WDmodel Documentation

Release 0.4

Gautham Narayan

Aug 01, 2019

CONTENTS

1 WDmodel **3**
- 1.1 About . . . 3
- 1.2 Compatibility . . . 3
- 1.3 Analysis . . . 3

2 Help **5**
- 2.1 Installing WDmodel . . . 5
  - 2.1.1 Installation Instructions . . . 5
  - 2.1.2 Extra . . . 7
- 2.2 Using WDmodel . . . 8
  - 2.2.1 Usage . . . 8
  - 2.2.2 Useful runtime options . . . 9
- 2.3 Analyzing WDmodel . . . 10
  - 2.3.1 Analysis . . . 10
- 2.4 WDmodel . . . 13
  - 2.4.1 WDmodel package . . . 13

3 Indices and tables **59**

Python Module Index **61**

Index **63**

WDmodel: Bayesian inference of white dwarf properties from spectra and photometry to establish spectrophotometric standards

CHAPTER

ONE

WDMODEL

1.1 About

`WDmodel` is a DA White Dwarf model atmosphere fitting code. It fits observed spectrophotometry of DA White Dwarfs to infer intrinsic model atmosphere parameters in the presence of dust and correlated spectroscopic flux calibration errors, thereby determining full SEDs for the white dwarf. Its primary scientific purpose is to establish a network of faint (V = 16.5–19 mag) standard stars, suitable for LSST and other wide-field photometric surveys, and tied to HST and the CALSPEC system, defined by the three primary standards, GD71, GD153 and G191B2B.

Click on the badges above for code, licensing and documentation.

1.2 Compatibility

The code has been tested on Python 3.6 on both OS X (El Capitan, Sierra and High Sierra) and Linux (Debian-derivatives). Python 3.6 is highly recommended. Python 2.7 should work, but consider upgrading because this package's dependencies are dropping 2.7 support. Send us email or open an issue if you need help!

1.3 Analysis

We've published the results from our combined Cycle 22 and Cycle 20 data, while ramping up for Cycle 25!

You can read about the analysis in Narayan et al., 2019

and the data in Calamida et al., 2019

The data from Cycle 20 and 22 are available on Zenodo

You can read also the first version of our preliminary analysis of four of the Cycle 20 objects here

That analysis was intended as a proof-of-concept and used custom IDL routines from Jay Holberg (U. Arizona) to infer DA intrinsic parameters and custom python code to fit the reddening parameters. This code is intended to (significantly) improve on that analysis.

CHAPTER

TWO

HELP

This document will help get you up and running with the `WDmodel` package.

For the most part, you can simply execute code in grey boxes to get things up and running, and ignore the text initially. Come back to it when you need help, or to configure the fitter.

2.1 Installing WDmodel

This document will step you through getting the `WDmodel` package installed on your system.

- ***Installation instructions***
  - *Get python*
  - *Get the code*
  - *Install everything*
  - *Get auxillary pysynphot files*
  - *Install the code*
- *Some extra notes*

2.1.1 Installation Instructions

Here's a minimal set of instructions to get up and running. We will eventually get this package up on PyPI and conda-forge, and that should make this even easier.

0. Install python:

We recommend using the anaconda python distribution to run this package. If you don't have it already, follow the instructions here

Make sure you added the conda/bin dir to your path!

If you elect to use your system python, or some other distribution, we will assume you know what you are doing, and you can, skip ahead.

1. Get the code:

Clone this repository

```
git clone https://github.com/gnarayan/WDmodel.git
cd WDmodel
```

2. Install everything:

a. Create a new environment from specification (Preferred! All dependencies resolved!)

```
conda env create -f docs/env/conda_environment_py36_[osx64|i686].yml
```

or

b. Create a new environment from scratch (Let conda figure out dependencies and you sort out potential issues)

```
cp docs/env/condarc.example ~/.condarc
conda create -n WDmodel
source activate WDmodel
conda install --yes --file dependencies_py36.txt
```

Setting up an environment vs setting up a known good environment

The `env` folder contains files to help get you setup using a consistent environment with all packages specified.

The `requirements_py[27|36].txt` files contains a list of required python packages and known working versions for each. They differ from the `dependencies_py[27|36].txt` files in the root directory in that those files specify packages and version ranges, rather than exact versions, to allow conda to resolve dependecies and pull updated versions.

Of course, the environment really needs more than just python packages, while `pip` only manages python packages. The conda environment files, `conda_environment_py[27|37]_[osx64|i686].yml` files can be used to create conda environments with exact versions of all the packages for python 2.7 or 3.6 on OS X or linux. This is the most reliable way to recreate the entire environment.

3. Get the latest HST CDBS files:

These are available over FTP from [ftp://archive.stsci.edu/pub/hst/pysynphot/]

Untar them wherever you like, and set the `PYSYN_CDBS` environment variable. You need at least `synphot1.tar.gz` and `synphot6.tar.gz`.

```
export PYSYN_CDBS=place_you_untarred_the_files
```

4. Install the package [optional]:

```
python setup.py install
```

2.1.2 Extra

The instructions should be enough to get up and running, even without `sudo` privileges. There's a few edge cases on cluster environments though. These notes may help:

Some extra notes on installation

If you followed the installation process detailed above, you shouldn't need these notes, but they are provided for users who may be running on environments they do not manage themselves.

- *Installing eigen3 without conda*
- *Installing OpenMPI and mpi4py without conda*
- *Installing on a cluster*

Installing eigen3 without conda

If eigen3 isn't on your system, and installing it with conda didn't work

For OS X do:

```
brew install eigen
```

or on a linux system with apt:

```
apt-get install libeigen3-dev
```

or compile it from source

Note that if you do install it in a custom location, you may have to compile celerite yourself.

```
pip install celerite --global-option=build_ext --global-option=-I/path/to/eigen3
```

Installing OpenMPI and mpi4py without conda

if no mpi is on your system, and installing it with conda didn't work

For OS X do:

```
brew install [mpich|mpich2|open-mpi]
```

on a linux system with apt:

```
apt-get install openmpi-bin
```

and if you had to resort to brew or apt, then finish with:

```
pip install mpi4py
```

Notes from installing on the Odyssey cluster at Harvard

These may be of use to get the code up and running with MPI on some other cluster. Good luck.

Odyssey uses the lmod system for module management, like many other clusters You can `module spider openmpi` to find what the openmpi modules.

The advantage to using this is distributing your computation over multiple nodes. The disadvantage is that you have to compile mpi4py yourself against the cluster mpi.

```
module load gcc/6.3.0-fasrc01 openmpi/2.0.2.40dc0399-fasrc01
wget https://bitbucket.org/mpi4py/mpi4py/downloads/mpi4py-2.0.0.tar.gz
tar xvzf mpi4py-2.0.0.tar.gz
cd mpi4py-2.0.0
python setup.py build --mpicc=$(which mpicc)
python setup.py build_exe --mpicc="$(which mpicc) --dynamic"
python setup.py install
```

2.2 Using WDmodel

This document will help you get comfortable using the `WDmodel` package.

- *Usage*
  - *Get data*
  - *Running single threaded*
  - *Running with MPI*
- *Useful options*
  - *Quick analysis*
  - *Initializing the fitter*
  - *Configuring the sampler*
  - *Resuming the fit*

2.2.1 Usage

This is the TL;DR version to get up and running.

1. Get the data:

Instructions will be available here when the paper is accepted. In the meantime there's a single test object in the spectroscopy directory. If you want more, Write your own HST proposal! :-P

2. Run a fit single threaded:

```
fit_WDmodel --specfile data/spectroscopy/yourfavorite.flm
```

This option is single threaded and slow, but useful to testing or quick exploratory analysis.

A more reasonable way to run things fast is to use mpi.

3. Run a fit as an MPI process:

```
mpirun -np 8 fit_WDmodel --mpi --specfile=file.flm [--ignorephot]
```

Note that `--mpi` **MUST** be specified in the options to `WDmodel` and you must start the process with `mpirun`

2.2.2 Useful runtime options

There's a large number of command line options to the fitter, and most of it's aspects can be configured. Some options make sense in concert with others, and here's a short summary of use cases.

Quick looks

The spectrum can be trimmed prior to fitting with the `--trimspec` option. You can also blotch over gaps and cosmic rays if your reduction was sloppy, and you just need a quick fit, but it's better to do this manually.

If there is no photometry data for the object, the fitter will barf unless `--ignorephot` is specified explicitly, so you know that the parameters are only constrained by the spectroscopy.

The fitter runs a MCMC to explore the posterior distribution of the model parameters given the data. If you are running with the above two options, chances are you are at the telescope, getting spectra, and doing quick look reductions, and you just want a rough idea of temperature and surface gravity to decide if you should get more signal, and eventually get HST photometry. The MCMC is overkill for this purpose so you can `--skipmcmc`, in which case, you'll get results using minuit. They'll be biased, and the errors will probably be too small, but they give you a ballpark estimate.

If you do want to use the MCMC anyway, you might like it to be faster. You can choose to use only every nth point in computing the log likelihood with `--everyn` - this is only intended for testing purposes, and should probably not be used for any final analysis. Note that the uncertainties increase as you'd expect with fewer points.

Setting the initial state

The fitter really runs minuit to refine initial supplied guesses for parameters. Every now at then, the guess prior to running minuit is so far off that you get rubbish out of minuit. This can be fixed by explicitly supplying a better initial guess. Of course, if you do that, you might wonder why even bother with minuit, and may wish to skip it entirely. This can be disabled with the `--skipminuit` option. If `--skipminuit` is used, a dl guess **MUST** be specified.

All of the parameter files can be supplied via a JSON parameter file supplied via the `--param_file` option, or using individual parameter options. An example parameter file is available in the module directory.

Configuring the sampler

You can change the sampler type (`-samptype`), number of chain temperatures (`--ntemps`), number of walkers (`--nwalkers`), burn in steps (`--nburnin`), production steps (`--nprod`), and proposal scale for the MCMC (`--ascale`). You can also thin the chain (`--thin`) and discard some fraction of samples from the start (`--discard`). The default sampler is the ensemble sampler from the `emcee` package. For a more conservative approach, we recommend the ptsampler with `ntemps=5`, `nwalkers=100`, `nprod=5000` (or more).

Resuming the fit

If the sampling needs to be interrupted, or crashes for whatever reason, the state is saved every 100 steps, and the sampling can be restarted with `--resume`. Note that you must have run at least the burn in and 100 steps for it to be possible to resume, and the state of the data, parameters, or chain configuration should not be changed externally (if

they need to be use `--redo` and rerun the fit). You can increase the length of the chain, and chain the visualization options when you `--resume` but the state of everything else is restored.

You can get a summary of all available options with `--help`

Useful routines

There are a few useful routines included in the `WDmodel` package. Using `WDmodel` itself will do the same thing as `fit_WDmodel`. If you need to look at results from a large number of fits, `print_WDmodel_result_table` and `print_WDmodel_residual_table` will print out tables of results and residuals. `make_WDmodel_slurm_batch_scripts` provides an example script to generate batch scripts for the SLURM system used on Harvard's Odyssey cluster. Adapt this for use with other job queue systems or clusters.

2.3 Analyzing WDmodel

This document describes the output produced by the `WDmodel` package.

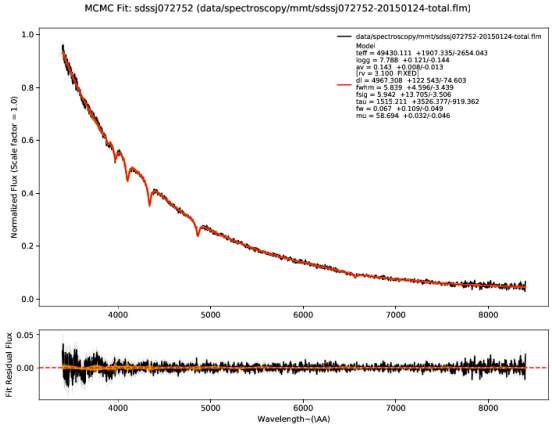
- *Analysis*
  - *The fit*
  - *Spectral flux calibration errors*
  - *Hydrogen Balmer line diagnostics*
  - *Posterior distributions*
  - *Output files*

2.3.1 Analysis

There's many different outputs (ascii files, bintables, plots) that are produced by the `WDmodel` package. We'll describe the plots first - it is a good idea to look at your data before using numbers from the analysis.

1. The fit:

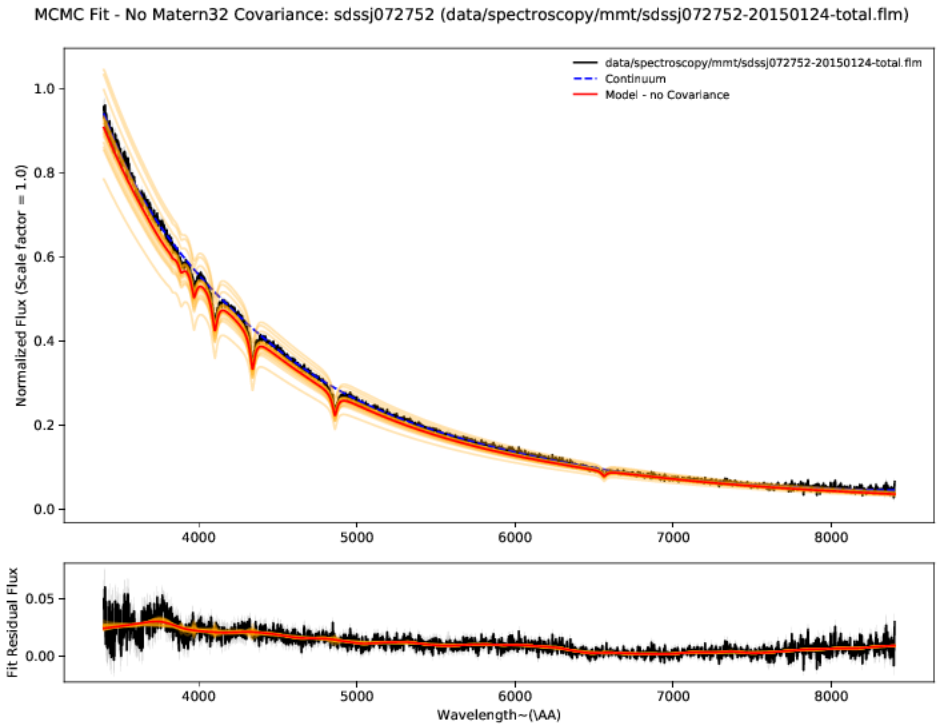
All the plots are stored in `<spec basename>_mcmc.pdf` in the output directory that is printed as you run the `WDmodel` fitter (default: `out/<object name>/<spec basename>/`.



The first plots show you the bottom line - the fit of the model (red) to the data - the observed photometry and spectroscopy (black). Residuals for both are shown in the lower panel. The model parameters inferred from the data are shown in the legend of the spectrum plot. Draws from the posterior are shown in orange. The number of these is customizable with `--ndraws`. Observational uncertainties are shown as shaded grey regions.

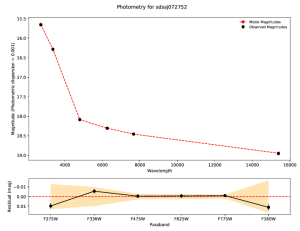
If both of these don't look reasonable, then the inferred parameters are probably meaningless. You should look at why the model is not in good agreement with the data. We've found this tends to happen if there's a significant flux excess at some wavelengths, indicating a companion or perhaps variability.

2. Spectral flux calibration errors:

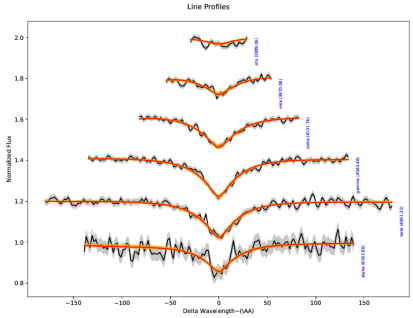


The `WDmodel` package uses a Gaussian process to model correlated flux calibration errors in the spectrum. These arise from a variety of sources (flat-fielding, background subtraction, extraction of the 2D spectrum with a polynomial, telluric feature removal, and flux calibration relative to some other spectrophotometric standard, which in turn is probably only good to a few percent). However, most of the processes that generate these errors would cause smooth and continuous deformations to the observed spectrum, and a single stationary covariance kernel is a useful and reasonable way to model the effects. The choice of kernel is customizable (`--covtype`, with default `Matern32` which has proven more than sufficient for data from four different spectrographs with very different optical designs).

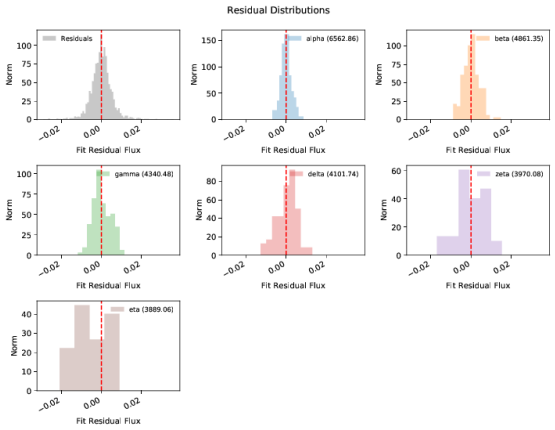
The residual plot shows the difference between the spectrum and best fit model without the Gaussian process applied. The residuals therefore show our estimate of the flux calibration errors and the Gaussian process model for them.



3. Hydrogen Balmer line diagnostics:



These plots illustrate the spectroscopic data and model specifically in the region of the Hydrogen Balmer lines. While the entire spectrum and all the photometry is fit simultaneously, we extract the Balmer lines, normalize their local continua to unity, and illustrate them separately here, offsetting each vertically a small amount for clarity.



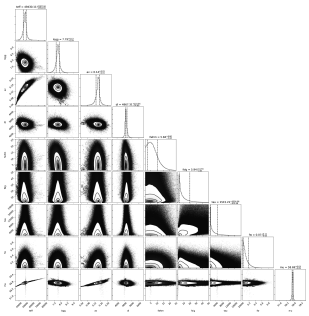
With the exception of the SDSS `autofit` package used for their white dwarf analysis (which isn't public in any case), every white dwarf atmosphere fitting code takes the approach of only fitting the Hydrogen Balmer lines to determine model parameters. This includes our own proof-of-concept analysis of Cycle 20 data. The argument goes that the inferred model parameters aren't sensitive to reddening if the local continuum is divided out, and the line profiles determine the temperature and surface gravity.

In reality, reddening also changes the shape of the line profiles, and to divide out the local continuum, a model for it had to be fit (typically a straight line across the line from regions defined "outside" the line profile wings). The properties of this local continuum *are* strongly correlated with reddening, and errors in the local continuum affect the inference of the model parameters, particularly at low S/N. This is the regime our program to observe faint white dwarfs operates in - low S/N with higher reddening. Any reasonable analysis of systematic errors should illustrate significant bias resulting from the naive analysis in the presence of correlated errors.

In other words, the approach doesn't avoid the problem, so much as shove it under a rug with the words "nuisance parameters" on top. This is why we adopted the more complex forward modeling approach in the `WDmodel` package. Unfortunately, Balmer profile fits are customary in the field, so after we forward model the data, we make a simple polynomial fit to the continuum (using our best understanding of what SDSS' `autofit` does), and extract out the Balmer lines purely for this visualization. This way the polynomial continuum model does have no affect on the inference, and if it goes wrong and the Balmer line profile plots look wonky, it doesn't actually matter.

If you mail the author about these plots, he will get annoyed and grumble about you, and probably reply with snark. He had no particular desire to even include these plots.

4. Posterior Distributions:



A corner plot of the posterior distribution. If the model and data are not in good agreement, then this is a good place to look. If you are running with `--samptype=ensemble` (the default), you might consider `--samptype=pt --ntemps=5 --nwalkers=100 --nprod=5000 --thin 10` to better sample the posterior, and map out any multi-modality.

5. Output Files:

This table describes the output files produced by the fitter.

File	Description
<spec basename>_inputs.hdf5	All inputs to fitter and visualization module. Restored on --resume
<spec basename>_params.json	Initial guess parameters. Refined by minuit if not --skipminuit
<spec basename>_minuit.pdf	Plot of initial guess model, if refined by minuit
<spec basename>_mcmc.hdf5	Full Markov Chain - positions, log posterior, chain attributes
<spec basename>_mcmc.pdf	Plot of model and data after MCMC
<spec basename>_result.json	Summary of inferred model parameters, errors, uncertainties after MCMC
<spec basename>_spec_model.dat	Table of the observed spectrum and inferred model spectrum
<spec basename>_phot_model.dat	Table of the observed photometry and inferred model photometry
<spec basename>_full_model.hdf5	Derived normalized SED of the object

See *Useful routines* for some useful routines to summarize fit results.

2.4 WDmodel

2.4.1 WDmodel package

WDmodel: Bayesian inference of white dwarf properties from spectra and photometry to establish spectrophotometric standards

Submodules

WDmodel.WDmodel module

DA White Dwarf Atmosphere Models and SED generator.

Model grid originally from J. Holberg using I. Hubeny's Tlusty code (v200) and custom Synspec routines, repackaged into HDF5 by G. Narayan.

class WDmodel.WDmodel.**WDmodel**(*grid_file=None*, *grid_name=None*, *rvmodel='f99'*)

Bases: object

DA White Dwarf Atmosphere Model and SED generator

Base class defines the routines to generate and work with DA White Dwarf model spectra. Requires a grid file of DA White Dwarf atmospheres. This grid file is included along with the package - TlustyGrids.hdf5 - and is the default.

Parameters

- **grid_file** (*str, optional*) – Filename of the HDF5 grid file to read. See *WDmodel.io.read_model_grid()* for format of the grid file. Default is TlustyGrids.hdf5, included with the *WDmodel* package.

- **grid_name** (*str, optional*) – Name of the HDF5 group containing the white dwarf model atmosphere grids in `grid_file`. Default is `default`
- **rvmodel** ({'ccm89','od94','f99','custom'}, optional) – Specify parametrization of the reddening law. Default is `'f99'`.

rvmodel	parametrization
`'ccm89'`	Cardelli, Clayton and Mathis (1989, ApJ, 345, 245)
`'od94'`	O'Donnell (1994, ApJ, 422, 158)
`'f99'`	Fitzpatrick (1999, PASP, 111, 63)
`'custom'`	Custom law from Jay Holberg (email, 20180424)

_lines
dictionary mapping Hydrogen Balmer series line names to line number, central wavelength in Angstrom, approximate line width and continuum region width around line. Used to extract Balmer lines from spectra for visualization.

Type dict

_grid_file
Filename of the HDF5 grid file that was read.

Type str

_grid_name
Name of the HDF5 group containing the white dwarf model atmosphere

Type str

_wave
Array of model grid wavelengths in Angstroms, sorted in ascending order

Type array-like

_ggrid
Array of model grid surface gravity values in dex, sorted in ascending order

Type array-like

_tgrid
Array of model grid temperature values in Kelvin, sorted in ascending order

Type array-like

_nwave
Size of the model grid wavelength array, `_wave`

Type int

_ngrav
Size of the model grid surface gravity array, `_ggrid`

Type int

_ntemp
Size of the model grid temperature array, `_tgrid`

Type int

_flux
Array of model grid fluxes, shape `(_nwave, _ntemp, _ngrav)`

Type array-like

_lwave
Array of model grid `log10` wavelengths for interpolation

Type array-like

_lflux
Array of model grid `log10` fluxes for interpolation, shape `(_ntemp, _ngrav, _nwave)`

Type array-like

_law

Type extinction function corresponding to `rvmodel`

Returns **out**

Return type *WDmodel.WDmodel.WDmodel* instance

Raises **ValueError** – If the supplied rvmodel is unknown

Notes

Virtually none of the attributes should be used directly since it is trivially possible to break the model by redefining them. Access to them is best through the functions connected to the models.

A custom user-specified grid file can be specified. See *WDmodel.io.read_model_grid()* for the format of the grid file.

Uses `scipy.interpolate.RegularGridInterpolator` to interpolate the models.

The class contains various convenience methods that begin with an underscore (_) that will not be imported by default. These are intended for internal use, and do not have the sanity checking and associated overhead of the public methods.

_WDmodel__init__rvmodel(*rvmodel='f99'*)

_WDmodel__init__tlusty(*grid_file=None*, *grid_name=None*)

__call__(*teff*, *logg*, *wave=None*, *log=False*, *strict=True*)
Returns the model flux given `teff` and `logg` at wavelengths `wave`

Wraps *WDmodel.WDmodel.WDmodel._get_model()* adding checking of inputs.

Parameters

- **teff** (*float*) – Desired model white dwarf atmosphere temperature (in Kelvin)
- **logg** (*float*) – Desired model white dwarf atmosphere surface gravity (in dex)
- **wave** (*array-like, optional*) – Desired wavelengths at which to compute the model atmosphere flux. If not supplied, the full model wavelength grid is returned.
- **log** (*bool, optional*) – Return the log10 flux, rather than the flux
- **strict** (*bool, optional*) – If strict, `teff` and `logg` out of model grid range raise a `ValueError`, otherwise raise a `RuntimeWarning` and set `teff`, `logg` to the nearest grid value.

Returns

- **wave** (*array-like*) – Valid output wavelengths
- **flux** (*array-like*) – Interpolated model flux at `teff`, `logg` and wavelengths `wave`

Raises **ValueError** – If `teff` or `logg` are out of range of model grid and `strict` is True or if there are any invalid wavelengths, or the requested wavelengths to do not overlap with the model grid

Notes

Unlike the corresponding private methods, the public methods implement checking of the inputs and returns the wavelengths in addition to the flux. Internally, we only use the private methods as the inputs only need to be checked once, and their state is not altered anywhere after.

__init__(*grid_file=None*, *grid_name=None*, *rvmodel='f99'*)
Initialize self. See help(type(self)) for accurate signature.

_custom_extinction(*wave*, *av*, *rv=3.1*, *unit='aa'*)
Return the extinction for `av`, `rv` at wavelengths `wave` for the custom reddening law defined by J. Holberg

Mimics the interface provided by *WDmodel.WDmodel.WDmodel._law* to calculate the extinction as a function of wavelength (in Angstroms), A_λ.

Parameters

- **wave** (*array-like*) – Array of wavelengths in Angstrom at which to compute extinction, sorted in ascending order
- **av** (*float*) – Extinction in the V band, A_V
- **rv** (*float, optional*) – Fixed to 3.1 for J. Holberg's custom reddening law

Returns **out** – Extinction at wavelengths `wave` for `av` and `rv`

Return type array-like

Notes

`av` should be >= 0.

_extract_from_indices(*w*, *f*, *ZE*, *df=None*)
Extracts slices of multiple arrays for the same set of indices.

Convenience function to extract elements of wavelength `w`, flux `f` and optionally flux uncertainty `df` using indices `ZE`

Parameters

- **w** (*array-like*) – Wavelength array from which to extract indices `ZE`
- **f** (*array-like*) – Flux array from which to extract indices `ZE`
- **ZE** (*array-like*) – indices to extract
- **df** (*None or array-like, optional*) – If array-like, extracted elements of this array are also returned

Returns

- **w** (*array-like*) – elements of input wavelength array at indices `ZE`
- **f** (*array-like*) – elements of input flux array at indices `ZE`
- **[df]** (*array-like*) – elements of input flux uncertainty array at indices `ZE` if optional input `df` is supplied

_extract_spectral_line(*w*, *f*, *line*, *df=None*)
Extracts slices of multiple arrays corresponding to a hydrogen Balmer line

Convenience function to extract elements of wavelength `w`, flux `f` and optionally flux uncertainty `df` for a hydrogen Balmer `line`. Wraps *WDmodel.WDmodel.WDmodel._get_line_indices()* and *WDmodel.WDmodel.WDmodel._extract_from_indices()*, both of which have their own reasons for existence as well.

Parameters

- **w** (*array-like*) – Wavelength array from which to extract elements corresponding to hydrogen Balmer `line`
- **f** (*array-like*) – Flux array from which to extract elements corresponding to hydrogen Balmer `line`
- **line** ({'alpha', 'beta', 'gamma', 'delta', 'zeta', 'eta'}) – Name of hydrogen Balmer line to extract. Properties are pre-defined in *WDmodel.WDmodel.WDmodel._lines*
- **df** (*None or array-like, optional*) – If array-like, extracted elements of this array are also returned

Returns

- **w** (*array-like*) – elements of input wavelength array for hydrogen Balmer feature `line`
- **f** (*array-like*) – elements of input flux array for hydrogen Balmer feature `line`
- **[df]** (*array-like*) – elements of input flux uncertainty array for hydrogen Balmer feature `line` if optional input `df` is supplied

Notes

Same as *WDmodel.WDmodel.WDmodel.extract_spectral_line()* without checking of inputs and therefore corresponding overhead. Used internally.

_get_full_obs_model(*teff*, *logg*, *av*, *fwhm*, *wave*, *rv=3.1*, *log=False*, *pixel_scale=1.0*)
Returns the observed model flux given `teff`, `logg`, `av`, `rv`, `fwhm` (for Gaussian instrumental broadening) at wavelengths, `wave` as well as the full SED.

Convenience function that does the same thing as *WDmodel.WDmodel.WDmodel._get_obs_model()*, but also returns the full SED without any instrumental broadening applied, appropriate for synthetic photometry.

Uses *WDmodel.WDmodel.WDmodel._get_model()* to get the unreddened model, and reddens it with *WDmodel.WDmodel.WDmodel.reddening()* and convolves it with a Gaussian kernel using `scipy.ndimage.filters.gaussian_filter1d()`

Parameters

- **teff** (*float*) – Desired model white dwarf atmosphere temperature (in Kelvin)
- **logg** (*float*) – Desired model white dwarf atmosphere surface gravity (in dex)
- **av** (*float*) – Extinction in the V band, A_V
- **fwhm** (*float*) – Instrumental FWHM in Angstrom
- **wave** (*array-like*) – Desired wavelengths at which to compute the model atmosphere flux.

- **rv** (*float, optional*) – The reddening law parameter, R_V, the ration of the V band extinction A_V to the reddening between the B and V bands, $E(B-V)$. Default is `3.1`, appropriate for stellar SEDs in the Milky Way.
- **log** (*bool, optional*) – Return the log10 flux, rather than the flux (what's actually interpolated)
- **pixel_scale** (*float, optional*) – Jacobian of the transformation between wavelength in Angstrom and pixels. In principle, this should be a vector, but virtually all spectral reduction packages resample the spectrum onto a uniform wavelength scale that is close to the native pixel scale of the spectrograph. Default is `1.`

Returns

- **flux** (*array-like*) – Interpolated model flux at `teff`, `logg` with reddening parametrized by `av`, `rv` and broadened by a Gaussian kernel defined by `fwhm` at wavelengths `wave`
- **mod** (`numpy.recarray` with `dtype=[('wave', '<f8'), ('flux', '<f8')]`) – Full model SED at `teff`, `logg` with reddening parametrized by `av`, `rv`

Notes

`fwhm` and `pixel_scale` must be > 0

classmethod _get_indices_in_range(*wave*, *WA*, *WB*, *W0=None*)

Returns indices of wavelength between blue and red wavelength limits and the central wavelength

Parameters

- **wave** (*array-like*) – Wavelengths array from which to extract indices
- **WA** (*float*) – blue limit of wavelengths to extract
- **WB** (*float*) – red limit of wavelenghts to extract
- **W0** (*float or None, optional*) – `None` or a central wavelength of range `[WA, WB]` to return. If `None`, the central wavelength is computed, else the input is simply returned.

Returns

- **W0** (*float*) – central wavelength of range `[WA, WB]`
- **ZE** (*array-like*) – indices of `wave` in range `[WA, WB]`

_get_line_indices(*wave*, *line*)

Returns the central wavelength and indices of wavelength corresponding to a hydrogen Balmer line

Parameters

- **wave** (*array-like*) – Wavelengths array from which to extract indices
- **line** (`{'alpha', 'beta', 'gamma', 'delta', 'zeta', 'eta'}`) – Name of hydrogen Balmer line to extract. Properties are pre-defined in *WDmodel.WDmodel.WDmodel._lines*

Returns

- **W0** (*float*) – central wavelength of line
- **ZE** (*array-like*) – indices of *wave* of line

Notes

No checking of input - will throw `KeyError` if line is not accepted value

_get_model(*teff*, *logg*, *wave=None*, *log=False*)
Returns the model flux given `teff` and `logg` at wavelengths `wave`

Simple 3-D interpolation of model grid. Computes unreddened, unnormalized, unconvolved, interpolated model flux. Uses `scipy.interpolate.RegularGridInterpolator` to generate the interpolated model. This output has been tested against *WDmodel.WDmodel.WDmodel._get_model_nosp()*.

Parameters

- **teff** (*float*) – Desired model white dwarf atmosphere temperature (in Kelvin)
- **logg** (*float*) – Desired model white dwarf atmosphere surface gravity (in dex)
- **wave** (*array-like, optional*) – Desired wavelengths at which to compute the model atmosphere flux. If not supplied, the full model wavelength grid is returned.
- **log** (*bool, optional*) – Return the log10 flux rather than the flux.

Returns **flux** – Interpolated model flux at `teff`, `logg` and wavelengths `wave`.

Return type array-like

Notes

Inputs `teff`, `logg` and `wave` must be within the bounds of the grid. See *WDmodel.WDmodel.WDmodel._wave*, *WDmodel.WDmodel.WDmodel._ggrid*, *WDmodel.WDmodel.WDmodel._tgrid*, for grid locations and limits.

_get_model_nosp(*teff*, *logg*, *wave=None*, *log=False*)
Returns the model flux given `teff` and `logg` at wavelengths `wave`

Simple 3-D interpolation of model grid. Computes unreddened, unnormalized, unconvolved, interpolated model flux. Not used, but serves as check of output of interpolation of `scipy.interpolate.RegularGridInterpolator` output.

Parameters

- **teff** (*float*) – Desired model white dwarf atmosphere temperature (in Kelvin)
- **logg** (*float*) – Desired model white dwarf atmosphere surface gravity (in dex)
- **wave** (*array-like, optional*) – Desired wavelengths at which to compute the model atmosphere flux. If not supplied, the full model wavelength grid is returned.
- **log** (*bool, optional*) – Return the log10 flux, rather than the flux.

Returns **flux** – Interpolated model flux at `teff`, `logg` and wavelengths `wave`

Return type array-like

Notes

Inputs `teff`, `logg` and `wave` must be within the bounds of the grid. See *WDmodel.WDmodel.WDmodel._wave*, *WDmodel.WDmodel.WDmodel._ggrid*, *WDmodel.WDmodel.WDmodel._tgrid*, for grid locations and limits.

This restriction is not imposed here for performance reasons, but is implicitly set by routines that call this method. The user is expected to verify this condition if this method is used outside the context of the *WDmodel* package. Caveat emptor.

_get_obs_model (*teff*, *logg*, *av*, *fwhm*, *wave*, *rv=3.1*, *log=False*, *pixel_scale=1.0*)

Returns the observed model flux given `teff`, `logg`, `av`, `rv`, `fwhm` (for Gaussian instrumental broadening) and wavelengths `wave`

Uses *WDmodel.WDmodel.WDmodel._get_model()* to get the unreddened model, and reddens it with *WDmodel.WDmodel.WDmodel.reddening()* and convolves it with a Gaussian kernel using `scipy.ndimage.filters.gaussian_filter1d()`

Parameters

- **teff** (*float*) – Desired model white dwarf atmosphere temperature (in Kelvin)
- **logg** (*float*) – Desired model white dwarf atmosphere surface gravity (in dex)
- **av** (*float*) – Extinction in the V band, A_V
- **fwhm** (*float*) – Instrumental FWHM in Angstrom
- **wave** (*array-like*) – Desired wavelengths at which to compute the model atmosphere flux.
- **rv** (*float, optional*) – The reddening law parameter, R_V, the ration of the V band extinction A_V to the reddening between the B and V bands, $E(B-V)$. Default is 3.1, appropriate for stellar SEDs in the Milky Way.
- **log** (*bool, optional*) – Return the log10 flux, rather than the flux (what's actually interpolated)
- **pixel_scale** (*float, optional*) – Jacobian of the transformation between wavelength in Angstrom and pixels. In principle, this should be a vector, but virtually all spectral reduction packages resample the spectrum onto a uniform wavelength scale that is close to the native pixel scale of the spectrograph. Default is `1.`

Returns **flux** – Interpolated model flux at `teff`, `logg` with reddening parametrized by `av`, `rv` and broadened by a Gaussian kernel defined by `fwhm` at wavelengths `wave`

Return type array-like

Notes

`fwhm` and `pixel_scale` must be > 0

_get_red_model (*teff*, *logg*, *av*, *wave*, *rv=3.1*, *log=False*)

Returns the reddened model flux given `teff`, `logg`, `av`, `rv` at wavelengths `wave`

Uses *WDmodel.WDmodel.WDmodel._get_model()* to get the unreddened model, and reddens it with *WDmodel.WDmodel.WDmodel.reddening()*

Parameters

- **teff** (*float*) – Desired model white dwarf atmosphere temperature (in Kelvin)
- **logg** (*float*) – Desired model white dwarf atmosphere surface gravity (in dex)
- **av** (*float*) – Extinction in the V band, A_V
- **wave** (*array-like*) – Desired wavelengths at which to compute the model atmosphere flux.

- **rv** (*float, optional*) – The reddening law parameter, R_V, the ration of the V band extinction A_V to the reddening between the B and V bands, $E(B-V)$. Default is 3.1, appropriate for stellar SEDs in the Milky Way.
- **log** (*bool, optional*) – Return the log10 flux, rather than the flux (what's actually interpolated)

Returns **flux** – Interpolated model flux at `teff`, `logg` with reddening parametrized by `av`, `rv` at wavelengths `wave`

Return type array-like

classmethod _wave_test(*wave*)

Raises an error if wavelengths are not valid

Parameters **wave** (*array-like*) – Array of wavelengths to test for validity

Raises **ValueError** – If wavelength array is empty, has negative values, or is not monotonic

extinction(*wave*, *av*, *rv=3.1*)

Return the extinction for `av`, `rv` at wavelengths `wave`

Uses the extinction function corresponding to the `rvmodel` parametrization set as *WDmodel.WDmodel.WDmodel._law* to calculate the extinction as a function of wavelength (in Angstroms), A_λ.

Parameters

- **wave** (*array-like*) – Array of wavelengths in Angstrom at which to compute extinction, sorted in ascending order
- **av** (*float*) – Extinction in the V band, A_V
- **rv** (*float, optional*) – The reddening law parameter, R_V, the ration of the V band extinction A_V to the reddening between the B and V bands, $E(B-V)$. Default is `3.1`, appropriate for stellar SEDs in the Milky Way.

Returns **out** – Extinction at wavelengths `wave` for `av` and `rv`

Return type array-like

Notes

`av` should be >= 0.

extract_spectral_line(*w*, *f*, *line*, *df=None*)

Extracts slices of multiple arrays corresponding to a hydrogen Balmer line

Convenience function to extract elements of wavelength `w`, flux `f` and optionally flux uncertainty `df` for a hydrogen Balmer line. Wraps *WDmodel.WDmodel.WDmodel._extract_spectral_line()* adding checking of inputs.

Parameters

- **w** (*array-like*) – Wavelength array from which to extract elements corresponding to hydrogen Balmer `line`
- **f** (*array-like*) – Flux array from which to extract elements corresponding to hydrogen Balmer `line`
- **line** ({'alpha', 'beta', 'gamma', 'delta', 'zeta', 'eta'}) – Name of hydrogen Balmer line to extract. Properties are pre-defined in *WDmodel.WDmodel.WDmodel._lines*

- **df** (*None or array-like, optional*) – If array-like, extracted elements of this array are also returned

Returns

- **w** (*array-like*) – elements of input wavelength array for hydrogen Balmer feature `line`
- **f** (*array-like*) – elements of input flux array for hydrogen Balmer feature `line`
- **[df]** (*array-like*) – elements of input flux uncertainty array for hydrogen Balmer feature `line` if optional input `df` is supplied

Raises **ValueError** – If `line` is not one of the first six of the Balmer series or If wavelengths are invalid of If there's a difference in shape of any of the arrays

get_model (*teff*, *logg*, *wave=None*, *log=False*, *strict=True*)

Returns the model flux given `teff` and `logg` at wavelengths `wave`

Wraps *WDmodel.WDmodel.WDmodel._get_model()* adding checking of inputs.

Parameters

- **teff** (*float*) – Desired model white dwarf atmosphere temperature (in Kelvin)
- **logg** (*float*) – Desired model white dwarf atmosphere surface gravity (in dex)
- **wave** (*array-like, optional*) – Desired wavelengths at which to compute the model atmosphere flux. If not supplied, the full model wavelength grid is returned.
- **log** (*bool, optional*) – Return the log10 flux, rather than the flux
- **strict** (*bool, optional*) – If strict, `teff` and `logg` out of model grid range raise a `ValueError`, otherwise raise a `RuntimeWarning` and set `teff`, `logg` to the nearest grid value.

Returns

- **wave** (*array-like*) – Valid output wavelengths
- **flux** (*array-like*) – Interpolated model flux at `teff`, `logg` and wavelengths `wave`

Raises **ValueError** – If `teff` or `logg` are out of range of model grid and `strict` is True or if there are any invalid wavelengths, or the requested wavelengths to do not overlap with the model grid

Notes

Unlike the corresponding private methods, the public methods implement checking of the inputs and returns the wavelengths in addition to the flux. Internally, we only use the private methods as the inputs only need to be checked once, and their state is not altered anywhere after.

get_obs_model (*teff*, *logg*, *av*, *fwhm*, *rv=3.1*, *wave=None*, *log=False*, *strict=True*, *pixel_scale=1.0*)

Returns the observed model flux given `teff`, `logg`, `av`, `rv`, `fwhm` (for Gaussian instrumental broadening) and wavelengths `wave`

Uses *WDmodel.WDmodel.WDmodel.get_red_model()* to get the reddened model and convolves it with a Gaussian kernel using `scipy.ndimage.filters.gaussian_filter1d()`

Parameters

- **teff** (*float*) – Desired model white dwarf atmosphere temperature (in Kelvin)
- **logg** (*float*) – Desired model white dwarf atmosphere surface gravity (in dex)
- **av** (*float*) – Extinction in the V band, A_V

- **fwhm** (*float*) – Instrumental FWHM in Angstrom
- **rv** (*float, optional*) – The reddening law parameter, R_V, the ration of the V band extinction A_V to the reddening between the B and V bands, $E(B-V)$. Default is `3.1`, appropriate for stellar SEDs in the Milky Way.
- **wave** (*array-like, optional*) – Desired wavelengths at which to compute the model atmosphere flux. If not supplied, the full model wavelength grid is returned.
- **log** (*bool, optional*) – Return the log10 flux, rather than the flux (what's actually interpolated)
- **strict** (*bool, optional*) – If strict, `teff` and `logg` out of model grid range raise a `ValueError`, otherwise raise a `RuntimeWarning` and set `teff`, `logg` to the nearest grid value.
- **pixel_scale** (*float, optional*) – Jacobian of the transformation between wavelength in Angstrom and pixels. In principle, this should be a vector, but virtually all spectral reduction packages resample the spectrum onto a uniform wavelength scale that is close to the native pixel scale of the spectrograph. Default is `1.`

Returns

- **wave** (*array-like*) – Valid output wavelengths
- **flux** (*array-like*) – Interpolated model flux at `teff`, `logg` with reddening parametrized by `av`, `rv` broadened by a Gaussian kernel defined by `fwhm` at wavelengths `wave`

Notes

`fwhm` and `pixel_scale` must be `> 0`

get_red_model (*teff*, *logg*, *av*, *rv=3.1*, *wave=None*, *log=False*, *strict=True*)

Returns the reddened model flux given `teff`, `logg`, `av`, `rv` at wavelengths `wave`

Uses *WDmodel.WDmodel.WDmodel.get_model()* to get the unreddened model, and reddens it with *WDmodel.WDmodel.WDmodel.reddening()*

Parameters

- **teff** (*float*) – Desired model white dwarf atmosphere temperature (in Kelvin)
- **logg** (*float*) – Desired model white dwarf atmosphere surface gravity (in dex)
- **av** (*float*) – Extinction in the V band, A_V
- **rv** (*float, optional*) – The reddening law parameter, R_V, the ration of the V band extinction A_V to the reddening between the B and V bands, $E(B-V)$. Default is `3.1`, appropriate for stellar SEDs in the Milky Way.
- **wave** (*array-like, optional*) – Desired wavelengths at which to compute the model atmosphere flux. If not supplied, the full model wavelength grid is returned.
- **log** (*bool, optional*) – Return the log10 flux, rather than the flux (what's actually interpolated)
- **strict** (*bool, optional*) – If strict, `teff` and `logg` out of model grid range raise a `ValueError`, otherwise raise a `RuntimeWarning` and set `teff`, `logg` to the nearest grid value.

Returns

- **wave** (*array-like*) – Valid output wavelengths

- **flux** (*array-like*) – Interpolated model flux at `teff`, `logg` with reddening parametrized by `av`, `rv` at wavelengths `wave`

Raises **ValueError** – If `av < 0` or `rv` not in `[1.7, 5.1]`

reddening(*wave*, *flux*, *av*, *rv=3.1*)

Redden a 1-D spectrum with extinction

Uses the extinction function corresponding to the `rvmodel` parametrization set in *WDmodel.WDmodel.WDmodel._WDmodel__init__rvmodel()* to calculate the extinction as a function of wavelength (in Angstroms), A_λ.

Parameters

- **wave** (*array-like*) – Array of wavelengths in Angstrom at which to compute extinction, sorted in ascending order
- **flux** (*array-like*) – Array of fluxes at `wave` at which to apply extinction
- **av** (*float*) – Extinction in the V band, A_V
- **rv** (*float, optional*) – The reddening law parameter, R_V, the ration of the V band extinction A_V to the reddening between the B and V bands, $E(B-V)$. Default is `3.1`, appropriate for stellar SEDs in the Milky Way.

Returns **out** – The reddened spectrum

Return type array-like

Notes

`av` and `flux` should be >= 0.

WDmodel.covariance module

Parametrizes the noise of the spectrum fit using a Gaussian process.

class WDmodel.covariance.**WDmodel_CovModel**(*errscale*, *covtype='Matern32'*, *coveps=1e-12*)

Bases: object

Parametrizes the noise of the spectrum fit using a Gaussian process.

This class models the covariance of the spectrum fit using a stationary Gaussian process conditioned on the spectrum flux residuals and spectrum flux uncertainties. The class allows the kernel of the Gaussian process to be set in a single location. A few different stationary kernels are supported. These choices are defined in `celerite.terms`.

Parameters

- **errscale** (*float*) – Chracteristic scale of the spectrum flux uncertainties. The kernel amplitude hyperparameters are reported as fractions of this number. If the spectrum flux is rescaled, this must be set appropriately to get the correct uncertainties. The *WDmodel* package uses the median of the spectrum flux uncertainty internally.
- **covtype** (`{'Matern32', 'SHO', 'Exp', 'White'}`) – The model to use to parametrize the covariance. Choices are defined in `celerite.terms` All choices except `'White'` parametrize the covariance using a stationary kernel with a characteristic amplitude `fsig` and scale `tau` + a white noise component with amplitude `fw`. Only the white noise component is used to condition the Gaussian process if `covtype` is `'White'`. If not specified or unknown, `'Matern32'` is used and a `RuntimeWarning` is raised.

- **coveps** (*float*) – If `covtype` is `'Matern32'` a `celerite.terms.Matern32Term` is used to approximate a Matern32 kernel with precision *coveps*. The default is `1e-12`. Ignored if any other `covtype` is specified.

_errscale
The input `errscale`

Type float

_covtype
The input `covtype`

Type str

_coveps
The input `coveps`

Type float

_ndim
The dimensionality of kernel used to parametrize the covariance

Type int

_k1
The non-trivial stationary component of the kernel

Type None or a term instance from `celerite.terms`

_k2
The white noise component of the kernel

Type `celerite.terms.JitterTerm`

_logQ
`1/sqrt(2)` - only set if `covtype` is `'SHO'`

Type float, conditional

Returns

Return type A *WDmodel.covariance.WDmodel_CovModel* instance

Notes

Virtually none of the attributes should be used directly since it is trivially possible to break the model by redefining them. Access to them is best through the functions connected to the models.

__init__(*errscale*, *covtype='Matern32'*, *coveps=1e-12*)
Initialize self. See help(type(self)) for accurate signature.

getgp(*wave*, *flux_err*, *fsig*, *tau*, *fw*)
Return the `celerite.GP` instance

Precomputes the covariance matrix of the Gaussian process specified by the functional form of the stationary kernel and the current values of the hyperparameters. Wraps `celerite.GP`.

Parameters

- **wave** (*array-like, optional*) – Wavelengths at which to condition the Gaussian process
- **flux_err** (*array-like*) – Flux uncertainty array on which to condition the Gaussian process

- **fsig** (*float*) – The fractional amplitude of the non-trivial stationary kernel. The true amplitude is scaled by *WDmodel.covariance.WDmodel_CovModel._errscale*
- **tau** (*float*) – The characteristic length scale of the non-trivial stationary kernel.
- **fw** (*float*) – The fractional amplitude of the white noise component of the kernel. The true amplitude is scaled by *WDmodel.covariance.WDmodel_CovModel._errscale*

Returns **gp** – The Gaussian process with covariance matrix precomputed at the location of the data

Return type `celerite.GP` instance

Notes

`fsig`, `tau` and `fw` all must be > 0. This constraint is not checked here, but is instead imposed by the samplers/optimizers used in the *WDmodel.fit* methods, and by bounds used to construct the `WDmodel.likelihood.WDmodel_Likelihood` instance using the *WDmodel.likelihood.setup_likelihood()* method.

lnlikelihood(*wave*, *res*, *flux_err*, *fsig*, *tau*, *fw*)

Return the log likelihood of the Gaussian process

Conditions the Gaussian process specified by the functional form of the stationary kernel and the current values of the hyperparameters on the data, and computes the log likelihood. Wraps `celerite.GP.log_likelihood()`.

Parameters

- **wave** (*array-like, optional*) – Wavelengths at which to condition the Gaussian process
- **res** (*array-like*) – Flux residual array on which to condition the Gaussian process. The kernel parametrization assumes that the mean model has been subtracted off.
- **flux_err** (*array-like*) – Flux uncertaintyarray on which to condition the Gaussian process
- **fsig** (*float*) – The fractional amplitude of the non-trivial stationary kernel. The true amplitude is scaled by *WDmodel.covariance.WDmodel_CovModel._errscale*
- **tau** (*float*) – The characteristic length scale of the non-trivial stationary kernel.
- **fw** (*float*) – The fractional amplitude of the white noise component of the kernel. The true amplitude is scaled by *WDmodel.covariance.WDmodel_CovModel._errscale*

Returns **lnlike** – The log likelihood of the Gaussian process conditioned on the data.

Return type float

See also:

getgp()

predict(*wave*, *res*, *flux_err*, *fsig*, *tau*, *fw*, *mean_only=False*)

Return the prediction for the Gaussian process

Conditions the Gaussian process specified by the parametrized with the functional form of the stationary kernel and the current values of the hyperparameters on the data, and computes returns the prediction at the same location as the data. Wraps `celerite.GP.predict()`.

Parameters

- **wave** (*array-like, optional*) – Wavelengths at which to condition the Gaussian process
- **res** (*array-like*) – Flux residual array on which to condition the Gaussian process. The kernel parametrization assumes that the mean model has been subtracted off.
- **flux_err** (*array-like*) – Flux uncertaintyarray on which to condition the Gaussian process
- **fsig** (*float*) – The fractional amplitude of the non-trivial stationary kernel. The true amplitude is scaled by *WDmodel.covariance.WDmodel_CovModel._errscale*
- **tau** (*float*) – The characteristic length scale of the non-trivial stationary kernel.
- **fw** (*float*) – The fractional amplitude of the white noise component of the kernel. The true amplitude is scaled by *WDmodel.covariance.WDmodel_CovModel._errscale*
- **mean_only** (*bool, optional*) – Return only the predicted mean, not the covariance matrix

Returns

- **wres** (*array-like*) – The prediction of the Gaussian process conditioned on the data at the same location i.e. the model.
- **cov** (*array-like, optional*) – The computed covariance matrix of the Gaussian process using the parametrized stationary kernel evaluated at the locations of the data.

See also:

getgp()

WDmodel.fit module

Core data processing and fitting/sampling routines

WDmodel.fit.**blotch_spectrum**(*spec*, *linedata*)

Automagically remove cosmic rays and gaps from spectrum

Parameters

- **spec** (numpy.recarray) – The spectrum with `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`
- **linedata** (numpy.recarray) – The observations of the spectrum corresponding to the hydrogen Balmer lines. Must have `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8'), ('line_mask', 'i4'), ('line_ind','i4')]` Produced by *orig_cut_lines()*

Returns spec – The blotched spectrum with `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`

Return type numpy.recarray

Notes

Some spectra have nasty cosmic rays or gaps in the data. This routine does a reasonable job blotching these by Wiener filtering the spectrum, marking features that differ significantly from the local variance in the region, and replace them with the filtered values. The hydrogen Balmer lines are preserved, so if your gap/cosmic ray lands on a line it will not be filtered. Additionally, filtering has edge effects, and these data are preserved as well. If you do blotch the spectrum, it is highly recommended that you use the bluelimit and redlimit options to trim the ends of the spectrum. Note that the spectrum will be rejected if it has flux or flux errors that are not finite or below zero. This is often the case with cosmic rays and gaps, so you will likely have to do some manual removal of these points.

YOU SHOULD PROBABLY PRE-PROCESS YOUR DATA YOURSELF BEFORE FITTING IT AND NOT BE LAZY! THIS ROUTINE ONLY EXISTS TO FIT QUICK LOOK SPECTRUM AT THE TELESCOPE, BEFORE FINAL REDUCTIONS!

`WDmodel.fit.`**`fit_model`**(*spec*, *phot*, *model*, *covmodel*, *pbs*, *params*, *objname*, *outdir*, *specfile*, *phot_dispersion=0.0*, *samptype='ensemble'*, *ascale=2.0*, *ntemps=1*, *nwalkers=300*, *nburnin=50*, *nprod=1000*, *everyn=1*, *thin=1*, *pool=None*, *resume=False*, *redo=False*)

Core routine that models the spectrum using the white dwarf model and a Gaussian process with a stationary kernel to account for any flux miscalibration, sampling the posterior using a MCMC.

Parameters

- **spec** (numpy.recarray) – The spectrum with `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`
- **phot** (numpy.recarray) – The photometry of `objname` with `dtype=[('pb', 'str'), ('mag', '<f8'), ('mag_err', '<f8')]`
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator
- **pbs** (*dict*) – Passband dictionary containing the passbands corresponding to phot.pb‘ and generated by *WDmodel.passband.get_pbmodel()*.
- **params** (*dict*) – A parameter dict such as that produced by *WDmodel.io.read_params()*
- **objname** (*str*) – object name - used to save output with correct name
- **outdir** (*str*) – controls where the chain file s written
- **specfile** (*str*) – Used in the title, and to set the name of the `outfile`
- **phot_dispersion** (*float, optional*) – Excess photometric dispersion to add in quadrature with the photometric uncertainties `phot.mag_err`. Use if the errors are grossly underestimated. Default is `0.`
- **samptype** (`{'ensemble', 'pt', 'gibbs'}`) – Which sampler to use. The default is `ensemble`.
- **ascale** (*float*) – The proposal scale for the sampler. Default is `2.`
- **ntemps** (*int*) – The number of temperatures to run walkers at. Only used if `samptype` is in `{'pt','gibbs'}` and set to `1.` for `ensemble`. See a short summary review for details. Default is `1.`
- **nwalkers** (*int*) – The number of Goodman and Weare walkers. Default is `300`.
- **nburnin** (*int*) – The number of steps to discard as burn-in for the Markov-Chain. Default is `500`.

- **nprod** (*int*) – The number of production steps in the Markov-Chain. Default is `1000`.
- **everyn** (*int, optional*) – If the posterior function is evaluated using only every nth observation from the data, this should be specified. Default is `1`.
- **thin** (*int*) – Only save every `thin` steps to the output Markov Chain. Useful, if brute force way of reducing correlation between samples.
- **pool** (*None or :py:class`emcee.utils.MPIPool`*) – If running with MPI, the pool object is used to distribute the computations among the child process
- **resume** (*bool*) – If `True`, restores state and resumes the chain for another `nprod` iterations.
- **redo** (*bool*) – If `True`, and a chain file and state file exist, simply clobbers them.

Returns

- **free_param_names** (*list*) – names of parameters that were fit for. Names correspond to keys in `params` and the order of parameters in `samples`.
- **samples** (*array-like*) – The flattened Markov Chain with the parameter positions. Shape is `(ntemps*nwalkers*nprod, nparam)`
- **samples_lnprob** (*array-like*) – The flattened log of the posterior corresponding to the positions in `samples`. Shape is `(ntemps*nwalkers*nprod, 1)`
- **everyn** (*int*) – Specifies sampling of the data used to compute the posterior. Provided in case we are using `resume` to continue the chain, and this value must be restored from the state file, rather than being supplied as a user input.
- **shape** (*tuple*) – Specifies the shape of the un-flattened chain. `(ntemps, nwalkers, nprod, nparam)` Provided in case we are using `resume` to continue the chain, and this value must be restored from the state file, rather than being supplied as a user input.

Raises **RuntimeError** – If `resume` is set without the chain having been run in the first place.

Notes

Uses an Ensemble MCMC (implemented by emcee) to generate samples from the posterior. Does a short burn-in around the initial guess model parameters - either `minuit` or user supplied values/defaults. Model parameters may be frozen/fixed. Parameters can have bounds limiting their range. Then runs a full production change. Chain state is saved after every 100 production steps, and may be continued after the first 100 steps if interrupted or found to be too short. Progress is indicated visually with a progress bar that is written to STDOUT.

See also:

WDmodel.likelihood, *WDmodel.covariance*

`WDmodel.fit.`**fix_pos**(*pos*, *free_param_names*, *params*)

Ensures that the initial positions of the `emcee` walkers are out of bounds

Parameters

- **pos** (*array-like*) – starting positions of all the walkers, such as that produced by `utils.sample_ball`
- **free_param_names** (*iterable*) – names of parameters that are free to float. Names must correspond to keys in `params`.
- **params** (*dict*) – A parameter dict such as that produced by *WDmodel.io.read_params()*

Returns **pos** – starting positions of all the walkers, fixed to guarantee that they are within `bounds` defined in `params`

Return type array-like

Notes

`emcee.utils.sample_ball()` creates random walkers that may be initialized out of bounds. These walkers get stuck as there is no step they can take that will make the change in loglikelihood finite. This makes the chain appear strongly correlated since all the samples of one walker are at a fixed location. This resolves the issue by assuming that the parameter `value` was within `bounds` to begin with. This routine does not do any checking of types, values or bounds. This check is done by *`WDmodel.io.get_params_from_argparse()`* before the fit. If you setup the fit using an external code, you should check these values.

See also:

`emcee.utils.sample_ball()`, *`WDmodel.io.get_params_from_argparse()`*

`WDmodel.fit.`**`get_fit_params_from_samples`**(*param_names*, *samples*, *samples_lnprob*, *params*, *ntemps=1*, *nwalkers=300*, *nprod=1000*, *discard=5*)

Get the marginalized parameters from the sample chain

Parameters

- **`param_names`** (*`list`*) – names of parameters that were fit for. Names correspond to keys in `params` and the order of parameters in `samples`.
- **`samples`** (*`array-like`*) – The flattened Markov Chain with the parameter positions. Shape is `(ntemps*nwalkers*nprod, nparam)`
- **`samples_lnprob`** (*`array-like`*) – The flattened log of the posterior corresponding to the positions in `samples`. Shape is `(ntemps*nwalkers*nprod, 1)`
- **`params`** (*`dict`*) – A parameter dict such as that produced by *`WDmodel.io.read_params()`*
- **`ntemps`** (*`int`*) – The number of temperatures chains were run at. Default is `1`.
- **`nwalkers`** (*`int`*) – The number of Goodman and Weare walkers used in the fit. Default is `300`.
- **`nprod`** (*`int`*) – The number of production steps in the Markov-Chain. Default is `1000`.
- **`discard`** (*`int`*) – percentage of nprod steps from the start of the chain to discard in analyzing samples

Returns

- **mcmc_params** (*dict*) – The output parameter dictionary with updated parameter estimates, errors and a scale. `params`.
- **out_samples** (*array-like*) – The flattened Markov Chain with the parameter positions with the first `%discard` tossed.
- **out_ samples_lnprob** (*array-like*) – The flattened log of the posterior corresponding to the positions in `samples` with the first `%discard` samples tossed.

See also:

`fit_model()`

WDmodel.fit.**hyper_param_guess**(*spec*, *phot*, *model*, *pbs*, *params*)

Makes a guess for the parameter mu after the initial fit by *quick_fit_spec_model()*

Parameters

- **spec** (numpy.recarray) – The spectrum with dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]
- **phot** (numpy.recarray) – The photometry of objname with dtype=[('pb', 'str'), ('mag', '<f8'), ('mag_err', '<f8')]
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator
- **pbs** (*dict*) – Passband dictionary containing the passbands corresponding to phot.pb‘ and generated by *WDmodel.passband.get_pbmodel()*.
- **params** (*dict*) – A parameter dict such as that produced by *WDmodel.io.read_params()*

Returns **out_params** – The output parameter dictionary with an initial guess for mu

Return type dict

Notes

Uses the initial guess of parameters from the spectrum fit by *quick_fit_spec_model()* to construct an initial guess of the SED, and computes mu (which looks like a distance modulus, but also includes a normalization for the radius of the DA white dwarf, and it's radius) as the median difference between the observed and synthetic photometry.

WDmodel.fit.**orig_cut_lines**(*spec*, *model*)

Cut out the hydrogen Balmer spectral lines defined in *WDmodel.WDmodel.WDmodel* from the spectrum.

The masking of Balmer lines is basic, and not very effective at high surface gravity or low temperature, or in the presence of non hydrogen lines. It's used to get a roughly masked set of data suitable for continuum detection, and is effective in the context of our ground-based spectroscopic followup campaign for HST GO 12967 and 13711 programs.

Parameters

- **spec** (numpy.recarray) – The spectrum with dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator

Returns

- **linedata** (numpy.recarray) – The observations of the spectrum corresponding to the hydrogen Balmer lines. Has dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8'), ('line_mask', 'i4'), (line_ind', 'i4')]
- **continuumdata** (numpy.recarray) – The continuum data. Has dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]

Notes

Does a coarse cut to remove hydrogen absorption lines from DA white dwarf spectra The line centroids, and widths are fixed and defined with the model grid This is insufficient, and particularly at high surface gravity and

low temperatures the lines are blended. This routine is intended to provide a rough starting point for the process of continuum determination.

WDmodel.fit.**polyfit_continuum**(*continuumdata*, *wave*)

Fit a polynomial to the DA white dwarf continuum to normalize it - purely for visualization purposes

Parameters

- **continuumdata** (numpy.recarray) – The continuum data. Must have dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')] Produced by running the spectrum through *WDmodel.fit.orig_cut_lines()* and extracting the pre-defined lines in the *WDmodel.WDmodel.WDmodel* instance.
- **wave** (*array-like*) – The full spectrum wavelength array on which to interpolate the continuum model

Returns cont_model – The continuum model Must have dtype=[('wave', '<f8'), ('flux', '<f8')]

Return type numpy.recarray

Notes

Roughly follows the algorithm described by the SDSS SSPP for a global continuum fit. Fits a red side and blue side at 5500 A separately to get a smooth polynomial representation. The red side uses a degree 5 polynomial and the blue side uses a degree 9 polynomial. Then "splices" them together - I don't actually know how SDSS does this, but we simply assert the two bits are the same function - and fits the full continuum to a degree 9 polynomial.

WDmodel.fit.**pre_process_spectrum**(*spec*, *bluelimit*, *redlimit*, *model*, *params*, *lamshift=0.0*, *vel=0.0*, *rebin=1*, *blotch=False*, *rescale=False*)

Pre-process the spectrum before fitting

Parameters

- **spec** (numpy.recarray) – The spectrum with dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]
- **bluelimit** (*None or float*) – Trim wavelengths bluer than this limit. Uses the bluest wavelength of spectrum if None
- **redlimit** (*None or float*) – Trim wavelengths redder than this limit. Uses the reddest wavelength of spectrum if None
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator
- **params** (*dict*) – A parameter dict such as that produced by *WDmodel.io.read_params()* Will be modified to adjust the spectrum normalization parameters dl limits if rescale is set
- **lamshift** (*float, optional*) – Apply a flat wavelength shift to the spectrum. Useful if the target was not properly centered in the slit, and the shift is not correlated with wavelength. Default is 0.
- **vel** (*float, optional*) – Apply a velocity shift to the spectrum. Default is 0.
- **rebin** (*int, optional*) – Integer factor by which to rebin the spectrum. Default is 1 (no rebinning).

- **blotch** (*bool, optional*) – Attempt to remove cosmic rays and gaps from spectrum. Only to be used for quick look analysis at the telescope.
- **rescale** (*bool, optional*) – Rescale the spectrum to make the median noise ~1. Has no effect on fitted parameters except spectrum flux normalization parameter dl but makes residual plots, histograms more easily interpretable as they can be compared to an N(0, 1) distribution.

Returns **spec** – The spectrum with dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]

Return type numpy.recarray

See also:

orig_cut_lines(), *blotch_spectrum()*, *rebin_spec_by_int_factor()*, *polyfit_continuum()*

WDmodel.fit.**quick_fit_spec_model**(*spec*, *model*, *params*)

Does a quick fit of the spectrum to get an initial guess of the fit parameters

Uses iminuit to do a rough diagonal fit - i.e. ignores covariance. For simplicity, also fixed FWHM and Rv (even when set to be fit). Therefore, only teff, logg, av, dl are fit for (at most). This isn't robust, but it's good enough for an initial guess.

Parameters

- **spec** (numpy.recarray) – The spectrum with dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator
- **params** (*dict*) – A parameter dict such as that produced by *WDmodel.io.read_params()*

Returns **migrad_params** – The output parameter dictionary with updated initial guesses stored in the value key. Same format as params.

Return type dict

Raises

- **RuntimeError** – If all of teff, logg, av, dl are set as fixed - there's nothing to fit.
- **RuntimeWarning** – If minuit.Minuit.migrad() or minuit.Minuit.hesse() indicate that the fit is unreliable

Notes

None of the starting values for the parameters maybe None EXCEPT c. This refines the starting guesses, and determines a reasonable value for c

WDmodel.fit.**rebin_spec_by_int_factor**(*spec*, *f=1*)

Rebins a spectrum by an integer factor f

Parameters

- **spec** (numpy.recarray) – The spectrum with dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]

- **f** (*int, optional*) – an integer factor to rebin the spectrum by. Default is `1` (no rebinning)

Returns **rspec** – The rebinned spectrum with `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`

Return type `numpy.recarray`

Notes

If the spectrum is not divisible by f, the edges are trimmed by discarding the remainder measurements from both ends. If the remainder itself is odd, the extra measurement is discarded from the blue side.

WDmodel.io module

I/O methods. All the submodules of the WDmodel package use this module for almost all I/O operations.

`WDmodel.io._read_ascii`(*filename*, ***kwargs*)

Read ASCII files

Read space separated ASCII file, with column names provided on first line (leading `#` optional). `kwargs` are passed along to `numpy.genfromtxt()`. Forces any string column data to be encoded in ASCII, rather than Unicode.

Parameters

- **filename** (*str*) – Filename of the ASCII file. Column names must be provided on the first line.
- **kwargs** (*dict*) – Extra options, passed directly to `numpy.genfromtxt()`

Returns **out** – Record array with the data. Field names correspond to column names in the file.

Return type `numpy.recarray`

See also:

`numpy.genfromtxt()`

`WDmodel.io.copy_params`(*params*)

Returns a deep copy of a dictionary. Necessary to ensure that dictionaries that nest dictionaries are properly updated.

Parameters **params** (*dict or Object*) – Any python object for which a deepcopy needs to be created. Typically a parameter dictionary such as that from *WDmodel.io.read_params()*

Returns **params** – A deepcopy of the object

Return type Object

Notes

Simple wrapper around `copy.deepcopy()`

`WDmodel.io.get_filepath`(*infile*)

Returns the full path to a file. If the path is relative, it is converted to absolute. If this file does not exist, it is treated as a file within the *WDmodel* package. If that file does not exist, an error is raised.

Parameters **infile** (*str*) – The name of the file to set the full path for

Returns **pkgfile** – The path to the file

Return type str

Raises **IOError** – If the `infile` could not be found at location or inside the *WDmodel* package.

`WDmodel.io.`**get_options**(*args*, *comm*)
Get command line options for the *WDmodel* fitter package

Parameters

- **args** (*array-like*) – list of the input command line arguments, typically from `sys.argv`
- **comm** (None or `mpi4py.mpi.MPI` instance) – Used to communicate options to all child processes if running with mpi

Returns

- **args** (*Namespace*) – Parsed command line options
- **pool** (None or :py:class‘emcee.utils.MPIPool‘) – If running with MPI, the pool object is used to distribute the computations among the child process

Raises **ValueError** – If any input value is invalid

`WDmodel.io.`**get_outfile**(*outdir*, *specfile*, *ext*, *check=False*, *redo=False*, *resume=False*)
Formats the output directory, spectrum filename, and an extension into an output filename.

Parameters

- **outdir** (*str*) – The output directory name for the output file
- **specfile** (*str*) – The spectrum filename
- **ext** (*str*) – The output file's extension
- **check** (*bool, optional*) – If `True`, check if the output file exists
- **redo** (*bool, optional*) – If `False` and the output file already exists, an error is raised
- **resume** (*bool, optional*) – If `False` and the output file already exists, an error is raised

Returns **outfile** – The output filename

Return type str

Raises **IOError** – If `check` is `True`, `redo` and `resume` are `False`, and `outfile` exists.

Notes

We set the output file based on the spectrum name, since we can have multiple spectra per object.

If `outdir` is configured by *set_objname_outdir_for_specfile()* for `specfile`, it'll include the object name.

See also:

set_objname_outdir_for_specfile()

`WDmodel.io.`**get_params_from_argparse**(*args*)
Converts an `argparse.Namespace` into an ordered parameter dictionary.

Parameters **args** (`argparse.Namespace`) – The parsed command-line options from *WDmodel.io.get_options()*

Returns **params** – The parameter dictionary

Return type collections.OrderedDict

Raises **RuntimeError** – If format of argparse.Namespace is invalid. or If parameter is fixed but value is None. or If parameter value is out of bounds.

Notes

Assumes that the argument parser options were names

- <param>_value : Value of the parameter (float or None)
- <param>_fix : Bool specifying if the parameter
- <param>_bounds : tuple with lower limit and upper limit

where <param> is one of WDmodel.io._PARAMETER_NAMES

See also:

WDmodel.io.read_params(), *WDmodel.io.get_options()*

WDmodel.io.**get_phot_for_obj**(*objname*, *filename*)

Gets the measured photometry for an object from a photometry lookup table.

Parameters

- **objname** (*str*) – Object name to look for photometry for
- **filename** (*str*) – The spectrum FWHM lookup table filename

Returns **phot** – The photometry of objname with dtype=[('pb', 'str'), ('mag', '<f8'), ('mag_err', '<f8')]

Return type numpy.recarray

Raises

- **RuntimeError** – If there are no matches in the photometry lookup file or if there are multiple matches for an object in the photometry lookup file
- **ValueError** – If the photometry or the photometry uncertainty values are not finite or if the photometry uncertainties are less <= 0

Notes

The lookup file must be readable by *read_phot()*

The column name with the object name objname expected to be obj

If column names for magnitudes are named <passband>, the column names for errors in magnitudes in passband must be 'd'+<passband_name>.

WDmodel.io.**get_pkgfile**(*infile*)

Returns the full path to a file inside the *WDmodel* package

Parameters **infile** (*str*) – The name of the file to set the full package filename for

Returns **pkgfile** – The path to the file within the package.

Return type str

Raises **IOError** – If the pkgfile could not be found inside the *WDmodel* package.

Notes

This allows the package to be installed anywhere, and the code to still determine the location to a file included with the package, such as the model grid file.

WDmodel.io.**get_spectrum_resolution**(*specfile*, *spectable*, *fwhm=None*, *lamshift=None*)
Gets the measured FWHM from a spectrum lookup table.

Parameters

- **specfile** (*str*) – The spectrum filename
- **spectable** (*str*) – The spectrum FWHM lookup table filename
- **fwhm** (*None or float, optional*) – If specified, this overrides the resolution provided in the lookup table. If `None` lookups the resultion from `spectable`.
- **lamshift** (*None or float, optional*) – If specified, this overrides the wavelength shift provided in the lookup table. If `None` lookups the wavelength shift from `spectable`.

Returns

- **fwhm** (*float*) – The FWHM of the spectrum file. This is typically used as an initial guess to the *WDmodel.fit* fitter routines.
- **lamshift** (*float*) – The wavelength shift to apply additively to the spectrum. This is not a fit parameter, and is treated as an input

Raises **RuntimeWarning** – If the `spectable` cannot be read, or the `specfile` name indicates that this is a test, or if there are no or multiple matches for `specfile` in the `spectable`

Notes

If the `specfile` is not found, it returns a default resolution of `5` Angstroms, appropriate for the instruments used in our program.

Note that there there's some hackish internal name fixing since T. Matheson's table spectrum names didn't match the spectrum filenames.

WDmodel.io.**make_outdirs**(*dirname*, *redo=False*, *resume=False*)
Makes output directories

Parameters

- **dirname** (*str*) – The output directory name to create
- **redo** (*bool, optional*) – If `False` the directory will not be created if it already exists, and an error is raised
- **resume** (*bool, optional*) – If `False` the directory will not be created if it already exists, and an error is raised

Returns **None** – If the output directory `dirname` is successfully created

Return type None

Raises

- **IOError** – If the output directory exists
- **OSError** – If the output directory could not be created

Notes

If the options are parsed by *get_options()* then only one of `redo` or `resume` can be set, as the options are mutually exclusive. If `redo` is set, the fit is redone from scratch, while `resume` restarts the MCMC sampling from the last saved chain position.

`WDmodel.io.`**`read_fit_inputs`**(*input_file*)

Read the fit input HDF5 file produced by *write_fit_inputs()* and return `numpy.recarray` instances with the data.

Parameters **`input_file`** (*str*) – The HDF5 fit inputs filename

Returns

- **spec** (`numpy.recarray`) – The spectrum with `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`
- **cont_model** (`numpy.recarray`) – The continuum model. Has the same structure as `spec`.
- **linedata** (`numpy.recarray`) – The observations of the spectrum corresponding to the hydrogen Balmer lines. Has `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8'), ('line_mask', 'i4')]`
- **continuumdata** (`numpy.recarray`) – Data used to generate the continuum model. Has the same structure as `spec`.
- **phot** (None or `numpy.recarray`) – `None` or the photometry with `dtype=[('pb', 'str'), ('mag', '<f8'), ('mag_err', '<f8')]`
- **fit_config** (*dict*) –

  Dictionary with various keys needed to configure the fitter

  - `rvmodel` : `{'ccm89','od94','f99', 'custom'}` - Parametrization of the reddening law.
  - `covtype` : `{'Matern32', 'SHO', 'Exp', 'White'}`- kernel type used to parametrize the covariance
  - `coveps` : float - Matern32 kernel precision
  - `phot_dispersion` : float - Excess dispersion to add in quadrature with photometric uncertainties
  - `scale_factor` : float - Flux scale factor

Raises

- **`IOError`** – If all the fit inputs could not be restored from the HDF5 `input_file`
- **`RuntimeWarning`** – If the `input_file` includes a `phot` group, but the data cannot be loaded.

See also:

write_fit_inputs()

`WDmodel.io.`**`read_full_model`**(*input_file*)

Read the full SED model from an output file.

Parameters **`input_file`** (*str*) – Input HDF5 SED model filename

Returns **spec** – Record array with the model SED. Has `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`

Return type numpy.recarray

Raises

- **KeyError** – If any of wave, flux or flux_err is not found in the file
- **ValueError** – If any value is not finite or if flux or flux_err have any values <= 0

WDmodel.io.**read_mcmc**(*input_file*)

Read the saved HDF5 Markov chain file and return samples, sample log probabilities and chain parameters

Parameters **input_file** (*str*) – The HDF5 Markov chain filename

Returns

- **samples** (*array-like*) – The model parameter sample chain
- **samples_lnprob** (*array-like*) – The log posterior corresponding to each of the samples
- **chain_params** (*dict*) –

  The chain parameter dictionary

  - param_names : list - list of model parameter names
  - samptype : {'ensemble','pt','gibbs'} - the sampler to use
  - ntemps : int - the number of chain temperatures
  - nwalkers : int - the number of Goodman & Ware walkers
  - nprod : int - the number of production steps of the chain
  - ndim : int - the number of model parameters in the chain
  - thin : int - the chain thinning if any
  - everyn : int - the sparse of spectrum sampling step size
  - ascale : float - the proposal scale for the sampler

Raises **IOError** – If a key in the fit_config output is missing

WDmodel.io.**read_model_grid**(*grid_file=None*, *grid_name=None*)

Read the Tlusty/Hubeny grid file

Parameters

- **grid_file** (*None or str*) – Filename of the Tlusty model grid HDF5 file. If None reads the TlustyGrids.hdf5 file included with the *WDmodel* package.
- **grid_name** (*None or str*) – Name of the group name in the HDF5 file to read the grid from. If None uses default

Returns

- **grid_file** (*str*) – Filename of the HDF5 grid file
- **grid_name** (*str*) – Name of the group within the HDF5 grid file with the grid arrays
- **wave** (*array-like*) – The wavelength array of the grid with shape (nwave,)
- **ggrid** (*array-like*) – The surface gravity array of the grid with shape (ngrav,)
- **tgrid** (*array-like*) – The temperature array of the grid with shape (ntemp,)
- **flux** (*array-like*) – The DA white dwarf model atmosphere flux array of the grid. Has shape (nwave, ngrav, ntemp)

Notes

There are no easy command line options to change this deliberately because changing the grid file essentially changes the entire model, and should not be done lightly, without careful comparison of the grids to quantify differences.

See also:

WDmodel.WDmodel

WDmodel.io.**read_params**(*param_file=None*)

Read a JSON file that configures the default guesses and bounds for the parameters, as well as if they should be fixed.

Parameters **param_file** (*str, optional*) – The name of the input parameter file. If not the default file provided with the package, `WDmodel_param_defaults.json`, is read.

Returns **params** – The dictionary with the parameter `values`, `bounds`, `scale` and if `fixed`. See notes for more detailed information on dictionary format and `WDmodel_param_defaults.json` for an example file for `param_file`.

Return type dict

Notes

params is a dict the parameter names, as defined with `WDmodel.io._PARAMETER_NAMES` as keys

Each key must have a dictionary with keys:

- `value` : value
- `fixed` : a bool specifying if the parameter is fixed (`True`) or allowed to vary (`False`)
- `scale` : a scale parameter used to set the step size in this dimension
- `bounds` : An upper and lower limit on parameter values

The default bounds are set by the grids available for the DA White Dwarf atmospheres, and by reasonable plausible ranges for the other parameters. Don't muck with them unless you really have good reason to.

This routine does not do any checking of types, values or bounds. This is done by *WDmodel.io.get_params_from_argparse()* before the fit. If you setup the fit using an external code, you should check these values.

WDmodel.io.**read_pbmap**(*filename*, ***kwargs*)

Read passband obsmode mapping table - wraps *_read_ascii()*

WDmodel.io.**read_phot**(*filename*, ***kwargs*)

Read photometry - wraps *_read_ascii()*

WDmodel.io.**read_reddening**(*filename*, ***kwargs*)

Read J. Holberg's custom reddening function - wraps *_read_ascii()*

WDmodel.io.**read_spec**(*filename*, ***kwargs*)

Read a spectrum

Wraps *_read_ascii()*, adding testing of the input arrays to check if the elements are finite, and if the errors and flux are strictly positive.

Parameters

- **filename** (*str*) – Filename of the ASCII file. Must have columns `wave`, `flux`, `flux_err`

- **kwargs** (*dict*) – Extra options, passed directly to `numpy.genfromtxt()`

Returns spec – Record array with the spectrum data. Has `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`

Return type `numpy.recarray`

Raises **ValueError** – If any value is not finite or if `flux` or `flux_err` have any values `<= 0`

See also:

`numpy.genfromtxt()`, *`_read_ascii()`*

`WDmodel.io.`**read_spectable**(*filename*, ***kwargs*)
Read spectrum FWHM table - wraps *`_read_ascii()`*

`WDmodel.io.`**set_objname_outdir_for_specfile**(*specfile*, *outdir=None*, *outroot=None*, *redo=False*, *resume=False*, *nocreate=False*)
Sets the short human readable object name and output directory

Parameters

- **specfile** (*`str`*) – The spectrum filename
- **outdir** (*`None or str, optional`*) – The output directory name to create. If `None` this is set based on `specfile`
- **outroot** (*`None or str, optional`*) – The output root directory under which to store the fits. If `None` the default is `'out'`
- **redo** (*`bool, optional`*) – If `False` the directory will not be created if it already exists, and an error is raised
- **resume** (*`bool, optional`*) – If `False` the directory will not be created if it already exists, and an error is raised
- **nocreate** (*`bool, optional`*) – If `True` then creation of output directories is not even attempted

Returns

- **objname** (*str*) – The human readable object name based on the spectrum
- **dirname** (*str*) – The output directory name created if successful

See also:

`make_outdirs()`

`WDmodel.io.`**write_fit_inputs**(*spec*, *phot*, *cont_model*, *linedata*, *continuumdata*, *rvmodel*, *covtype*, *coveps*, *phot_dispersion*, *scale_factor*, *outfile*)
Save all the inputs to the fitter to a file

This file is enough to resume the fit with the same input, redoing the output, or restoring from a failure.

Parameters

- **spec** (`numpy.recarray`) – The spectrum with `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`
- **phot** (None or `numpy.recarray`) – `None` or the photometry with `dtype=[('pb', 'str'), ('mag', '<f8'), ('mag_err', '<f8')]`
- **cont_model** (`numpy.recarray`) – The continuum model. Must have the same structure as `spec`. Produced by *`WDmodel.fit.pre_process_spectrum()`*. Used by *`WDmodel.viz`*

- **linedata** (numpy.recarray) – The observations of the spectrum corresponding to the hydrogen Balmer lines. Must have `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8'), ('line_mask', 'i4')]` Produced by *WDmodel.fit.pre_process_spectrum()* Used by *WDmodel.viz*
- **continuumdata** (numpy.recarray) – Data used to generate the continuum model. Must have the same structure as `spec`. Produced by *WDmodel.fit.pre_process_spectrum()*
- **rvmodel** (`{'ccm89','od94','f99', 'custom'}`) – Parametrization of the reddening law. Used to initialize *WDmodel.WDmodel.WDmodel()* instance.
- **covtype** (`{'Matern32', 'SHO', 'Exp', 'White'}`) – stationary kernel type used to parametrize the covariance in *WDmodel.covariance.WDmodel_CovModel*
- **coveps** (*float*) – If `covtype` is `'Matern32'` a celerite.terms.Matern32Term is used to approximate a Matern32 kernel with precision *coveps*.
- **phot_dispersion** (*float, optional*) – Excess photometric dispersion to add in quadrature with the photometric uncertainties `phot.mag_err` in `WDmodel.likelihood.WDmodel_Likelihood`.
- **scale_factor** (*float*) – Factor by which the flux must be scaled. Critical to getting the right uncertainties.
- **outfile** (*str*) – Output HDF5 filename

Notes

The outputs are stored in a HDF5 file with groups

- `spec` - storing the spectrum and `scale_factor`
- `cont_model` - stores the continuum model
- `linedata` - stores the hydrogen Balmer line data
- `continuumdata` - stores the data used to generate `cont_model`
- `fit_config` - stores `covtype`, `coveps` and `rvmodel` as attributes
- `phot` - only created if `phot` is not `None`, stores `phot`, `phot_dispersion`

`WDmodel.io.`**write_full_model**(*full_model*, *outfile*)

Write the full SED model to an output file.

Parameters

- **full_model** (numpy.recarray) – The SED model with `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`
- **outfile** (*str*) – Output HDF5 SED model filename

Notes

The output is written into a group `model` with datasets

- `wave` : array-like - the SED model wavelength
- `flux` : array-like - the SED model flux
- `flux_err` : array-like - the SED model flux uncertainty

WDmodel.io.**write_params**(*params*, *outfile*)
Dumps the parameter dictionary params to a JSON file

Parameters

- **params** (*dict*) – A parameter dict such as that produced by *WDmodel.io.read_params()*
- **outfile** (*str*) – Output filename to save the parameter dict as a JSON file.

Notes

params is a dict the parameter names, as defined with `WDmodel.io._PARAMETER_NAMES` as keys

Each key must have a dictionary with keys:

- `value` : value
- `fixed` : a bool specifying if the parameter is fixed (`True`) or allowed to vary (`False`)
- `scale` : a scale parameter used to set the step size in this dimension
- `bounds` : An upper and lower limit on parameter values

Any extra keys are simply written as-is JSON doesn't preserve ordering necessarily. This is imposed by *WDmodel.io.read_params()*

See also:

WDmodel.io.read_params()

WDmodel.io.**write_phot_model**(*phot*, *model_mags*, *outfile*)
Write the photometry, model photometry and residuals to an output file.

Parameters

- **phot** (None or `numpy.recarray`) – `None` or the photometry with `dtype=[('pb', 'str'), ('mag', '<f8'), ('mag_err', '<f8')]`
- **model_mags** (None or `numpy.recarray`) – The model magnitudes. Has `dtype=[('pb', 'str'), ('mag', '<f8')]`
- **outfile** (*str*) – Output space-separated text filename

Notes

The data is saved to a space-separated ASCII text file with 6 decimal places of precision.

The order of the columns is

- `pb` : array-like - the observation's passband
- `mag` : array-like - the observed magnitude
- `mag_err` : array-like - the observed magnitude uncertainty
- `model_mag` : array-like - the model magnitude
- `res_mag` : array-like - the magnitude residual

WDmodel.io.**write_spectrum_model**(*spec*, *model_spec*, *outfile*)
Write the spectrum and the model spectrum and residuals to an output file.

Parameters

- **spec** (numpy.recarray) – The spectrum with dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]
- **model_spec** (numpy.recarray) – The model spectrum. Has dtype=[('wave', '<f8'), ('flux', '<f8'), ('norm_flux', '<f8'), ('flux_err', '<f8')]
- **outfile** (*str*) – Output space-separated text filename

Notes

The data is saved to a space-separated ASCII text file with 8 decimal places of precision.

The order of the columns is

- wave : array-like - the spectrum wavelength
- flux : array-like - the observed flux
- flux_err : array-like - the observed flux uncertainty
- norm_flux : array-like - the model flux without the Gaussian process covariance model
- model_flux : array-like - the model flux
- model_flux_err : array-like - the model flux uncertainty
- res_flux : array-like - the flux residual

WDmodel.likelihood module

Classes defining the likelihood and the posterior probability of the model given the data

class WDmodel.likelihood.**WDmodel_Posterior**(*spec*, *phot*, *model*, *covmodel*, *pbs*, *lnlike*, *pixel_scale=1.0*, *phot_dispersion=0.0*)

Bases: object

Classes defining the posterior probability of the model given the data

An instance of this class is used to store the data and model, and evaluate the likelihood and prior to compute the posterior.

Parameters

- **spec** (numpy.recarray) – The spectrum with dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]
- **phot** (None or numpy.recarray) – The photometry with dtype=[('pb', 'str'), ('mag', '<f8'), ('mag_err', '<f8')]
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator
- **covmodel** (*WDmodel.covariance.WDmodel_CovModel* instance) – The parametrized model for the covariance of the spectrum spec
- **pbs** (*dict*) – Passband dictionary containing the passbands corresponding to phot.pb and generated by *WDmodel.passband.get_pbmodel()*.
- **lnlike** (WDmodel_Likelihood instance) – Instance of the likelihood function class, such as that produced by *WDmodel.likelihood.setup_likelihood()*

- **pixel_scale** (*float, optional*) – Jacobian of the transformation between wavelength in Angstrom and pixels. In principle, this should be a vector, but virtually all spectral reduction packages resample the spectrum onto a uniform wavelength scale that is close to the native pixel scale of the spectrograph. Default is `1.`
- **phot_dispersion** (*float, optional*) – Excess photometric dispersion to add in quadrature with the photometric uncertainties `phot.mag_err`. Use if the errors are grossly underestimated. Default is `0.`

spec
The spectrum with `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`

Type `numpy.recarray`

wave_scale
length of the wavelength array `wave` in Angstroms

Type float

phot
The photometry with `dtype=[('pb', 'str'), ('mag', '<f8'), ('mag_err', '<f8')]`

Type None or `numpy.recarray`

model
The DA White Dwarf SED model generator

Type *WDmodel.WDmodel.WDmodel* instance

covmodel
The parametrized model for the covariance of the spectrum `spec`

Type *WDmodel.covariance.WDmodel_CovModel* instance

pbs
Passband dictionary containing the passbands corresponding to `phot.pb` and generated by *WDmodel.passband.get_pbmodel()*.

Type dict

_lnlike
Instance of the likelihood function class, such as that produced by *WDmodel.likelihood.setup_likelihood()*

Type `WDmodel_Likelihood` instance

pixel_scale
Jacobian of the transformation between wavelength in Angstrom and pixels. In principle, this should be a vector, but virtually all spectral reduction packages resample the spectrum onto a uniform wavelength scale that is close to the native pixel scale of the spectrograph. Default is `1.`

Type float

phot_dispersion
Excess photometric dispersion to add in quadrature with the photometric uncertainties `phot.mag_err`. Use if the errors are grossly underestimated. Default is `0.`

Type float, optional

p0
initial values of all the model parameters, including fixed parameters

Type dict

Returns **lnpost** – It is this instance that is passed to the samplers/optimizers in the *WDmodel.fit* module. Those methods evaluate the posterior probability of the model parameters given the data.

Return type *WDmodel_Posterior* instance

Notes

Wraps `celerite.modeling.Model.log_prior()` which imposes a boundscheck and returns `-inf`. This is not an issue as the samplers used in the methods in *WDmodel.fit*.

__call__(*theta*, *prior=False*, *likelihood=False*)
Evalulates the log posterior of the model parameters given the data

Parameters

- **theta** (*array-like*) – Vector of the non-frozen model parameters. The order of the parameters is defined by `WDmodel_Likelihood.parameter_names`.
- **prior** (*bool, optional*) – Only return the value of the log prior given the model parameters
- **likelihood** (*bool, optional*) – Only return the value of the log likelihood given the model parameters if the prior is finite

Returns **lnpost** – the log posterior of the model parameters given the data

Return type float

__init__(*spec*, *phot*, *model*, *covmodel*, *pbs*, *lnlike*, *pixel_scale=1.0*, *phot_dispersion=0.0*)
Initialize self. See help(type(self)) for accurate signature.

_lnprior()
Evalulates the log likelihood of the model parameters given the data.

Implements an lnprior function which imposes weakly informative priors on the model parameters.

Parameters **theta** (*array-like*) – Vector of the non-frozen model parameters. The order of the parameters is defined by `WDmodel_Likelihood.parameter_names`.

Returns **lnprior** – the log likelihood of the model parameters given the data

Return type float

Notes

The prior on `av` is the 'glos' prior

The prior on `rv` is a Gaussian with mean 3.1 and standard deviation 0.18. This is adopted from Schlafly et al., 2014 PS1 analysis. Note that they report 3.31, but they aren't really measuring E(B-V) with PS1. Their sigma should be consistent despite the different filter set.

The prior on `fsig` and `fw` - the fractional amplitudes of the non-trivial stationary and white components of the kernel used to parametrize the covariance is half-Cauchy since we don't want it to be less than zero

There is no explicit prior on `tau` i.e. a tophat prior, defined by the bounds

The `fwhm` has a lower bound set at the value below which the spectrum isn't being convolved anymore. We never run into this bound since real spectra have physical instrumental broadening. This prevents `fwhm` from going to zero for fitting poorly simulated spectra generated from simply resampling the model grid.

The prior on all other parameters are broad Gaussians

Wraps `celerite.modeling.Model.log_prior()` which imposes a boundscheck and returns `-inf`. This is not an issue as the samplers used in the methods in *WDmodel.fit*.

lnlike(*theta*)
Evalulates the log likelihood of the model parameters given the data.

Convenience function that can return the value of the likelihood even if the prior is not finite unlike *WDmodel_Posterior.__call__()* for debugging.

Parameters **theta** (*array-like*) – Vector of the non-frozen model parameters. The order of the parameters is defined by `WDmodel_Likelihood.parameter_names`.

Returns **lnlike** – the log likelihood of the model parameters given the data

Return type float

lnprior(*theta*)
Evalulates the log prior of the model parameters.

Convenience function that can return the value of the prior defined to make the interface consistent with the *WDmodel_Posterior.lnlike()* method. Just a thin wrapper around *WDmodel_Posterior._lnprior()* which is what is actually evalulated by *WDmodel_Posterior.__call__()*.

Parameters **theta** (*array-like*) – Vector of the non-frozen model parameters. The order of the parameters is defined by `WDmodel_Likelihood.parameter_names`.

Returns **lnprior** – the log prior of the model parameters

Return type float

`WDmodel.likelihood.`**setup_likelihood**(*params*)
Setup the form of the likelihood of the data given the model.

Parameters **params** (*dict*) – A parameter dictionary used to configure the `WDmodel_Likelihood` instance. The format of the dict is defined by *WDmodel.io.read_params()*.

Returns **lnlike** – An instance of the likelihood function class.

Return type `WDmodel.likelihood.WDmodel_Likelihood`

WDmodel.main module

The WDmodel package is designed to infer the SED of DA white dwarfs given spectra and photometry. This main module wraps all the other modules, and their classes and methods to implement the alogrithm.

`WDmodel.main.`**main**(*inargs=None*)
Entry point for the *WDmodel* fitter package.

Parameters **inargs** (*dict, optional*) – Input arguments to configure the fit. If not specified `sys.argv` is used. inargs must be parseable by *WDmodel.io.get_options()*.

Raises **RuntimeError** – If user attempts to resume the fit without having run it first

Notes

The package is structured into several modules and classes

Module	Model Component
WDmodel.io	I/O methods
WDmodel.WDmodel.WDmodel	SED generator
WDmodel.passband	Throughput model
WDmodel.covariance.WDmodel_CovModel	Noise model
WDmodel.likelihood.WDmodel_Likelihood	Likelihood function
WDmodel.likelihood.WDmodel_Posterior	Posterior function
WDmodel.fit	"Fitting" methods
WDmodel.viz	Viz methods

This method implements our algorithm to infer the DA White Dwarf properties and construct the SED model given the data using the methods and classes listed above. Once the data is read, the model is configured, and the liklihood and posterior functions constructed, the fitter methods evaluate the model parameters given the data, using the samplers in emcee. *WDmodel.mossampler* provides an overloaded emcee.PTSampler with a more reliable auto-correlation estimate. Finally, the result is output along with various plots.

WDmodel.main.**mpi_excepthook**(*excepttype*, *exceptvalue*, *traceback*)

Overload sys.excepthook() when using mpi4py.MPI to terminate all MPI processes when an Exception is raised.

WDmodel.mossampler module

Overridden PTSampler with random Gibbs selection, more-reliable acor.

Original Author: James Guillochon for the mosfit package

Modified to update kwargs, docstrings for full compatibility with PTSampler by G. Narayan

WDmodel.passband module

Instrumental throughput models and calibration and synthetic photometry routines

WDmodel.passband.**chop_syn_spec_pb**(*spec*, *model_mag*, *pb*, *model*)

Trims the pysynphot bandpass pb to non-zero throughput, computes the zeropoint of the passband given the SED spec, and model magnitude of spec in the passband

Parameters

- **spec** (numpy.recarray) – The spectrum. Typically a standard which has a known model_mag. This can be a real source such as Vega, BD+174708, or one of the three CALSPEC standards, or an idealized synthetic source such as AB. Must have dtype=[('wave', '<f8'), ('flux', '<f8')]
- **model_mag** (*float*) – The apparent magnitude of the spectrum through the passband. The difference between the apparent magnitude and the synthetic magnitude is the synthetic zeropoint.
- **pb** (numpy.recarray) – The passband transmission. Must have dtype=[('wave', '<f8'), ('throughput', '<f8')]
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator

Returns

- **outpb** (numpy.recarray) – The passband transmission with zero throughput entries trimmed. Has dtype=[('wave', '<f8'), ('throughput', '<f8')]
- **outzp** (*float*) – The synthetic zeropoint of the passband pb such that the source with spectrum spec will have apparent magnitude model_mag through pb. With the synthetic zeropoint computed, the synthetic magnitude of any source can be converted into an apparent magnitude and can be passed to *WDmodel.passband.synphot()*.

See also:

WDmodel.passband.interp_passband(), *WDmodel.passband.synphot()*

WDmodel.passband.**get_model_synmags**(*model_spec*, *pbs*, *mu=0.0*)

Computes the synthetic magnitudes of spectrum model_spec through the passbands pbs, and optionally applies a common offset, mu

Wrapper around *WDmodel.passband.synphot()*.

Parameters

- **spec** (numpy.recarray) – The spectrum. Must have dtype=[('wave', '<f8'), ('flux', '<f8')]
- **pbs** (*dict*) – Passband dictionary containing the passbands corresponding to phot.pb‘ and generated by *WDmodel.passband.get_pbmodel()*.
- **mu** (*float, optional*) – Common achromatic photometric offset to apply to the synthetic magnitudes in al the passbands. Would be equal to the distance modulus if model_spec were normalized to return the true absolute magnitude of the source.

Returns **model_mags** – The model magnitudes. Has dtype=[('pb', 'str'), ('mag', '<f8')]

Return type None or numpy.recarray

WDmodel.passband.**get_pbmodel**(*pbnames*, *model*, *pbfile=None*, *mag_type=None*, *mag_zero=0.0*)

Converts passband names pbnames into passband models based on the mapping of name to pysynphot obsmode strings in pbfile.

Parameters

- **pbnames** (*array-like*) – List of passband names to get throughput models for Each name is resolved by first looking in pbfile (if provided) If an entry is found, that entry is treated as an obsmode for pysynphot. If the entry cannot be treated as an obsmode, we attempt to treat as an ASCII file. If neither is possible, an error is raised.
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator All the passbands are interpolated onto the wavelengths of the SED model.
- **pbfile** (*str, optional*) – Filename containing mapping between pbnames and pysynphot obsmode string, as well as the standard that has 0 magnitude in the system (either ''Vega" or ''AB''). The obsmode may also be the fullpath to a file that is readable by pysynphot
- **mag_type** (*str, optional*) – One of ''vegamag" or ''abmag" Used to specify the standard that has mag_zero magnitude in the passband. If magsys is specified in pbfile, that overrides this option. Must be the same for all passbands listed in pbname that do not have magsys specified in pbfile If pbnames require multiple mag_type, concatentate the output.
- **mag_zero** (*float, optional*) – Magnitude of the standard in the passband If magzero is specified in pbfile, that overrides this option. Must be the same for all

passbands listed in pbname that do not have magzero specified in pbfile If pbnames require multiple mag_zero, concatentate the output.

Returns **out** – Output passband model dictionary. Has passband name pb from pbnames as key.

Return type dict

Raises **RuntimeError** – If a bandpass cannot be loaded

Notes

Each item of out is a tuple with

- pb : (numpy.recarray) The passband transmission with zero throughput entries trimmed. Has dtype=[('wave', '<f8'), ('throughput', '<f8')]
- transmission : (array-like) The non-zero passband transmission interpolated onto overlapping model wavelengths
- ind : (array-like) Indices of model wavelength that overlap with this passband
- zp : (float) mag_type zeropoint of this passband
- avgwave : (float) Passband average/reference wavelength

pbfile must be readable by *WDmodel.io.read_pbmap()* and must return a numpy.recarray with‘‘dtype=[(‘pb’, ‘str’),(‘obsmode’, ‘str’)]‘‘

If there is no entry in pbfile for a passband, then we attempt to use the passband name pb as obsmode string as is.

Trims the bandpass to entries with non-zero transmission and determines the VEGAMAG/ABMAG zeropoint for the passband - i.e. zp that gives mag_Vega/AB=mag_zero in all passbands.

See also:

WDmodel.io.read_pbmap(), *WDmodel.passband.chop_syn_spec_pb()*

WDmodel.passband.**interp_passband**(*wave*, *pb*, *model*)

Find the indices of the wavelength array wave, that overlap with the passband pb and interpolates the passband onto the wavelengths.

Parameters

- **wave** (*array-like*) – The wavelength array. Must satisfy *WDmodel.WDmodel.WDmodel._wave_test()*
- **pb** (numpy.recarray) – The passband transmission. Must have dtype=[('wave', '<f8'), ('throughput', '<f8')]
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator

Returns

- **transmission** (*array-like*) – The transmission of the passband interpolated on to overlapping elements of wave
- **ind** (*array-like*) – Indices of wavelength wave that overlap with the passband pb. Produced by *WDmodel.WDmodel.WDmodel._get_indices_in_range()* Satisfies transmission.shape == wave[ind].shape

Notes

The passband `pb` is interpolated on to the wavelength arrray `wave`. `wave` is typically the wavelengths of a spectrum, and have much better sampling than passband transmission curves. Only the wavelengths `wave` that overlap the passband are taken, and the passband transmission is then linearly interpolated on to these wavelengths. This prescription has been checked against `pysynphot` to return synthetic magnitudes that agree to be `< 1E-6`, while *`WDmodel.passband.synphot()`* is very significantly faster than `pysynphot.observation.Observation.effstim()`.

`WDmodel.passband.`**`synflux`**(*spec*, *ind*, *pb*)

Compute the synthetic flux of spectrum `spec` through passband `pb`

Parameters

- **`spec`** (`numpy.recarray`) – The spectrum. Must have `dtype=[('wave', '<f8'), ('flux', '<f8')]`
- **`ind`** (*`array-like`*) – Indices of spectrum `spec` that overlap with the passband `pb`. Can be produced by *`WDmodel.passband.interp_passband()`*
- **`pb`** (*`array-like`*) – The passband transmission. Must satisfy `pb.shape == spec[ind].flux.shape`

Returns **flux** – The normalized flux of the spectrum through the passband

Return type float

Notes

The passband is assumed to be dimensionless photon transmission efficiency.

Routine is intended to be a mch faster implementation of `pysynphot.observation.Observation.effstim()`, since it is called over and over by the samplers as a function of model parameters.

Uses `numpy.trapz()` for interpolation.

See also:

`WDmodel.passband.interp_passband()`

`WDmodel.passband.`**`synphot`**(*spec*, *ind*, *pb*, *zp=0.0*)

Compute the synthetic magnitude of spectrum `spec` through passband `pb`

Parameters

- **`spec`** (`numpy.recarray`) – The spectrum. Must have `dtype=[('wave', '<f8'), ('flux', '<f8')]`
- **`ind`** (*`array-like`*) – Indices of spectrum `spec` that overlap with the passband `pb`. Can be produced by *`WDmodel.passband.interp_passband()`*
- **`pb`** (*`array-like`*) – The passband transmission. Must satisfy `pb.shape == spec[ind].flux.shape`
- **`zp`** (*`float, optional`*) – The zeropoint to apply to the synthetic flux

Returns **mag** – The synthetic magnitude of the spectrum through the passband

Return type float

See also:

`WDmodel.passband.synflux()`, *`WDmodel.passband.interp_passband()`*

WDmodel.viz module

Routines to visualize the DA White Dwarf model atmosphere fit

WDmodel.viz.**plot_mcmc_line_fit**(*spec*, *linedata*, *model*, *cont_model*, *draws*, *balmer=None*)

Plot a comparison of the normalized hydrogen Balmer lines of the spectrum and model

Note that we fit the full spectrum, not just the lines. The lines are extracted using a coarse continuum fit in *WDmodel.fit.pre_process_spectrum()*. This fit is purely cosmetic and in no way contributes to the likelihood. It's particularly useful to detect small velocity offsets or wavelength calibration errors.

Parameters

- **spec** (numpy.recarray) – The spectrum. Must have `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`
- **linedata** (numpy.recarray) – The observations of the spectrum corresponding to the hydrogen Balmer lines. Must have `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8'), ('line_mask', 'i4'), ('line_ind', 'i4')]`
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator
- **cont_model** (numpy.recarray) – The continuuum model. Must have the same structure as `spec` Produced by *WDmodel.fit.pre_process_spectrum()*
- **draws** (*array-like*) – produced by *plot_mcmc_spectrum_fit()* - see notes for content.
- **balmer** (*array-like, optional*) – list of Balmer lines to plot - elements must be in range `[1, 6]` These correspond to the lines defined in *WDmodel.WDmodel.WDmodel._lines*. Default is `range(1, 7)`

Returns

- **fig** (matplotlib.figure.Figure instance) – The output figure containing the line profile plot
- **fig2** (matplotlib.figure.Figure instance) – The output figure containing histograms of the line residuals

See also:

WDmodel.viz.plot_mcmc_spectrum_fit()

WDmodel.viz.**plot_mcmc_model**(*spec*, *phot*, *linedata*, *scale_factor*, *phot_dispersion*, *objname*, *outdir*, *specfile*, *model*, *covmodel*, *cont_model*, *pbs*, *params*, *param_names*, *samples*, *samples_lnprob*, *covtype='Matern32'*, *balmer=None*, *ndraws=21*, *everyn=1*, *savefig=False*)

Make all the plots to visualize the full fit of the DA White Dwarf data

Wraps *plot_mcmc_spectrum_fit()*, *plot_mcmc_photometry_res()*, *plot_mcmc_spectrum_nogp_fit()*, *plot_mcmc_line_fit()* and corner.corner() and saves all the plots to a combined PDF, and optionally individual PDFs.

Parameters

- **spec** (numpy.recarray) – The spectrum. Must have `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`
- **phot** (None or numpy.recarray) – The photometry. Must have `dtype=[('pb', 'str'), ('mag', '<f8'), ('mag_err', '<f8')]`

- **linedata** (numpy.recarray) – The observations of the spectrum corresponding to the hydrogen Balmer lines. Must have `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8'), ('line_mask', 'i4'), ('line_ind', 'i4')]`
- **scale_factor** (*float*) – factor by which the flux was scaled for y-axis label
- **phot_dispersion** (*float, optional*) – Excess photometric dispersion to add in quadrature with the photometric uncertainties `phot.mag_err`. Use if the errors are grossly underestimated. Default is `0.`
- **objname** (*str*) – object name - used to title plots
- **outdir** (*str*) – controls where the plot is written out if `savefig=True`
- **specfile** (*str*) – Used in the title, and to set the name of the `outfile` if `savefig=True`
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator
- **covmodel** (*WDmodel.covariance.WDmodel_CovModel* instance) – The parametrized model for the covariance of the spectrum `spec`
- **cont_model** (numpy.recarray) – The continuuum model. Must have the same structure as `spec` Produced by *WDmodel.fit.pre_process_spectrum()*
- **pbs** (*dict*) – Passband dictionary containing the passbands corresponding to `phot.pb` and generated by *WDmodel.passband.get_pbmodel()*.
- **params** (*dict*) – dictionary of parameters with keywords `value`, `fixed`, `scale`, `bounds` for each. Same format as returned from *WDmodel.io.read_params()*
- **param_names** (*array-like*) – Ordered list of free parameter names
- **samples** (*array-like*) – Samples from the flattened Markov Chain with shape `(N, len(param_names))`
- **samples_lnprob** (*array-like*) – Log Posterior corresponding to `samples` from the flattened Markov Chain with shape `(N,)`
- **covtype** (`{'Matern32', 'SHO', 'Exp', 'White'}`) – stationary kernel type used to parametrize the covariance in *WDmodel.covariance.WDmodel_CovModel*
- **balmer** (*array-like, optional*) – list of Balmer lines to plot - elements must be in range `[1, 6]` These correspond to the lines defined in *WDmodel.WDmodel.WDmodel._lines*. Default is `range(1, 7)`
- **ndraws** (*int, optional*) – Number of draws to make from the Markov Chain to overplot. Higher numbers provide a better sense of the uncertainty in the model at the cost of speed and a larger, slower to render output plot.
- **everyn** (*int, optional*) – If the posterior function was evaluated using only every nth observation from the data, this should be specified to visually indicate the observations used.
- **savefig** (*bool*) – if True, save the individual figures

Returns

- **model_spec** (numpy.recarray) – The model spectrum. Has `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8'), ('norm_flux', '<f8')]` and same shape as input `spec`. The `norm_flux` attribute has the model flux without the Gaussian process prediction applied.

- **SED_model** (numpy.recarray) – The SED model spectrum. Has dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]
- **model_mags** (None or numpy.recarray) – If there is observed photometry, this contains the model magnitudes. Has dtype=[('pb', 'str'), ('mag', '<f8')]

WDmodel.viz.**plot_mcmc_photometry_res**(*objname*, *phot*, *phot_dispersion*, *model*, *pbs*, *draws*)

Plot the observed DA white dwarf photometry as well as the "best-fit" model magnitudes

Parameters

- **objname** (*str*) – object name - used to title plots
- **phot** (None or numpy.recarray) – The photometry. Must have dtype=[('pb', 'str'), ('mag', '<f8'), ('mag_err', '<f8')]
- **phot_dispersion** (*float, optional*) – Excess photometric dispersion to add in quadrature with the photometric uncertainties phot.mag_err. Use if the errors are grossly underestimated. Default is 0.
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator
- **pbs** (*dict*) – Passband dictionary containing the passbands corresponding to phot.pb and generated by *WDmodel.passband.get_pbmodel()*.
- **draws** (*array-like*) – produced by *plot_mcmc_spectrum_fit()* - see notes for content.

Returns

- **fig** (matplotlib.figure.Figure instance) – The output figure
- **mag_draws** (*array-like*) – The magnitudes corresponding to the parameters draws from the Markov Chain used in fig

Notes

Each element of mag_draws contains

- wres - the difference between the observed and synthetic magnitudes
- model_mags - the model magnitudes corresponding to the current model parameters
- mu - the flux normalization parameter that must be added to the model_mags

See also:

WDmodel.viz.plot_mcmc_spectrum_fit()

WDmodel.viz.**plot_mcmc_spectrum_fit**(*spec*, *objname*, *specfile*, *scale_factor*, *model*, *covmodel*, *result*, *param_names*, *samples*, *ndraws=21*, *everyn=1*)

Plot the spectrum of the DA White Dwarf and the "best fit" model

The full fit parametrizes the covariance model using a stationary Gaussian process as defined by *WDmodel.covariance.WDmodel_CovModel*. The posterior function constructed in *WDmodel.likelihood.WDmodel_Posterior* is evaluated by the sampler in the *WDmodel.fit.fit_model()* method. The median value is reported as the best-fit value for each of the fit parameters in WDmodel.likelihood.WDmodel_Likelihood.parameter_names.

Parameters

- **spec** (numpy.recarray) – The spectrum. Must have `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`
- **objname** (*str*) – object name - used to title plots
- **outdir** (*str*) – controls where the plot is written out if `save=True`
- **specfile** (*str*) – Used in the title, and to set the name of the `outfile` if `save=True`
- **scale_factor** (*float*) – factor by which the flux was scaled for y-axis label
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator
- **covmodel** (*WDmodel.covariance.WDmodel_CovModel* instance) – The parametrized model for the covariance of the spectrum `spec`
- **result** (*dict*) – dictionary of parameters with keywords `value`, `fixed`, `scale`, `bounds` for each. Same format as returned from *WDmodel.io.read_params()*
- **param_names** (*array-like*) – Ordered list of free parameter names
- **samples** (*array-like*) – Samples from the flattened Markov Chain with shape `(N, len(param_names))`
- **ndraws** (*int, optional*) – Number of draws to make from the Markov Chain to overplot. Higher numbers provide a better sense of the uncertainty in the model at the cost of speed and a larger, slower to render output plot.
- **everyn** (*int, optional*) – If the posterior function was evaluated using only every nth observation from the data, this should be specified to visually indicate the observations used.

Returns

- **fig** (matplotlib.figure.Figure instance) – The output figure
- **draws** (*array-like*) – The actual draws from the Markov Chain used in `fig`

Notes

It's faster to draw samples from the posterior in one location, and pass along the same samples to all the methods in *WDmodel.viz*.

Consequently, most require `draws` as an input. This makes all the plots connected, and none will return if an error is thrown here, but this is the correct behavior as all of them are visualizing one aspect of the same fit.

Each element of `draws` contains

- `smoothedmod` - the model spectrum
- `wres` - the prediction from the Gaussian process
- `wres_err` - the diagonal of the covariance matrix for the prediction from the Gaussian process
- `full_mod` - the full model SED, in order to compute the synthetic photometry
- `out_draw` - the dictionary of model parameters from this draw. Same format as `result`.

`WDmodel.viz.`**plot_mcmc_spectrum_nogp_fit**(*spec*, *objname*, *specfile*, *scale_factor*, *cont_model*, *draws*, *covtype='Matern32'*, *everyn=1*)

Plot the spectrum of the DA White Dwarf and the "best fit" model without the Gaussian process

Unlike *plot_mcmc_spectrum_fit()* this version does not apply the prediction from the Gaussian process to the spectrum model to match the observed spectrum. This visualization is useful to indicate if the Gaussian process - i.e. the kernel choice `covtype` used to parametrize the covariance is - is appropriate.

Parameters

- **spec** (numpy.recarray) – The spectrum. Must have `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`
- **objname** (*str*) – object name - used to title plots
- **outdir** (*str*) – controls where the plot is written out if `save=True`
- **specfile** (*str*) – Used in the title, and to set the name of the outfile if `save=True`
- **scale_factor** (*float*) – factor by which the flux was scaled for y-axis label
- **cont_model** (numpy.recarray) – The continuuum model. Must have the same structure as `spec` Produced by *WDmodel.fit.pre_process_spectrum()*
- **draws** (*array-like*) – produced by *plot_mcmc_spectrum_fit()* - see notes for content.
- **covtype** (`{'Matern32', 'SHO', 'Exp', 'White'}`) – stationary kernel type used to parametrize the covariance in *WDmodel.covariance.WDmodel_CovModel*
- **everyn** (*int, optional*) – If the posterior function was evaluated using only every nth observation from the data, this should be specified to visually indicate the observations used.

Returns **fig** – The output figure

Return type matplotlib.figure.Figure instance

See also:

WDmodel.viz.plot_mcmc_spectrum_fit()

WDmodel.viz.**plot_minuit_spectrum_fit**(*spec*, *objname*, *outdir*, *specfile*, *scale_factor*, *model*, *result*, *save=True*)

Plot the MLE fit of the spectrum with the model, assuming uncorrelated noise.

Parameters

- **spec** (numpy.recarray) – The spectrum. Must have `dtype=[('wave', '<f8'), ('flux', '<f8'), ('flux_err', '<f8')]`
- **objname** (*str*) – object name - used to title plots
- **outdir** (*str*) – controls where the plot is written out if `save=True`
- **specfile** (*str*) – Used in the title, and to set the name of the `outfile` if `save=True`
- **scale_factor** (*float*) – factor by which the flux was scaled for y-axis label
- **model** (*WDmodel.WDmodel.WDmodel* instance) – The DA White Dwarf SED model generator
- **result** (*dict*) – dictionary of parameters with keywords `value`, `fixed`, `scale`, `bounds` for each. Same format as returned from *WDmodel.io.read_params()*
- **save** (*bool*) – if True, save the file

Returns **fig**

Return type matplotlib.figure.Figure instance

Notes

The MLE fit uses `iminuit.Minuit.migrad()` to fit the spectrum with the model. This fit doesn't try to account for the covariance in the data, and is not expected to be great - just fast, and capable of setting a reasonable initial guess. If it is apparent from the plot that this fit is very far off, refine the initial guess to the fitter.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

W

WDmodel, 13
WDmodel.covariance, 24
WDmodel.fit, 27
WDmodel.io, 34
WDmodel.likelihood, 44
WDmodel.main, 47
WDmodel.mossampler, 48
WDmodel.passband, 48
WDmodel.viz, 52
WDmodel.WDmodel, 13

INDEX

Symbols

_WDmodel__init__rvmodel() (*WDmodel.WDmodel.WDmodel method*), 15
_WDmodel__init__tlusty() (*WDmodel.WDmodel.WDmodel method*), 15
__call__() (*WDmodel.WDmodel.WDmodel method*), 15
__call__() (*WDmodel.likelihood.WDmodel_Posterior method*), 46
__init__() (*WDmodel.WDmodel.WDmodel method*), 16
__init__() (*WDmodel.covariance.WDmodel_CovModel method*), 25
__init__() (*WDmodel.likelihood.WDmodel_Posterior method*), 46
_coveps (*WDmodel.covariance.WDmodel_CovModel attribute*), 25
_covtype (*WDmodel.covariance.WDmodel_CovModel attribute*), 25
_custom_extinction() (*WDmodel.WDmodel.WDmodel method*), 16
_errscale (*WDmodel.covariance.WDmodel_CovModel attribute*), 25
_extract_from_indices() (*WDmodel.WDmodel.WDmodel method*), 16
_extract_spectral_line() (*WDmodel.WDmodel.WDmodel method*), 16
_flux (*WDmodel.WDmodel.WDmodel attribute*), 14
_get_full_obs_model() (*WDmodel.WDmodel.WDmodel method*), 17
_get_indices_in_range() (*WDmodel.WDmodel.WDmodel class method*), 18
_get_line_indices() (*WDmodel.WDmodel.WDmodel method*), 18
_get_model() (*WDmodel.WDmodel.WDmodel method*), 19
_get_model_nosp() (*WDmodel.WDmodel.WDmodel method*), 19
_get_obs_model() (*WDmodel.WDmodel.WDmodel method*), 20
_get_red_model() (*WDmodel.WDmodel.WDmodel method*), 20
_ggrid (*WDmodel.WDmodel.WDmodel attribute*), 14
_grid_file (*WDmodel.WDmodel.WDmodel attribute*), 14
_grid_name (*WDmodel.WDmodel.WDmodel attribute*), 14
_k1 (*WDmodel.covariance.WDmodel_CovModel attribute*), 25
_k2 (*WDmodel.covariance.WDmodel_CovModel attribute*), 25
_law (*WDmodel.WDmodel.WDmodel attribute*), 15
_lflux (*WDmodel.WDmodel.WDmodel attribute*), 15
_lines (*WDmodel.WDmodel.WDmodel attribute*), 14
_lnlike (*WDmodel.likelihood.WDmodel_Posterior attribute*), 45
_lnprior() (*WDmodel.likelihood.WDmodel_Posterior method*), 46
_logQ (*WDmodel.covariance.WDmodel_CovModel attribute*), 25
_lwave (*WDmodel.WDmodel.WDmodel attribute*), 14
_ndim (*WDmodel.covariance.WDmodel_CovModel attribute*), 25
_ngrav (*WDmodel.WDmodel.WDmodel attribute*), 14
_ntemp (*WDmodel.WDmodel.WDmodel attribute*), 14
_nwave (*WDmodel.WDmodel.WDmodel attribute*), 14
_read_ascii() (*in module WDmodel.io*), 34
_tgrid (*WDmodel.WDmodel.WDmodel attribute*), 14
_wave (*WDmodel.WDmodel.WDmodel attribute*), 14
_wave_test() (*WDmodel.WDmodel.WDmodel class method*), 21

B

blotch_spectrum() (*in module WDmodel.fit*), 27

C

chop_syn_spec_pb() (*in module WDmodel.passband*), 48
copy_params() (*in module WDmodel.io*), 34
covmodel (*WDmodel.likelihood.WDmodel_Posterior attribute*), 45

E

extinction() (*WDmodel.WDmodel.WDmodel method*), 21
extract_spectral_line() (*WDmodel.WDmodel.WDmodel method*), 21

F

fit_model() (*in module WDmodel.fit*), 28
fix_pos() (*in module WDmodel.fit*), 29

G

get_filepath() (*in module WDmodel.io*), 34
get_fit_params_from_samples() (*in module WDmodel.fit*), 30
get_model() (*WDmodel.WDmodel.WDmodel method*), 22
get_model_synmags() (*in module WDmodel.passband*), 49
get_obs_model() (*WDmodel.WDmodel.WDmodel method*), 22
get_options() (*in module WDmodel.io*), 35
get_outfile() (*in module WDmodel.io*), 35
get_params_from_argparse() (*in module WDmodel.io*), 35
get_pbmodel() (*in module WDmodel.passband*), 49
get_phot_for_obj() (*in module WDmodel.io*), 36
get_pkgfile() (*in module WDmodel.io*), 36
get_red_model() (*WDmodel.WDmodel.WDmodel method*), 23
get_spectrum_resolution() (*in module WDmodel.io*), 37
getgp() (*WDmodel.covariance.WDmodel_CovModel method*), 25

H

hyper_param_guess() (*in module WDmodel.fit*), 30

I

interp_passband() (*in module WDmodel.passband*), 50

L

lnlike() (*WDmodel.likelihood.WDmodel_Posterior method*), 47
lnlikelihood() (*WDmodel.covariance.WDmodel_CovModel method*), 26
lnprior() (*WDmodel.likelihood.WDmodel_Posterior method*), 47

M

main() (*in module WDmodel.main*), 47
make_outdirs() (*in module WDmodel.io*), 37
model (*WDmodel.likelihood.WDmodel_Posterior attribute*), 45
mpi_excepthook() (*in module WDmodel.main*), 48

O

orig_cut_lines() (*in module WDmodel.fit*), 31

P

p0 (*WDmodel.likelihood.WDmodel_Posterior attribute*), 45
pbs (*WDmodel.likelihood.WDmodel_Posterior attribute*), 45
phot (*WDmodel.likelihood.WDmodel_Posterior attribute*), 45
phot_dispersion (*WDmodel.likelihood.WDmodel_Posterior attribute*), 45
pixel_scale (*WDmodel.likelihood.WDmodel_Posterior attribute*), 45
plot_mcmc_line_fit() (*in module WDmodel.viz*), 52
plot_mcmc_model() (*in module WDmodel.viz*), 52
plot_mcmc_photometry_res() (*in module WDmodel.viz*), 54
plot_mcmc_spectrum_fit() (*in module WDmodel.viz*), 54
plot_mcmc_spectrum_nogp_fit() (*in module WDmodel.viz*), 55
plot_minuit_spectrum_fit() (*in module WDmodel.viz*), 56
polyfit_continuum() (*in module WDmodel.fit*), 32
pre_process_spectrum() (*in module WDmodel.fit*), 32
predict() (*WDmodel.covariance.WDmodel_CovModel method*), 26

Q

quick_fit_spec_model() (*in module WDmodel.fit*), 33

R

read_fit_inputs() (*in module WDmodel.io*), 38
read_full_model() (*in module WDmodel.io*), 38
read_mcmc() (*in module WDmodel.io*), 39
read_model_grid() (*in module WDmodel.io*), 39
read_params() (*in module WDmodel.io*), 40
read_pbmap() (*in module WDmodel.io*), 40
read_phot() (*in module WDmodel.io*), 40
read_reddening() (*in module WDmodel.io*), 40
read_spec() (*in module WDmodel.io*), 40
read_spectable() (*in module WDmodel.io*), 41
rebin_spec_by_int_factor() (*in module WDmodel.fit*), 33

reddening() (*WDmodel.WDmodel.WDmodel method*), 24

S

set_objname_outdir_for_specfile() (*in module WDmodel.io*), 41
setup_likelihood() (*in module WDmodel.likelihood*), 47
spec (*WDmodel.likelihood.WDmodel_Posterior attribute*), 45
synflux() (*in module WDmodel.passband*), 51
synphot() (*in module WDmodel.passband*), 51

W

wave_scale (*WDmodel.likelihood.WDmodel_Posterior attribute*), 45
WDmodel (*class in WDmodel.WDmodel*), 13
WDmodel (*module*), 1, 13
WDmodel.covariance (*module*), 24
WDmodel.fit (*module*), 27
WDmodel.io (*module*), 34
WDmodel.likelihood (*module*), 44
WDmodel.main (*module*), 47
WDmodel.mossampler (*module*), 48
WDmodel.passband (*module*), 48
WDmodel.viz (*module*), 52
WDmodel.WDmodel (*module*), 13
WDmodel_CovModel (*class in WDmodel.covariance*), 24
WDmodel_Posterior (*class in WDmodel.likelihood*), 44
write_fit_inputs() (*in module WDmodel.io*), 41
write_full_model() (*in module WDmodel.io*), 42
write_params() (*in module WDmodel.io*), 43
write_phot_model() (*in module WDmodel.io*), 43
write_spectrum_model() (*in module WDmodel.io*), 43