aesop Documentation

Release 0.0.dev089

Brett M. Morris, Trevor Dorn-Wallenstein

Dec 11, 2018

Contents:

I Installing aesop 3

II IRAF 7

III Getting started 11

IV Reference/API 21

V What is aesop? 39

VI Typical Workflow 43

VII Indices and tables 47

aesop is a data reduction toolkit for echelle spectra from the Astrophysical Research Consortium (ARC) 3.5 m Telescope Echelle Spectrograph (ARCES) at Apache Point Observatory (APO). You can view the source code and submit issues via GitHub.

Part I

Installing aesop

Clone the aesop repository from GitHub, and go into the top-level aesop directory:

```
git clone https://github.com/bmorris3/aesop
cd aesop
```

Install aesop with:

```
python setup.py install
```

You can ensure that the package was installed successfully by running the tests from a Python interpreter with the following commands:

```
import aesop
aesop.test()
```

Part II

IRAF

CHAPTER 1

Contents

- *Downloading and installing AstroConda with IRAF*
- *Preparing your data*
- *Reducing your data*

1.1 Downloading and installing AstroConda with IRAF

Go to the AstroConda installation page, and follow the instructions to download AstroConda with IRAF. Unfortunately this only works with Python 2.7.

1.2 Preparing your data

`ReduceARCES` will expect your flats in the red to start with "redflat", your flats in the blue to start with "blueflat", your biases to start with "bias", and your ThAr lamp exposures to start with "ThAr".

1.3 Reducing your data

First, make sure your iraf27 (or whatever you named it when installing AstroConda with IRAF) environment is activated by doing

```
source activate iraf27
```

from a bash terminal.

Copy all of the files in the `aesop/iraf` directory into the directory where your raw data and cals are. From that directory, do

```
mkiraf
```

which will create `login.cl` in that directory. If prompted to specify a terminal, say `xgterm`. Edit `login.cl` to include the following imports

```
imred
ccdred
echelle
crutil
astutil
images
imgeom
twodspec
apextract
onedspec
```

Next, open an xgterm by doing `xgterm`, and do

```
cl
cl < ReduceARCES.cl
```

This will extract the spectrum from your data. When prompted to edit the parameters of the tasks that `ReduceARCES.cl` uses, simply do `:q`. You'll have to do this twice at the beginning of the reduction and a few more times in the middle.

Now in order to use `aesop`, follow the tutorial in *Getting started*

Part III

Getting started

CHAPTER 2

Contents

- *Opening example spectra*
- *Normalizing your spectra*
- *Merge all orders into a 1D spectrum*
- *Putting it all together*

2.1 Opening example spectra

If you want to do this with your own data, make sure it is reduced by following the steps in *IRAF*

Example ARCES spectra are available online for you to work with. You can download them via Python like this:

```
>>> from astropy.utils.data import download_file

>>> target_url = 'http://staff.washington.edu/bmmorris/docs/KIC8462852.0065.wfrmcpc.fits'
>>> spectroscopic_standard_url = 'http://staff.washington.edu/bmmorris/docs/BD28_4211.0034.wfrmcpc.fits'

>>> target_path = download_file(target_url, show_progress=False)
>>> standard_path = download_file(spectroscopic_standard_url, show_progress=False)
```

This will download temporary copies of the famous Kepler target KIC 8462852, also known as Boyajian's Star, and spectroscopic standard O star BD+28 4211. We first create an `EchelleSpectrum` object for each star:

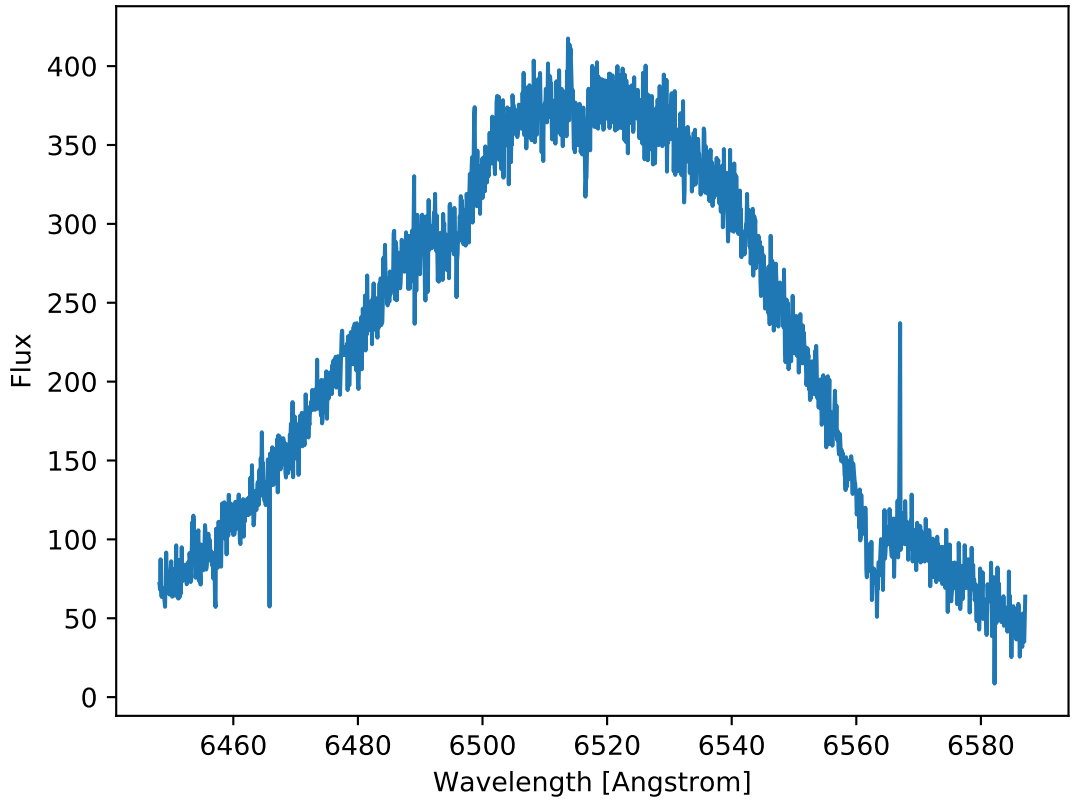
```
>>> from aesop import EchelleSpectrum

>>> target_spectrum = EchelleSpectrum.from_fits(target_path)
>>> standard_spectrum = EchelleSpectrum.from_fits(standard_path)
```

You can check basic metadata for an `EchelleSpectrum` object by printing it:

```
>>> print(target_spectrum)
<EchelleSpectrum: 107 orders, 3506.8-10612.5 Angstrom>
```

The EchelleSpectrum object behaves a bit like a Python list – it supports indexing, where the index counts the order number, starting with index zero for the order with the shortest wavelengths. Elements of the EchelleSpectrum are Spectrum1D objects. Suppose you want to make a quick plot of the 73rd order of the target's echelle spectrum:

```
order73 = target_spectrum[73]
order73.plot() # doctest: +SKIP
```



You can see the H-alpha absorption in this O star at 6562 Angstroms.

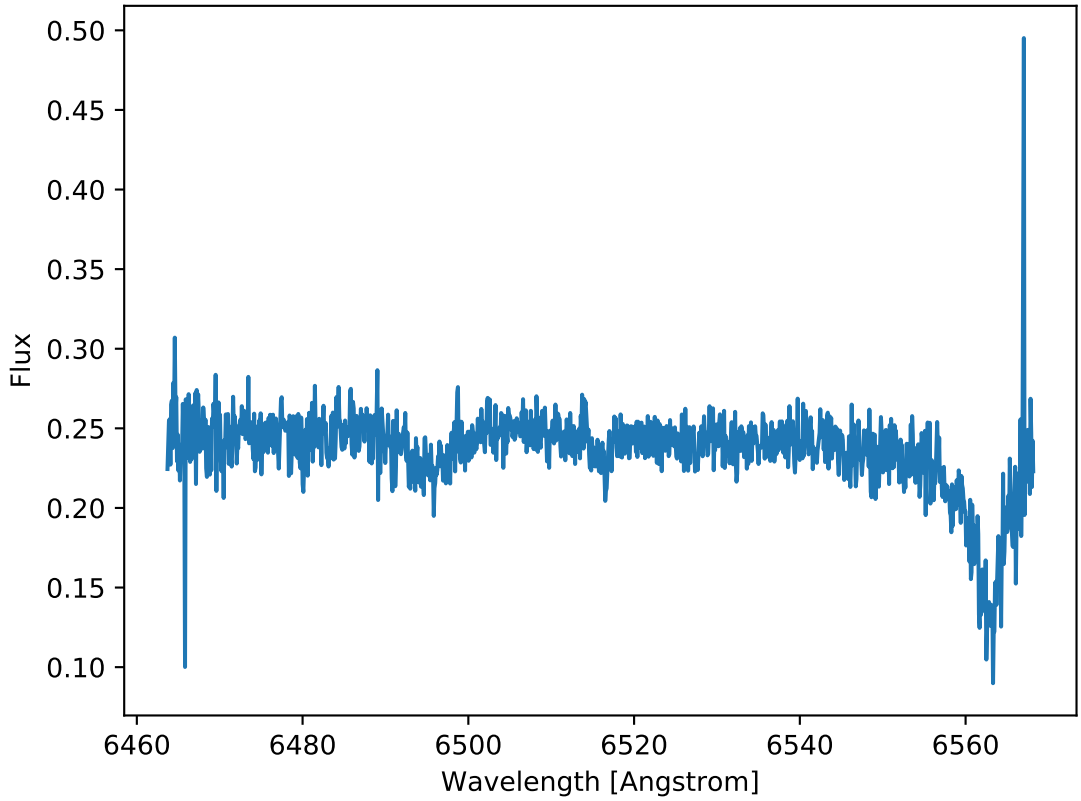
2.2 Normalizing your spectra

You can continuum-normalize your echelle spectra with two methods:

- continuum_normalize_from_standard will remove the blaze function from each echelle order by fitting polynomials to the continuum of each order of a spectroscopic standard star, and then remove those polynomials from each order of the target star
- continuum_normalize_lstsq will attempt to remove the blaze function from each echelle order by using a robust least-squares fit to the continuum in each order of the target spectrum – no spectroscopic standard observation is required by this method.

Let's first use continuum_normalize_from_standard on our previous example to see order containing the H-alpha line after the blaze function has been mostly removed:

```
>>> target_spectrum.continuum_normalize_from_standard(standard_spectrum,
...                                                   polynomial_order=8)
>>> target_spectrum[73].plot()
```



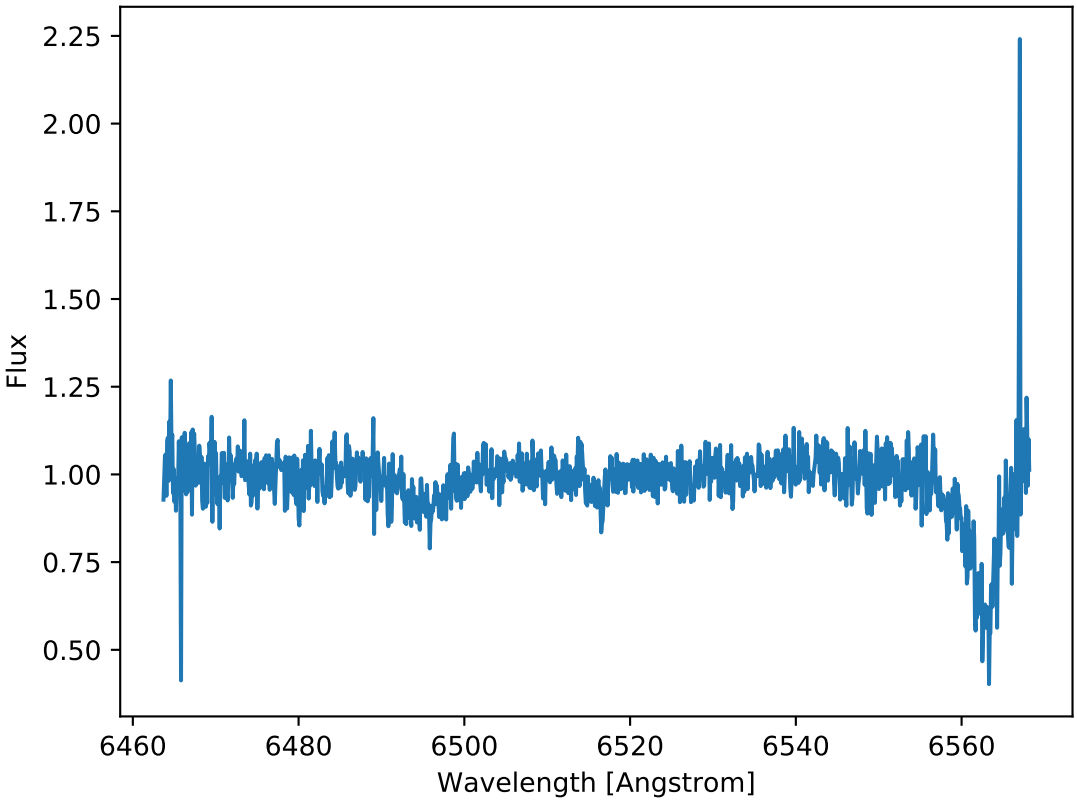
As you can see in this example, the standard star normalization will approximately flatten the continuum, but not normalize it to unity. We can now flatten the continuum and normalize it to unity with the other continuum normalization method, `continuum_normalize_lstsq`:

```
>>> target_spectrum.continuum_normalize_lstsq(polynomial_order=2)
>>> target_spectrum[73].plot()
```

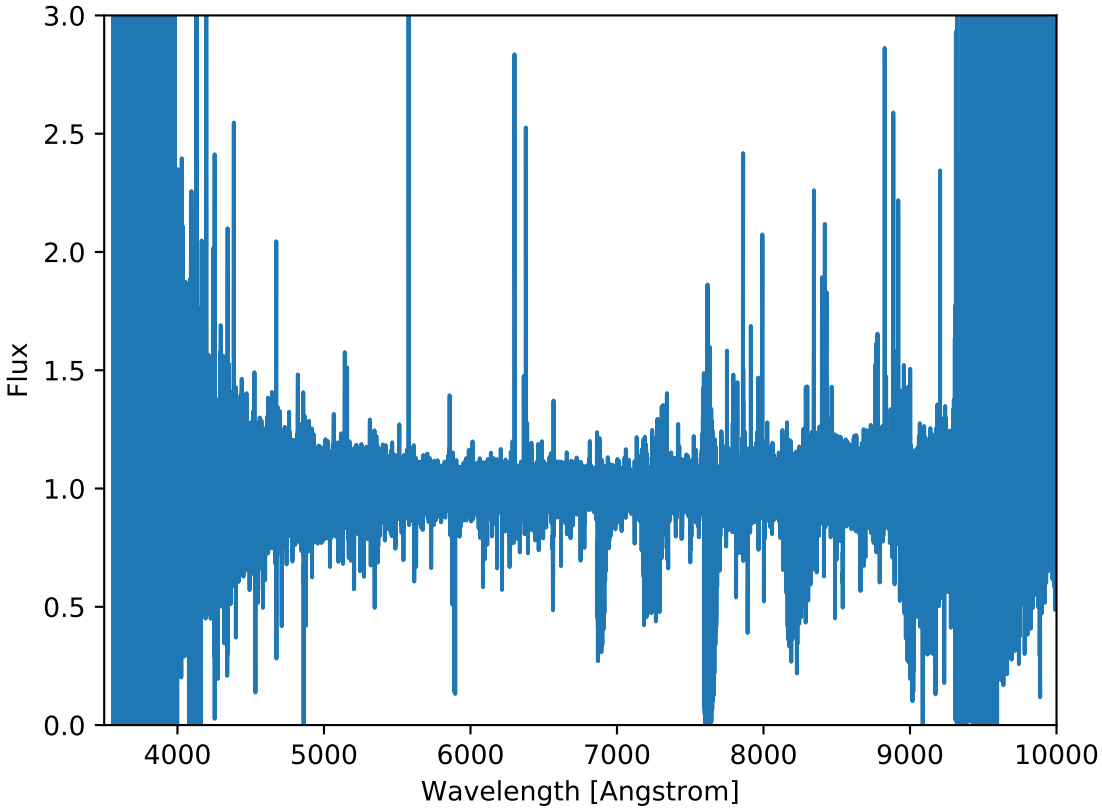
2.3 Merge all orders into a 1D spectrum

Now that you have a great normalized echelle spectrum, let's collapse the echelle spectrum down to one, big 1D spectrum, using `to_Spectrum1D`, which will give us a `Spectrum1D` object:

```
>>> spec1d = target_spectrum.to_Spectrum1D()
>>> print(spec1d)
<Spectrum1D: 3561.8-10390.9 Angstrom>

>>> spec1d.plot()
```


2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
Flux
6460
6480
6500
6520
6540
6560
Wavelength [Angstrom]

Of course, this plot is going to look a bit bonkers because there is a lot of noise in the extreme red and blue, cosmic rays here and there, and whopping telluric absorption. Here's what it looks like:



2.4 Putting it all together

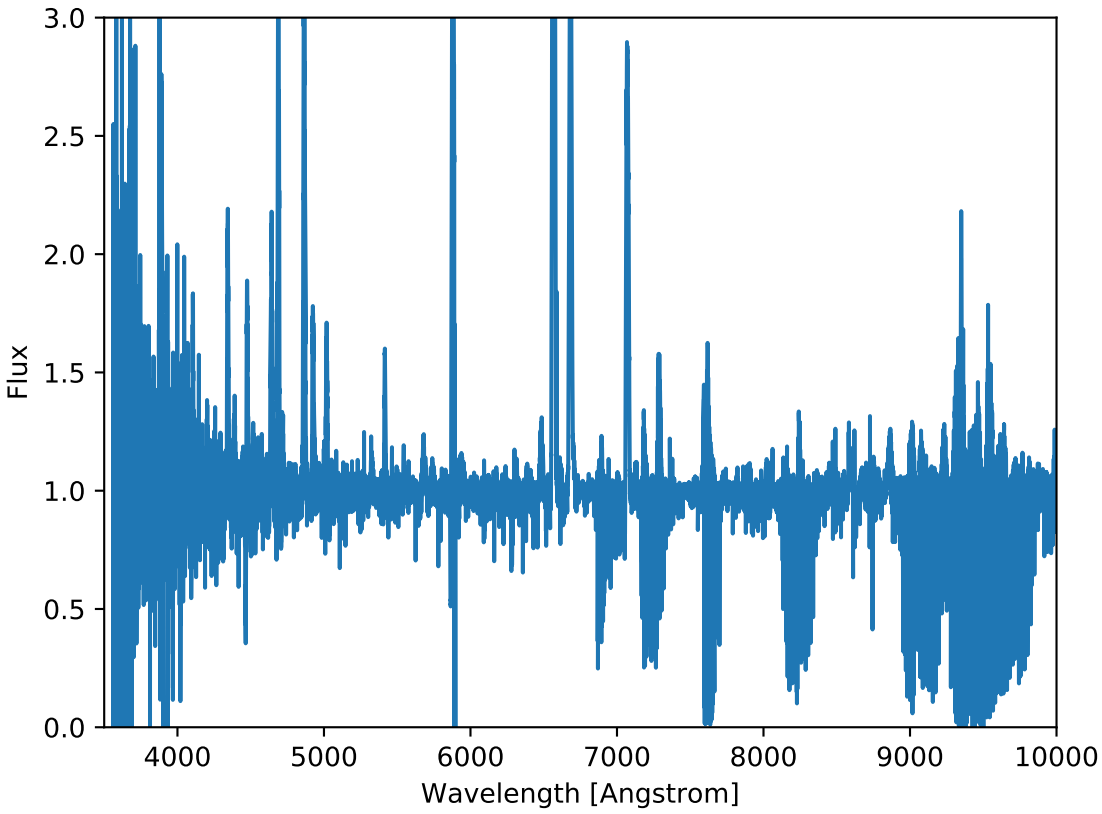
Now let's do all of this analysis on a very different star – this time a Wolf-Rayet star:

```
>>> # Download example spectra
>>> from astropy.utils.data import download_file

>>> target_url = 'http://staff.washington.edu/tzdw/wr_echelle/wr124.0001.wfrmcpc.fits'
>>> spectroscopic_standard_url = 'http://staff.washington.edu/tzdw/wr_echelle/HIP107864.0003.wfrmcpc.
↪fits'

>>> target_path = download_file(target_url, show_progress=False)
>>> standard_path = download_file(spectroscopic_standard_url, show_progress=False)

>>> # Open those example spectra with aesop
>>> from aesop import EchelleSpectrum

>>> target_spectrum = EchelleSpectrum.from_fits(target_path)
>>> standard_spectrum = EchelleSpectrum.from_fits(standard_path)
```

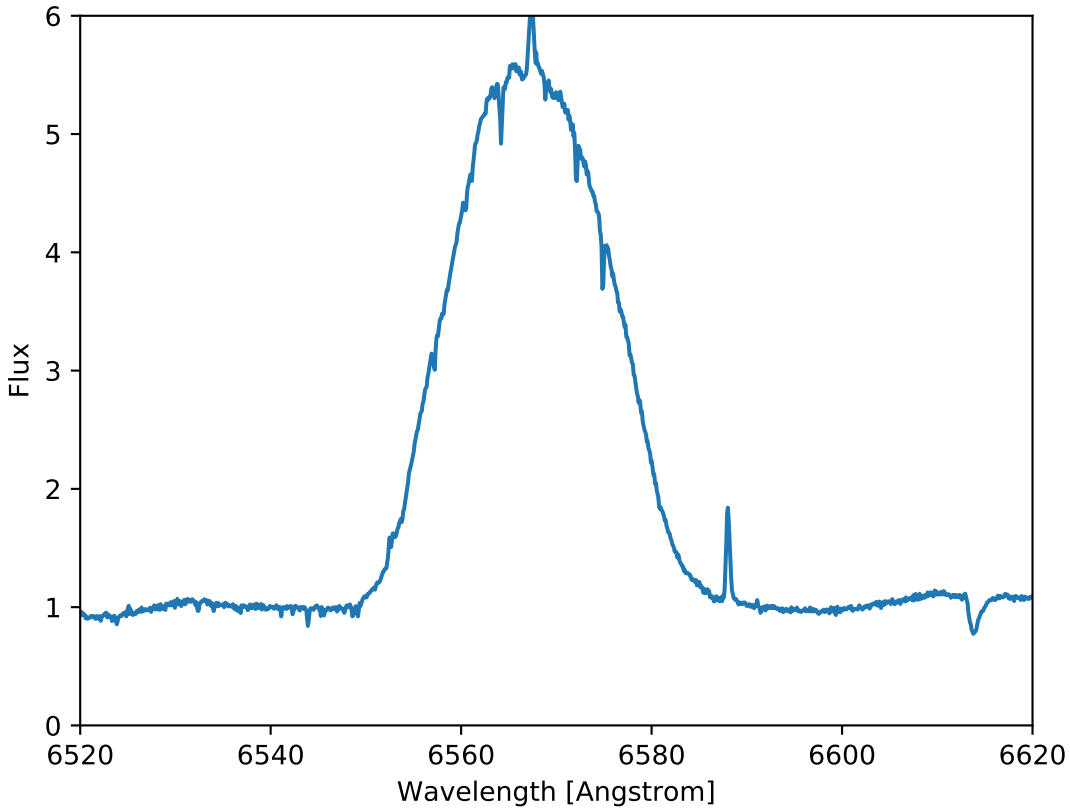
(continues on next page)

(continued from previous page)

```
>>> # Continuum normalize the spectra against a standard, and with lstsq
>>> target_spectrum.continuum_normalize_from_standard(standard_spectrum,
...                                                   polynomial_order=8)
>>> target_spectrum.continuum_normalize_lstsq(polynomial_order=2)

>>> # Plot the concatenated 1D spectrum
>>> spec1d = target_spectrum.to_Spectrum1D()
>>> spec1d.plot()
```



But note that these spectra have lots of emission, which our continuum normalization preserves:


Flux
Wavelength [Angstrom]
0
1
2
3
4
5
6
6520
6540
6560
6580
6600
6620

Part IV

Reference/API

CHAPTER 3

aesop Package

3.1 Functions

cross_corr(target_spectrum, model_spectrum, ...)	Cross correlate an observed spectrum with a model.
get_phoenix_model_spectrum(T_eff[, log_g, cache])	Download a PHOENIX model atmosphere spectrum for a star with given properties.
get_spectrum_mask(spectrum[, cutoff, plot])	Fit the raw, pre-normalized spectrum's blaze function with a Gaussian.
glob_spectra_paths(data_dir, target_names)	Collect paths to spectrum FITS files.
integrate_spectrum_trapz(spectrum, ...[, ...])	Integrate the area under a spectrum.
interpolate_spectrum(spectrum, new_wavelengths)	Linearly interpolate a spectrum onto a new wavelength grid.
json_to_stars(json_path)	Loads JSON archive into list of StarProps objects.
query_for_T_eff(identifier)	Get the approximate effective temperature of a star.
query_for_spectral_type(identifier[, ...])	Search SIMBAD for the spectral type of a star.
slice_spectrum(spectrum, min_wavelength, ...)	Return a slice of a spectrum on a smaller wavelength range.
stars_to_json(star_list[, output_path])	Save list of stellar properties to a JSON file.
test([package, test_path, args, plugins, ...])	Run the tests using py.test.
uncalibrated_s_index(spectrum[, plots])	Calculate the uncalibrated S-index from an Echelle spectrum.

3.1.1 cross_corr

aesop.**cross_corr**(*target_spectrum*, *model_spectrum*, *kernel_width*)

Cross correlate an observed spectrum with a model.

Convolve the model with a Gaussian kernel.

Parameters

target_spectrum : Spectrum1D

Observed spectrum of star

model_spectrum : `Spectrum1D`

Model spectrum of star

kernel_width : float

Smooth the model spectrum with a kernel of this width, in units of the wavelength step size in the model

Returns

———-

wavelength_shift : `Quantity`

Wavelength shift required to shift the target spectrum to the rest-frame

3.1.2 get_phoenix_model_spectrum

aesop.**get_phoenix_model_spectrum**(*T_eff*, *log_g=4.5*, *cache=True*)

Download a PHOENIX model atmosphere spectrum for a star with given properties.

Parameters

T_eff : float

Effective temperature. The nearest grid-temperature will be selected.

log_g : float

This must be a log g included in the grid for the effective temperature nearest `T_eff`.

cache : bool

Cache the result to the local astropy cache. Default is `True`.

Returns

spectrum : `Spectrum1D`

Model spectrum

3.1.3 get_spectrum_mask

aesop.**get_spectrum_mask**(*spectrum*, *cutoff=1.5*, *plot=False*)

Fit the raw, pre-normalized spectrum's blaze function with a Gaussian. Use a

Parameters

spectrum : `Spectrum1D`

Spectrum to fit

cutoff : float

Mask channels greater than `cuttoff`-sigma away from the peak flux.

Returns

mask : `ndarray`

Mask that excludes channels greater than `cuttoff`-sigma away from the peak flux.

3.1.4 glob_spectra_paths

aesop.**glob_spectra_paths**(*data_dir*, *target_names*)
Collect paths to spectrum FITS files.

Parameters
data_dir : str or list

Paths to the directories containing spectrum FITS files

target_names : list

String patterns that match the beginning of files with targets to collect.

Returns
spectra_paths : list

List of paths to spectrum FITS files

3.1.5 integrate_spectrum_trapz

aesop.**integrate_spectrum_trapz**(*spectrum*, *center_wavelength*, *width*, *weighting=False*, *plot=False*)
Integrate the area under a spectrum.

Parameters
spectrum : EchelleSpectrum

Spectrum to integrate under

center_wavelength : Quantity

Center of region to integrate

width : Quantity

Width about the center to integrate

wavelength_offset : float

Offset wavelengths by this amount, which is useful if a wavelength solution refinement as been made.

weighting : bool

Apply a triangular weighting function to the fluxes

Returns
integral : float

Integral under the spectrum

error : float

Square-root of the sum of the fluxes within the bounds of the integral

3.1.6 interpolate_spectrum

aesop.**interpolate_spectrum**(*spectrum*, *new_wavelengths*)
Linearly interpolate a spectrum onto a new wavelength grid.

Parameters
spectrum : Spectrum1D

Spectrum to interpolate onto new wavelengths

new_wavelengths : `Quantity`

New wavelengths to interpolate the spectrum onto

Returns

interp_spec : `Spectrum1D`

Interpolated spectrum.

3.1.7 json_to_stars

aesop.**json_to_stars**(*json_path*)

Loads JSON archive into list of `StarProps` objects.

Parameters

json_path : str

Path to saved stellar properties

Returns

stars : list of `StarProps`

List of stellar properties.

3.1.8 query_for_T_eff

aesop.**query_for_T_eff**(*identifier*)

Get the approximate effective temperature of a star.

Query SIMBAD for the spectral type of the target, convert spectral type to approximate effective temperature, in general. If the target is in the K2 EPIC, use the EPIC Teff.

Parameters

identifier : str

Name of target

Returns

T_eff : int

Approximate effective temperature of the star.

3.1.9 query_for_spectral_type

aesop.**query_for_spectral_type**(*identifier*, *only_first_two_characters=True*, *default_sptype=u'G0'*)

Search SIMBAD for the spectral type of a star.

If no spectral type is found, the default return value is `"G0"`.

Parameters

identifier : str

Name of target

only_first_two_characters : bool

Return only first two characters of spectral type?

default_sptype : str

Spectral type returned when none is found on SIMBAD

Returns

sptype : str

Spectral type of the star.

3.1.10 slice_spectrum

aesop.**slice_spectrum**(*spectrum*, *min_wavelength*, *max_wavelength*, *norm=None*)

Return a slice of a spectrum on a smaller wavelength range.

Parameters

spectrum : `Spectrum1D`

Spectrum to slice.

min_wavelength : `Quantity`

Minimum wavelength to include in new slice

max_wavelength : `Quantity`

Maximum wavelength to include in new slice

norm : float

Normalize the new slice fluxes by `norm` divided by the maximum flux of the new slice.

Returns

sliced_spectrum : `Spectrum1D`

3.1.11 stars_to_json

aesop.**stars_to_json**(*star_list*, *output_path=u'star_data.json'*)

Save list of stellar properties to a JSON file.

Parameters

star_list : list of `StarProps`

Star properties to save to json

output_path : str

File path to output

3.1.12 test

aesop.**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. Requires the psutil package.

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.

3.1.13 uncalibrated_s_index

aesop.**uncalibrated_s_index**(*spectrum*, *plots=False*)

Calculate the uncalibrated S-index from an Echelle spectrum.

Parameters

spectrum : `EchelleSpectrum`

Normalized target spectrum

Returns

s_ind : `SIndex`

S-index. This value is intrinsic to the instrument you're using.

3.2 Classes

`EchelleSpectrum`(spectrum_list[, header, ...])	Echelle spectrum of one or more spectral orders.
`FitParameter`(value[, err_upper, err_lower, ...])	Fit results for a fitting parameter, with asymmetrical errors.
`Measurement`([value, err, time, default_err, ...])	
`Spectrum1D`([wavelength, flux, name, mask, ...])	Simple 1D spectrum object.
`StarProps`([name, s_apo, s_mwo, time])	S-index properties for a star

3.2.1 EchelleSpectrum

class aesop.**EchelleSpectrum**(*spectrum_list*, *header=None*, *name=None*, *fits_path=None*, *time=None*)

Bases: `object`

Echelle spectrum of one or more spectral orders.

The spectral orders will be indexed in order of increasing wavelength.

Parameters

spectrum_list : list of `Spectrum1D` objects

List of `Spectrum1D` objects for the spectra in each echelle order.

header : `astropy.io.fits.header.Header` (optional)

FITS header object associated with the echelle spectrum.

name : str (optional)

Name of the target or a name for the spectrum

fits_path : str (optional)

Path where FITS file was opened from.

time : `Time` (optional)

Time at which the spectrum was taken

Methods Summary

`barycentric_correction`([time, skycoord, ...])	Barycentric velocity correction, code from StuartLittlefair (https://gist.github.com/StuartLittlefair/5aaf476c5d7b52d20aa9544cfaa936a1)
`continuum_normalize_from_standard`(..., [, ...])	Normalize the spectrum by a polynomial fit to the standard's spectrum.
`continuum_normalize_lstsq`(polynomial_order)	Normalize the spectrum with a robust least-squares polynomial fit to the spectrum of each order.

Continued on next page

Table 3 – continued from previous page

fit_order(spectral_order, polynomial_order)	Fit a spectral order with a polynomial.
from_fits(path)	Load an echelle spectrum from a FITS file.
get_order(order)	Get the spectrum from a specific spectral order
offset_wavelength_solution(wavelength_offset)	Offset the wavelengths by a constant amount in each order.
predict_continuum(spectral_order, fit_params)	Predict continuum spectrum given results from a polynomial fit from EchelleSpectrum.fit_order.
rv_wavelength_shift(spectral_order[, T_eff, ...])	Solve for the radial velocity wavelength shift.
rv_wavelength_shift_ransac([min_order, ...])	Solve for the radial velocity wavelength shift of every order in the echelle spectrum, then do a RANSAC (outlier rejecting) linear fit to the wavelength correction between orders min_order and max_order.
to_Spectrum1D()	Convert this echelle spectrum into a simple 1D spectrum.

Methods Documentation

barycentric_correction(*time=None*, *skycoord=None*, *location=None*)

Barycentric velocity correction, code from StuartLittlefair (https://gist.github.com/StuartLittlefair/5aaf476c5d7b52d20aa9544cfaa936a1)

Uses the ephemeris set with `astropy.coordinates.solar_system_ephemeris.set` for corrections.

For more information see solar_system_ephemeris.

Will attempt to get the necessary info from the header if possible, otherwise requires time, skycoord, and location parameters to be set.

Parameters

time : Time

The time of observation, optional

skycoord: '~astropy.coordinates.SkyCoord'

The sky location to calculate the correction for, optional.

location: '~astropy.coordinates.EarthLocation', optional

The location of the observatory to calculate the correction for.

Returns

barycentric_velocity : Quantity

The velocity correction that was added to the wavelength arrays of each order.

continuum_normalize_from_standard(*standard_spectrum*, *polynomial_order*, *only_orders=None*, *plot_masking=False*, *plot_fit=False*)

Normalize the spectrum by a polynomial fit to the standard's spectrum.

Parameters

standard_spectrum : EchelleSpectrum

Spectrum of the standard object

polynomial_order : int

Fit the standard's spectrum with a polynomial of this order

only_orders : ndarray

Only do the continuum normalization for these echelle orders.

plot_masking : bool

Plot the masked-out low S/N regions

plot_fit : bool

Plot the polynomial fit to the standard star spectrum

continuum_normalize_lstsq(*polynomial_order*, *only_orders=None*, *plot=False*, *fscale_mad_factor=0.2*)

Normalize the spectrum with a robust least-squares polynomial fit to the spectrum of each order.

Parameters

polynomial_order : int

Fit the standard's spectrum with a polynomial of this order

only_orders : `ndarray` (optional)

Only do the continuum normalization for these echelle orders.

plot_masking : bool (optional)

Plot the masked-out low S/N regions

plot_fit : bool (optional)

Plot the polynomial fit to the standard star spectrum

fscale_mad_factor : float (optional)

The robust least-squares fitter will reject outliers by keeping the standard deviation of inliers close to `fscale_mad_factor` times the median absolute deviation (MAD) of the fluxes.

fit_order(*spectral_order*, *polynomial_order*, *plots=False*)

Fit a spectral order with a polynomial.

Ignore fluxes near the CaII H & K wavelengths.

Parameters

spectral_order : int

Spectral order index

polynomial_order : int

Polynomial order

Returns

fit_params : `ndarray`

Best-fit polynomial coefficients

classmethod from_fits(*path*)

Load an echelle spectrum from a FITS file.

Parameters

path : str

Path to the FITS file

get_order(*order*)

Get the spectrum from a specific spectral order

Parameters
order : int

Echelle order to return

Returns
spectrum : `Spectrum1D`

One order from the echelle spectrum

offset_wavelength_solution(*wavelength_offset*)
Offset the wavelengths by a constant amount in each order.

Parameters
wavelength_offset : `Quantity` or list

Offset the wavelengths by this amount. If `wavelength_offset` is a list, each value will be treated as an offset for on echelle order, otherwise a single `wavelength_offset` will be applied to every order.

predict_continuum(*spectral_order*, *fit_params*)
Predict continuum spectrum given results from a polynomial fit from `EchelleSpectrum.fit_order`.

Parameters
spectral_order : int

Spectral order index

fit_params : `ndarray`

Best-fit polynomial coefficients

Returns
flux_fit : `ndarray`

Predicted flux in the continuum for this order

rv_wavelength_shift(*spectral_order*, *T_eff=None*, *plot=False*)
Solve for the radial velocity wavelength shift.

Parameters
spectral_order : int

Echelle spectrum order to shift

rv_wavelength_shift_ransac(*min_order=10*, *max_order=45*, *T_eff=4700*)
Solve for the radial velocity wavelength shift of every order in the echelle spectrum, then do a RANSAC (outlier rejecting) linear fit to the wavelength correction between orders `min_order` and `max_order`.

Parameters
min_order : int

Index of the bluest order to fit in the wavelength correction

max_order : int

Index of the reddest order to fit in the wavelength correction

T_eff : int

Effective temperature of the PHOENIX model atmosphere to use in the cross-correlation.

Returns
wl : `Quantity`

Wavelength corrections for each order.

to_Spectrum1D()

Convert this echelle spectrum into a simple 1D spectrum.

In wavelength regions where two spectral orders overlap, take the mean of the overlapping region.

Parameters

mad_outlier_factor : float

Mask positive outliers `max_outlier_factor` times the median absolute deviation plus the median of the fluxes.

Returns

spectrum : `Spectrum1D`

Simple 1D spectrum.

3.2.2 FitParameter

class aesop.**FitParameter**(*value*, *err_upper=None*, *err_lower=None*, *default_err=10000000000.0*)

Bases: `object`

Fit results for a fitting parameter, with asymmetrical errors.

Parameters

value : {`Quantity`, `ndarray`, float}

err_upper : {`Quantity`, `ndarray`, float}

err_lower : {`Quantity`, `ndarray`, float}

default_err : float

Methods Summary

`from_text`(path)	
`to_text`(path)	

Methods Documentation

classmethod from_text(*path*)

to_text(*path*)

3.2.3 Measurement

class aesop.**Measurement**(*value=None*, *err=None*, *time=None*, *default_err=10000000000.0*, *meta=None*)

Bases: `object`

Methods Summary

from_dict(dictionary)	
from_min_max(min, max)	
to_latex()	

Methods Documentation

classmethod from_dict(*dictionary*)

classmethod from_min_max(*min*, *max*)

to_latex()

3.2.4 Spectrum1D

class aesop.**Spectrum1D**(*wavelength=None*, *flux=None*, *name=None*, *mask=None*, *wcs=None*, *meta={}*, *time=None*, *continuum_normalized=None*)

Bases: object

Simple 1D spectrum object.

A Spectrum1D object can be used to describe one order of an echelle spectrum, for example.

If the spectrum is initialized with wavelength``s that are not strictly increasing, ``Spectrum1D will sort the wavelength, flux and mask arrays so that wavelength is monotonically increasing.

Parameters

wavelength : ndarray

Wavelengths

flux : ndarray

Fluxes

name : str (optional)

Name for the spectrum

mask : ndarray (optional)

Boolean mask of the same shape as flux

wcs : Spectrum1DLookupWCS (optional)

Store the WCS parameters

meta : dict (optional)

Metadata dictionary.

continuum_normalized : bool (optional)

Is this spectrum continuum normalized?

Attributes Summary

masked_flux	
masked_wavelength	

Methods Summary

flux_calibrate(flux_calibrated_spectrum, ...)	Calculates coefficients of sensitivity function, then returns flux-calibrated spectrum
flux_calibrate_parameters(... [, plots])	Interpolate high-res spectrum to low-res flux calibrated spectrum, then fit the ratio with a polynomial to flux calibrate.
from_array(wavelength, flux[, ...])	Initialize a spectrum with the same call signature as from_array.
from_specutils(spectrum1d[, name])	Convert a Spectrum1D object into our Spectrum1D object.
mask_outliers([reject_negative, mad_clip, ...])	Identify outliers, update the mask attribute.
plot([ax])	Plot the spectrum.

Attributes Documentation

masked_flux

masked_wavelength

Methods Documentation

flux_calibrate(*flux_calibrated_spectrum*, *polynomial_order*)
Calculates coefficients of sensitivity function, then returns flux-calibrated spectrum

Parameters
flux_calibrated_spectrum : Spectrum1D

Already flux calibrated low-resolution spectrum of the same object

polynomial_order : int

Order of polynomial fit

Returns
transformed_spectrum : Spectrum1D

Spectrum transformed with sensitivity polynomial

flux_calibrate_parameters(*flux_calibrated_spectrum*, *polynomial_order*, *plots=False*)
Interpolate high-res spectrum to low-res flux calibrated spectrum, then fit the ratio with a polynomial to flux calibrate. Returns polynomial coefficients

Parameters
flux_calibrated_spectrum : Spectrum1D

Already flux calibrated low-resolution spectrum of the same object

polynomial_order : int

Order of polynomial fit

plots : bool

If True, plot the sensitivity data and the fit

Returns

fit_params : `ndarray`

Best-fit polynomial coefficients

classmethod from_array(*wavelength*, *flux*, *dispersion_unit=None*, *name=None*, ***kwargs*)

Initialize a spectrum with the same call signature as `from_array`.

Parameters

wavelength : `Quantity`

Spectrum wavelengths

flux : `Quantity` or `ndarray`

Spectrum fluxes

dispersion_unit : `Unit` (optional)

Unit of the wavelength

name : str (optional)

Name of the target/spectrum

classmethod from_specutils(*spectrum1d*, *name=None*, ***kwargs*)

Convert a `Spectrum1D` object into our Spectrum1D object.

Parameters

spectrum1d : `Spectrum1D`

Input spectrum

name : str

Target/spectrum name

mask_outliers(*reject_negative=True*, *mad_clip=True*, *mad_outlier_factor=3*)

Identify outliers, update the `mask` attribute.

Parameters

reject_negative : bool (optional)

Reject fluxes < -0.5. Default is `True`.

mad_clip : bool

Reject fluxes more than `mad_outlier_factor` times the median absolute deviation (MAD) from the continuum flux.

mad_outlier_factor : float

MAD-masking factor – fluxes more than `mad_outlier_factor` away from the continuum flux will be masked.

plot(*ax=None*, ***kwargs*)

Plot the spectrum.

Parameters

ax : `Axes` (optional)

The `Axes` to draw on, if provided.

kwargs

All other keyword arguments are passed to plot

3.2.5 StarProps

class aesop.**StarProps**(*name=None*, *s_apo=None*, *s_mwo=None*, *time=None*)

Bases: object

S-index properties for a star

Attributes Summary

s_mwo

Methods Summary

from_dict(dictionary)
get_s_mwo()

Attributes Documentation

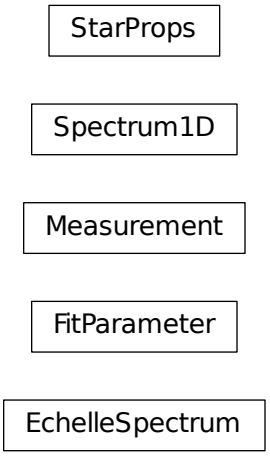
s_mwo

Methods Documentation

classmethod from_dict(*dictionary*)

get_s_mwo()

3.3 Class Inheritance Diagram



Part V

What is aesop?

In the interest of reproducing established results, we choose not to re-implement the canonical IRAF reduction routine ReduceARCES.cl, and to limit aesop's functionality to processing after the aperture extraction and initial wavelength solution.

The current version of aesop is useful for the following tasks:

- Managing and manipulating spectra with convenient data structures (`EchelleSpectrum`, `Spectrum1D`)
- Removing radial velocities from echelle orders via cross-correlation with PHOENIX model spectra
- Normalizing out the blaze-function in each echelle order given observations of a spectroscopic standard
- Further normalizing each order for a flat continuum with robust least-squares
- Concatenating the spectra in each order, and taking the mean between adjacent orders where they overlap, to produce a 1D spectrum from 3000-10000 Angstroms

Part VI

Typical Workflow

In general, aesop users will follow this procedure:

1. Install aesop
2. Run the ReduceARCES.cl IRAF script to reduce your echelle spectra (see *IRAF*)
3. Open the extracted 1D spectra for each spectral order with aesop (see *Getting started*)

Part VII

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

aesop, 23

Index

A

aesop (*module*), 23

B

barycentric_correction() (*aesop.EchelleSpectrum method*), 30

C

continuum_normalize_from_standard() (*aesop.EchelleSpectrum method*), 30
continuum_normalize_lstsq() (*aesop.EchelleSpectrum method*), 31
cross_corr() (*in module aesop*), 23

E

EchelleSpectrum (*class in aesop*), 29

F

fit_order() (*aesop.EchelleSpectrum method*), 31
FitParameter (*class in aesop*), 33
flux_calibrate() (*aesop.Spectrum1D method*), 35
flux_calibrate_parameters() (*aesop.Spectrum1D method*), 35
from_array() (*aesop.Spectrum1D class method*), 36
from_dict() (*aesop.Measurement class method*), 34
from_dict() (*aesop.StarProps class method*), 37
from_fits() (*aesop.EchelleSpectrum class method*), 31
from_min_max() (*aesop.Measurement class method*), 34
from_specutils() (*aesop.Spectrum1D class method*), 36
from_text() (*aesop.FitParameter class method*), 33

G

get_order() (*aesop.EchelleSpectrum method*), 31
get_phoenix_model_spectrum() (*in module aesop*), 24
get_s_mwo() (*aesop.StarProps method*), 37
get_spectrum_mask() (*in module aesop*), 24
glob_spectra_paths() (*in module aesop*), 25

I

integrate_spectrum_trapz() (*in module aesop*), 25
interpolate_spectrum() (*in module aesop*), 25

J

json_to_stars() (*in module aesop*), 26

M

mask_outliers() (*aesop.Spectrum1D method*), 36
masked_flux (*aesop.Spectrum1D attribute*), 35
masked_wavelength (*aesop.Spectrum1D attribute*), 35
Measurement (*class in aesop*), 33

O

offset_wavelength_solution() (*aesop.EchelleSpectrum method*), 32

P

plot() (*aesop.Spectrum1D method*), 36
predict_continuum() (*aesop.EchelleSpectrum method*), 32

Q

query_for_spectral_type() (*in module aesop*), 26
query_for_T_eff() (*in module aesop*), 26

R

rv_wavelength_shift() (*aesop.EchelleSpectrum method*), 32
rv_wavelength_shift_ransac() (*aesop.EchelleSpectrum method*), 32

S

s_mwo (*aesop.StarProps attribute*), 37
slice_spectrum() (*in module aesop*), 27
Spectrum1D (*class in aesop*), 34
StarProps (*class in aesop*), 37
stars_to_json() (*in module aesop*), 27

T

test() (*in module aesop*), 27
to_latex() (*aesop.Measurement method*), 34
to_Spectrum1D() (*aesop.EchelleSpectrum method*), 33
to_text() (*aesop.FitParameter method*), 33

U

uncalibrated_s_index() (*in module aesop*), 28