pyhrs Documentation

Release 0.0.dev126

Steve Crawford

March 31, 2015

Contents

Welcome to the pyhrs documentation! pyhrs is a package for the reduction and analysis of data from the High Resolution Spectrograph on the Southern African Large Telescope

pyhrs is an ffiliated package for the AstroPy package. The documentation for this package is here:

Part I

HRS Data Reduction (`pyhrs`)

CHAPTER 1

Introduction

Note: pyhrs works only with astropy version 1.0.0 or later. It also requires ccdproc version 0.3.0 or later.

The pyhrs package provides steps for reducing and extracting data from the High Resolution Spectrograph on the Southern African Large Telescope. The package includes the following sub-packages to help with the processing and reduction of the data:

- A class describing a single HRS order, HRSOrder, that includes position, wavelength, and flux properties
- A class describing the HRS spectrograph, HRSModel, to allow for accurate modeling of the HRS Spectrograph
- Raw data can be processed using hrsprocess for both the blue and red arms.
- Orders can be identified in the images using create_orderframe
- wavelength_calibrate_arc can be used to calculate the wavelength calibration.

Once the data are reduced and calibrated, extraction of the object spectra can proceed based on the prefered method of the user.

CHAPTER 2

Processing your data

For more information about how to process your data, please check out

2.1 Processing HRS Data

`hrsprocess` includes steps for the basic CCD processing necessary for HRS data. The code provides a wrapper for tasks from `ccdproc` to provide specific reductions for HRS data. In addition, it provides several functions for creating calibration frames for the reduction of HRS data.

Note: `hrsprcess` expects files to follow the SALT naming conventions

2.1.1 Processing Data frames

Data frames can be process using the tasks `blue_process` and `red_process`. The user can select from several options included in these programs, but certain aspects are hard wired to provide convenient functions for data reductions. For example, to process blue data:

```
>>> from pyhrs.hrsprocess import blue_process
>>> ccd = blue_process('H201411170015.fits', masterbias=masterbias)
```

This will return an `ccdproc.CCDData` object that has had the overscan corrected, trimmed, gain corrected, had the master bias subtracted, and positioned such that the orders increase from the bottom to the top and the dispersion goes from the left to the right. Flatfielding and calibration from a spectrophotometric standard will only be applied in later steps.

Convenience functions for Processing Science Data:

- `blue_process`: process data from the HRS blue camera
- `red_process`: process data from the HRS red camera
- `hrs_process`: convenience function for processing HRS data

The functions all pass appropriate parameters to `ccd_process`. This tasks wraps functions from `ccdproc` for processing CCD images. `ccd_process` has a number of steps, which are all optional, that include overscan subtraction, trimming, creating error frames, masking, gain correction, and subtracting a master bias.

2.1.2 Processing Calibration Frames

Calibration frames can also be created using several convenience functions. For example, passing a list of filenames to `create_masterbias` will process the data and combine them to create the master bias frame.

```
>>> from pyhrs.hrsprocess import create_masterbias
>>> masterbias = create_masterbias(['H201411170015.fits', 'H201411170016.fits']
```

This will process each frame and return a masterbias `ccdproc.CCDData` object.

In addition, masterflats can be produced using:

```
>>> from pyhrs.hrsprocess import create_masterflat
>>> masterflat = create_masterflat(['H201411170020.fits', 'H201411170021.fits', 'H201411100022.fits']
```

2.2 Finding Orders

A critical step to the reduction of HRS data is finding orders in the images. Typically, images of flat fields or of a bright object can be used to identify orders in the frame. The data must first be processed. Unfortunately, a truly perfect data set for the identification of the orders is rarely available as the response function across the CCD can widely vary.

Below we outline some tasks that can be used for identifying orders in HRS images and the steps that are used to process the data to make the identification possible.

2.2.1 Normalizing a flat field frame

Due to the changing response function across the CCD, a fiber flat image can show significant vignetting in both the vertical and horizontal direction. To remove the vignetting in the vertical direction, the `normalize_image` task can be used.

The `normalize_image` task fits a function to a fiber flat image after masking the image. Either a 1D or 2D function can be fit to the image, and the fitting function should be specified by the user. For the best performance, it is best to either apply an already existing order frame or to smooth the image and mask areas of low response.

Here is an example of the steps needed to normalize an image:

```
>>> from astropy.io import fits
>>> from astropy import modeling as mod
>>> from scipy import ndimage as nd
>>> from pyhrs import normalize_image

>>> hdu = fits.open('HFLAT.fits')
>>> image = nd.filters.maximum_filter(image, 10)
>>> mask = (image > 500)
>>> norm = normalize_image(image, mod.models.Legendre1D(10), mask=mask)
>>> norm[norm<10000] = 0
```

This will produce a `ndarry` where good areas all have the same value and bad areas will have values set to zero. This significantly simplifies the process of identifying orders in the image. However, orders at the very top of the image with little or no signal will still be difficult to detect.

2.2.2 Creating an order frame

The next step is to create an order frame. An order frame is defined as an image where each pixel associated with an order is identified and labeled with that order. To produce the order frame, an initial detection kernal (based on

a user input) is convolved with a single column in the image. The first maximum identified is assocatied with the initial input order given by the user. To identify the full 2D shape of the order, all pixels above a certain threshold and connected are identified. These pixels are then given the value of the initial order. Once the order is identified, all pixels associated with this order are set to zero. The detection kernal is then updated based on the 1D shape of this order, the order is incremented, and the process is repeated until all orders are identified in the frame.

All of these steps are accomplished running the `create_orderframe` task. An example of running this task is the follow:

```
>>> norm[norm>500] = 500
>>> xc = int(norm.shape[0]/2)
>>> detect_kern = norm[30:110, xc]
>>> frame = create_orderframe(norm, 84, xc, detect_kern, y_start=30, y_limit=4050)
```

This will produce the order frame for the blue arm in medium resolution mode. It will identify all orders up to the limit of y-position of 4050. For the red arm, the first order is 53 for the medium resolution mode.

2.3 Wavelength Calibration

Wavelength calibration requires the identification of known lines in a spectrum of an arc to determine the transformation between pixel space and wavelength.

2.3.1 HRSModel

`HRSModel` is a class for producing synthetic HRS spectra. HRSModel is based on the `PySpectrograph.Spectrograph` class. It includes a simple model for both arms that is based on the instrument confirguration and the spectrograph equation. After adjusting for offsets in the fiber position relative to the CCD, the model can return an approximation for the transformation that is accurate to within a few pixels.

The residuals between the model and the actual solution though are well described by a quadratic equation. This quadratic does slowly vary accross the orders and is different between the two arms. Due to the change in the fiber position between the different resolutions, this quadratic can change between the different configurations as well.

For these reasons, the initial guess for the wavelength solution is based on `HRSModel` plus a quadratic correction. The correction can either be calculated manual or by automatically fitting a single row of an order.

2.3.2 Calibrating a single order

To calibrate a single order, the following steps are carried out:

1. Curvature due to the optical distortion is removed from the spectra and a square representation of the 2D spectra is created. Only integer shifts are applied to the data
2. A model of the spectrograph is created based on the order, camera, and xpos offset that are supplied. A small correction described by a quadratic equation is added to the transformation calculated from the model.
3. In each row of the data, peaks are extracted and matched with a line in the atlas of wavelengths that is provided. Due to the accuracy of the initial guess, lines are matched to within an angstrom and any line that might be blended is rejected.
4. Once the first set of peaks and lines are matched up, a new solution is calculated for the given row. Then the processes of matching lines and determining a wavelength solution is repeated using this new solution. The best result from each line is saved.

5. Using all of the matched lines from all lines, a 'best' solution is determined. Everything but the zeroth order parameter of the fit is fixed to a slowly varying value based on the overall solution to all lines. See `fit_solution` for more details.
6. Based on the best solution found, the process is repeated for each row but only determing the zeropoint.
7. Based on the solution found, a wavelength is assigned to each pixel

All of these steps are carried out by `wavelength_calibrate_order`. In the end, this task returns an `HRSOrder` object with wavelengths correspond to every pixel where a good solution was found. In addition, it also returns the x-position, wavelength, and solution for the initial row.

2.3.3 Calibrating an arc

For full automated calibration of an arc, `wavelength_calibrate_arc` can be used. In this task, it applies `wavelength_calibrate_order` to each of the orders in the frame. It uses `pyhrs.HRSModel` for the first guess but takes the quadratic correction from the solution of the nearest order. It starts with the initial order and the first row is also set by the user.

Part II

Processing HRS Data

hrsprocess includes steps for the basic CCD processing necessary for HRS data. The code provides a wrapper for tasks from ccdproc to provide specific reductions for HRS data. In addition, it provides several functions for creating calibration frames for the reduction of HRS data.

Note: hrsprcess expects files to follow the SALT naming conventions

CHAPTER 3

Processing Data frames

Data frames can be process using the tasks `blue_process` and `red_process`. The user can select from several options included in these programs, but certain aspects are hard wired to provide convenient functions for data reductions. For example, to process blue data:

```
>>> from pyhrs.hrsprocess import blue_process
>>> ccd = blue_process('H201411170015.fits', masterbias=masterbias)
```

This will return an `ccdproc.CCDData` object that has had the overscan corrected, trimmed, gain corrected, had the master bias subtracted, and positioned such that the orders increase from the bottom to the top and the dispersion goes from the left to the right. Flatfielding and calibration from a spectrophotometric standard will only be applied in later steps.

Convenience functions for Processing Science Data:

- `blue_process`: process data from the HRS blue camera
- `red_process`: process data from the HRS red camera
- `hrs_process`: convenience function for processing HRS data

The functions all pass appropriate parameters to `ccd_process`. This tasks wraps functions from `ccdproc` for processing CCD images. `ccd_process` has a number of steps, which are all optional, that include overscan subtraction, trimming, creating error frames, masking, gain correction, and subtracting a master bias.

CHAPTER 4

Processing Calibration Frames

Calibration frames can also be created using several convenience functions. For example, passing a list of filenames to `create_masterbias` will process the data and combine them to create the master bias frame.

```
>>> from pyhrs.hrsprocess import create_masterbias
>>> masterbias = create_masterbias(['H201411170015.fits', 'H201411170016.fits']
```

This will process each frame and return a masterbias `ccdproc.CCDData` object.

In addition, masterflats can be produced using:

```
>>> from pyhrs.hrsprocess import create_masterflat
>>> masterflat = create_masterflat(['H201411170020.fits', 'H201411170021.fits', 'H201411100022.fits']
```

Part III

Finding Orders

A critical step to the reduction of HRS data is finding orders in the images. Typically, images of flat fields or of a bright object can be used to identify orders in the frame. The data must first be processed. Unfortunately, a truly perfect data set for the identification of the orders is rarely available as the response function across the CCD can widely vary.

Below we outline some tasks that can be used for identifying orders in HRS images and the steps that are used to process the data to make the identification possible.

CHAPTER 5

Normalizing a flat field frame

Due to the changing response function across the CCD, a fiber flat image can show significant vignetting in both the vertical and horizontal direction. To remove the vignetting in the vertical direction, the `normalize_image` task can be used.

The `normalize_image` task fits a function to a fiber flat image after masking the image. Either a 1D or 2D function can be fit to the image, and the fitting function should be specified by the user. For the best performance, it is best to either apply an already existing order frame or to smooth the image and mask areas of low response.

Here is an example of the steps needed to normalize an image:

```
>>> from astropy.io import fits
>>> from astropy import modeling as mod
>>> from scipy import ndimage as nd
>>> from pyhrs import normalize_image

>>> hdu = fits.open('HFLAT.fits')
>>> image = nd.filters.maximum_filter(image, 10)
>>> mask = (image > 500)
>>> norm = normalize_image(image, mod.models.Legendre1D(10), mask=mask)
>>> norm[norm<10000] = 0
```

This will produce a `ndarry` where good areas all have the same value and bad areas will have values set to zero. This significantly simplifies the process of identifying orders in the image. However, orders at the very top of the image with little or no signal will still be difficult to detect.

CHAPTER 6

Creating an order frame

The next step is to create an order frame. An order frame is defined as an image where each pixel associated with an order is identified and labeled with that order. To produce the order frame, an initial detection kernal (based on a user input) is convolved with a single column in the image. The first maximum identified is assocatied with the initial input order given by the user. To identify the full 2D shape of the order, all pixels above a certain threshold and connected are identified. These pixels are then given the value of the initial order. Once the order is identified, all pixels associated with this order are set to zero. The detection kernal is then updated based on the 1D shape of this order, the order is incremented, and the process is repeated until all orders are identified in the frame.

All of these steps are accomplished running the `create_orderframe` task. An example of running this task is the follow:

```
>>> norm[norm>500] = 500
>>> xc = int(norm.shape[0]/2)
>>> detect_kern = norm[30:110, xc]
>>> frame = create_orderframe(norm, 84, xc, detect_kern, y_start=30, y_limit=4050)
```

This will produce the order frame for the blue arm in medium resolution mode. It will identify all orders up to the limit of y-position of 4050. For the red arm, the first order is 53 for the medium resolution mode.

Part IV

Wavelength Calibration

Wavelength calibration requires the identification of known lines in a spectrum of an arc to determine the transformation between pixel space and wavelength.

CHAPTER 7

HRSModel

HRSModel is a class for producing synthetic HRS spectra. HRSModel is based on the PySpectrograph.Spectrograph class. It includes a simple model for both arms that is based on the instrument confirguration and the spectrograph equation. After adjusting for offsets in the fiber position relative to the CCD, the model can return an approximation for the transformation that is accurate to within a few pixels.

The residuals between the model and the actual solution though are well described by a quadratic equation. This quadratic does slowly vary accross the orders and is different between the two arms. Due to the change in the fiber position between the different resolutions, this quadratic can change between the different configurations as well.

For these reasons, the initial guess for the wavelength solution is based on HRSModel plus a quadratic correction. The correction can either be calculated manual or by automatically fitting a single row of an order.

CHAPTER 8

Calibrating a single order

To calibrate a single order, the following steps are carried out:

1. Curvature due to the optical distortion is removed from the spectra and a square representation of the 2D spectra is created. Only integer shifts are applied to the data
2. A model of the spectrograph is created based on the order, camera, and xpos offset that are supplied. A small correction described by a quadratic equation is added to the transformation calculated from the model.
3. In each row of the data, peaks are extracted and matched with a line in the atlas of wavelengths that is provided. Due to the accuracy of the initial guess, lines are matched to within an angstrom and any line that might be blended is rejected.
4. Once the first set of peaks and lines are matched up, a new solution is calculated for the given row. Then the processes of matching lines and determining a wavelength solution is repeated using this new solution. The best result from each line is saved.
5. Using all of the matched lines from all lines, a 'best' solution is determined. Everything but the zeroth order parameter of the fit is fixed to a slowly varying value based on the overall solution to all lines. See `fit_solution` for more details.
6. Based on the best solution found, the process is repeated for each row but only determing the zeropoint.
7. Based on the solution found, a wavelength is assigned to each pixel

All of these steps are carried out by `wavelength_calibrate_order`. In the end, this task returns an `HRSOrder` object with wavelengths correspond to every pixel where a good solution was found. In addition, it also returns the x-position, wavelength, and solution for the initial row.

CHAPTER 9

Calibrating an arc

For full automated calibration of an arc, `wavelength_calibrate_arc` can be used. In this task, it applies `wavelength_calibrate_order` to each of the orders in the frame. It uses `pyhrs.HRSModel` for the first guess but takes the quadratic correction from the solution of the nearest order. It starts with the initial order and the first row is also set by the user.

Part V

Extracting Spectra

Part VI

PyHRS

The PyHRS package is for the reduction of data from the High Resolution Spectrograph on the Southern African Large Telescope. The goals of the package are to provide tools to be able to produce scientific quality reductions for the the low, medium, and high resolution modes for HRS and to prepare data for more specialized code for the reduction of high stability observations.

The package includes the following classes and functions: - HRSModel - hrsprocess - HRSOrder - hrstools

CHAPTER 10

HRSModel

HRSModel is a class for producing synthetic HRS spectra. HRSModel is based on the `PySpectrograph.Spectrograph` class. It only includes a simple model based on the instrument confirguration and the spectrograph equation.

CHAPTER 11

hrs_process

`hrsprocess` includes steps for the basic CCD processing necessary for HRS data. It also includes steps necessary for creating calibration frames.

CHAPTER 12

HRSOrder

HRSOrder is a class descirbe a single order from an HRS image. The order then has different tools for identifying regions, extracting orders, and defining properties of different orders such as wavelengths and calibrations.

CHAPTER 13

hrsextract

hrsextract includes all steps necessary to extract a single, one-dimensional HRS spectrum.

CHAPTER 14

HRStools

HRStools includes generally utilies used across different functions and classes.

14.1 Reference/API

CHAPTER 15

pyhrs Module

pyhrs is a package for reducing data from the High Resolution Spectrograph on the Southern African Large Telescope

background(b_arr[, niter])	Determine the background for an array
blue_process(infile[, masterbias, error, ...])	Process a blue frame
calc_weights(x, y, m[, yerr])	Calculate weights for each value based on deviation from best fit model
ccd_process(ccd[, oscan, trim, error, ...])	Perform basic processing on ccd data.
create_masterbias(image_list)	Create a master bias frame from a list of images
create_masterflat(image_list[, masterbias])	Create a master flat frame from a list of images
create_orderframe(data, first_order, xc, ...)	Create an order frame from from an observation.
fit_order(data, detect_kernal, xc[, order, ...])	Given an array and an overlapping detect_kernal,
fit_wavelength_solution(sol_dict)	Determine the best fit solution and re-fit each line with that solution
hrs_process(image_name[, ampsec, oscansec, ...])	Processing required for HRS observations.
iterfit1D(x, y, fitter, model[, yerr, ...])	Iteratively fit a function.
match_lines(xarr, farr, sw, sf, ws[, rw, ...])	Match lines in the spectra with specific wavleengths
ncor(x, y)	Calculate the normalized correlation of two arrays
normalize_image(data, func_init, mask[, ...])	Normalize an HRS image.
red_process(infile[, masterbias, error, rdnoise])	Process a blue frame
test([package, test_path, args, plugins, ...])	Run the tests using py.test.
wavelength_calibrate_arc(arc, order_frame, ...)	Wavelength calibrate an arc spectrum from HRS
wavelength_calibrate_order(hrs, slines, ...)	Wavelength calibration of a single order from the HRS arc spectra
xcross_fit(warr, farr, sw_arr, sf_arr[, dw, nw])	Calculate a zeropoint shift between the observed arc

15.1 background

pyhrs.**background**(*b_arr*, *niter=3*)

Determine the background for an array

Parameters

b_arr: numpy.ndarray

Array for the determination of the background

niter: int

Number of iterations for sigma clipping

Returns

bkgrd: float

median background value after sigma clipping

bkstd: float

Estimated standard deviation based on the median absolute deviation

15.2 blue_process

pyhrs.**blue_process**(*infile*, *masterbias=None*, *error=False*, *rdnoise=None*)

Process a blue frame

15.3 calc_weights

pyhrs.**calc_weights**(*x*, *y*, *m*, *yerr=None*)

Calculate weights for each value based on deviation from best fit model

Parameters

x: numpy.ndarray

Arrray of x-values

y: numpy.ndarray

Arrray of y-values

model: ~astropy.modeling.model

A model to be fit

yerr: numpy.ndarray

[Optional] Array of uncertainties for the y-value

Returns

weights: numpy.ndarray

Weights for each parameter

15.4 ccd_process

pyhrs.**ccd_process**(*ccd*, *oscan=None*, *trim=None*, *error=False*, *masterbias=None*, *bad_pixel_mask=None*, *gain=None*, *rdnoise=None*, *oscan_median=True*, *oscan_model=None*)

Perform basic processing on ccd data.

The following steps can be included:

- overscan correction
- trimming of the image
- create edeviation frame
- gain correction
- add a mask to the data
- subtraction of master bias

The task returns a processed `ccdproc.CCDData` object.

Parameters

ccd: ‘ccdproc.CCDData‘

Frame to be reduced

oscan: None, str, or, ‘~ccdproc.ccddata.CCDData‘

For no overscan correction, set to None. Otherwise proivde a region of `ccd` from which the overscan is extracted, using the FITS conventions for index order and index start, or a slice from `ccd` that contains the overscan.

trim: None or str

For no trim correction, set to None. Otherwise proivde a region of `ccd` from which the image should be trimmed, using the FITS conventions for index order and index start.

error: boolean

If True, create an uncertainty array for ccd

masterbias: None, ‘~numpy.ndarray‘, or ‘~ccdproc.CCDData‘

A materbias frame to be subtracted from ccd.

bad_pixel_mask: None or ‘~numpy.ndarray‘

A bad pixel mask for the data. The bad pixel mask should be in given such that bad pixels havea value of 1 and good pixels a value of 0.

gain: None or ‘~astropy.Quantity‘

Gain value to multiple the image by to convert to electrons

rdnoise: None or ‘~astropy.Quantity‘

Read noise for the observations. The read noise should be in `electron`

oscan_median : bool, optional

If true, takes the median of each line. Otherwise, uses the mean

oscan_model : `Model`, optional

Model to fit to the data. If None, returns the values calculated by the median or the mean.

Returns

ccd: `ccdproc.CCDData`

Reduded ccd

Examples

1.To overscan, trim, and gain correct a data set:

```
>>> import numpy as np
>>> from astropy import units as u
>>> from hrsprocess import ccd_process
>>> ccd = CCDData(np.ones([100, 100]), unit=u.adu)
>>> nccd = ccd_process(ccd, oscan='[1:10,1:100]', trim='[10:100, 1,100]',
                       error=False, gain=2.0*u.electron/u.adu)
```

15.5 create_masterbias

pyhrs.**create_masterbias**(*image_list*)

Create a master bias frame from a list of images

Parameters

image_list: list

List contain the file names to be processed

Returns

masterbias: ccddata.CCDData

Combine master bias from the biases supplied in image_list

15.6 create_masterflat

pyhrs.**create_masterflat**(*image_list*, *masterbias=None*)

Create a master flat frame from a list of images

Parameters

image_list: list

List contain the file names to be processed

masterbias: None, '~numpy.ndarray', or '~ccdproc.CCDData'

A materbias frame to be subtracted from ccd.

Returns

masterflat: ccddata.CCDData

Combine master flat from the flats supplied in image_list

15.7 create_orderframe

pyhrs.**create_orderframe**(*data*, *first_order*, *xc*, *detect_kernal*, *smooth_length=15*, *y_start=0*, *y_limit=None*)

Create an order frame from from an observation.

A one dimensional detect_kernal is correlated with a column in the image. The kernal steps through y-space until a match is made. Once a best fit is found, the order is extracted to include all pixels that are detected to be part of that order. Once all pixels have been extracted, they are set to zero in the original frame. The detection kernal is updated by the new order detected

Parameters

data: ~numpy.ndarray

An image with the different orders illuminated. Any processing of this image should have been performed prior to running `create_orderframe`.

first_order: int

The first order to appear in the image starting from the bottom of the image

xc: int

The x-position to extract a 1-D map of the orders

detect_kern: ~numpy.ndarray

The initial detection kernal which have the shape of a single order.

smooth_length: int

The length to smooth the images by prior to processing them

y_start: int

The initial value to start searching for the first maximum

y_limit: int

The limit in y-positions for automatically finding the orders.

Returns

order_frame: ~numpy.ndarray

An image with each of the order identified by their number

Notes

Currently no orders are extrcted above y_limit and the code still needs to be updated to handle those higher orders

15.8 fit_order

pyhrs.**fit_order**(*data*, *detect_kernal*, *xc*, *order=3*, *ratio=0.5*)

Given an array and an overlapping detect_kernal, determine two polynomials that would outline the top and bottom of the order

Parameters

data: ~numpy.ndarray

Image of the orders

detect_kernal: ~numpy.ndarray

An array aligned with data that has the approximate outline of the order. The data shoud have a value of one for where the order is.

xc: int

x-position to determine the width of the order

order: int

Order to use for the polynomial fit.

ratio: float

Limit at which to determine an order. It is the ratio of the flux in the pixle to the flux at the peak.

Returns

y_l: `Polynomial1D`

A polynomial that outlines the bottom of the order

y_u: `Polynomial1D`

A polynomial that outlines the top of the order

15.9 fit_wavelength_solution

pyhrs.**fit_wavelength_solution**(*sol_dict*)

Determine the best fit solution and re-fit each line with that solution

The following steps are used to determine the best wavelength solution: 1. The coefficients of the solution to each row are fit by a line 2. The coefficients for each row are then replaced by the best-fit values 3. The wavelenght zeropoint is then re-calculated for each row

15.10 hrs_process

pyhrs.**hrs_process**(*image_name*, *ampsec=[]*, *oscansec=[]*, *trimsec=[]*, *masterbias=None*, *error=False*, *bad_pixel_mask=None*, *flip=False*, *rdnoise=None*, *oscan_median=True*, *oscan_model=None*)

Processing required for HRS observations. If the images have multiple

amps, then this will process each part of the image and recombine them into for the final results

Parameters

image_name: str

Name of file to be processed

ampsec: list

List of ampsections. This list should have the same length as the number of amps in the data set. The sections should be given in the format of fits_sections (see below).

oscansec: list

List of overscan sections. This list should have the same length as the number of amps in the data set. The sections should be given in the format of fits_sections (see below).

trimsec: list

List of overscan sections. This list should have the same length as the number of amps in the data set. The sections should be given in the format of fits_sections (see below).

error: boolean

If True, create an uncertainty array for ccd

masterbias: None, '~numpy.ndarray', or '~ccdproc.CCDData'

A materbias frame to be subtracted from ccd.

bad_pixel_mask: None or '~numpy.ndarray'

A bad pixel mask for the data. The bad pixel mask should be in given such that bad pixels havea value of 1 and good pixels a value of 0.

flip: boolean

If True, the image will be flipped such that the orders run from the bottom of the image to the top and the dispersion runs from the left to the right.

rdnoise: None or '~astropy.Quantity'

Read noise for the observations. The read noise should be in `electron`

oscan_median : bool, optional

If true, takes the median of each line. Otherwise, uses the mean

oscan_model : `Model`, optional

Model to fit to the data. If None, returns the values calculated by the median or the mean.

Returns

ccd: `CCDData`

Data processed and

Notes

The format of the `fits_section` string follow the rules for slices that are consistent with the FITS standard (v3) and IRAF usage of keywords like TRIMSEC and BIASSEC. Its indexes are one-based, instead of the python-standard zero-based, and the first index is the one that increases most rapidly as you move through the array in memory order, opposite the python ordering.

The 'fits_section' argument is provided as a convenience for those who are processing files that contain TRIMSEC and BIASSEC. The preferred, more pythonic, way of specifying the overscan is to do it by indexing the data array directly with the `overscan` argument.

15.11 iterfit1D

pyhrs.**iterfit1D**(*x*, *y*, *fitter*, *model*, *yerr=None*, *thresh=5*, *niter=5*)

Iteratively fit a function.

Outlyiers will have a reduced weight in the fit, and then the fit will be repeated niter times to determine the best fits

Parameters

x: numpy.ndarray

Arrray of x-values

y: numpy.ndarray

Arrray of y-values

fitter: ~astropy.modeling.fitting

Method to fit the model

model: ~astropy.modeling.model

A model to be fit

Returns

m: ~astropy.modeling.model

Model fit after reducing the weight of outlyiers

15.12 match_lines

pyhrs.**match_lines**(*xarr*, *farr*, *sw*, *sf*, *ws*, *rw=5*, *npoints=20*, *xlimit=1.0*, *slimit=1.0*, *wlimit=1.0*)

Match lines in the spectra with specific wavleengths

Match lines works by finding the closest peak based on the x-position transformed by ws that is within wlimit of a known line.

Parameters

xarr: numpy.ndarray

pixel positions

farr: numpy.ndarray

flux values at xarr positions

sw: numpy.ndarray

wavelengths of known arc lines

sf: numpy.ndarray

relative fluxes at those wavelengths

ws: function

Function converting xarr into wavelengths. It should be defined such that wavelength = ws(xarr)

rw: float

Radius around peak to extract for fitting the center

npoints: int

The maximum number of points to bright points to fit.

xlimit: float

Maximum shift in line centroid when fitting

slimit: float

Minimum scale for line when fitting

wlimit: float

Minimum separation in wavelength between peak and line

Returns

mx: numpy.ndarray

x-position for matched lines

mw: numpy.ndarray

Wavelength position for matched lines

15.13 ncor

pyhrs.**ncor**(*x*, *y*)

Calculate the normalized correlation of two arrays

Parameters

x: numpy.ndarray

Arrray of x-values

y: numpy.ndarray

Arrray of y-values

Returns

ncor: float

Normalize correctation value for two arrays

15.14 normalize_image

pyhrs.**normalize_image**(*data*, *func_init*, *mask*, *fitter=<class 'astropy.modeling.fitting.LinearLSQFitter'>*, *normalize=True*)

Normalize an HRS image.

The tasks takes an image and will fit a function to the overall shape to it. The task will only fit to the illuminated orders and if an order_frame is provided it will use that to identify the areas it should fit to. Otherwise, it will filter the image such that only the maximum areas are fit.

This function will then be divided out of the image and return a normalized image if requested.

Parameters

data: numpy.ndarray

Data to be normalized

mask: numpy.ndarray

If a numpy.ndarray, this will be used to determine areas to be used for the fit.

func_init: ~astropy.modeling.models

Function to fit to the image

fitter: ~astropy.modeling.fitting

Fitter function

normalize: boolean

If normalize is True, it will return data normalized by the function fit to it. If normalize is False, it will return an array representing the function fit to data.

Returns

ndata: numpy.ndarray

If normalize is True, it will return data normalized by the function fit to it. If normalize is False, it will return an array representing the function fit to data.

15.15 red_process

pyhrs.**red_process**(*infile*, *masterbias=None*, *error=None*, *rdnoise=None*)

Process a blue frame

15.16 test

pyhrs.**test**(*package=None*, *test_path=None*, *args=None*, *plugins=None*, *verbose=False*, *pastebin=None*, *remote_data=False*, *pep8=False*, *pdb=False*, *coverage=False*, *open_files=False*, ***kwargs*)

Run the tests using py.test. A proper set of arguments is constructed and passed to pytest.main.

Parameters

package : str, optional

The name of a specific package to test, e.g. 'io.fits' or 'utils'. If nothing is specified all default tests are run.

test_path : str, optional

Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.

args : str, optional

Additional arguments to be passed to pytest.main in the `args` keyword argument.

plugins : list, optional

Plugins to be passed to pytest.main in the `plugins` keyword argument.

verbose : bool, optional

Convenience option to turn on verbose output from py.test. Passing True is the same as specifying `'-v'` in `args`.

pastebin : {'failed','all',None}, optional

Convenience option for turning on py.test pastebin output. Set to `'failed'` to upload info for failed tests, or `'all'` to upload info for all tests.

remote_data : bool, optional

Controls whether to run tests marked with @remote_data. These tests use online data and are not run by default. Set to True to run these tests.

pep8 : bool, optional

Turn on PEP8 checking via the pytest-pep8 plugin and disable normal tests. Same as specifying `'--pep8 -k pep8'` in `args`.

pdb : bool, optional

Turn on PDB post-mortem analysis for failing tests. Same as specifying `'--pdb'` in `args`.

coverage : bool, optional

Generate a test coverage report. The result will be placed in the directory htmlcov.

open_files : bool, optional

Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Works only on platforms with a working `lsof` command.

parallel : int, optional

When provided, run the tests in parallel on the specified number of CPUs. If parallel is negative, it will use the all the cores on the machine. Requires the pytest-xdist plugin installed. Only available when using Astropy 0.3 or later.

kwargs

Any additional keywords passed into this function will be passed on to the astropy test runner. This allows use of test-related functionality implemented in later versions of astropy without explicitly updating the package template.

15.17 wavelength_calibrate_arc

pyhrs.**wavelength_calibrate_arc**(*arc*, *order_frame*, *slines*, *sfluxes*, *first_order*, *hrs_model*, *ws_init*, *fit_ws*, *y0=50*, *wavelength_shift=None*, *xlimit=1.0*, *slimit=1.0*, *wlimit=0.5*, *min_order=54*)

Wavelength calibrate an arc spectrum from HRS

'wavelength_calibrate_order' will be applied to each order in 'order_frame'a Once all orders have been processed, it will return an array where the wavelength is specified at each x- and y-position.

Parameters

arc: ~ccdproc.CCDData

Arc frame to be calibrated

order_frame: ~ccdproc.CCDData

Frame containting the positions of each of the orders

slines: numpy.ndarray

wavelengths of known arc lines

sfluxes: numpy.ndarray

relative fluxes at those wavelengths

first_order: int

First order to be processed

hrs_model: ~HRSModel

A model for the spectrograph for the given arc

ws_init: ~astropy.modeling.model

A initial model decribe the trasnformation from x-position to wavelength

fit_ws: ~astropy.modeling.fitting

Method to fit the model

y0: int

First row in which to determine the solution

wavelength_shift: ~astropy.modeling.model or None

For the row given by y0, this is the correction needed to be applied to the model wavelengths to provide a closer match to the observed arc.

npoints: int

The maximum number of points to bright points to fit.

xlimit: float

Maximum shift in line centroid when fitting

slimit: float

Minimum scale for line when fitting

wlimit: float

Minimum separation in wavelength between peak and line

15.18 wavelength_calibrate_order

pyhrs.**wavelength_calibrate_order**(*hrs*, *slines*, *sfluxes*, *ws_init*, *fit_ws*, *y0=50*, *npoints=30*, *xlimit=1.0*, *slimit=1.0*, *wlimit=0.5*)

Wavelength calibration of a single order from the HRS arc spectra

The calibration proceeds through following steps: 1. Curvature due to the optical distortion is removed from the spectra and

a square representation of the 2D spectra is created. Only integer shifts are applied to the data

2.A model of the spectrograph is created based on the order, camera, and xpos offset that are supplied.

3.In each row of the data, peaks are extracted and matched with a line in the atlas of wavelengths that is provided (slines, sflux). For the details of the matching process, see the match_arc function.

4.Once the first set of peaks and lines are matched up, a new solution is calculated for the given row. Then the processes of matching lines and determining a wavelength solution is repeated. The best result from each line is saved.

5.Using all of the matched lines from all lines, a 'best' solution is determined. Everything but the zeroth order parameter of the fit is fixed to a slowly varying value based on the overall solution to all lines. See fit_solution for more details.

6.Based on the best solution found, the process is repeated for each row but only determing the zeropoint.

7.Based on the solution found, a wavelength is assigned to each pixel

Parameters

hrs: ~HRSOrder

Object describing a single HRS order. It should already contain the defined order and the flux from the arc for that order

slines: numpy.ndarray

wavelengths of known arc lines

sfluxes: numpy.ndarray

relative fluxes at those wavelengths

ws_init: ~astropy.modeling.model

A initial model decribe the trasnformation from x-position to wavelength

fit_ws: ~astropy.modeling.fitting

Method to fit the model

y0: int

First row for determine the solution

npoints: int

The maximum number of points to bright points to fit.

xlimit: float

Maximum shift in line centroid when fitting

slimit: float

Minimum scale for line when fitting

wlimit: float

Minimum separation in wavelength between peak and line

Returns

hrs: ~HRSOrder

An HRSOrder with a calibrated wavelength property

15.19 xcross_fit

pyhrs.**xcross_fit**(*warr*, *farr*, *sw_arr*, *sf_arr*, *dw=1.0*, *nw=100*)

Calculate a zeropoint shift between the observed arc
and the line list of values

Parameters

warr: numpy.ndarray

Estimated wavelength for arc

farr: numpy.ndarray

Flux at each arc position

sw_arr: numpy.ndarray

Wavelength of known lines

sf_arr: numpy.ndarray

Flux of known lines

dw: float

Value to search over. The search will be done from -dw to +dw

nw: int

Number of steps in the search

Returns

warr: numpy.ndarray

Wavelength after correcting for shift from fiducial values

CCD([name, height, width, xpos, ypos, ...])	Defines a CCD by x and y position, size, and pixel size.
Detector([name, ccd, zpos, xpos, ypos, ...])	A class that describing the Detector.
Grating([name, spacing, order, height, ...])	A class that describing gratings.
HRSModel([grating_name, camera_name, slit, ...])	HRSModel is a class that describes the High Resolution Specotrgraph on SALT
HRSOrder(order[, region, flux, wavelength, ...])	A class describing a single order for a High Resolutoin Spectrograph observation.
Optics([name, diameter, focallength, width, ...])	A class that describing optics.
Slit([name, height, width, zpos, xpos, ...])	A class that describing the slit.
Spectrograph([camang, gratang, grating, ...])	A class describing a spectrograph and functions related to a spectrograph.
SpectrographError	Exception Raised for Spectrograph errors

15.20 HRSModel

class pyhrs.**HRSModel**(*grating_name='hrs'*, *camera_name='hrdet'*, *slit=2.0*, *order=83*, *gamma=None*, *xbin=1*, *ybin=1*, *xpos=0.0*, *ypos=0.0*)

Bases: PySpectrograph.Spectrograph.Spectrograph.Spectrograph

HRSModel is a class that describes the High Resolution Specotrgraph on SALT

Methods Summary

alpha([da])	Return the value of alpha for the spectrograph
beta([db])	Return the value of beta for the spectrograph
get_wavelength(xarr[, gamma])	For a given spectrograph configuration, return the wavelength coordinate associa
set_camera([name, focallength])	
set_collimator([name, focallength])	
set_detector([name, geom, xbin, ybin, xpos, ...])	
set_grating([name, order])	
set_order(order)	
set_slit([slitang])	
set_telescope([name])	

Methods Documentation

alpha(*da=0.0*)

Return the value of alpha for the spectrograph

beta(*db=0*)

Return the value of beta for the spectrograph

Beta_o=(1+fA)*(camang)-gratang+beta_o

get_wavelength(*xarr*, *gamma=0.0*)

For a given spectrograph configuration, return the wavelength coordinate associated with a pixel coordinate.

xarr: 1-D Array of pixel coordinates gamma: Value of gamma for the row being analyzed

returns an array of wavelengths in mm

set_camera(*name='hrdet'*, *focallength=None*)

set_collimator(*name='hrs'*, *focallength=2000.0*)

set_detector(*name='hrdet'*, *geom=None*, *xbin=1*, *ybin=1*, *xpos=0*, *ypos=0*)

set_grating(*name=None*, *order=83*)

set_order(*order*)

set_slit(*slitang=2.2*)

set_telescope(*name='SALT'*)

15.21 HRSOrder

class pyhrs.**HRSOrder**(*order*, *region=None*, *flux=None*, *wavelength=None*, *flux_unit=None*, *wavelength_unit=None*, *order_type=None*)

Bases: object

A class describing a single order for a High Resolutoin Spectrograph observation.

Parameters

order: integer

Order of the HRS observations

region: list, tuple, or '~numpy.ndarray'

region is an object that contains coordinates for pixels in the image which are part of this order. It should be a list containing two arrays with the coordinates listed in each array.

flux: '~numpy.ndarray'

Fluxes corresponding to each pixel coordinate in region.

wavelength: '~numpy.ndarray'

Wavelengths corresponding to each pixel coordinate in region.

order_type: str

Type of order for the Order of the HRS observations

flux_unit: '~astropy.units.UnitBase' instance or str, optional

The units of the flux.

wavelength_unit: '~astropy.units.UnitBase' instance or str, optional

The units of the wavelength

Attributes Summary

flux	
flux_unit	
order	
order_type	
region	
wavelength	
wavelength_unit	

Methods Summary

extract_spectrum()	Extract 1D spectrum from the information provided so far and
set_flux_from_array(data[, flux_unit])	Given an array of data of fluxes, set the fluxes for

Continued on

Table 15.5 – continued from previous page

set_order_from_array(data)	Given an array of data which has an order specified at each pixel,
set_wavelength_from_array(data, wavelength_unit)	Given an array of wavelengths, set the wavelength for each pixel coordinate
set_wavelength_from_model(model, params, ...)	Given an array of wavelengths, set the wavelength for each pixel coordinate

Attributes Documentation

flux

flux_unit

order

order_type

region

wavelength

wavelength_unit

Methods Documentation

extract_spectrum()
Extract 1D spectrum from the information provided so far and createa Spectrum1D object

set_flux_from_array(*data*, *flux_unit=None*)

Given an array of data of fluxes, set the fluxes for
the region at the given order for HRSOrder

Parameters
data: '~numpy.ndarray'

data is an 2D array with a flux value specified at each pixel.

flux_unit: '~astropy.units.UnitBase' instance or str, optional

The units of the flux.

set_order_from_array(*data*)

Given an array of data which has an order specified at each pixel,
set the region at the given order for HRSOrder

Parameters
data: '~numpy.ndarray'

data is an 2D array with an order value specified at each pixel. If no order is available for a given pixel, the pixel should have a value of zero.

`set_wavelength_from_array`(*data*, *wavelength_unit*)

Given an array of wavelengths, set the wavelength for
each pixel coordinate in `region`.

Parameters

data: ‘~numpy.ndarray‘

data is an 2D array with a wavelength value specified at each pixel

wavelength_unit: ‘~astropy.units.UnitBase‘ instance or str, optional

The units of the wavelength

`set_wavelength_from_model`(*model*, *params*, *wavelength_unit*, ***kwargs*)

Given an array of wavelengths, set the wavelength for
each pixel coordinate in `region`.

Parameters

model: function

model is a callable function that will create a corresponding wavelength for each pixel in `region`. The function can either be 1D or 2D. If it is 2D, the x-coordinate should be the first argument.

params: ‘~numpy.ndarray‘

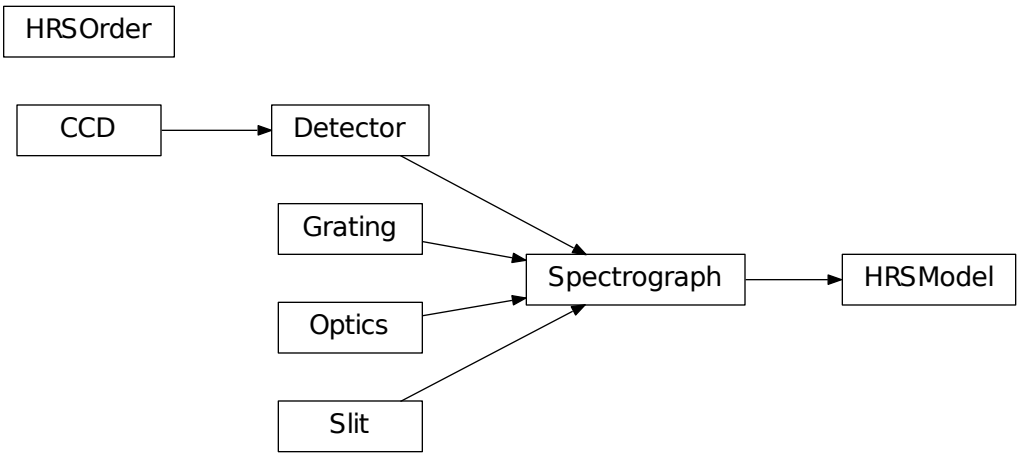
Either a 1D or 2D list of parameters with the number of elements corresponding to the number of pixles. Typically, if model is a 1D function, this would be the x-coordinated from `region`. Otherwise, this would be expected to be `region`.

wavelength_unit: ‘~astropy.units.UnitBase‘ instance or str, optional

The units of the wavelength

****kwargs:**

All additional keywords to be passed to model


HRSOrder
CCD
Detector
Grating
Spectrograph
HRSModel
Optics
Slit

p