Eniric Documentation

Release 1.0.0

Jason Neal, Pedro Figueira

Jun 11, 2020

Contents:

CHAPTER 1

Installation

You can install `eniric` by cloning the current `master` branch.

It is recommended to use a `conda` or `virtualenv` environment. To use the most up-to-date packages install the pinned requirements from the `requirements_dev.txt` file

```
git clone https://github.com/jason-neal/eniric
cd eniric
pip install -r requirements_dev.txt
python setup.py develop
```

Installation from the github repository should also be possible.

```
pip install https://github.com/jason-neal/eniric/archive/master.zip#egg=eniric
```

or for the develop branch

```
pip install https://github.com/jason-neal/eniric/archive/develop.zip#egg=eniric
```

1.1 Requirements

The requirements for `eniric` are:

- astropy
- joblib>=0.12.3
- matplotlib
- multiprocess
- numpy>=0.15.4
- pandas
- pyyaml

- oyaml
- scipy
- Starfish
- tqdm

There are some specific version requirements given in `requirements.txt` but the most recent versions are pinned in `requirements_dev.txt` to install these run

```
pip install -r requirements_dev.txt
```

from the cloned `eniric` repository.

1.1.1 Starfish

`Eniric` makes use of Starfish's `GridTools` module to load the synthetic libraries: PHOENIX-ACES and BT-Settl models.

`Starfish` is currently going through API changes and not yet available on pypi.

A custom fixed branch of Starfish can be used on both Linux and Windows. `Starfish` should be automatically installed during the installation of `eniric`, but if not you can install it with.

```
pip install https://github.com/jason-neal/Starfish/archive/eniric_suitable.zip
↪#egg=astrostarfish
```

If issues arise regarding `Starfish` see github.com/iancze/Starfish:,

Other requirements required with Starfish are:

- corner
- cycler
- cython
- emcee
- h5py
- kiwisolver
- pyparsing
- python-dateutil
- pyyaml

1.2 OS

`Eniric` has been tested to work on both **Linux** and **Windows**.

1.3 Eniric Data

There are some data files not included in the `eniric` repository that are *necessary for testing*. These can be easily downloaded using the provided scripts.

```
$ download_eniric_data.sh
```

Note: If you have an issue connecting to dropbox you can also try adding the `--no-check-certificate` flag.

or on **Windows** in a PowerShell

```
ps_download_eniric_data.ps1
```

This includes an atmospheric transmission spectrum, located at `data/atmos/Average_TAPAS_2014.dat`, which can be used for spectral masking.

Note: This is to keep the size of the git repository small.

1.4 Testing

To test `eniric` is installed try

```
python -c "import eniric"
```

To run the test suite run `pytest` from the root directory of the repository (requires pytest). This will result in an output similar to:

```
 $ pytest

 ============================= test session starts ==============================
 platform linux -- Python 3.6.7, pytest-4.3.0, py-1.7.0, pluggy-0.8.0
 hypothesis profile 'default' -> database=DirectoryBasedExampleDatabase('/home/travis/
↪build/jason-neal/eniric/.hypothesis/examples')
 rootdir: /home/travis/build/jason-neal/eniric, inifile: setup.cfg
 plugins: cov-2.6.1, hypothesis-4.7.17
 collected 718 items

 ...

 ======= 610 passed, 84 xfailed, 24 xpassed, 2 warnings in 33.00 seconds ========
The command "pytest" exited with 0.
```

The requirements for the test suite can be installed from the root of the repository using

```
$ python setup.py install .[test]
```

Note: A users copied `config.yaml` file in the repository home directory may interfere with the test results, causing some failures.

CHAPTER 2

Configuration

Configuration is preformed using a `config.yaml` placed in the working directory you wish to run `eniric` from. It configures the path locations to the spectral libraries and allows for user defined spectral bands.

If a `config.yaml` file does not exist then the default is used, located at `eniric/config.yaml`.

You can copy the default configuration to a directory located at `<path_to_dir>` using `config.copy_file(<path_to_dir>)`

For example, to copy it to the current directory from the command line use:

```
$ python -c "from eniric import config; config.copy_file('.')"
```

The configuration values can be changed in python and saved back to the config file. For example

```
from eniric import config
config.paths["precision"] = ["new", "path", "to", "precision"]  # or "new/path/to/
↪precision"
config.update()
```

will update precision path in the configuration file.

Note: The default `config.yaml` file cannot be overwritten.

2.1 Eniric configuration

The default configuration file is shown below. The comments explain the keywords needed.

```
# Default YAML configuration script for Eniric.

# Paths to locations used by eniric, relative to config.yaml.
# Paths can either be strings of list of strings which will be passed
```

(continues on next page)

(continued from previous page)

```yaml
# to :py:func:`os.path.join` (This is for os independence)
paths:
  phoenix_raw: ["..", "data", "phoenix-raw"] # Path to Phoenix-ACES spectra
  btsettl_raw: ["..", "data", "btsettl-raw"] # Path to BT-Settl spectra
  atmmodel: ["..", "data", "atmmodel"] # Path of atmosphere model directory
  precision_results: ["..", "data", "precision"]  # A place to put precision results

# Wavelength bands.
bands:
  all: ["VIS", "GAP", "Z", "Y", "J", "H", "K", "CONT", "NIR", "TEST"] # "all" name␣
↪available in scripts
# Already included wavelength bands  (see `eniric.utilities.py`)
# "VIS": (0.38, 0.78),
# "GAP": (0.78, 0.83),
# "Z": (0.83, 0.93),
# "Y": (1.0, 1.1),
# "J": (1.17, 1.33),
# "H": (1.5, 1.75),
# "K": (2.07, 2.35),
# "CONT": (0.45, 1.05),
# "NIR": (0.83, 2.35)

# Add your custom band limits here and include in "all" list above.
# Limits must be contained in lists.
custom_bands:
  TEST: [2.1, 2.2]

# Location of the joblib memory cache.
cache:
  # location: None  # Disables cache
  location: [".joblib"]

# Properties of the atmospheric model used.
atmmodel:
  # Base name of telluric model
  base: "Average_TAPAS_2014"

# Necessary configuration keywords for Starfish can be included in this file.
name: "default"
```

2.2 Config Class

class eniric._config.**Config**(*path*)

Creates a persistent Config object from the give file.

This class is not meant to be instantiated by the User, but rather to be used via `eniric.config`.

Parameters **path** (*str or path-like*) – The filename for creating the Config. Must be YAML.

copy_file(*directory=os.getcwd()*, *switch=True*)

Copies the master config file to the given directory.

Parameters

- **directory** (*str or path-like*) – The directory to copy the `config.yaml` file

to. Default is os.getcwd().

- **switch** (*bool*) – If True, will switch the current config to use the copied file. Default is True

Example

```
eniric.config.copy_file()
```

change_file(*filename*)
Change the current configuration to use the given YAML file.

Parameters **filename** (*str or path-like*) – The YAML file to switch to using for config.

Example

```
eniric.config.change_file('new_config.yaml')
```

copy_file(*directory=None*, *switch=True*)
Copies the master config file to the given directory.

Parameters

- **directory** (*str or path-like*) – The directory to copy the `config.yaml` file to. Default is os.getcwd().
- **switch** (*bool*) – If True, will switch the current config to use the copied file. Default is True

Example

```
eniric.config.copy_file()
```

get_pathdir()
Directory of the config file.

pathdir
Directory of the config file.

update(*d=None*, ***kwargs*)
Update the config values and save to the config file.

CHAPTER 3

Basic Usage

To calculate the RV precision given a spectrum with `wavelength` and `flux` you can use the `rv_precsion` function.

```
from eniric.precision import rv_precision
rv = rv_precision(wavelength, flux)
```

or for the flux independent spectral `quality`, Q,

```
from eniric.precision import quality
q = quality(wavelength, flux)
```

eniric.precision.**rv_precision**(*wavelength: numpy.ndarray*, *flux: numpy.ndarray*, *mask: Optional[numpy.ndarray] = None*, ***kwargs*) → astropy.units.quantity.Quantity

Calculate the theoretical RV precision achievable on a spectrum.

Parameters

- **wavelength** (*array-like or Quantity*) – Wavelength of spectrum.
- **flux** (*array-like or Quantity*) – Flux of spectrum.
- **mask** (*array-like, Quantity or None*) – Masking function array to apply to the pixel weights.
- **kwargs** – Kwargs for sqrt_sum_wis

Returns **RVrms** – Radial velocity precision of spectra in m/s.

Return type astropy.Quantity

eniric.precision.**quality**(*wavelength: numpy.ndarray*, *flux: numpy.ndarray*, *mask: Optional[numpy.ndarray] = None*, ***kwargs*) → float

Calculation of the spectral Quality, Q, for a spectrum.

```
Q = sqrt{sum{W(i)}} / sqrt{sum{A_0{i}}
```

The spectral quality, Q, is independent of the flux level and is only a function of the spectral profile.

Parameters

- **wavelength** (*array-like or Quantity*) – Wavelength of spectrum.
- **flux** (*array-like or Quantity*) – Flux of spectrum.
- **mask** (*array-like, Quantity or None*) – Masking function array to apply to the pixel weights.
- **kwargs** – Kwargs for sqrt_sum_wis (including mask).

Returns **q** – Spectral quality.

Return type float

3.1 Masking

It is possible to include custom pixel masks both `rv_recision()` and `quality()`, to selectively select, exclude, or weight the spectra. These are pass it as the 3rd argument to either function.

```
rv = rv_precision(wavelength, flux, mask=my_mask)
q = quality(wavelength, flux, mask=my_mask)
```

Not supplying a mask or setting `mask=None` is equivalent to a mask of all `1`'s which will have no impact on the precision (i.e. all pixels are weighted equally).

To use the default (supplied) telluric line model to apply spectral masking you can use the Atmosphere class:

```
from eniric.atmosphere import Atmosphere
# Assuming K is the correct band, and you want to mask for seasonal variation.
atm = Atmosphere.from_band("K", bary=True)

# Obtain closet telluric model values at the wavelength values (this telluric mask is␣
↪super sampled).
atm = atm.at(wavelength)

# Boolean telluric line mask at 2% deep.
rv2 = rv_precision(wavelength, flux, mask=atm.mask)

# Perfect telluric correction mask.
rv3 = rv_precision(wavelength, flux, mask=atm.transmission**2)
```

For more details about about the masks used here see *Telluric Masking*.

CHAPTER 4

Broadening

`broaden` is a module to perform broadening of the stellar spectra.

Contents

Two mechanisms are used to broaden the spectra:

4.1 Rotational broadening

- Convolution of the spectra with a rotational broadening function with a given velocity `vsini` (in km/s). The default limb darkening coefficient is ´epsilon= 0.6´.

`eniric.broaden.`**`rotational_convolution`**(**args*, ***kwargs*)

Memoized version of rotational_convolution(wavelength:numpy.ndarray, extended_wav:numpy.ndarray, extended_flux:numpy.ndarray, vsini:float, *, epsilon:float=0.6, normalize:bool=True, num_procs:Union[int, joblib.parallel.Parallel, NoneType]=None, verbose:bool=True) -> numpy.ndarray

Perform Rotational convolution.

Parameters

- **`wavelength`** (*`ndarray`*) – Wavelength array.
- **`extended_wav`** (*`ndarray`*) – Extended wavelength array to avoid boundary issues.
- **`extended_flux`** (*`ndarray`*) – Photon flux for the extended wavelength array.

- **vsini** (*float*) – Rotational velocity in km/s.
- **epsilon** (*float*) – Limb darkening coefficient. Default is 0.6.
- **normalize** (*bool*) – Area normalize the broadening kernel. This corrects for kernel area with unequal wavelength spacing. Default is True.
- **num_procs** (*int, None or joblib.parallel.Parallel.*) – Number of processes to use, n_job parameter in joblib. If num_procs = 1, then a single core is used. Can also be a joblib.parallel.Parallel instance. Default is None.
- **verbose** (*bool*) – Show the tqdm progress bar. Default is True.

Returns **convolved_flux** – The convolved flux evaluated at the wavelength array.

Return type ndarray

4.1.1 Rotation kernel

eniric.broaden.**rotation_kernel**(*delta_lambdas: numpy.ndarray*, *delta_lambda_l: float*, *vsini: float*, *epsilon: float*) → numpy.ndarray

Rotation kernel for a given line center.

Parameters

- **delta_lambdas** (*array*) – Wavelength difference from the line center lambda.
- **delta_lambda_l** (*float*) – Maximum line shift of line center by vsini.
- **vsini** (*float*) – Projected rotational velocity in km/s.
- **epsilon** (*float*) – Linear limb-darkening coefficient, between 0 and 1.

Returns **kernel** – Rotational kernel.

Return type array

Notes

Gray, D. F. (2005). The Observation and Analysis of Stellar Photospheres. 3rd ed. Cambridge University Press.

4.2 Instrumental broadening

- Convolution by a Gaussian with a FWHM equivalent to the resolving power `R` at each wavelength. The convolution is extended either side to a `fwhm_lim` of 5 by default.

eniric.broaden.**resolution_convolution**(**args*, ***kwargs*)

Memoized version of resolution_convolution(wavelength:numpy.ndarray, extended_wav:numpy.ndarray, extended_flux:numpy.ndarray, R:float, *, fwhm_lim:float=5.0, normalize:bool=True, num_procs:Union[int, joblib.parallel.Parallel, NoneType]=None, verbose:bool=True) -> numpy.ndarray

Perform Resolution convolution.

Parameters

- **wavelength** (*ndarray*) – Wavelength array.
- **extended_wav** (*ndarray*) – Extended wavelength array to avoid boundary issues.
- **extended_flux** (*ndarray*) – Photon flux for the extended wavelength array.

- **R** (*float*) – Resolution of Guassian instrumental profile.
- **fwhm_lim** (*float*) – FWHM limit for instrument broadening. Default is 5.0.
- **normalize** (*bool*) – Area normalize the broadening kernel. This corrects for kernel area with unequal wavelength spacing. Default is True.
- **num_procs** (*int, None or joblib.parallel.Parallel.*) – Number of processes to use, n_job parameter in joblib. If num_procs = 1, then a single core is used. Can also be a joblib.parallel.Parallel instance.
- **verbose** (*bool*) – Show the tqdm progress bar. Default is True.

Returns **convolved_flux** – The convolved flux evaluated at the wavelength array.

Return type ndarray

4.2.1 Gaussian kernel

eniric.broaden.**unitary_gaussian**(*x: Union[range, int, numpy.ndarray], center: Union[float, int, str], fwhm: Union[float, int, str]*) → numpy.ndarray

Gaussian kernal of area 1.

Parameters

- **x** (*array-like*) – Position array.
- **center** (*float*) – Central position of Gaussian.
- **fwhm** (*float*) – Full Width at Half Maximum.

Returns **kernel** – Gaussian kernel.

Return type array-like

4.2.2 Circular Fibre

This is the convolution kernel for a circular fibre. .. autofunction:: oned_circle_kernel

When analyzing the spectral libraries, rotational broadening is preformed first, followed by the instrumental broadening.

Our convolution functions use wavelength dependent kernels and do not require uniform spacing between points, unlike PyAstronomy. This means our convolutions are slower but are more precise. We compare the convolutions in the Convolution_speeds.ipynb notebook.

4.3 Combined convolution

Resolution following rotation

eniric.broaden.**convolution**(**args*, ***kwargs*)

Memoized version of convolution(wav:numpy.ndarray, flux:numpy.ndarray, vsini:float, R:float, band:str='All', *, epsilon:float=0.6, fwhm_lim:float=5.0, num_procs:Union[int, joblib.parallel.Parallel, NoneType]=None, normalize:bool=True, verbose:bool=True)

Perform rotational then Instrumental broadening with convolutions.

Parameters

- **wav** (*ndarray*) – Wavelength array.

- **flux** (*ndarray*) – Flux array.
- **vsini** (*float*) – Rotational velocity in km/s.
- **R** (*int*) – Resolution of instrumental profile.
- **band** (*str*) – Wavelength band to choose, default is "All".
- **epsilon** (*float*) – Limb darkening coefficient. Default is 0.6.
- **fwhm_lim** (*float*) – FWHM limit for instrument broadening. Default is 5.0.
- **normalize** (*bool*) – Area normalize the broadening kernel. This corrects for kernel area with unequal wavelength spacing. Default is True.
- **num_procs** (*int, None or joblib.parallel.Parallel.*) – Number of processes to use, n_job parameter in joblib. If num_procs = 1, then a single core is used. Can also be a joblib.parallel.Parallel instance.
- **verbose** (*bool*) – Show the tqdm progress bar. Default is True.

Returns

- **wav_band** (*ndarray*) – Wavelength for the selected band.
- **flux_band** (*ndarray*) – Original flux for the selected band.
- **flux_conv** (*ndarray*) – Convolved flux for the selected band.

4.4 Caching

Convolution results are cached using `joblib`, see https://joblib.readthedocs.io/en/latest/memory.html

Caching of the convolution stages is performed to avoid re-computation of this slow component when possible using ‘‘*joblib.Memory*‘ <https://joblib.readthedocs.io/en/latest/memory.html>‘_. The caching directory defaults to `~/.joblib` but can be changed in the configuration file `config.yaml`.

Caching can be disabled by setting `location=None` in `config.yml`.

CHAPTER 5

Atmospheric Transmission

For ground-based observations it is important to understand the how the atmospheric absorption affects the Radial Velocity precision. `Eniric` does this in two scenarios; masking out all wavelength regions affected by telluric lines over adjusting for barycentric motion, or assuming perfect atmospheric correction, in which the variance of the photon noise is adjusted.

For this `eniric` contains a average telluric transmission model. It is an average model was created from weekly transmission spectra in 2014 simulated with the TAPAS software (Bertaux et al 2014) as detailed in (Figueira et al. 2016). The main parameters are

- La Silla Observatory
- airmass = 1.2 (z = 33.5 degrees)
- R = 100,000
- sampling = 10

This average model that is used is available at `data/atmmodel/Average_TAPAS_2014.txt`. It has 4 columns, wavelength (in nano-meters), transmission (between 0-1), std, mask (0 or 1). The mask is 0 for telluric lines deeper than 2%, 1 elsewhere. The std is the standard deviation of the transmission across the year but is not used in deriving precision.

You may supply your own transmission atmospheric models if you wish., e.g. `my_telluric_model.txt` To make individual precision calculations faster this model can be split into the separate spectroscopic bands using the scrip `scripts/split_atmmodels.py`, making smaller files to load later, but taking up more disk space.

e.g. Use `split_atmmodels.py -h` to get the description of flags to use.

```
split_atmmodels.py My_telluric_model.txt -b H K
```

Will create the subsets of `my_telluric_model_H.txt` and `my_telluric_model_K.txt` from `my_teluric_model.txt`. If the the band subset cannot be found the `my_telluric_model.txt` will be used as a fallback.

You can configure the location of and atmmodel to use in config.yaml.

5.1 Telluric Masking

In real observations telluric lines contaminate the spectra and need to be removed. Typically this is done by excluding waveelengths that are near deep (e.g. >2%) telluric lines. Figueria et al. (2016) condsidered three different conditions which can be

The atmospheric absorption can be used to mask

The three default cases treated in Figueira et al. (2016) were no telluric corection, masking, assumption of perfect telluric correction.

The masks M for these three cases are as follows, given the atmospheretic transmission T for each wavelenght or pixel, i.

- **Condition 1** No telluric treatment, theoretical precision of the spectrum.

$$M(i) = 1$$

- **Condition 2** Masking out telluric lines deeper than τ (e.g. $\tau = 0.98$ for 2%).

$$M(i) = \begin{cases} 0, & T(i) < \tau \\ 1, & T(i) \geq \tau \end{cases}$$

- **Condition 3** The assumption of perfect telluric correction. The telluric lines have been completely removed however the flux variance at the locations of the lines increases due to the lower flux received. This can be implemented with the following mask

$$M(i) = T(i)^2$$

For examples using these masks see *Masking* or the usage in `phoenix_precision.py`.

The mask for Condition 2 can be created using Atmosphere class

`Atmosphere.`**`mask_transmission`**(*depth: float = 2.0*) → None

Mask the transmission below given depth. e.g. 2%.

Updates the mask.

Parameters **`depth`** (*`float`*) – Telluric line depth percentage to mask out. Default is 2.0. E.g. depth=2 will mask transmission deeper than 2%.

5.2 Barycentric Motion

To exclude wavelength regions that will be affected by telluric lines at some point during the year you can extend out the telluric mask.

```
new_mask = barycentric_broaden(wav, transmission, mask)
```

This extends the regions that are masked out, you can check that the mask continues to mask out deep lines like so…

```
assert np.all(transmission[mask] > 0.98)      # Check old mask works
assert np.all(transmission[new_mask] > 0.98) # Check new mask
# More points should be zero in extended mask
assert np.sum(new_mask) < np.sum(mask)
```

The fraction `(np.sum(new_mask) - np.sum(mask)) /np.sum(mask)` can also indicate the increase in masked out wavelengths.

5.3 Telluric Preparation

The default telluric transmission spectrum provided with `eniric` is given in `data\Atmodel\Average_TAPAS_2014.dat`. This is quite unweildly, opening and slicing of the large telluric spectrum reduces performance.

To prepare this for computations it is better to split the spectra into the individual wavelength bands and apply the barycentric shifts.

```
split_atmmodel.py
barycenter_broaden_atmmodel.py
```

These two scripts will split the large telluirc spectra into the bands specified in the `config.yaml`. They default to a mask of 2% depth. To change the masking cutoff depth use the `--cutoff-depth` flag.

```
split_atmmodel.py --cutoff-depth 4
```

The `Atmosphere` class will try to load the individaul band spectra but will fall back to the base file.

5.4 Module

class eniric.atmosphere.**Atmosphere**(*wavelength*, *transmission*, *mask=None*, *std=None*, *shifted=False*, *verbose=False*)

Atmospheric transmission object.

Stores wavelength and atmospheric transmission arrays. Enables telluric masking and accounting for barycentric motion.

wl
Wavelength array.

Type ndarray

transmission
Atmospheric transmission (between 0 and 1).

Type ndarray

std
Standard deviation of transmission.

Type ndarray

mask
Transmission mask (1's are kept).

Type ndarray

shifted
Indicate shifted mask. Default is False.

Type bool

verbose
Enable verbose. Default is False.

Type bool

Constructors

----

from_file(*atmmodel*)
Read in atmospheric model and prepare.

from_band(*band*, *bary=False*)
Read in atmospheric model for given band.

to_file(*fname*, *header*, *fmt*)
Save the atmospheric model to a txt file.

at(*wave*)
Return the transmission value at the closest points to wave.

wave_select(*wl_min*, *wl_max*)
Slice Atmosphere between two wavelengths.

band_select(*band*)
Slice Atmosphere to a given band.

copy()
Make a copy of atmosphere object.

mask_transmission(*depth*)
Mask the transmission below given depth. e.g. 2%

barycenter_broaden(*rv*, *consecutive_test*)
Sweep telluric mask symmetrically by +/- a velocity.

broaden(*resolution*, **kwargs*)
Instrument broadening of the atmospheric transmission profile.

Configuration()

Two things can be set for the Atmosphere class in the `config.yaml` file.

The path to the atmosphere data

e.g.

paths: atmmodel: "path/to/atmmodel/directory"

The name for the atmosphere model *.dat file

atmmodel: base: "Average_TAPAS_2014"

at(*wave*)
Return the transmission value at the closest points to wave.

This assumes that the atmosphere model is sampled at a higher rate than the stellar spectra.

For instance, the default Tapas model has a sampling if 10 compared to 3.

Parameters **wave** (*ndarray*) – Wavelengths at which to return closest atmosphere values.

band_select(*band*)
Slice Atmosphere to a given Band.

barycenter_broaden(*rv: float = 30.0*, *consecutive_test: bool = False*)
Sweep telluric mask symmetrically by +/- a velocity.

Updates the objects mask.

Parameters

- **rv** (*float*) – Velocity to extend masks in km/s. Default is 30 km/s.
- **consecutive_test** (*bool*) – Checks for 3 consecutive zeros to mask out transmission. Default is False.

broaden(*resolution: float*, *fwhm_lim: float = 5*, *num_procs=None*)
Instrument broadening of the atmospheric transmission profile.

This does not change any created masks.

Parameters

- **resolution** (*float*) – Instrumental resolution R.
- **fwhm_lim** (*int/float*) – Number of FWHM to extend the wings of the convolution kernel.
- **num_procs** (*Optional[int]*) – Number of processors to use. Default = total processors - 1

copy()
Make a copy of atmosphere object.

classmethod from_band(*band: str*, *bary: bool = False*)
Read in atmospheric model for given band.

Alternate constructor for Atmosphere. Base on "base" path in config.yaml.

Parameters

- **band** (*str*) – Name of atmosphere file.
- **bary** (*bool*) – Barycentric shift the mask.

classmethod from_file(*atmmodel: str*)
Read in atmospheric model and prepare.

Alternate constructor for Atmosphere.

Parameters **atmmodel** (*str*) – Name of atmosphere file.

mask_transmission(*depth: float = 2.0*) → None
Mask the transmission below given depth. e.g. 2%.

Updates the mask.

Parameters **depth** (*float*) – Telluric line depth percentage to mask out. Default is 2.0. E.g. depth=2 will mask transmission deeper than 2%.

to_file(*fname: str*, *header: Optional[List[str]] = None*, *fmt: str = '%11.8f'*)
Save the atmospheric model to a txt file.

Converts micron back into nanometers to be consistent with from_file().

Parameters

- **fname** (*str*) – Name of atmosphere file to save to.
- **header** – Header lines to add.
- **fmt** (*str*) – String formatting

wave_select(*wl_min*, *wl_max*)
Slice Atmosphere between two wavelengths.

eniric.atmosphere.**consecutive_truths**(*condition: numpy.ndarray*) → numpy.ndarray
Length of consecutive true values in an bool array.

Parameters **condition** (*ndarray of bool*) – True False array of a condition.

Returns **len_consecutive** – Array of lengths of consecutive true values of the condition.

Return type ndarray of ints

Notes

Solution found at http://stackoverflow.com/questions/24342047

CHAPTER 6

Normalization

The theoretical RV precision (only due to photon noise) is inversely dependent on the spectral flux. That is as the flux increases the error on the RV measurement decreases.

The flux of a observed spectra is dependant on a number of factors e.g.

- Star luminosity
- Integration time
- Telescope area
- Instrument efficiency

The photon flux can be scaled by multiplicative constant which affects the photon SNR and the radial velocity precision.

To compare the relative theoretical precision between different synthetic spectra they are normalized consistently.

This is achieved by normalizing to a specific SNR per resolution element at a particular location.

By default this is a SNR of 100 at the center of the "J" band. Other band centers or wavelengths as well as other SNR values can be chosen.

The following functions are used to determine the normalization constant to divide a spectrum by to achieve the desired normalization.

eniric.snr_normalization.**snr_constant_wav**(*wav: numpy.ndarray*, *flux: numpy.ndarray*, *wav_ref: float*, *snr: Union[int, float] = 100*, *sampling: Union[int, float] = 3.0*, *verbose: bool = False*) → float

Determine the normalization constant to achieve a SNR at given wavelength.

SNR estimated by the square root of the number of photons in a resolution element.

Parameters

- **wav** (*ndarray*) – Wavelength array.
- **flux** (*ndarray*) – Photon flux array (photons/s/cm**2).

- **wav_ref** (*float*) – Wavelength to set the SNR per resolution element.
- **snr** (*int, float*) – SNR per resolution element to achieve. Default is 100.
- **sampling** (*int or float*) – Number of pixels per resolution element. Default is 3.
- **verbose** (*bool*) – Enable verbose. Default is False.

Returns **norm_value** – Normalization value to divide the flux by to achieve the desired SNR "SNR" in resolution element (defined by "sampling") around the wavelength "wav_ref".

Return type float

Note: If sampling is a float it will rounded to the nearest integer for indexing.

---

eniric.snr_normalization.**snr_constant_band**(*wav: numpy.ndarray*, *flux: numpy.ndarray*, *snr: Union[int, float] = 100*, *band: str = 'J'*, *sampling: Union[int, float] = 3.0*, *verbose: bool = False*) → float

Determine the normalization constant to achieve a SNR in the middle of a given band.

SNR estimated by the square root of the number of photons in a resolution element.

Parameters

- **wav** (*ndarray*) – Wavelength array in microns.
- **flux** (*ndarray*) – Photon flux array (photons/s/cm**2).
- **snr** (*int or float*) – SNR per resolution element to achieve. Default is 100.
- **band** (*str*) – Band to use for normalization. Default is "J".
- **sampling** (*int or float*) – Number of pixels per resolution element. Default is 3.
- **verbose** (*bool*) – Enable verbose. Default is False.

Returns **norm_value** – Normalization value to divide spectrum by to achieve a signal-to-noise level of snr within an resolution element in the middle of the band.

Return type float

Note: This is a wrapper around *snr_constant_wav*, using the band center.

Warning: Wavelength is expected in microns!

eniric.snr_normalization.**sampling_index**(*index: int*, *sampling: int = 3*, *array_length: Optional[int] = None*) → numpy.ndarray

Get a small number of index values around the given index value.

Parameters

- **index** (*int*) – The index value which to select values around.
- **sampling** (*int*) – Number of index values to return (sampling per resolution element). Must be an integer. Default is 3.
- **array_length** (*int or None*) – Length of array the indexes will be used in. To not exceed array length. Default is None.

Returns **indexes** – The index values.

Return type ndarray

CHAPTER 7

Resampling

In Figueria et al. (2016) and *scripts.phoenix_precision* the synthetic spectra are resampled to 3 pixels per resolution element.

This is done using the *eniric.resample* module.

Functions for spectral resampling.

eniric.resample.**log_chunks**(*wavelength: numpy.ndarray*, *percent: float*) → numpy.ndarray

Define the bounds at which $(Delta lambda)/lambda = X%$.

Allows spectrum to be split into chunks in which the size is X% of the given wavelength. This takes logarithmic steps with a base of (1+X/100).

Parameters

- **wavelength** (*ndarry*) – Wavelength array.
- **percent** (*float*) – Base step in percentage.

Returns **logspace** – Array of points with a growth in wavelength spanned of a given percent.

Return type ndarray

eniric.resample.**log_resample**(*wavelength*, *sampling: Union[int, float]*, *resolution: Union[int, float]*) → numpy.ndarray

Resample spectrum with a given sampling per resolution element.

Uses faster method using log and powers of a base. The base is (1.0 + 1.0/(sampling*resolution).

Parameters

- **wavelength** (*numpy.ndarray*) – Wavelength array.
- **sampling** (*int, float*) – Points to sample per resolution element
- **resolution** (*int, float*) – Instrumental resolution

Returns **logspace** – Array of points with a set sampling per resolution element.

Return type ndarray

Note: Almost equivalent to using ‘‘np.logspace(np.log(wavelength)/np.log(base), np.log(wavelength)/np.log(base),

np.log(wavelength_end / wavelength_start) / np.log(base), base)‘‘.

eniric.resample.**wl_logspace**(*start*, *stop*, *base*, *end_point: bool = False*)

Like np.logspace but start and stop in wavelength units.

Parameters

- **start** (*float*) – Starting point (in real units)
- **stop** (*float*) – End point (in real units)
- **base** (*float*) – Logarithmic base to jump between points.
- **end_point** (*bool*) – Make sure to include/go past the end point

Returns **logspace** – Array of points with a spacing such that x[ii+1] = x[ii] * base between start and stop (or stop*base if end_point = True).

Return type ndarray

CHAPTER 8

Theoretical Precision of Synthetic Spectra

The theoretical precision of M-dwarf synthetic spectra is extensively explored in Figueira et al. (2016). `Eniric` extends this work to any spectra in the PHOENIX-ACES or the BT-Settl-CIFIST2011_2015 spectral libraries.

The script `phoenix_precision.py` is provided to easily generate synthetic RV precision values, similarly to and beyond the work of Figueira et al. (2016). The precision of a spectrum can be obtained by providing its library parameters Teff, logg, Fe/H and, alpha. This has been mainly used on M-dwarf spectra with temperatures < 4000K, but higher temperatures also work.

For the library selection and loading of the spectra Starfish's Grid Tools is used. The spectral libraries need to be pre-downloaded, and the `raw_path` to their location configured in the `config.yaml` file in the working directory. For more information see the Starfish documentation here.

8.1 Example Usage

`phoenix_precision.py` is called by passing the stellar and other parameters on the command line. Several flags can take multiple values separated by a space. All combinations of the parameters supplied will be computed.

The available inputs parameters are:

```
$ phoenix_precision.py -h

usage: phoenix_precision.py [-h] [-t TEMP [TEMP ...]] [-l LOGG [LOGG ...]]
                            [-m METAL [METAL ...]] [-a ALPHA [ALPHA ...]]
                            [-s SAMPLING [SAMPLING ...]]
                            [-r RESOLUTION [RESOLUTION ...]]
                            [-v VSINI [VSINI ...]]
                            [-b {K,H,J,Y,Z,TEST} [{K,H,J,Y,Z,TEST} ...]]
                            [--model {aces,btsettl,phoenix}] [--snr SNR]
                            [--ref_band {SELF,K,H,J,Y,Z,TEST}]
                            [--num_procs NUM_PROCS] [-o OUTPUT]
                            [--rv RV [RV ...]] [--add_rv] [--air] [--correct]
                            [-V] [--disable_normalization]
```

(continues on next page)

(continued from previous page)

```
Calculate precision and quality for synthetic spectra.

optional arguments:
  -h, --help            show this help message and exit
  -t TEMP [TEMP ...], --temp TEMP [TEMP ...]
                        Temperature, default=[3900].
  -l LOGG [LOGG ...], --logg LOGG [LOGG ...]
                        Logg, default = [4.5].
  -m METAL [METAL ...], --metal METAL [METAL ...]
                        Metallicity, default=[0.0].
  -a ALPHA [ALPHA ...], --alpha ALPHA [ALPHA ...]
                        Alpha, default=[0.0].
  -s SAMPLING [SAMPLING ...], --sampling SAMPLING [SAMPLING ...]
                        Sampling, default=[3.0].
  -r RESOLUTION [RESOLUTION ...], --resolution RESOLUTION [RESOLUTION ...]
                        Instrumental resolution, default=[50000]
  -v VSINI [VSINI ...], --vsini VSINI [VSINI ...]
                        Rotational Velocity, default = [1.0]
  -b {K,H,J,Y,Z,TEST} [{K,H,J,Y,Z,TEST} ...], --bands {K,H,J,Y,Z,TEST} [{K,H,J,Y,Z,
↪TEST} ...]
                        Wavelength bands to select, default=['J'].
  --model {aces,btsettl,phoenix}
                        Spectral models to use, default='aces'.
  --snr SNR             Mid-band SNR scaling, default=100.
  --ref_band {SELF,K,H,J,Y,Z,TEST}
                        SNR reference band, default='J'.
                        'self' scales each band relative to the SNR itself.
  --num_procs NUM_PROCS
                        Number of processors to use,
                        default = (Total cores - 1)
  -o OUTPUT, --output OUTPUT
                        Filename for result file, default='precisions.csv'.
  --rv RV [RV ...]      Radial velocity value, default=[0.0]
  --add_rv              Include a radial velocity shift.
  --air                 Convert to air wavelengths.
  --correct             Apply Artigau et al. (2018) RV band corrections.
  -V, --verbose         Enable verbose.
  --disable_normalization
                        Disable the convolution normalization.
```

Note: The parameter space is multiplicative so the runtime increases when increasing the number of values for each parameter.

8.2 Output File

The script returns a table with input parameters and the calculated precisions for all three tellruic conditions presented in Figueira et al. (2016) for the parameter combinations requested. E.g:

```
# precisions.csv
temp,logg,feh,alpha,band,resolution,vsini,sampling,correctflag,quality,cond1,cond2,
↪cond3
3900,4.5,0.0,0.0,K,100k,1.0,3.0,0,4916,7.4,33.6,8.0
```

The different columns of the output file are given in the table below.

Col.	Name	Description
1	temp	Library stellar effective temperature (K).
2	logg	Library stellar surface gravity.
3	feh	Library stellar metallicity.
4	alpha	Library stellar alpha ratio.
5	band	Wavelength band letter.
6	resolution	Instrumental resolution.
7	vsini	Stellar rotation (km/s).
8	sampling	Spectral sampling - N points per resolution element.
9	correctflag	Indicate if Artigau et al. (2018) precision correction is applied.
10	quality	Theoretical spectral quality.
11	cond1	RV precision with no masking (m/s). (Condition 1)
12	cond2	RV precision with binary masking (m/s). (Condition 2)
13	cond3	RV precision with transmission masking (m/s). (Condition 3)

The first 9 columns uniquely identify a set of input parameter values, this is used to avoid repeating an identical computaion. In this way `precsions.csv` can be appended to with new values, while keeping the other values, if desired.

8.3 Calculating Precisions

Below are some specific examples of using `phoenix_precision.py`.

This script has been used to generate nIR RV precision values across the M-dwarf temperature range. These were requested by the NIRPS and SPIRou consortia for use as into their respective Exposure Time Calculators. The commands to use to generate these datasets are provided below.

8.3.1 Figueira et al. (2016)

To reproduce the calculations of Figueira et al. (2016) you can use the

```
phoenix_precision.py -t 2600 2800 3500 3900 -m 0.0, -l 4.5 -r 60000 80000 100000 -v 1.
↪0 5.0 10.0 --snr 100 -b Z Y J H K --ref_band J
```

8.3.2 NIRPS

For the NIRPS ETC precisions were calculated for the whole M-dwarf range between 2500 and 4000 K. These were centred on the H-band centering with a SNR of 100. This also included R=75000 tailored to the NIRPS instrument.

```
phoenix_precision.py -t 2500, 2600, 2700, 2800, 2900, 3000, 3100, 3200, 3300, 3400,␣
↪3500, 3600, 3700, 3800, 3900, 4000
 -m 0.0, -l 5.0 --snr 100 -b Z Y J H K --ref_band H -r 60000 75000 80000 100000 -v 1.
↪0 5.0 10.0
```

Note: The PHOENIX-ACES models for this range need to have been downloaded.

8.3.3 SPIRou

For the SPIRou ETC the parameter combinations are the same as Figueira et al. (2016) but calculated relative to a SNR of 100 in each respective bands.

```
phoenix_precision.py -t 2600, 2900, 3500, 3900 -m 0.0, -l 4.5 --snr 100 -b Z Y J H K -
↪-ref_band self
```

8.4 BT-SETTL

To use the BT-Settl sectral library use the `--model` flag.

```
phoenix_precision.py -t 2600, 2900, 3500, 3900 -b Z Y J H K --model btsettl
```

8.5 Precision Corrections

Artigau et al. (2018) provided corrections to the precision obtained from syntheithc mnodel, after a comparision to obsrevered spectra.

The multiplicative correction factors for each band can be obtained with `correct_artigau_2018()`, e.g.:

```
rv_K = 1 # m/s
correction_factor = correct_artigau_2018(band="K")
rv_K_corrected = rv_K * correction_factor
```

Table 3 from Artigau et al. (2018):

> Multiplicative correction factors to be applied on the RV precision derived from stellar models. These values correspond to the square-root of the flux-weighted mean Q ratio between observation and models for each bandpass. The nominal values are for a comparison with the default model described here, but we also explore the impact of other physical parameter choices.

	Nominal	Variants			
[Fe/H]	0.5	0.0			
logg	5.0		5.5		
Teff (K)	3200			3400	3000
g	0.66	0.63	0.69	0.82	0.51
r	0.82	0.76	0.84	1.08	0.60
i	0.94	0.81	0.99	1.26	0.73
z	1.27	1.09	1.20	1.82	0.96
Y	0.29	0.30	0.27	0.30	0.25
J	0.38	0.40	0.31	0.54	0.37
H	1.37	0.95	1.82	1.42	1.23
K	1.47	1.06	2.00	1.66	1.27

eniric.corrections.**correct_artigau_2018**(*band: str*) → float

Apply Artigau et al. (2018) Table 3 nominal Barnard's Star corrections.

Parameters **band** (*str*) – Wavelength band.

Returns **correction** – Correction value to multiply synthetic model precisions by to get to "real" values.

Return type float

Warning: Returns the corrections from the nominal (second) column only.

CHAPTER 9

Scripts

Scripts provided with eniric.

9.1 phoenix_precision.py

Script to generate RV precision of synthetic spectra, see *Calculating Precisions*.

scripts.phoenix_precision.**check_model**(*model: str*) → str

Check model is 'aces' or 'btsettl'.

Parameters **model** (*str*) – Model name. Should be either 'aces' or 'btsettl'.

Returns **model** – Valid model output; either 'aces' or 'btsettl'.

Return type str

scripts.phoenix_precision.**convolve_and_resample**(*wav: numpy.ndarray*, *flux: numpy.ndarray*, *vsini: float*, *R: float*, *band: str*, *sampling: float*, ***conv_kwargs*) → Tuple[numpy.ndarray, numpy.ndarray]

Convolve and resample functions together.

Returns

- **wav_grid** (*ndarray*) – Resampled wavelength array
- **sampled_flux** (*ndarray*) – Convolved and resampled flux array

scripts.phoenix_precision.**do_analysis**(*star_params*, *vsini: float*, *R: float*, *band: str*, *sampling: float = 3.0*, *conv_kwargs=None*, *snr: float = 100.0*, *ref_band: str = 'J'*, *rv: float = 0.0*, *air: bool = False*, *model: str = 'aces'*, *verbose: bool = False*) → Tuple[astropy.units.quantity.Quantity, ...]

Calculate RV precision and Quality for specific parameter set.

Parameters

- **star_param** – Stellar parameters [temp, logg, feh, alpha] for phoenix model libraries.
- **vsini** (*float*) – Stellar equatorial rotation.
- **R** (*float*) – Instrumental resolution.
- **band** (*str*) – Spectral band.
- **sampling** (*float (default=False)*) – Per pixel sampling (after convolutions)
- **conv_kwargs** (*Dict (default=None)*) – Arguments specific for the convolutions, 'epsilon', 'fwhm_lim', 'num_procs', 'normalize', 'verbose'.
- **snr** (*float (default=100)*) – SNR normalization level. SNR per pixel and the center of the ref_band.
- **ref_band** (*str (default="J")*) – Reference band for SNR normalization.
- **rv** (*float*) – Radial velocity in km/s (default = 0.0).
- **air** (*bool*) – Get model in air wavelengths (default=False).
- **model** (*str*) – Name of synthetic library (aces, btsettl) to use. Default = 'aces'.
- **verbose** – Enable verbose (default=False).

Returns

- **q** (*astropy.Quality*) – Spectral quality.
- **result_1** (*astropy.Quality*) – RV precision under condition 1.
- **result_2** (*astropy.Quality*) – RV precision under condition 2.
- **result_3** (*astropy.Quality*) – RV precision under condition 3.

Notes

We apply the radial velocity doppler shift after

- convolution (rotation and resolution)
- resampling
- SNR normalization.

in this way the RV only effects the precision due to the telluric mask interaction. Physically the RV should be applied between the rotational and instrumental convolution but we assume this effect is negligible.

scripts.phoenix_precision.**get_already_computed**(*filename: str*, *add_rv: bool = False*) → List[str]

Get the string of already computed model/parameters from the result file.

scripts.phoenix_precision.**header_row**(*add_rv=False*) → str

Header row for output file.

scripts.phoenix_precision.**is_already_computed**(*computed_values: List[str], model, pars, add_rv: bool = False, correct: bool = False, verbose=False*) → bool

Check if any combinations have already been preformed. Correct is boolean for applied Artigau correction.

scripts.phoenix_precision.**model_format_args**(*model*, *pars*)

Format the model and parameter args to save in output file. Change to int, str and float.

model in [temp, logg, fe/h, alpha] pars in order (R, band, vsini, sample).

Can now also optionally handle a 5th parameter RV.

scripts.phoenix_precision.**select_csv_columns**(*line: str*, *ncols: int = 8*) → str
Select first ncols in a line from a csv.

Parameters **ncols** (*int*) – Number of column to select.

Returns **selected_cols** – Selected ncols of csv.

Return type str

9.2 barycenter_broaden_atmmodel

Doppler shift the Tapas atmosphere model and save to files. This makes the RV precision calculations faster.

scripts.barycenter_broaden_atmmodel.**main**(*bands: Optional[List[str]] = None*, *verbose: bool = False*) → None
Preform the barycentric shifting of atmosphere masks and saves result.

This saves time in the precision determination code.

Parameters **bands** (*list of str*) – Wavelength bands to perform barycenter shifts on. Default is all bands.

9.3 split_atmmodel

Script to split the large atmospheric model transmission spectra into the separate bands. This create smaller files to load for each band for individual bands only.

scripts.split_atmmodel.**check_positive**(*value: str*) → float
Check if input is positive.

http://stackoverflow.com/questions/14117415.

Parameters **value** (*str*) – A input string from argparse to check if it is a positive number.

Returns **float_value** – The value if it is positive as a float.

Return type float

Raises ArgumentTypeError: – If value is not a positive number.

scripts.split_atmmodel.**main**(*model: str = 'Average_TAPAS_2014.dat'*, *bands: Optional[List[str]] = None*, *new_name: Optional[str] = None*, *data_dir: Optional[str] = None*, *rv_extend: float = 100.0*, *cutoff_depth: float = 2.0*, *verbose: bool = False*)
Split the large atmospheric model transmission spectra into the separate bands.

Keeps wavelength of atmosphere model as nanometers.

Parameters

- **model** (*str*) – Telluric model file to load. It has columns wavelength, flux, std_flux, mask.
- **bands** (*list[str]*) – List bands to split model into separate files.
- **new_name** (*str*) – New file name base.
- **data_dir** (*Optional[str]*) – Directory for results. Can also be given in config.yaml "paths:atmmodel:"...

- **rv_extend** (*float*) – Absolute RV to extend wavelength range of telluric band. To later apply barycenter shifting. Default is 100.
- **cutoff_depth** (*float*) – Telluric line depth cutoff percent. Default = 2.0.

Returns **exit_status** – Unix-like exit. Non-zero indicates a failure occurred.

Return type int

9.4 precision_four_panel.py

Plot a Figueira et al. (2016) Figure 1 like plot.

scripts.precision_four_panel.**cumulative_df**(*df*, *full_cum=False*)

Calculated cumulative RV precision across bands. The precision of "Z", "ZY", "ZYJ", "ZYJH", "ZYJHK" bands.

Parameters

- **df** (*pandas.DataFrame*) – DataFrame.
- **full_cum** (*bool*) – Include "YJHK", "JHK", "HK", "K" grouping also. Default is False.

scripts.precision_four_panel.**cumulative_plot**(*precision_file*, *teffs=None*, *logg=4.5*, *fe_h=0.0*, *vsini=1.0*, *sampling=3*, *full_cum=False*)

RV precision with cumulative bands.

full_cum: bool Cumlative over entire range ["Z","ZY", "ZYJ", "ZYJH", "ZYJHK","YJHK", "JHK","HK","K"]

Saves figure to `plots/`.

Parameters

- **precision_file** (*str*) – Name of phoenix_precision.py output.
- **teffs** (*List of int or None*) – Stellar temperatures. Default is [3900, 3500, 2800, 2600].
- **logg** (*int*) – Stellar Logg. Default is 4.5.
- **fe_h** (*int*) – Stellar metallicity. Default is 0.0.
- **vsini** (*float*) – Rotational velocity. Default is 1.0.
- **sampling** – Spectral sampling. Default is 3.
- **full_cum** (*bool*) – Cumulative over entire range. Default is False.

scripts.precision_four_panel.**filter_df**(*df*, *filter_dict*, *drop_list=None*)

Filter DataFrame by dictionary of key and values.

scripts.precision_four_panel.**load_dataframe**(*filename*)

Load in precision file, clean up spaces in csv.

Parameters **filename** (*str*) – Name of phoenix_precision.py output.

Returns **df** – DataFrame of data.

Return type pandas.DataFrame

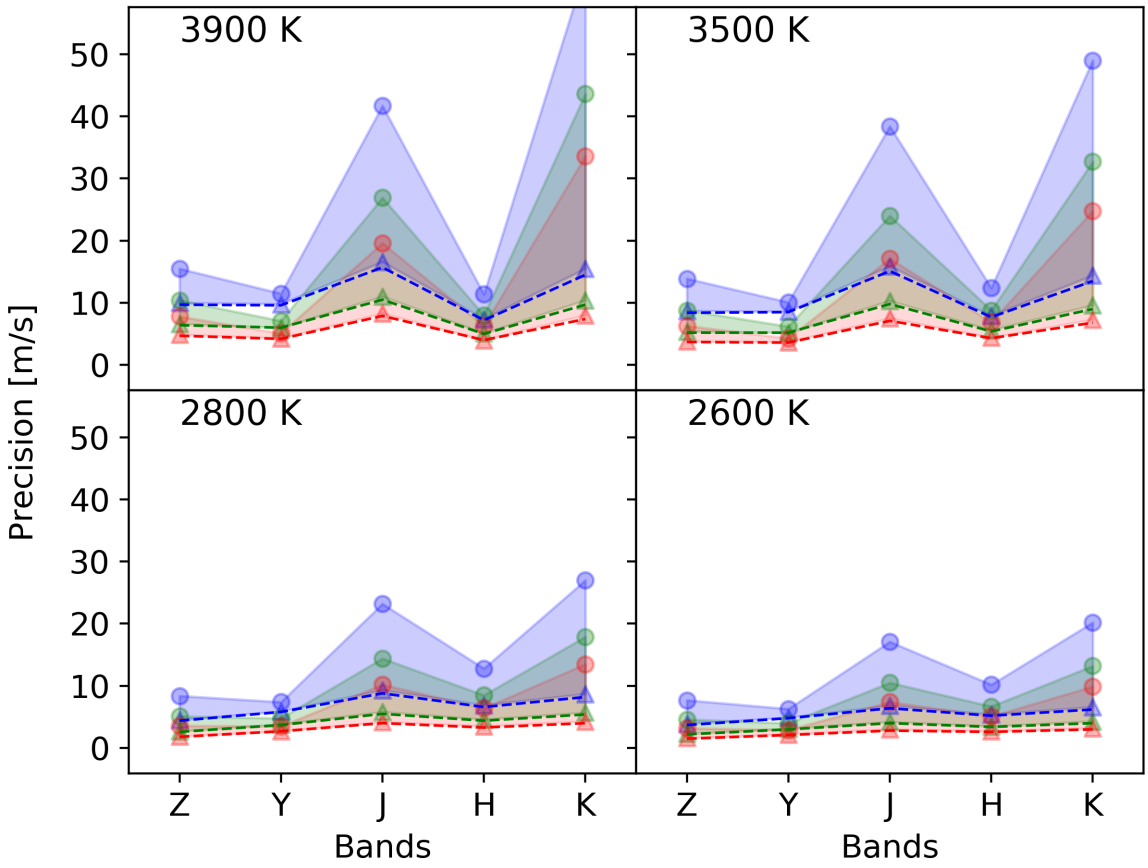
scripts.precision_four_panel.**plot_precision**(*precision_file*, *teffs=None*, *logg=4.5*, *fe_h=0.0*, *vsini=1.0*, *sampling=3*)

Plot precision 4 panel with RV precision.

Saves figure to `plots/`.

Parameters

- **precision_file** (*str*) – Name of phoenix_precision.py output.
- **teffs** (*List of int or None*) – Stellar temperatures. Default is [3900, 3500, 2800, 2600].
- **logg** (*int*) – Stellar Logg. Default is 4.5.
- **fe_h** (*int*) – Stellar metallicity. Default is 0.0.
- **vsini** (*float*) – Rotational velocity. Default is 1.0.
- **sampling** – Spectral sampling. Default is 3.



9.5 csv2tsv.py

Simple script using pandas to transform from *.csv to *.tsv.

9.6 untar_here.py

Bundled script to un-tar the eniric data downloaded.

Uses the tarfile module to extract the data.

9.7 Download Scripts

Scripts to download the eniric data and phoenix data. They can be run from the command line.

```
scripts/download/download_eniric_data.sh
```

This is also available as a powershell script.

```
scripts/download/ps_download_eniric_data.ps1
```

The test data from the PHOENIX-ACES library is specifically downloaded using Starfish utilities in

```
scripts/download/download_test_aces.py.
```

CHAPTER 10

Example Notebooks

There are several Jupyter Notebooks which focus on different aspects of eniric.

Below are the links to their location in github or rendered with nbveiwer.

These are:

- Atmosphere model (Github) (nbviewer)
- Compare precisions (Github) (nbviewer)
- Incremental_precision.ipynb (Github) (nbviewer)
- Precision_example.ipynb (Github) (nbviewer)
- Precison_with_doppler-K-band.ipynb (Github) (nbviewer)
- Precison_with_doppler-Z-band.ipynb (Github) (nbviewer)
- Effect of spectral gradient (Github) (nbviewer)
- Split_verse_Weighted_masking (Github) (nbviewer)

10.1 Misc

- **Convolution speeds (Github) (nbviewer)** Compares convolution speed between `eniric` and PyAstronomy.
- Extract paper precisions (Github) (nbviewer) Uses `tabula` to extract the published precision values from Appendix 1 of Figueira et al. (2016).

CHAPTER 11

Utilities

Collection of utility functions for eniric.

eniric.utilities.**band_limits**(*band: str*) → Tuple[float, float]
Get wavelength limits of band in microns.

Parameters **band** (*str*) – Band letter to get wavelength range for.

Returns

- **wav_min** (*float*) – Lower wavelength bound of band in microns
- **wav_max** (*float*) – Upper wavelength bound of band in microns

eniric.utilities.**band_middle**(*band: str*) → float
Calculate band mid-point.

band: str Band label

Returns **middle** – Wavelength at middle band.

Return type float

eniric.utilities.**band_selector**(*wav: numpy.ndarray*, *flux: numpy.ndarray*, *band: str*) → Tuple[numpy.ndarray, numpy.ndarray]
Select a specific wavelength band.

Parameters

- **wav** (*array-like*) – Wavelength values.
- **flux** (*array-like*) – Flux values.
- **band** (*str*) – Band letter to select, upper or lower case is supported. Options are ("ALL" or ""), "VIS", "GAP", "Z", "Y", "J", "H", "K".

eniric.utilities.**cpu_minus_one**() → int
Get one less than number of CPUs available.

Returns **num_cpu_minus_1** – One less than number of CPUs, or one.

Return type int

eniric.utilities.**doppler_limits**(*rvmax*, *wmin*, *wmax*)

Calculate wavelength limits to apply if preforming doppler shift.

To avoid any edge effects within wmin and wmax after doppler shift.

Parameters

- **rvmax** (*float*) – Maximium absolute RV offset in km/s. Uses np.abs() to constrain as absolute.
- **wmin** (*float*) – Lower wavelength limit.
- **wmax** (*float*) – Upper wavelength limit.

Returns

- **new_wmin** (*float*) – Lower wavelength bound shifted by -rvmax
- **new_wmax** (*float*) – Lower wavelength bound shifted by +rvmax

eniric.utilities.**doppler_shift_flux**(*wavelength: numpy.ndarray*, *flux: numpy.ndarray*, *vel: float*, *new_wav: Optional[numpy.ndarray] = None*)

Doppler shift flux by a given velocity, return it at the original wavelengths (non-relativistic).

Apply Doppler shift to the wavelength values of the spectrum using the velocity value provided and the relation (Deltalambda / lambda = v / c)

Then linearly interpolate the flux with the new wavelength to the old wavelengths.

Parameters

- **wavelength** (*ndarray*) – Wavelength vector
- **flux** (*ndarray*) – Flux vector
- **vel** (*float*) – Velocity to Doppler shift by in km/s.
- **new_wav** (*Optional[ndarray]*) – New wavelength array to evaluate the doppler shifted flux at. If None then defaults to new_wav=wavelength.

Returns **new_flux** – Doppler-shifted flux evaluated at new_wav.

Return type ndarray

eniric.utilities.**doppler_shift_wav**(*wavelength: numpy.ndarray*, *vel: float*)

Doppler shift wavelength by a given velocity (non-relativistic).

Apply Doppler shift to the wavelength values of the spectrum using the velocity value provided and the relation (Deltalambda / lambda = v / c)

Parameters

- **wavelength** (*ndarray*) – Wavelength vector
- **vel** (*float*) – Velocity to Doppler shift by in km/s.

Notes

The Doppler shift is calculated using the relation [Deltalambda / lambda = v / c]

Where RV is the radial velocity (in km/s), (lambda_0)‘ is the rest wavelength and (Deltalambda) is the wavelength shift, (lambda_{shift} - lambda_0)

eniric.utilities.**issequenceforme**(*obj*)

eniric.utilities.**load_aces_spectrum**(*params: Union[numpy.ndarray, List[float]], photons: bool = True, air: bool = False, wl_range: Union[List[float], Tuple[float, float]] = (3000, 54000)*)

Load a Phoenix spectrum from the phoenix library using STARFISH.

Parameters

- **params** (*ndarray*) – [temp, logg, metallicity(, alpha)]
- **photons** (*bool*) – Necessary conversions into photons for precisions.
- **air** (*bool*) – Convert to air wavelengths (default = False).
- **wl_range** (*(float, float)*) – Min/Max wavelength range to load. Default = (3000, 54000) Angstrom.

Returns

- **wav_micron** (*ndarray*) – Wavelength in microns
- **flux_micron** (*ndarray*) – Photon counts per (cm**2 s) or SED/micron (within a multiplicative constant 1/(h*c)).
- **Spectra available from http** (*//phoenix.astro.physik.uni-goettingen.de*)

eniric.utilities.**load_btsettl_spectrum**(*params: Union[numpy.ndarray, List[float]], photons: bool = True, air: bool = False, wl_range: Union[List[float], Tuple[float, float]] = (3000, 30000)*)

Load a BT-Settl spectrum from the CIFIST2011 library using STARFISH.

Parameters

- **params** (*ndarray*) – [temp, logg]. Metallicity = 0, alpha = 0
- **photons** (*bool*) – Necessary conversions into photons for precisions.
- **air** (*bool*) – Convert to air wavelengths (default = False).
- **wl_range** (*(float, float)*) – Min/Max wavelength range to load. Default = (3000, 30000) Angstrom.

Returns

- **wav_micron** (*ndarray*) – Wavelength in microns
- **flux_micron** (*ndarray*) – Photon counts per (cm**2 s) or SED/micron. (within a multiplicative constant 1/(h*c))

Notes

From BT-SETTL readme: CIFIST2011_2015: published version of the BT-Settl grid (Baraffe et al. 2015, Allard et al. 2015. This grid will be the most complete of the CIFIST2011 grids above, but currently: Teff = 1200 - 7000K, logg = 2.5 - 5.5, [M/H] = 0.0.

Available from https://phoenix.ens-lyon.fr/Grids/BT-Settl/CIFIST2011_2015/FITS/

The BT-Settl models are Sampled at a higher rate than PHOENIX-ACES (>10 X). Therefore we cut it by factor of 10 during loading to give them similar number of points. This makes the convolutions go 10X faster.

eniric.utilities.**mask_between**(*x: numpy.ndarray*, *xmin: float*, *xmax: float*) → numpy.ndarray

Create boolean mask of x between xmin and xmax.

eniric.utilities.**moving_average**(*x: numpy.ndarray, window_size: Union[int, float]*) → numpy.ndarray
Moving average.

eniric.utilities.**res2int**(*res: Any*) → int
Convert from "100k" or "100000" to 100000.

eniric.utilities.**res2str**(*res: Any*) → str
Convert from "100000" or 100000 to "100k".

eniric.utilities.**resolutions2ints**(*resolution: Sequence[Any]*) → List[int]
List of resolutions to list of integer resolutions.

Convert from ["100k", "30000"] to [100000, 30000].

eniric.utilities.**resolutions2strs**(*resolution: Sequence[Any]*) → List[str]
List of resolutions to list of string resolutions.

Convert from ["100000", 10000] to ["100k", "10k"].

eniric.utilities.**rv_cumulative**(*rv_vector: Union[List, numpy.ndarray], single: bool = False*) → List[float]
Function that calculates the cumulative RV vector weighted_error.

eniric.utilities.**rv_cumulative_full**(*rv_vector: Union[List, numpy.ndarray]*) → numpy.ndarray
Function that calculates the cumulative RV vector weighted_error. In both directions.

eniric.utilities.**silent_remove**(*filename: str*) → None
Remove file without failing when it doesn't exist.

eniric.utilities.**wav_selector**(*wav: Union[numpy.ndarray, List[float]], flux: Union[numpy.ndarray, List[float]], wav_min: float, wav_max: float*) → Tuple[numpy.ndarray, numpy.ndarray]
function that returns wavelength and flux within a giving range

Parameters

- **wav** (*array-like*) – Wavelength array.
- **flux** (*array-like*) – Flux array.
- **wav_min** (*float*) – Lower bound wavelength value.
- **wav_max** (*float*) – Upper bound wavelength value.

Returns

- **wav_sel** (*array*) – New wavelength array within bounds wav_min, wav_max
- **flux_sel** (*array*) – New wavelength array within bounds wav_min, wav_max

eniric.utilities.**weighted_error**(*rv_vector: Union[List[float], numpy.ndarray]*) → float
Function that calculates the average weighted error from a vector of errors.

Eniric is a Python 3.6+ software written to calculate the theoretical Radial Velocity (RV) precision of Near-InfraRed (NIR) stellar spectra. Eniric is an overhaul and extension to the code initially used in Figueira et al. 2016 to analysis the precision of M-dwarf stars in the NIR, extending its ability to use any synthetic spectra from the PHOENIX-ACES and BT-Settl libraries, or user provided spectra, and making it easier and faster to use. Extending the performance and usability, it is able to be used on any synthetic spectra from the PHOENIX-ACES and BT-Settl (CIFIST2011-2015) libraries.

To get started see the *Installation*, or the *Basic Usage*.

CHAPTER 12

Features

`Eniric` contains a number of tools to transform and analyse synthetic or observed spectra.

- ***Spectral broadening*:** Allows for Rotational and Instrumental broadening of synthetic spectra given a rotation speed `vsini` and resolution `R`.
- ***Atmospheric transmission* masking:** Analyzing the RV precision attainable under the different masking conditions presented in Figueira et al. 2016.

  The three conditions specifically treated are:

  - No contamination or treatment of atmospheric transmission
  - Masking all regions affected by atmospheric absorption of a given depth % over the course of the year.
  - Assuming perfect telluric correction in which the variance of the measured flux is impacted.
- **Spectral *Resampling*** Allows for resampling of synthetic spectra to `N` pixels per FWHM. Default is 3.
- ***SNR normalization*** Normalize spectral flux to a defined SNR level.
- **Photometric band selection** Analysis splitable into individual photometric bands `Z`, `Y`, `J`, `H`, `K`. User definable.
- ***Theoretical RV precision*** Compute spectral RV precision and spectral quality.
- **Incremental quality/precision** Determine the RV precision and spectral quality on narrow wavelength slices across the entire spectrum, similar to that present in Figure 1 of both Bouchy et al.(2001) and Artigau et al. (2018).
- **Analyse relative precision of synthetic libraries** The RV precision of are present relative to a specified SNR per pixel in the center of a photometric band. The default as used in Figueira et al. 2016 is a SNR of 100 at the center of the J-band.
  - **Available through Starfish's grid_tools.**
    - PHOENIX-ACES
    - BT-Settl

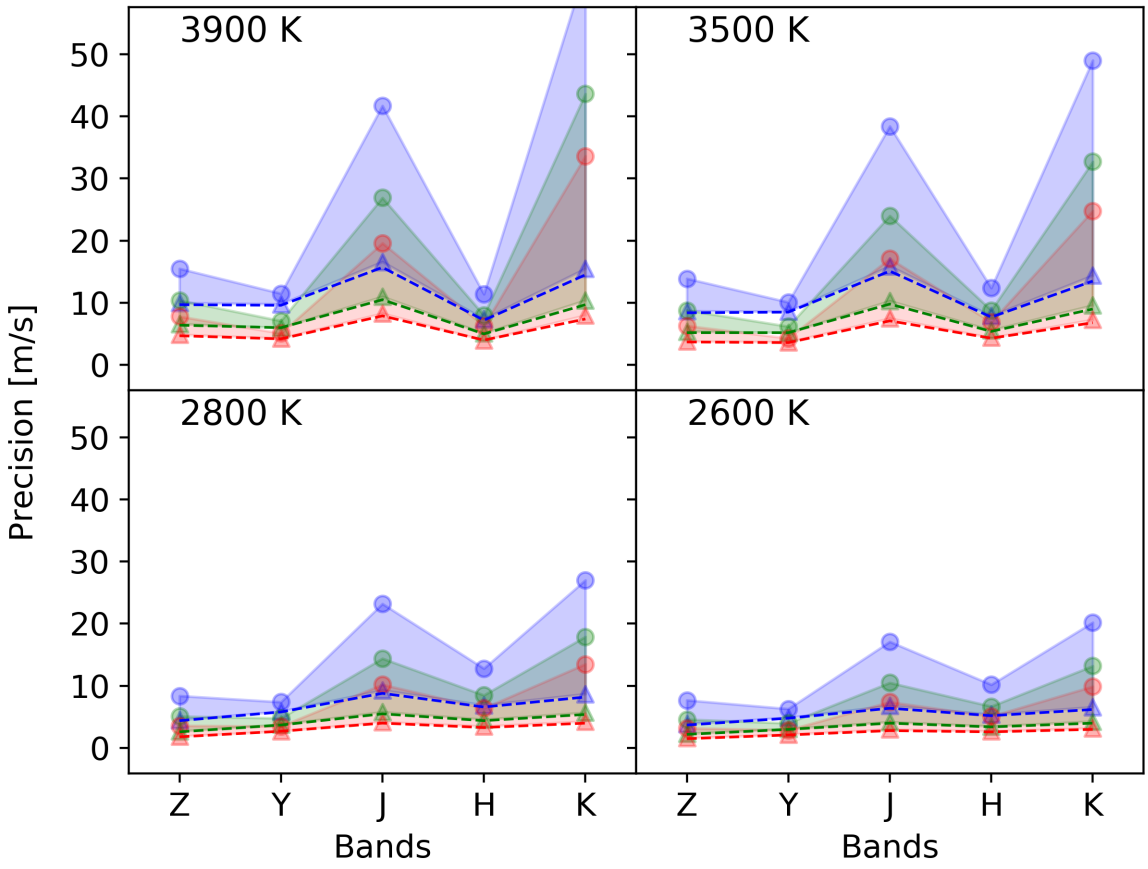


Fig. 1: Precision achieved with `eniric` as a function of spectral band for stars with a rotational velocity of vsini=1.0 km/s and temperatures 3900 K, 3500 K, 2800 K, 2600 K, corresponding to spectral types M0, M3, M6, and M9 respectively. The dashed line represents the theoretical limits imposed by condition 1, and the filled area represents the values within the limits set by conditions 2 (circles) and 3 (triangles); blue, green, and red represent the results obtained for resolutions of 60000, 80000, and 100000, respectively. The spectra were normalized to have a SNR of 100 per resolution element as measured at the center of the J-band. This is similar to Figure 1 from Figueira et al. 2016 but with updated precision values.

CHAPTER 13

Background

The theoretical background and first version of the code used in Eniric was presented in Figueira et al. 2016.

```
P. Figueira, V. Zh. Adibekyan, M. Oshagh, J. J. Neal, B. Rojas-Ayala, C. Lovis, C.␣
↪Melo, F. Pepe, N. C. Santos, M. Tsantaki, 2016,
Radial velocity information content of M dwarf spectra in the near-infrared,
Astronomy and Astrophysics, 586, A101
```

After *Installation* and *Configuration* the updated results for Figueira et al. 2016 can be reproduced by `eniric` with the following command.

```
phoenix_precision.py -t 3900 3500 2800 2600 -l 4.5 -m 0.0 -r 60000 80000 100000 -v 1.
↪0 5.0 10.0 -b Z Y J H K
```

CHAPTER 14

Support

If you are having issues, please let us know.

You can submit an issue on Github.

CHAPTER 15

License

The project is licensed under the MIT License.

15.1 Indices and tables

- genindex
- modindex
- search

Python Module Index

e

s

Index