms of column index. Default is 36.
- **aperture_left_width** (*integer*) – The half-width of the box aperture in pixels away from the aperture_center. Default is 4, and so this aperture would include the aperture_center, say column 36, and 4 columns to the left of this.
- **aperture_right_width** (*integer*) – The half-width of the box aperture in pixels away from the aperture_center. Default is 4, and so this aperture would include the aperture_center, say column 36, and 4 columns to the right of this.
- **draw_psf_fits** (*boolean*) – Plot Gaussina fits to the PSF.
- **draw_aperture** (*boolean*) – Plot the defined aperture, within which optimal extraction is applied.
- **draw_spectra** (*boolean*) – Plot the extracted 1D spectra.

Returns

wavelengths, spectra, spectra_uncertainties, trace_widths – Arrays of wavelengths (n_rows,), spectra (n_ints, n_rows), spectra_uncertainties (n_ints, n_rows), trace_widths (n_ints,).

Return type

tuple(np.ndarray, np.ndarray, np.ndarray, np.ndarray)

class exotic_miri.stage_2.**AlignSpectraStep**(**args: Any*, ***kwargs: Any*)

Bases: Step

Align spectra step.

process(*input*, *spec*, *spec_unc*)

Align the 1D stellar spectra. This step measures, by cross-correlation, the x- and y-positions of the spectral trace through time. These positions are returned and the spectra can optionally be realigned by the y-positions.

Parameters

- **input** (*jwst.datamodels.CubeModel*) – This is a rateints.fits loaded data segment.
- **align_spectra** (*boolean*) – If True the spectra are realigned by the measured y-positions. Default is True.
- **draw_cross_correlation_fits** (*boolean*) – Plot the cross-correlation function and the fit to this determining the trace position.
- **draw_trace_positions** (*boolean*) – Plot the measured trace positions.

Returns

spectra, spectra_uncertainties, x_shifts, y_shifts – The time-series spectra and their uncertainties each with shape (n_ints, n_wavelengths) and the measured trace shifts in x and y each with shape (n_ints,).

Return type

tuple(np.ndarray, np.ndarray, np.ndarray, np.ndarray)

CHAPTER

SEVEN

CITATION

If you make use of ExoTiC-MIRI in your research, please cite Grant et al. 2023:

```
@software{grant_david_2023_8211207,
  author       = {Grant, David and
                  Valentine, Daniel E. and
                  Wakeford, Hannah R.},
  title        = {Exo-TiC/ExoTiC-MIRI: ExoTiC-MIRI v1.0.0},
  month        = aug,
  year         = 2023,
  publisher    = {Zenodo},
  version      = {v1.0.0},
  doi          = {10.5281/zenodo.8211207},
  url          = {https://doi.org/10.5281/zenodo.8211207}
}
```

CHAPTER

EIGHT

ACKNOWLEDGEMENTS

Built by David Grant, Daniel Valentine, Hannah Wakeford, and contributors.

If you make use of ExoTiC-MIRI in your research, see the *citation page* for info on how to cite this package. You can find other software from the Exoplanet Timeseries Characterisation (ExoTiC) ecosystem over on GitHub.

PYTHON MODULE INDEX

e

INDEX

S